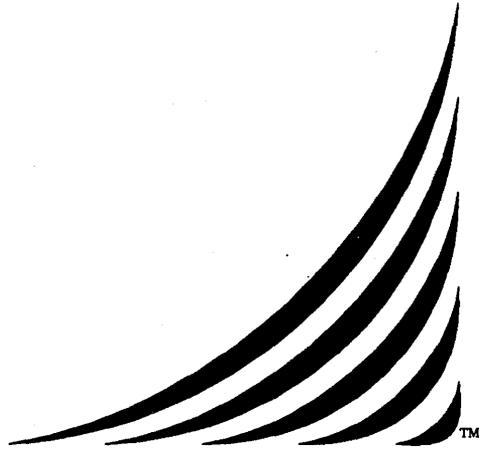


V12



MAINSAIL[®]

Documentation User's Guide, Overview,
and Master Index



MAINSAIL® Documentation User's Guide

MAINSAIL Overview

Master Index

24 March 1989



Copyright (c) 1981, 1983, 1984, 1985, 1986, 1987, 1989, by XIDAK, Inc., Menlo Park, California.

The software described herein is the property of XIDAK, Inc., with all rights reserved, and is a confidential trade secret of XIDAK. The software described herein may be used only under license from XIDAK.

MAINSAIL is a registered trademark of XIDAK, Inc. MAINDEBUG, MAINEDIT, MAINMEDIA, MAINPM, Structure Blaster, TDB, and SQL/T are trademarks of XIDAK, Inc.

CONCENTRIX is a trademark of Alliant Computer Systems Corporation.

Amdahl, Universal Time-Sharing System, and UTS are trademarks of Amdahl Corporation.

Aegis, Apollo, DOMAIN, GMR, and GPR are trademarks of Apollo Computer Inc.

UNIX and UNIX System V are trademarks of AT&T.

DASHER, DG/UX, ECLIPSE, ECLIPSE MV/4000, ECLIPSE MV/8000, ECLIPSE MV/10000, and ECLIPSE MV/20000 are trademarks of Data General Corporation.

DEC, PDP, TOPS-10, TOPS-20, VAX-11, VAX, MicroVAX, MicroVMS, ULTRIX-32, and VAX/VMS are trademarks of Digital Equipment Corporation.

EMBOS and ELXSI System 6400 are trademarks of ELXSI, Inc.

The KERMIT File Transfer Protocol was named after the star of THE MUPPET SHOW television series. The name is used by permission of Henson Associates, Inc.

HP-UX and Vectra are trademarks of Hewlett-Packard Company.

Intel is a trademark of Intel Corporation.

CLIPPER, CLIX, Intergraph, InterPro 32, and InterPro 32C are trademarks of Intergraph Corporation.

System/370, VM/SP CMS, and CMS are trademarks of International Business Machines Corporation.

MC68000, M68000, MC68020, and MC68881 are trademarks of Motorola Semiconductor Products Inc.

ROS and Ridge 32 are trademarks of Ridge Computers.

SPARC, Sun Microsystems, Sun Workstation, and the combination of Sun with a numeric suffix are trademarks of Sun Microsystems, Inc.

WIN/TCP is a trademark of The Wollongong Group, Inc.

WY-50, WY-60, WY-75, and WY-100 are trademarks of Wyse Technology.

Some XIDAK documentation is published in the typefaces "Times" and "Helvetica", used by permission of Apple Computer, Inc., under its license with the Allied Corporation. Helvetica and Times are trademarks of the Allied Corporation, valid under applicable law.

The use herein of any of the above trademarks does not create any right, title, or interest in or to the trademarks.

Table of Contents

1. Documentation Overview	1
1.1. Introductory Documents and Language Documentation	1
1.2. MAINSAIL Environment Documentation	1
1.3. System-Specific Documentation	2
1.4. Additional Documents Available from XIDAK	2
1.5. Online Documentation	2
1.6. Recommended Reading Order	2
I. MAINSAIL(R) Overview	7
2. Introduction	8
II. The MAINSAIL Language	9
3. Introduction to the MAINSAIL Language	10
4. Basic Concepts	12
4.1. Character Set	12
4.2. Comments	12
4.3. Compiletime Evaluation	12
4.4. Low-Level Storage Manipulation	12
4.5. Log and Command Files	12
5. Data Types	13
5.1. Boolean	13
5.2. Integer and Long Integer	13
5.3. Real and Long Real	13
5.4. Bits and Long Bits	13
5.5. String	13
5.6. Pointer	14
5.7. Address	14
5.8. Charadr	14
5.9. Conversion	15
6. Expressions	16
6.1. Constants	16
6.2. Variables	16
6.3. Procedure Expression	16

6.4.	Substrings	17
6.5.	If Expression	17
6.6.	Assignment Expression	17
6.7.	Universal Operations	17
6.8.	Comparison Operations	18
6.9.	Arithmetic Operations	18
6.10.	Bitwise Operations	18
6.11.	String Operations	19
6.12.	Operator Precedence	19
6.13.	Dotted Operators	20
7.	Statements	21
7.1.	Assignment Statement	21
7.2.	Expression Statement	21
7.3.	Procedure Statement	21
7.4.	Return Statement	21
7.5.	Begin Statement	21
7.6.	If Statement	22
7.7.	Case Statement	22
7.8.	Iterative Statement	22
7.9.	Done Statement	23
7.10.	Continue Statement	23
7.11.	Empty Statement	23
8.	Declarations	24
8.1.	Scope of Identifiers	24
8.2.	Simple Variable Declarations	24
8.3.	"OWN" Qualifier	24
9.	Arrays	25
9.1.	Array Declarations	25
9.2.	Array Allocation	25
9.3.	Array Initialization	26
9.4.	Accessing an Array Element	26
9.5.	Array Variables	26
10.	Classes and Records	27
10.1.	Records	27
10.2.	Classes	27
10.3.	Record Allocation and Disposal	27
10.4.	Classified Pointers	28
10.5.	Unclassified Pointers	28
10.6.	Accessing Fields of Records	28
10.7.	Explicit Classes in Field Variables	28
10.8.	Prefix Classes	28

11. Procedures	30
11.1. Procedure Declarations	30
11.2. Procedure Calls	30
11.3. Procedure Parameters	30
11.4. Optional Arguments	31
11.5. Repeatable Arguments	31
11.6. Forward Procedures	31
11.7. Compiletime Libraries	31
11.8. Inline Procedures	32
11.9. Generic Procedures	32
12. Modules	34
12.1. Module Allocation and Communication	35
12.2. Module Syntax	36
12.3. Combining Module Declarations in One File	37
12.4. Objmod Libraries	37
13. Macros	38
13.1. "DEFINE"	38
13.2. Macro Constants	39
13.3. Interactive Define and Redefine	39
13.4. Macro Calls	39
14. Compiler Directives and Conditional Compilation	40
14.1. "MESSAGE"	40
14.2. "SOURCEFILE"	40
14.3. Checking, Arithmetic Checking, and Optimization	40
14.4. "IFC", "THENC", "\$EFC", "ELSEC", and "ENDC"	41
14.5. "DCL"	41
14.6. "DSP"	41
14.7. "\$STYPEOF", "\$CLASSOF", "\$ISCONSTANT"	41
14.8. Scanning Directives	41
14.9. "NEEDBODY" and "NEEDANYBODIES"	42
14.10. "\$DIRECTIVE" Directives	42
15. Intmods.	43
16. Exceptions	44
16.1. Handle Statement	44
16.2. Handling Exceptions	45
16.3. Propagating Exceptions	45
16.4. Information about the Current Exception	46
16.5. Nested Exceptions.	46

17. Coroutines	47
17.1. Coroutine Implementation	47
17.2. Coroutines and Exceptions	48
18. Files	50
18.1. File I/O	50
18.2. Text Files	50
18.3. Terminal I/O	51
18.4. Memory Files and Data Sinks	51
19. Areas	52
19.1. Advantages of Areas	52
19.2. Area Facilities	52
19.3. Area Caveats	52
20. System Procedures, Macros, and Variables	54
20.1. System Procedures, Variables, and Macros Summary	54
21. Sample MAINSAIL Code	67
III. The MAINSAIL Environment	73
22. The MAINSAIL Compiler	74
22.1. Code Generators	74
22.2. Disassemblers	74
22.3. Foreign Language Interface	74
23. MAINDEBUG, the MAINSAIL Debugger	79
24. MAINEDIT, a Portable Text Editor	82
25. MAINPM, the MAINSAIL Performance Monitor	92
26. The MAINSAIL Structure Blaster	94
27. MAINSAIL STREAMS	96
IV. Master Index	97
28. Master Index	98

List of Examples

17-1.	Generator/Processor Coroutines	48
21-1.	HSHMOD Declarations	67
21-2.	HSHMOD Source Text.	68
25-1.	Timing and Statement Counts Table.	92
25-2.	Source Text with Statement Counts	93

List of Tables

1.6-1.	Reading Order for New MAINSAIL Programmers	3
1.6-2.	Introductory MAINSAIL Documents and Language Documentation	4
1.6-3.	Documentation on the MAINSAIL Environment (Combined in the "MAINSAIL Tools User's Guides", File Name "toolu.doc")	5
1.6-4.	System-Dependent MAINSAIL Documentation (Combined in the "MAINSAIL System-Specific User's Guides", File Name "osu.doc")	6
6.12-1.	Operator Precedence	19
20.1-1.	System Procedures, Macros, and Variables Summary	54
22-1.	MAINSAIL Compiler Subcommands	75
22.1-1.	Computer Systems on Which MAINSAIL Is Supported	78
23-1.	Debugging Command Summary	80
24-1.	Available Display Modules	83
26-1.	Selected Structure Blaster Procedure Headers	95

1. Documentation Overview

XIDAK, Inc., has organized the documentation on its MAINSAIL language and MAINSAIL programming environment into a series of manuals and guides. The present guide is intended both for the new user of MAINSAIL who wishes to learn how to use MAINSAIL and wants to know what to read first, and also for the experienced MAINSAIL user who needs a specific piece of information and wishes to know what manual to consult.

A summary of the MAINSAIL documentation is provided in Tables 1.6-2, 1.6-3, and 1.6-4.

1.1. Introductory Documents and Language Documentation

The documents listed in Table 1.6-2 are designed to provide information to the new user of MAINSAIL and to serve as references on the MAINSAIL language.

If you do not know anything about MAINSAIL, but are familiar with other programming languages and systems, you should consult the "MAINSAIL Overview". It provides a high-level view of the facilities available from MAINSAIL.

If you have access to a computer that runs MAINSAIL and wish to begin writing MAINSAIL code right away, you should follow the examples in the "MAINSAIL Tutorial". The first part of the "MAINSAIL Tutorial" assumes a minimum of previous programming experience, although it may be of interest to experienced programmers as well. The second part of the "MAINSAIL Tutorial" provides details on MAINSAIL's implementation and suggestions for making programs written in MAINSAIL as maintainable, efficient, and portable as possible.

The "MAINSAIL Language Manual" is a thorough but concise reference on the syntax and semantics of the MAINSAIL language. It is intended for the experienced programmer who wishes to know the precise definition of a feature of MAINSAIL. If you are just learning about MAINSAIL, you will find the "MAINSAIL Language Manual" makes more sense after you have read the "MAINSAIL Overview" or the "MAINSAIL Tutorial".

1.2. MAINSAIL Environment Documentation

The guides in Table 1.6-3 describe various programs written in MAINSAIL that constitute the MAINSAIL programming environment (often called "the MAINSAIL environment" or "the MAINSAIL system"). These are utilities that are useful in writing MAINSAIL programs; some of them (e.g., MAINEDIT, the text editor) may be used to accomplish tasks unrelated to MAINSAIL programming.

1.3. System-Specific Documentation

The guides listed in Table 1.6-4 provide information specific to the use of MAINSAIL on particular operating systems. They discuss those details of MAINSAIL that depend on the host system. Since MAINSAIL is designed for portability, these guides are small. They tell how to invoke MAINSAIL on the given operating system, and explain system-dependent features of MAINSAIL, such as MAINSAIL's interaction with the file system and MAINSAIL's handling of system-dependent exceptions.

1.4. Additional Documents Available from XIDAK

On most releases of MAINSAIL, XIDAK issues a release note that describes any changes since the previous release note.

If you are considering the purchase of a MAINSAIL system or any of its components, you should request a copy of the most recent "XIDAK Product Catalog" from XIDAK. From time to time, XIDAK also issues product data sheets and other information of interest to potential purchasers of MAINSAIL. These do not contain any technical information that is not also available in the standard documents listed in this guide.

1.5. Online Documentation

With each standard MAINSAIL system, XIDAK includes current versions of its documentation on the distribution medium. The file names under which the documents appear are listed in quotes in Tables 1.6-2, 1.6-3, and 1.6-4.

1.6. Recommended Reading Order

If you are using MAINSAIL for the first time, you should read the documents in Table 1.6-1 in the order shown.

The "MAINSAIL Overview" describes the MAINSAIL language and MAINSAIL environment to the new user. The operating-system-specific user's guide describes how to invoke MAINSAIL on your system. You need to know how to invoke MAINSAIL before starting the tutorial. The first part of the tutorial guides you step-by-step through the writing of some simple MAINSAIL programs and the use of a number of the MAINSAIL utilities.

MAINEDIT, the MAINSAIL compiler, MAINEX, CONF, MODLIB, and MAINDEBUG are among the most important components of the MAINSAIL environment. The tutorial provides an introduction to each of these, but more information is found in the appropriate user's guides.

1. MAINSAIL Overview (skim)
2. Operating-system-specific user's guide for your system (skim)
3. MAINEDIT User's Guide (if you are using MAINEDIT)
4. MAINSAIL Tutorial (first half)
5. MAINSAIL Compiler User's Guide, first two chapters
6. MAINSAIL Utilities User's Guide, chapters on MAINEX, CONF, and MODLIB
7. MAINDEBUG User's Guide (if you are using MAINDEBUG)
8. MAINSAIL Tutorial (second half)

Table 1.6-1. Reading Order for New MAINSAIL Programmers

The second part of the tutorial contains suggestions for the construction of efficient, portable MAINSAIL programs and for sophisticated use of the tools in the MAINSAIL environment. It is of interest to the programmer who has mastered the basics of the MAINSAIL language.

<u>Title ("File Name")</u>	<u>Function</u>
MAINSAIL Documentation User's Guide, MAINSAIL Overview, and Master Index ("mslov.doc")	Gives an overview of the other documents. Summary of the main features of the MAINSAIL language and programming environment, intended as an overview for those evaluating MAINSAIL or using it for the first time. Master index to all MAINSAIL documents.
MAINSAIL Tutorial ("mtut.doc")	Step-by-step instructions for and examples of writing MAINSAIL programs, and tips for writing good MAINSAIL code.
MAINSAIL Language Manual ("mlanm.doc")	Comprehensive reference on the MAINSAIL language.

Table 1.6-2. Introductory MAINSAIL Documents and Language Documentation

<u>Title</u>	<u>Function</u>
MAINSAIL Compiler User's Guide	Describes the MAINSAIL compiler and related utilities, including the MAINSAIL disassemblers.
MAINDEBUG User's Guide	Describes the MAINSAIL portable source-level debugger.
MAINEDIT User's Guide	Describes the MAINSAIL full-screen text editor.
MAINKERMIT User's Guide	Describes a MAINSAIL implementation of the KERMIT file transfer program
MAINPM User's Guide	Describes the MAINSAIL performance monitor, which allows the programmer to determine where the inefficiencies in a program lie
MAINSAIL Structure Blaster User's Guide	Describes the routines used for fast, simple input/output of MAINSAIL data structures
MAINSAIL STREAMS User's Guide	Describes STREAMS, a package for portable distributed applications
MAINSAIL Utilities User's Guide	Describes miscellaneous (but important) components of the MAINSAIL environment

Table 1.6-3. Documentation on the MAINSAIL Environment (Combined in the "MAINSAIL Tools User's Guides", File Name "toolu.doc")

The following operating-system-specific MAINSAIL user's guides are available as of March, 1989:

Title

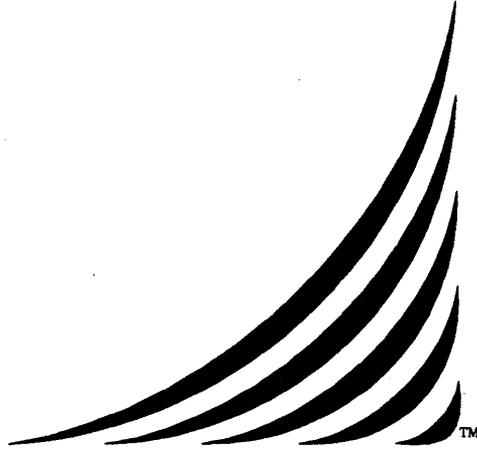
Aegis MAINSAIL User's Guide

VM/SP CMS MAINSAIL User's Guide

UNIX MAINSAIL User's Guide

VAX/VMS MAINSAIL User's Guide

Table 1.6-4. System-Dependent MAINSAIL Documentation (Combined in the "MAINSAIL System-Specific User's Guides", File Name "osu.doc")



MAINSAIL[®] Overview

24 March 1989

xitakTM

2. Introduction

The "MAINSAIL Overview" summarizes the features of the MAINSAIL language; the MAINSAIL compiler; MAINDEBUG, the MAINSAIL debugger; MAINEDIT, the MAINSAIL text editor; MAINPM, the MAINSAIL performance monitor; the MAINSAIL Structure Blaster; and MAINSAIL STREAMS. These components, together with a number of utility programs, make up the MAINSAIL system, a powerful, highly portable programming environment supported and marketed by XIDAK, Inc.

The MAINSAIL programming environment provides a complete set of tools to support the entire software development cycle. Since these tools are themselves written in MAINSAIL, they operate identically on every computing system on which MAINSAIL is supported. Programmers, once trained to use these tools, can move their program development from one computing system to another without having to learn new tools each time they move.

MAINSAIL is a sophisticated language primarily intended for large projects. Its flexible structure encourages a high-level, object-oriented programming style. Large MAINSAIL software systems are easier to write and maintain than systems written in other commercially available programming languages, both because of the variety of tools provided to support large-system development and because of the inherent clarity and power of the language itself.

The "MAINSAIL Overview" is recommended reading for new users of MAINSAIL and for potential purchasers of XIDAK products. The MAINSAIL language is described in detail in the "MAINSAIL Language Manual" and the "MAINSAIL Tutorial"; other XIDAK products have their own user's guides. Consult the "MAINSAIL Documentation User's Guide and Master Index" or a current "XIDAK Product Catalog" for more information on XIDAK's documentation products.

The features of the language and other components of the system are not described exhaustively in this document; only the most commonly used facilities have been covered.

The MAINSAIL Language

3. Introduction to the MAINSAIL Language

MAINSAIL was originally developed at Stanford University under the auspices of the SUMEX Computer Project by the founders of XIDAK, Inc. It is an ALGOL-like programming language derived from the SAIL programming language. SAIL was developed at the Artificial Intelligence Laboratory of Stanford University. MAINSAIL retains some of the most popular features of SAIL, such as variable-length strings and garbage collection, but eliminates all the machine-dependent characteristics of SAIL.

MAINSAIL was specifically designed to provide true source-level portability. The entire MAINSAIL compiler and the bulk of the runtime system are written in MAINSAIL, minimizing the effort required by XIDAK to move MAINSAIL to a new processor or operating system. MAINSAIL's unprecedented level of portability allows XIDAK to guarantee a uniform programming environment across all MAINSAIL implementations.

MAINSAIL is an extremely powerful programming language offering features not found in other popular programming languages such as C, Pascal, Modula-2, or Ada. Independently of its complete portability, MAINSAIL is an outstanding choice for most programming tasks, even those that are not intrinsically portable.

MAINSAIL provides a complete I/O interface that supports sequential and random access to files of text or data. Terminal interaction is also part of the I/O interface.

MAINSAIL provides a very powerful module facility similar to the "class" concept of SIMULA. Modules are both the unit of compilation and the unit of execution. They can be used as packaging devices or for the implementation of abstract data types. Modules can be embedded in data structures by means of pointers, and multiple copies of a module can be dynamically allocated and deallocated.

Other powerful features include dynamic arrays and records, all subject to the automatic storage reclamation strategy known as "garbage collection", which is usually available only in very-high-level languages, such as LISP.

Unlike most commercially available programming languages, MAINSAIL programs are not statically linked. A statically linked language requires all code that might be used in a particular execution to be loaded when any program written in that language begins execution. In MAINSAIL, object modules are loaded as needed; in a large system, this may result in far less code being in memory if much of the code is needed only occasionally. The code constituting a MAINSAIL "program" is therefore determined dynamically.

MAINSAIL was designed to simplify the management and maintenance of large programming projects involving many programmers. The compiler and runtime system provide extensive facilities for incremental development of components and subsequent integration of components into a complete system.

MAINSAIL is a "natural" programming language; programmers can program the way they think. Programmers need not revert to programming "tricks" or obscure coding techniques to get their jobs done. MAINSAIL's clear, clean syntax makes it easy to understand MAINSAIL programs. This dramatically reduces the resources required for both program development and program maintenance.

MAINSAIL is a broad-spectrum programming language. Its utility spans the range of applications, from systems programming, such as compilers and text editors, through scientific and technical applications, to business and financial applications. MAINSAIL runs on a diverse set of processors, from mainframes to microprocessors. Great care has been taken to provide a language definition that supports compatible implementations across such a wide range of machines. An approach that uses the same compiler, runtime system, and support software across all implementations is the key to such compatibility.

4. Basic Concepts

4.1. Character Set

MAINSAIL does not specify the exact character set of the machines on which it runs. Instead, guarantees are given that must hold for all character sets under which MAINSAIL is implemented. System procedures are provided to complement these assumptions. Predefined string constants are provided for horizontal tab, end-of-line, and end-of-page characters.

4.2. Comments

A comment starts with "#" and extends to the end of a line. Several methods are available for commenting out a large body of text; for example, a directive is provided to skip entire pages.

4.3. Compiletime Evaluation

Most expressions consisting entirely of constant operands are evaluated at compiletime. The compiler uses variable-length string representations in performing such evaluation so that any host machine limitations do not affect the precision of arithmetic results.

4.4. Low-Level Storage Manipulation

Storage is measured in "storage units" and "character units". Storage units are independent of the byte or word size of the processor on which MAINSAIL executes. Character units are always eight-bit bytes. A number of predefined identifiers specify the characteristics of the target processor, permitting low-level code to be written in a highly portable fashion.

4.5. Log and Command Files

A command file (cmdFile) and a logging file (logFile) are utilized for standard input and output. These files are normally associated with "TTY", which represents primary input and output (usually the user's terminal), but may be redirected to other files so that a program can utilize any file for what appears to the program as terminal-oriented I/O.

5. Data Types

There are eleven MAINSAIL data types. Each data type includes a "zero" value, referred to as "Zero", which is represented in memory as an all-zero bit pattern.

5.1. Boolean

True or false. Boolean Zero is false.

5.2. Integer and Long Integer

An integer is guaranteed (at least) the range -32767 through 32767 (representable in 16 bits). A long integer is guaranteed (at least) the range -2147483647 through +2147483647 (representable in 32 bits). (Long) integer Zero is 0.

5.3. Real and Long Real

A real is guaranteed a fraction of at least 6 full decimal digits, and the exponent is guaranteed to range at least from 1.0E-38 to 1.0E+38. For a long real, the fraction is guaranteed to consist of at least 11 full decimal digits, and the exponent range is at least as large as that of a real. (Long) real Zero is 0.0.

5.4. Bits and Long Bits

These data types are for representing sequences of bits. A bits consists of (at least) 16 bits and a long bits consists of (at least) 32 bits. These data types may take part in bit operations such as masking, shifting and testing. (Long) bits Zero has all bits equal to 0-bit.

5.5. String

A string represents a variable-length sequence of characters. A string variable is implemented as a descriptor that gives the current length (number of characters) of the string, and the location of the first character. MAINSAIL guarantees that a string may contain up to 32766

characters. The characters themselves usually reside in an area of memory known as "string space", where they are subject to a storage reclamation method known as "garbage collection".

A string constant is a sequence of characters enclosed in double quotes. A double quote is made part of a string constant by using two double quotes. String Zero is "", the string with no characters.

5.6. Pointer

A pointer is a data type for referencing dynamically allocated objects such as records or modules. These dynamically allocated objects are subject to garbage collection. Pointer Zero is nullPointer, which points to no object.

5.7. Address

Address is a data type for representing arbitrary memory addresses. An address can reference all data types. To access individual characters, the data type "charadr" must be used. Address Zero is nullAddress.

A classified address variable, e.g., "a" declared as "ADDRESS(c) a", can be used in field variables of the form a.f, where f is a field of the class c. This allows a class to be used as a storage template placed over memory starting at the address contained in "a".

Memory is viewed as a linear sequence of addressable cells ("storage units"). Addresses are ordered with respect to the relative position of the referenced cells. This order is used when comparing addresses.

5.8. Charadr

Charadr ("character address") is a data type for representing the location of a character. Charadr is distinct from address since some machines address words that may contain several characters. Charadr Zero is nullCharadr. Charadr provides a more primitive handling of characters than the string data type.

Charadr and address are used only by programs manipulating the contents of memory in a low-level fashion; many applications do not use address and charadr at all.

5.9. Conversion

There is no implicit data type conversion; instead, explicit system procedures are provided for data type conversions.

6. Expressions

An expression provides the means of accessing and computing values.

6.1. Constants

Constants are the predefined values for each data type, e.g., "FALSE", "37", or "3.14159". They may be represented symbolically as macro constants.

6.2. Variables

There are five kinds of variables:

- (Non-own) local variable: allocated dynamically upon procedure entry and deallocated upon procedure exit; accessible only within declaring procedure.
- Own variable: allocated dynamically upon module allocation and deallocated upon module deallocation; accessible only within module (and only within procedure if declared within procedure).
- Interface variable: allocated the same as an own variable; accessible within declaring module and from other modules.
- Subscripted variable: element of an array. All arrays are dynamically allocated.
- Field variable: field of a record, data section, or storage template. All records, data sections, and scratch memory are dynamically allocated.

6.3. Procedure Expression

A procedure expression is a procedure call used as an expression. The invoked procedure must be typed, i.e., declared as returning a value.

6.4. Substrings

"s[i TO j]" is the string consisting of characters i through j of the string s. "s[i FOR j]" is the string consisting of characters i through i+j-1 of s. The integer expressions i or j may contain the keyword "INF", which stands for the length of s.

6.5. If Expression

An If Expression selects among several possible values. It has the general form:

```
IF e1 THEN v1
EF e2 THEN v2
...
EL vn
```

where "EF" abbreviates "ELSE IF", and "EL" abbreviates "ELSE". The "EF" clauses may be omitted. All the v_i must be of the same type, and this type is the type of the If Expression.

6.6. Assignment Expression

An Assignment Expression has the form "v := e", where v is a variable and e is an expression of the same data type as v. The result is the value of e. "_" may be used in place of ":=".

6.7. Universal Operations

The following operations apply to all data types:

NOT e	IF e THEN FALSE ELSE TRUE
v := e	assignment expression
e1 OR e2	IF e1 THEN TRUE EF e2 THEN TRUE EL FALSE
e1 AND e2	IF NOT e1 THEN FALSE EF e2 THEN TRUE EL FALSE
e1 = e2	equal
e1 NEQ e2	not equal (" \neq " may be used for NEQ)

e2 is evaluated only if necessary for "OR" and "AND".

6.8. Comparison Operations

The following operations apply only to those data types that have an ordering, i.e., (long) integer, (long) real, string, address, and charadr:

e1 < e2	less than
e1 LEQ e2	less than or equal (" \leq " may be used for LEQ)
e1 > e2	greater than
e1 GEQ e2	greater than or equal (" \geq " may be used for GEQ)
e1 MIN e2	minimum
e1 MAX e2	maximum

6.9. Arithmetic Operations

The following operations apply to (long) integer and (long) real, except as otherwise specified. Both arguments must be the same type except as otherwise indicated for " \wedge ":

- e	negative of e
e1 \wedge e2	exponentiation (allowed type combinations: i \wedge i, li \wedge i, r \wedge i, lr \wedge i, r \wedge r, lr \wedge r)
e1 + e2	sum
e1 - e2	difference
e1 * e2	product
e1 / e2	(long) real quotient
e1 DIV e2	(long) integer quotient (discard remainder)
e1 MOD e2	(long) integer modulus (remainder)

"**" may be used in place of " \wedge ".

6.10. Bitwise Operations

The following operations apply to (long) bits. Both arguments must be the same type, except that e2 is an integer for "SHL" and "SHR":

e1 TST e2	TRUE if any 1-bit in e2 is a 1-bit in e1
e1 NTST e2	TRUE if no 1-bit in e2 is a 1-bit in e1
e1 TSTA e2	TRUE if all 1-bits in e2 are 1-bits in e1
e1 NTSTA e2	TRUE if not all 1-bits in e2 are 1-bits in e1
e1 IOR e2	inclusive or
e1 XOR e2	exclusive or
e1 MSK e2	Clear any bits in e1 that are 0-bits in e2
e1 CLR e2	Clear any bits in e1 that are 1-bits in e2
e1 SHL e2	shift left (" \ll " may be used in place of SHL)
e1 SHR e2	shift right (" \gg " may be used in place of SHR)
e1 ! e2	= e1 IOR e2, except ! has higher precedence

6.11. String Operations

The following operation applies to strings:

e1 & e2	concatenation: the string made up of the characters of e1 followed by the characters of e2
---------	--

6.12. Operator Precedence

Table 6.12-1 shows the precedence of the operators. Operators on the same line have equal precedence.

OR	(least precedence -- least binding)								
AND									
NOT									
=	NEQ	<	LEQ	>	GEQ	TST	NTST	TSTA	NTSTA
:=									
MIN	MAX								
+	- (binary)	IOR	XOR	MSK	CLR				
*	/	&	DIV	MOD	SHL	SHR			
!	^								
- (unary)	(most precedence -- most binding)								

Table 6.12-1. Operator Precedence

Operators of equal precedence are associated from left to right (except for assignment). The order of evaluation of the operands is in general not specified. The precedence of ":@" was chosen so that it could be used in expressions without the need for parentheses in most common cases.

6.13. Dotted Operators

The expression "v .op e" is a short form of "v := v op e", except that v is evaluated just once. For example:

$$v[i .+ 1] .+ 5$$

adds 5 to v[j], where $j = i + 1$; in addition, i is incremented. Almost all operators can be "dotted" this way.

7. Statements

This chapter describes eleven of the thirteen MAINSAIL statements; the other two, the Init and Handle Statements, are described in Chapters 9 and 16, respectively.

7.1. Assignment Statement

An Assignment Statement has the form of an Assignment Expression, except that it occurs as a statement rather than as an expression.

7.2. Expression Statement

An Expression Statement is a dotted expression used as a statement; e.g., "i .- 1" is a statement that decrements i.

7.3. Procedure Statement

A Procedure Statement has the form of a Procedure Expression, except that the procedure may be untyped. If it is typed, the result is discarded.

7.4. Return Statement

A Return Statement has the form "RETURN" for an untyped procedure, and "RETURN(e)" for a typed procedure, where the expression "e" is of the same type as declared for the procedure. It causes immediate termination of the procedure's execution, returning the specified value.

7.5. Begin Statement

A Begin Statement has the form "BEGIN s1; ...; sn END", where the si are statements. It is a means of grouping a list of statements into a single statement. A name can be given to a Begin Statement by inserting a string constant after the "BEGIN", in which case the same name must be inserted after the matching "END".

7.6. If Statement

An If Statement has the general form:

```
IF e1 THEN s1
EF e2 THEN s2
...
EF en THEN sn
EL s
```

The "EF" and "EL" lines can be omitted. "EF" abbreviates "ELSE IF", and "EL" abbreviates "ELSE" (the longer forms can be used). An "ELSE" ("EL") or "EF" matches with the innermost unmatched "IF" or "EF".

7.7. Case Statement

A Case Statement is illustrated by:

```
CASE e OFB
  [c1]                s1;
  [c2 TO c3]          s2;
  [c4][c5 TO c6]      s3;
  [ ]                 s4;
END
```

"e" is an integer expression, the c_i are integer constants, and the s_i are statements. "OFB" is an abbreviation for "OF BEGIN", which may be used instead. If e has the value c_1 , s_1 is executed. If e is between c_2 and c_3 , inclusive, s_2 is executed. If e is c_4 between c_5 and c_6 , s_3 is executed. Otherwise, s_4 is executed. In general, any number of case selectors may be utilized for a given statement, as two are shown for s_3 . The catchall "[]" case may be omitted, in which case an error occurs if no statement is selected. Only the selected s_i is executed. s_i may be a Begin Statement, i.e., a list of statements to be executed.

7.8. Iterative Statement

An Iterative Statement has the general form:

```
FOR i := e1 UPTO e2   WHILE e3           DO s   UNTIL e4
(FOR-clause)          (WHILE-clause)         (UNTIL-clause)
```

where *i* is an integer variable, *e1* and *e2* are (long) integer expressions, *e3* and *e4* are any expressions, *s* is any statement, and "UPTO" may be replaced with "DOWNTO". Any of the clauses may be omitted; thus, there are eight possible forms (ignoring the distinction between "UPTO" and "DOWNTO").

"DO *s*" alone means repeatedly execute *s* until some action terminates the Iterative Statement, most likely a Done or Return Statement or an exception. "UPTO" increments *i* by 1, and "DOWNTO" decrements *i* by 1. *e3* is evaluated before each iteration, and *e4* after each iteration. *e2* is evaluated just once, before any iterations. An "UNTIL" is matched with the innermost unmatched "DO". A name may be given to the Iterative Statement by placing a string constant after "DO", "DO BEGIN" or "DOB" (an abbreviation for "DO BEGIN"). In the latter two cases, the same name must be placed after the matching "END". Such a name may be used by the Done or Continue Statement as described below.

7.9. Done Statement

A Done Statement has the form "DONE", or:

```
DONE "c"
```

where "c" is the name of an enclosing Iterative Statement. The referenced Iterative Statement is terminated; in the absence of "c", the innermost enclosing Iterative Statement is terminated.

7.10. Continue Statement

A Continue Statement has the form of the Done Statement, except that "CONTINUE" replaces "DONE". The referenced Iterative Statement is continued (tests in the clauses are performed, and if they pass, the iterated statement is re-started) as if the iterated statement had terminated normally.

7.11. Empty Statement

The Empty Statement consists of nothing at all. It serves as a place holder in certain situations; for example, it allows superfluous semicolons between statements.

8. Declarations

Identifiers must be declared before they are referenced.

8.1. Scope of Identifiers

An identifier declared within a procedure is accessible only within that procedure. An identifier declared outside any procedure is accessible within procedures in the same module that follow the declaration, except those that redeclare it.

8.2. Simple Variable Declarations

A simple variable declaration has the form "type v1, ..., vn", where the vi are identifiers, and type is the name of a data type. In addition, the type keywords "POINTER" and "ADDRESS" may be followed by a parenthesized class name.

8.3. "OWN" Qualifier

The "OWN" qualifier specifies that a local variable is to be allocated as if it were declared in the outer block. This means that it is allocated and deallocated along with the module data, rather than upon procedure entry and exit. This allows a procedure to have a private variable that can retain information across procedure calls.

9. Arrays

9.1. Array Declarations

An array declaration has one of the forms:

```
type ARRAY(l1 TO u1, ..., lm TO um) v1, ..., vn
```

or:

```
type LONG ARRAY(l1 TO u1, ..., lm TO um) v1, ..., vn
```

For reasons of efficiency MAINSAIL supports two sizes of array. Short arrays can use only integer subscripts while long arrays can use either long integer or integer subscripts. Subscript calculations for short arrays are performed with integer arithmetic; long array subscript calculations are performed with long integer arithmetic. Since long integer arithmetic is slower than integer arithmetic on some systems, short arrays permit greater runtime efficiency, but cannot be as large as long arrays.

l_i and u_i specify the bounds of the i th dimension. Up to three dimensions are allowed. Each l_i or u_i is either a (long) integer constant, with $l_i \leq u_i$, or "*" to indicate that the bound is not known at the point of declaration (it must be given when the array is allocated).

The data type and/or the parenthesized bounds list may be omitted, in which case the array cannot be used for element access. It may still be used as a parameter or assigned to or compared with some other array.

The array Zero is nullArray.

9.2. Array Allocation

It is the programmer's responsibility to allocate an array before an element is accessed. MAINSAIL differs in this respect from most languages, which automatically allocate and deallocate arrays according to the context in which the array is declared.

An array is allocated by "new(a,l1,u1,...,ln,un)", where a is the array to be allocated, and the l_i and u_i are expressions for the bounds. Bounds that can be determined from the array

declaration may be omitted. An array may be allocated any number of times. Each new allocation replaces the old one; no elements are copied. All elements of a newly allocated array are initialized to Zero.

MAINSAIL's explicit array allocation permits a program to decide dynamically when to allocate arrays and how large each must be. For example, a procedure can allocate arrays of a size dependent upon input data. Many arrays can be declared, only some of which are actually allocated during a given program execution.

Arrays may be explicitly deallocated, or the programmer may allow the garbage collector to free the storage occupied by unused arrays.

9.3. Array Initialization

The Init Statement may be used to initialize an array. The array must already have been allocated. The general form of the Init Statement is:

```
INIT v (c1, ..., cn)
```

where v is the array to be initialized, and the ci are constant expressions of v's type.

9.4. Accessing an Array Element

An array element is accessed by a variable of the form "a[i]", "a[i,j]", or "a[i,j,k]", depending on the number of dimensions of the array a. i, j, and k are (long) integer expressions that must be within the bounds declared for the corresponding dimensions.

9.5. Array Variables

An array variable is implemented as a pointer to the array's elements. Array variables may be assigned, passed as parameters, and compared. In each case, only the array variable itself (i.e., the pointer to the array elements) takes part in the operation. Thus, a single array's elements may be pointed to by many array variables.

10. Classes and Records

10.1. Records

A record is a data structure that differs from an array in that its components, called fields, may be of differing data types and are accessed by name instead of by subscripts.

Records are dynamically allocated; the programmer does not declare records, and there is no way to create a record at compiletime. Just "classes", which give templates for the structure of records created at runtime, and pointers, which point to records, are declared. All records are allocated during program execution under direct control of the program.

10.2. Classes

Each record is an instance of a programmer-declared class that serves as a template describing the various fields of the record. The most common form of a class declaration is:

```
CLASS v (declarations of fields of class)
```

Class declarations may occur only in the outer declarations of a module. The fields of a class can be variables of any data type. The order in which they occur in the class declaration is the order in which they occur in memory. Thus, a class can be a template to be overlaid on an already-existing data structure, such as might be created by a procedure in some other language and passed to MAINSAIL.

Classes are also used to describe module interfaces; such classes may include procedure fields.

10.3. Record Allocation and Disposal

Any number of new records of a class may be created at runtime by calls to the system procedure "new". "p := new(c)" allocates a record that conforms to the class c and sets the pointer p to reference the record. All the fields of the newly allocated record are cleared.

The storage occupied by a record pointed to by p is freed by "dispose(p)". Records no longer pointed to by any pointer are automatically disposed by the MAINSAIL garbage collector.

Arrays and module data sections may also be disposed.

10.4. Classified Pointers

A pointer *p* can be declared as "POINTER(*c*) *p*" to indicate that it will point only to records of class "*c*". Such a pointer is a "classified pointer". The compiler ensures that classified pointers are not mistakenly used to refer to records in other classes.

10.5. Unclassified Pointers

The class in a pointer variable declaration may be omitted if the pointer is to be used for an unknown class, or for several different classes. The programmer must be especially careful when using unclassified pointers since class checking is not provided for them.

10.6. Accessing Fields of Records

A field *f* of a record of a class *c* pointed to by a pointer *p*, declared as "POINTER(*c*) *p*", is accessed by the field variable "*p.f*". "*p*" is the base part, and "*f*" the field part.

The base part may itself be a field variable. For example, "*p.q.f*" has the base part "*p.q*", where *q* is a pointer field of the record pointed to by *p*. Base parts may also be subscripted variables (e.g., "*a[i].f*"), procedure calls (e.g., "*proc(parms).f*"), or parenthesized pointer expressions (e.g., "*(p := q).f*").

10.7. Explicit Classes in Field Variables

The variable "*p:c.f*", where *p* is a pointer expression, *c* is a class and *f* is a field of *c*, provides a means of explicitly specifying the class of a pointer at the point of use as a base part.

10.8. Prefix Classes

A class can inherit its initial fields from a previously declared class, called its "prefix class". The form of a declaration for such a class is:

```
CLASS(c1) c2 (declarations for additional fields)
```

In this case *c1* is the prefix class. If *c1*'s declaration were:

```
CLASS c1 (INTEGER i,j,k)
```

then the effect on c2 would be the same as:

```
CLASS c2 (INTEGER i,j,k; declarations for additional fields)
```

Two classes are "related" if one is a prefix class of the other or if they are the same class. Pointer assignments and argument-parameter matching are allowed only between related classes.

MAINSAIL's prefix classes play a role similar to that of Pascal's "variant records", though prefix classes are a simpler concept and require no runtime overhead. Prefix classes allow classes that share some fields to have these common fields "abstracted out" into a separate class. Procedures that manipulate pointers to a prefix class cannot mistakenly access fields of prefixed classes.

11. Procedures

11.1. Procedure Declarations

The basic form of a procedure declaration is:

```
type PROCEDURE v (declarations list for parameters);  
procedure body
```

"type" is present only if the procedure is to return a value; the parameter list may be omitted if there are no parameters. The procedure body is either a statement, or a list of local variable declarations followed by a list of statements, all within a "BEGIN"- "END" pair. Procedures cannot be statically nested; i.e., a procedure body cannot contain a procedure. The initial values of the local variables are not defined, except for uses and modifies parameters, as described below.

11.2. Procedure Calls

A procedure call has the form "p(e1,...,en)", where p is a procedure, and the ei are arguments. The order of evaluation of the ei is not specified. If p has no arguments, the parenthesized list may be omitted. Any procedure may be invoked recursively.

11.3. Procedure Parameters

There are three kinds of parameters, distinguished in the parameter declarations by the qualifiers "USES", "PRODUCES", and "MODIFIES". A uses parameter (the default) is initialized by the value of the argument. A produces parameter is not initialized by its argument, but instead sends its final value back to the argument (which must be a variable) when the procedure returns. A modifies parameter has the characteristics of both uses and produces parameters, i.e., is initialized by the argument and sends its value back to the argument.

A parameter behaves like a local variable inside the body of the procedure. In particular, modifies and produces parameters are not passed as the address of the corresponding arguments, as is the case for the "reference" and "name" parameters of ALGOL or FORTRAN.

11.4. Optional Arguments

Trailing parameters may be qualified with "OPTIONAL" to indicate that their arguments may be omitted in procedure calls, in which case the compiler passes Zero values of the appropriate data type. This allows little-used parameters to be left out of most calls.

11.5. Repeatable Arguments

Trailing parameters of an untyped procedure may be qualified with "REPEATABLE" to indicate that a call may give several sets of arguments for the repeatable group of parameters. The compiler acts as if several calls had been explicitly made, with arguments before the repeated ones computed just once, and passed for each call. This allows constructs such as "write(f,a,b,c)", which is equivalent to:

```
write(f,a); write(f,b); write(f,c)
```

except that f may be evaluated just once.

11.6. Forward Procedures

A procedure must be declared before it can be called. However, if two procedures call each other, a vicious circle results since each must be declared before the other. To get around this problem, one of the procedures is first given a "forward" declaration, which is like a normal declaration except it is qualified with "FORWARD", and just the procedure header (not the body) is given. Later the procedure is declared as usual; the compiler automatically figures out that a previously declared forward procedure is now being given a body. Calls to the procedure may appear at any point after the forward declaration.

11.7. Compiletime Libraries

A related use of forward procedure declarations is to specify the name of the file that contains the complete procedure declaration. This is done by "FORWARD(f)" where f is a string constant for the name of the file. If at the end of compilation the procedure has been called, but no body has been declared for it, the compiler automatically compiles the file f, expecting to encounter the procedure's declaration.

This mechanism allows the creation of "compiletime libraries" that contain full procedure declarations for commonly used procedures and are automatically accessed by the compiler to

obtain bodies of called procedures. In most cases, however, intrinsics are a better way of organizing a set of procedures; see Chapter 15.

11.8. Inline Procedures

Procedures or procedure calls may be marked with the keyword "INLINE", which causes the procedure or particular call to be expanded inline. This avoids the usual procedure call overhead at the expense of producing more code. It is appropriate to declare small, frequently-called procedures inline, since the procedure call overhead may represent a substantial fraction of the execution time of the procedure.

11.9. Generic Procedures

A generic procedure allows a single identifier to represent several procedures. A particular one is chosen in a procedure call as determined by the data types of the arguments. This is a convenience to the programmer in that it allows the same name to be used for related procedures.

An example of a generic procedure declaration is:

```
GENERIC PROCEDURE p "p1,p2,...,pn"
```

where the p_i are procedure identifiers. Whenever "p" is used in a procedure call, the compiler acts as if "p1" had been used instead, except that if some "error" occurs (e.g., argument-parameter type mismatch), the compiler "backs up" and acts as if p2 had been used instead of p1. If another "error" occurs, the compiler proceeds to p3, and so forth, until a p_i is found that causes no error (the compiler complains if no such p_i is found).

The generic mechanism can be combined with repeatable arguments to provide a quite general procedure calling capability with no execution time overhead. For example, the generic system procedure "write" allows any number and any type of arguments to be written to several different kinds of files, or to a string, or to memory, all based on the generic and repeatable mechanisms.

Generic procedures (including predefined system procedures) may be extended by the user. For example, if a generic procedure is originally declared as:

```
GENERIC PROCEDURE p "a,b,c"
```

then the declaration:

GENERIC PROCEDURE p "x,y,z"

is concatenated with the original declaration, so that the effect is as if p had been declared as:

GENERIC PROCEDURE p "x,y,z,a,b,c"

12. Modules

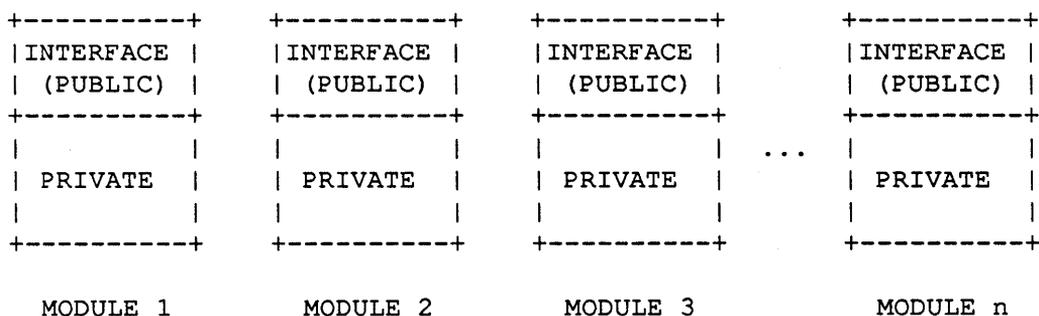
Modules are used to divide a program into small, manageable units that are separately compiled and manipulated during execution. A program is a collection of modules, some of which are contributed by the programmer, and others by MAINSAIL's runtime system.

Many instances of a module may exist, and modules may be manipulated by code that knows only a limited amount about the module. These capabilities permit an "object-oriented" style of programming, in which each instance of a MAINSAIL module is treated as an object.

MAINSAIL's approach to modules goes well beyond that found in other algorithmic programming languages, and provides the programmer with previously unavailable capabilities, such as dynamic linking and loading, which have a profound effect on program organization and portability.

Unlike most programming languages, MAINSAIL does not utilize a "link" step prior to execution. Instead, the modules are brought into memory as needed during execution and MAINSAIL provides all the facilities for intermodule communication. The programmer need never specify what modules make up a program; a program is an open-ended collection of modules the identity of which need not be determined until execution time. The runtime selection of modules provides a degree of flexibility lacking in statically linked systems. It also frees MAINSAIL from any dependence on machine-specific linkers, with their attendant restrictions and peculiarities.

Only the currently executing module need be in memory; MAINSAIL automatically swaps modules in and out of memory during execution. It tries to keep a "working set" of modules in memory. Modules are compiled into position-independent code so that they may reside anywhere in memory, and may even be moved about in memory during execution.



A module is written according to the following general layout:

```
+-----+
| BEGIN "modNam" |
| +-----+ |
| | outer |
| | declarations 1 |
| +-----+ |
| +-----+ |
| | procedure 1 |
| +-----+ |
| | |
| | . |
| | . |
| | . |
| +-----+ |
| | outer |
| | declarations n |
| +-----+ |
| +-----+ |
| | procedure n |
| +-----+ |
| END "modNam" |
+-----+
```

"modNam" gives the name of the module, which must be an identifier of six characters or fewer. Since the runtime system uses a module's name to identify it, every module in a program must have a unique name.

The "outer declarations" declare variables to be accessible throughout the remainder of the module, but not in any other modules. The outer declarations of a module m must include declarations for all modules referenced by m. In addition, m must declare itself if it has any interface fields.

12.1. Module Allocation and Communication

Modules communicate through "interface fields", which are the variables and procedures of each module that the programmer declares to be accessible from other modules. The interface fields of a module are analogous to the interface fields of a record. Data (non-procedure) interface fields reside in a data structure called a "data section". Data sections exist in "bound" and "nonbound" forms.

Nonbound data sections are allocated by means of the procedure "new". Nonbound data section interface fields are always accessed with explicit pointers; the syntax used is identical to that for records. A module may have more than one nonbound data section.

Bound data sections are allocated by means of "bind". The interface field of a bound data section may be accessed by name (with no module or pointer prefix) if the reference is unambiguous; bound data sections may also be accessed with explicit pointers. A module may not have more than one bound data section at any given time. Bound data sections allow the programmer to write code without knowing the name of the module in which an interface variable or procedure actually resides, so that a system of many modules can be reconfigured with minimal source code changes.

Data sections contain the outer and local own variables of a module as well as the interface fields, but only the interface variables are accessible from outside the module.

When a module's data section is allocated with "bind" or "new", the interface and own variables in the data section are cleared, and the initial procedure is invoked. MAINSAIL automatically allocates a bound data section the first time one of its interface procedures is called, if it has not already been allocated.

At the end of program execution, MAINSAIL automatically invokes the final procedures associated with any active modules and then closes any open files.

12.2. Module Syntax

The most common form of a module declaration is:

```
MODULE v (declarations of interface fields)
```

where v is the module's name. Interface fields may be variables and/or procedures, in any order. The declaration of an interface procedure gives only the header. It serves as a forward declaration for the procedure. The procedure body must be given within the module v, where the procedure is declared as usual.

A module's fields can also be supplied by means of a class, with a declaration such as "MODULE(c) m (additional fields)", where c is a class that specifies the first fields of m's interface and the parenthesized list of additional field declarations is optional.

Each module may contain at most one typeless and parameterless "initial procedure" that is to be called whenever a data section for the module is allocated. This procedure is qualified with the keyword "INITIAL". A module invoked from the MAINSAIL executive leads to execution of an entire program by executing the module's initial procedure. Thus, initial procedures play

two roles: one is to perform any kind of initialization for a module, and the other is to lead to execution of what the user views as a program. As far as the MAINSAIL runtime system is concerned, a program is just a set of modules that are brought into memory and initialized.

Each module may contain a single typeless and parameterless "final procedure" that is automatically invoked when the module is disposed. This procedure is qualified with the keyword "FINAL".

12.3. Combining Module Declarations in One File

The outer declarations of a module *m* include declarations for all modules accessed by *m*. A module must be declared identically (up to the last interface field accessed) in all modules that access any of its interface fields.

To ensure consistent module declarations, and to save retyping and updating the declarations in every module of a program, all module declarations for a related group of modules can be put in a single file, and that file included in the compilation of all the modules by means of the "SOURCEFILE" compiler directive. Alternatively, the declarations may be stored in a compiler symbol table file (an *intmod*) for faster processing at compiletime. MAINSAIL's compiler directives have been chosen to provide flexibility in the construction of the source text for large programs.

12.4. Objmod Libraries

An *objmod* (object module) library is a file that contains object modules. A librarian program, *MODLIB*, is provided for maintaining *objmod* libraries. Any number of *objmod* libraries may be open for execution. MAINSAIL automatically searches all open libraries to find a module. If a module is not found in any *objmod* library, a file name is formed from the name of the module, and an attempt is made to obtain the object module from that file. If that file cannot be opened, MAINSAIL may ask the user during execution where the module resides.

A means is provided for indicating what file contains an object module, and for dynamically associating the "true" module name for a "dummy" module name, thereby allowing programs to be written without knowledge of the specific physical module that provides a given service.

13. Macros

A macro allows an identifier to represent either a constant or arbitrary text. Each occurrence of the macro identifier (a macro call) is replaced by the compiler with the associated constant or text specified when the macro was defined.

13.1. "DEFINE"

The form of a simple macro definition is:

```
DEFINE v1 = macroBody1, ..., vn = macroBodyN;
```

where the v_i are identifiers and macro bodies are constants or "bracketed text", e.g., "[xxx]" where "xxx" is arbitrary text.

The form of a definition for a macro with parameters is:

```
DEFINE v(v1, ..., vn) = [text];
```

where the macro identifier v is followed by a parenthesized list of parameter identifiers (the v_i) that may be used within the bracketed text. Subsequent occurrences of v (i.e., macro calls) are followed by a parenthesized list of arguments, much like a procedure call. Each occurrence of the identifier v_i within the bracketed text (even within string constants and comments) is replaced with the corresponding argument text.

A macro definition may occur virtually anywhere in a program, even in the midst of an expression, for example. However, a macro definition cannot occur in the midst of another definition, except within bracketed text.

Macro calls may be used anywhere, even in subsequent macro definitions.

"REDEFINE" is used to change the body of a previously defined macro. For example, "DEFINE $n = 0$; ... REDEFINE $n = n + 1$ " does compiletime counting.

13.2. Macro Constants

Macro constants (constants used as macro bodies) may be constants of any data type. Defining a macro identifier to be equal to a constant (or constant expression) allows a meaningful name to be used in the program text instead of a "bare" constant. If the value of the constant expression needs to be changed, then only the macro definition has to be changed instead of all occurrences of the constant expression.

13.3. Interactive Define and Redefine

A macro definition may omit the "=" and subsequent macro body, in which case the compiler prompts for the macro body. Such interactive definitions allow the programmer to make limited contributions to the source text during compilation, in lieu of having to edit the source text just prior to compilation. For example, "DEFINE debugOption;" prompts with "DEFINE DEBUGOPTION =" during compilation. The user types in a value "v", and the compilation proceeds as if it had instead encountered "DEFINE debugOption = v;".

13.4. Macro Calls

A "macro call" is the occurrence of a macro identifier at any point in a program after it has been defined. It directs the compiler to scan the body of the macro as if it appeared in place of the macro call.

If the macro was defined with parameters, a parenthesized list of macro arguments separated with commas may appear after the macro identifier. The macro arguments replace all occurrences of the corresponding parameter identifiers in the macro body.

14. Compiler Directives and Conditional Compilation

A compiler directive indicates which source text is to be compiled or conveys information to the compiler that is used while compiling the program.

A compiler directive may occur wherever a declaration or statement may occur (except that "BEGINSCAN" must be the first thing on a page), and must be terminated with a semicolon.

14.1. "MESSAGE"

"MESSAGE c" writes the string constant "c" onto a new line of logFile during compilation.

14.2. "SOURCEFILE"

"SOURCEFILE c" directs the compiler to compile the file with name "c" as if it appeared in place of the directive. Sourcefile nesting may occur to any level. The name "c" can be obtained interactively; e.g.:

```
DEFINE f; SOURCEFILE f;
```

prompts the user for the name of the source file, then uses the string constant macro f in the source file directive.

14.3. Checking, Arithmetic Checking, and Optimization

Checking directs the compiler to emit code to check certain conditions at runtime that cannot be determined at compiletime. Arithmetic checking directs the compiler to emit code to check for (long) integer overflows, if such code is necessary. Optimization directs the compiler to make efforts to produce the best code it knows how (at the expense of a longer compilation time).

Directives are provided to control checking, arithmetic checking, and optimization at the module level as well as on a per-procedure or even per-expression basis.

14.4. "IFC", "THENC", "\$EFC", "ELSEC", and "ENDC"

The conditional compilation form:

```
IFC c THENC text1 $EFC c2 THENC text2 ELSEC text3 ENDC
```

where *c* is an expression evaluated at compiletime, causes the compiler to compile *text1* and ignore *text2* and *text3* if *c* is non-Zero. If *c* is Zero and *c2* non-Zero, then *text2* is compiled and *text1* and *text3* ignored. If *c* and *c2* are both Zero, *text3* is compiled and *text1* and *text2* ignored. There may be zero, one, or many "\$EFC *ci* THENC *texti*" parts, and "ELSEC *text3*" may be omitted. IFC's may be nested to any depth.

"\$CASEC" and "\$DOC" provide compiletime analogues of Case and Iterative Statements, just as "IFC" is analogous to the If Statement.

14.5. "DCL"

"DCL(<identifier>)" is true if the identifier has been declared or defined and false otherwise. "DCL" is useful in conjunction with conditional compilation.

14.6. "DSP"

"DSP(*c.f*)" returns the offset in storage units to field "*f*" of class "*c*".

14.7. "\$TYPEOF", "\$CLASSOF", "\$ISCONSTANT"

"\$TYPEOF(*x*)" returns the type code for the expression *x*. "\$CLASSOF(*x*)" returns the class name of the expression *x*. "\$ISCONSTANT(*x*)" returns true if and only if the expression *x* is constant, i.e., can be evaluated at compiletime. All three are pseudo-procedures evaluated at compiletime, and all three discard the expression *x* without actually evaluating it.

14.8. Scanning Directives

The scanning directives are "BEGINSCAN", "SKIPSCAN", and "DONESCAN". "SKIPSCAN" and "BEGINSCAN" may be followed by string constant names. "SKIPSCAN" specifies that source text is to be skipped up to the next appropriately named "BEGINSCAN" directive. "BEGINSCAN" is used to mark places in the source text where a "SKIPSCAN"

search may terminate. "DONESCAN" causes the compiler to terminate compilation of the current file as if end-of-file had been reached.

These directives allow essentially arbitrary use of text files as repositories of fragments of source text that are "scooped up" during the compilation of many different modules. When combined with the flexible conditional compilation directives, the programmer can piece together a program in almost any manner from any number of files, in accordance with options specified interactively during compilation.

14.9. "NEEDBODY" and "NEEDANYBODIES"

The compiletime pseudo-procedures "NEEDBODY" and "NEEDANYBODIES" are used in conjunction with the "FORWARD" qualifier to determine whether a forward procedure needs a body, i.e., has been called but has not yet been given a declaration containing the procedure body. They are typically used in compiletime libraries to ensure that the compiler sees just those procedures for which bodies are needed.

The form "NEEDBODY(id)" is true if id is the name of a procedure that has been declared "FORWARD" and has appeared in a procedure call, but has not been declared with a body.

"NEEDANYBODIES(c)" is true if there are any procedure bodies in the file "c" that need to be compiled.

The form "NEEDANYBODIES" is equivalent to "NEEDANYBODIES(c)" where c is the name of the file that caused the current automatic sourcefile.

14.10. "\$DIRECTIVE" Directives

"\$DIRECTIVE" is a directive that takes a string argument. The string argument begins with a keyword that functions as a directive. "\$DIRECTIVE" is used for a number of directives with miscellaneous functions; many of them allow compiler subcommands to be inserted into the source text of a module.

15. Intmods

Intmods allow the compiler's symbol table for a module to be preserved. The symbols in it may be used in the compilation of other modules or by MAINSAIL system programs such as MAINDEBUG and MAINPM. When the symbols are used in a compilation, an intmod may contain symbols used by many modules. Such an intmod replaces a "header" file, which would be recompiled once for each module that uses its symbols, with a module compiled just once to produce the intmod. A header file is a file that contains definitions and declarations (e.g., of classes and modules) of use to several modules.

An intmod is produced for a module in which the compiler encounters the "SAVEON" directive. The form of the "SAVEON" directive is "SAVEON c", where "c" is a string constant expression for the name of the intmod on which the symbol table is to be saved.

The "RESTOREFROM m" directive may be used to make an intmod m "visible" to another module, meaning that the symbols in it can be used in the other module (the symbols themselves are also said to be visible). Symbols that are not visible can be used only with a special qualifier of the form:

```
<intmod name>${<symbol name>
```

Directives exist to specify that only certain symbols in an intmod should be made visible (usable without the qualifier) to modules that make the intmod visible, thereby providing information hiding. For example, if only certain procedures from an intmod are to be visible, the programmer may still include supporting procedures in the intmod without making the identifiers for those procedures visible in other modules (possibly conflicting with identifiers from the other module).

Identifiers from an intmod may be outer declarations of the intmod (variables, classes, modules, macros, etc.) and procedures. The procedures are incorporated into the using module only if they are called.

16. Exceptions

An exception is an unusual or erroneous condition that occurs during a program's execution. When an exception occurs, it may cause the execution of a statement called an exception handler. A handler may, for some exceptions, repair the error and resume execution at the place where the exception occurred, or it may recover from the error by aborting the execution of one or more nested statements (including procedure invocations), or it may propagate the exception to another handler. If there is no handler for an exception, the MAINSAIL runtime system reports an error by calling the system procedure `errMsg`.

Exceptions are divided into two categories: those predefined by MAINSAIL and those known only to user programs. User exceptions must be explicitly caused by the user program by means of the system procedure `$raise`; predefined exceptions are caused by MAINSAIL.

16.1. Handle Statement

A Handle Statement associates an exception handler with a statement in the program (called the handled statement) and executes the handled statement. If an exception occurs during the execution of the handled statement, that statement's execution is interrupted and the handler is executed. If no exception occurs, the handler is ignored.

The general form of the Handle Statement is:

```
$HANDLE s1 $WITH s2
```

in which `s1` and `s2` are statements. The statement `s1` is the handled statement and `s2` is the handler.

When a Handle Statement is executed, its statement `s1` is initiated. If an exception occurs during `s1`'s execution (which may entail several levels of procedure calls), and the exception has not been handled by another Handle Statement initiated during `s1`'s execution, then `s1`'s execution is suspended and the exception handler `s2` is executed. Otherwise, `s2` is ignored.

A handler can either recover from an exception and allow the program's execution to continue or it can propagate the exception to another exception handler. In the first case, the handler is said to have handled the exception.

16.2. Handling Exceptions

There are two ways a handler can allow a program's execution to continue:

- If the exception that occurred was caused by a call to the system procedure \$raise, the handler can resume execution at the place where the exception occurred by calling the system procedure \$raiseReturn. This terminates the handler's execution. When s1 terminates after its execution is resumed, s2 is ignored. If the exception that occurred was not caused by a call to \$raise, a runtime error occurs if \$raiseReturn is called to continue from the exception.
- The handler can terminate the Handle Statement's execution in one of three ways:
 1. It can resume execution at the statement following the Handle Statement, if any, by having control fall out of the handler.
 2. If the Handle Statement is contained within an Iterative Statement, the handler can terminate the Handle Statement's execution and either repeat or terminate the execution of the Iterative Statement by means of a Continue or Done Statement.
 3. The handler can terminate the Handle Statement's execution and return from the procedure containing the Handle Statement by means of a Return Statement.

When a handler terminates the execution of its Handle Statement, the handled statement s1, the execution of which was suspended by the occurrence of the exception, is aborted, along with all other statements initiated as a result of s1's execution (and all procedures thus invoked). When a procedure is aborted in this manner, if it contains any active handled statements, MAINSAIL raises the predefined exception \$abortProcedureExcpt and executes the handlers associated with the handled statements. This gives each procedure a chance to do any cleaning up that might be necessary before it is aborted.

16.3. Propagating Exceptions

If a handler is unable to handle an exception, it can propagate the exception to the next handler by calling the system procedure \$raise with no arguments. If there is no next handler within the user program, the MAINSAIL runtime system reports an error by calling the system procedure errMsg.

16.4. Information about the Current Exception

A handler can obtain the name of the current exception by calling the system macro `$exceptionName`. Other information about the current exception is available by calling `$exceptionStringArg1`, `$exceptionStringArg2`, `$exceptionPointerArg`, `$exceptionBits`, and `$exceptionCoroutine`.

16.5. Nested Exceptions

Exceptions can be nested. If an exception occurs during a handler's execution, the handler's execution is suspended and a handler for the new exception is searched for and initiated, as described above. If the new handler resumes execution at the place where the new exception occurred, the previous exception is restored to being the current exception and the execution of its handler continues. If the new handler aborts the execution of its Handle Statement and that Handle Statement caused the execution of the previous handler's Handle Statement, the execution of the previous handler's Handle Statement is also aborted.

17. Coroutines

A "coroutine" is a context that preserves the state of a procedure so that its execution may be "resumed" at the preserved state by a procedure in some other coroutine. System procedures are provided to create, resume, and kill coroutines.

A coroutine may be thought of as a "thread of execution" that progresses independently of other threads of execution in an interleaved fashion. Thus, a coroutine executes for a while and then explicitly resumes some other coroutine. That new coroutine executes for a while, and then explicitly resumes another coroutine, perhaps the one that resumed it. Coroutines must explicitly resume other coroutines; i.e., coroutines do not execute in parallel, and there is no automatic timer interrupt that causes resumption of coroutines. However, user or system procedures may resume coroutines in such a way as to give the illusion of parallel execution.

Example 17-1 shows a use of coroutines. `generateNextNode` generates a node in some data structure, and `processNextNode` processes the node.

The procedure `processNextNode` first creates a coroutine for the generator. The arguments indicate the data section and procedure name where the first resumption is to start. `processNextNode` then uses the returned pointer to resume the coroutine to get each node, and finally to kill it. There can be any number of procedures like `processNextNode` that use `generateNextNode`. `generateNextNode` sets the outer variable `nextNode` to point to the next node, and then resumes the coroutine in which `processNextNode` is running. Outer or interface variables must be used for communication among coroutines, since `$resumeCoroutine` does not have any parameters for intercoroutine communication.

17.1. Coroutine Implementation

A coroutine consists of a stack to hold the procedure frames and a record of the predeclared class `$coroutine`, which contains information about the coroutine.

A tree structure is imposed on coroutine records based on a parent-child relationship. Each coroutine record has a link `$up` that points to its parent coroutine record, i.e., the coroutine that created it. The link `$down` points to the first-created (oldest) child. `$right` points to the next younger sibling, and `$left` points to the next older sibling.

The `$up`, `$down`, `$left`, and `$right` links are "structural" links in that they depend on the order of coroutine creation. In addition, `$coroutine` records are maintained on a "dynamic" list by

```

POINTER(node) nextNode;
POINTER($coroutine) processCoroutine;

PROCEDURE generateNextNode;
BEGIN
...
DOB ...
    nextNode := ...;
    $resumeCoroutine(processCoroutine);
    ...
    END;
END;

PROCEDURE processNextNode;
BEGIN
POINTER($coroutine) p;
...
processCoroutine := $thisCoroutine;
p := $createCoroutine(thisDataSection, "generateNextNode");
DOB ...
    $resumeCoroutine(p);
    ... use nextNode here ...
    END;
$killCoroutine(p);
...
END;

```

Example 17-1. Generator/Processor Coroutines

means the \$prev and \$next links. Each time a coroutine is resumed, it is moved to the head of this list. The head of the list is pointed to by the predeclared variable \$thisCoroutine, so that \$thisCoroutine points to the record for the currently executing coroutine, and "\$thisCoroutine.\$next" points to the coroutine that most recently resumed the current one.

17.2. Coroutines and Exceptions

A parameter to \$raise may be used to initiate handling of an exception in a coroutine other than the current one.

Exceptions not handled in a child coroutine are propagated to the coroutine's ancestors; first to its parent, then its grandparent, and so on, until the root coroutine "MAINSAIL" is reached. \$raiseReturn erases traces of the current exception in all affected coroutines and resumes the coroutine from which the exception was raised.

18. Files

A file is an ordered series of data with a beginning position, a current position, and possibly an ending position. A file may reside on some external medium (e.g., an operating system's file structure) that is not defined by MAINSAIL or under MAINSAIL's complete control.

Some files may exist independently of the execution of a program, so that a program can create a file that can later be accessed by another program. Thus, files can provide continuity from one program execution to another.

Every file has a name, which is represented in a MAINSAIL program as a string.

MAINSAIL distinguishes between "text files" and "data files". A text file is composed of character units, a data file of storage units. MAINSAIL also distinguishes two methods of access to a file: sequential and random. The current position in a sequential file is updated to point to the next datum as each datum is read or written, in order, starting from the beginning. The current position of a random file may be explicitly changed to be anywhere within the file by means of the procedure "setPos".

18.1. File I/O

System procedures are provided for opening, closing, deleting, and renaming files, for reading from a file, and for writing to a file.

Before a file can be used by a program, it must be "opened" by a call to the system procedure "open". Arguments to the open procedure specify the file name and indicate how the file is to be accessed (sequentially or randomly, for input and/or output, etc.). A file is "closed" by a call to the system procedure "close" to indicate that it will no longer be used by the program, unless opened again.

A file is referenced in a MAINSAIL program with a pointer returned by the open procedure. The pointer belongs to one of the predeclared classes "textFile" or "dataFile".

18.2. Text Files

When a (long) integer, (long) real, or (long) bits value is written to a text file (with the system procedure "write"), an automatic conversion is made to the appropriate string representation of

that value, and then the string is written to the file. MAINSAIL does not insert any additional characters such as blanks or tabs, so that the user has full control over the layout of the file.

When the non-string data types are read from a text file (with the system procedure "read"), a scan for an appropriate string representation takes place, and when found, a conversion is made to the appropriate internal representation.

A string is read from a file by using "read", which gets the next line; "fldRead", which reads a string of a specified width; or "scan", which employs a scan table to break on a specified set of characters.

18.3. Terminal I/O

"TTY" is the file name associated with the user's terminal, or with the operating-system-dependent standard input. "ttyRead" and "ttyWrite" are system procedures used for explicit communication with "TTY". ttyRead reads a line typed by the user, and ttyWrite writes a string to the terminal.

18.4. Memory Files and Data Sinks

Files maintained in memory may be accessed with any name beginning with "MEM>" (except that the ">" character may be replaced with another character on some operating systems). A data sink file may be accessed with a name beginning with "NUL>". Data written to such a file are discarded; on input, a data sink file always acts as if end-of-file had been reached.

19. Areas

Areas provide a method for precise control of memory management. Data structures (records, arrays, data sections, and string text) may be grouped into areas, and the memory management of each area controlled separately. Areas are appropriate for programs that deallocate large conglomerations of data at a time. Careless use of areas, however, can cause bugs that are very difficult to track. Programs can be written without reference to areas; the use of areas is purely an optimization.

19.1. Advantages of Areas

An area can contain a single large, complex data structure. When the data structure is no longer needed, the entire area can be disposed. The disposal of an entire area has several advantages over individually disposing each chunk (record, array, or data section) in it:

- It is faster.
- Entire pages become free, rather than individual chunks (the memory manager makes better use of free pages than of free chunks).
- It is easy to be sure that entire data structure has been freed.

An area may also be marked as not subject to garbage collection. This is useful if no garbage is being generated in the data structures in the area, since garbage collections can be shorter.

19.2. Area Facilities

Procedures are provided to allocate areas, clear them (remove all data from them), and deallocate them. Parameters are provided to most procedures that allocate chunks or string text to specify the area into which the data are to go. If the area parameters are omitted, data automatically go into the common area, `$defaultArea`.

19.3. Area Caveats

To avoid bugs that are difficult to track and reproduce, the programmer must be careful about the following:

- A pointer or string referencing data in a disposed area is said to be "dangling". Use of a dangling pointer or string has undefined effects.
- A program must not pass to a system procedure or macro or assign to a system variable any pointer or string referencing data in an area that is to be disposed or cleared before MAINSAIL exits.

Because these rules are not always easy to follow, the use of areas is recommended only for experienced programmers writing programs in which the benefits of areas are really needed.

20. System Procedures, Macros, and Variables

Predefined procedures, macros, and variables provide services that support the execution of MAINSAIL programs. Much of the power of MAINSAIL comes from the large number of system procedures.

20.1. System Procedures, Variables, and Macros Summary

Table 20.1-1 contains a summary of all MAINSAIL system procedures, variables, and macros.

<code>open</code>	open a file
<code>\$reOpen</code>	open a file with new open bits
<code>close</code>	close a file
<code>\$closedFile</code>	determine whether a file has been closed
<code>\$createUniqueFile</code>	create file with unique name
<code>\$devModBrk</code>	device module name break character
<code>\$devModBrkStr</code>	string consisting of <code>\$devModBrk</code>
<code>\$delete</code>	delete a file
<code>\$rename</code>	rename a file
<code>\$copyFile</code>	copy (part of) one file to another.
<code>\$truncateFile</code>	truncate a file to given length
<code>getPos</code>	get file position
<code>setPos</code>	set file position
<code>relPos</code>	set relative file position
<code>\$getEofPos</code>	get end-of-file position of byte-stream file

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

eof	true when positioned at or beyond end-of-file
\$gotValue	determine if actually read last value; better way of checking for end-of-file
read	read values
write	write values
\$storageUnitRead	read a number of data efficiently from a file
\$storageUnitWrite	write a number of data efficiently to a file
\$characterRead	read a number of characters efficiently from a file
\$characterWrite	write a number of characters efficiently to a file
\$pageRead	read a page of data from a file
\$pageWrite	write a page of data to a file
cRead	read a character from file, string, or charadr
cWrite	write characters to file, string, or charadr
\$clearFileCache	uncache all or part of file
\$queryFileCacheParms	information about file cache
\$setFileCacheParms	control file cache
\$concat	concatenate strings (same as "&" operator)
\$dup	perform multiple concatenations
rcRead	reverse character read (from the end of a string)
rcWrite	reverse character write (to the beginning of a string)

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

fldRead	read a string field
fldWrite	write a string field
ttyRead	read a line from "TTY"
ttyWrite	write values to "TTY"
ttycWrite	write characters to "TTY"
\$removeBoolean	parse boolean string
\$removeBits	parse bits string
\$removeInteger	parse integer string
\$removeReal	parse real string
confirm	get yes/no confirmation
cmdMatch	match a command (command recognition)
errMsg	raise an exception and/or write a message and get a response
cmdFile	standard input file
logFile	standard output file
enterLogicalName	establish logical file name
lookUpLogicalName	find logical file name
\$setSearchPath	set file searchpath
\$globalLookup	look up global symbol
\$globalEnter	enter global symbol
\$globalRemove	remove global symbol
\$registerException	register an exception name so that it can be raised in response to an errMsg prompt
\$deRegisterException	undo \$registerException
\$newException	assign a unique exception name
\$raise	raise an exception
\$raiseReturn	terminate an exception handler

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

\$exceptionBits	return information about current exception
\$exceptionName	return name of current exception
\$exceptionCoroutine	return raising coroutine of current exception
\$exceptionPointerArg	return pointer argument of current exception
\$exceptionStringArg1, \$exceptionStringArg2	return a string argument of current exception
scanSet	set up scan bit
\$scanSet	set up scan integer
scanRel	release scan bits or integers
scan	scan a file or string according to a scan specification
\$removeLeadingBlankSpace, \$removeTrailingBlankSpace	remove blank space from string
\$removeWord	remove non-blank chars from string
\$formParagraph	fill and justify string

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

\$cvbo	convert to boolean
cvi	convert to integer
cvli	convert to long integer
cvr	convert to real
cvlr	convert to long real
cvb	convert to bits
cvlb	convert to long bits
cvs	convert to string
cvp	convert to pointer
cva	convert to address
cvc	convert to charadr
cvAry	convert to array
cvcs	convert a character code to a single-character string
cvu	convert to upper case
cvl	convert to lower case
\$length	length of result of cvs
first	first character of a string
last	last character of a string
length	number of characters in a string
compare	-1, 0 or 1 as result of (optionally "caseless") comparison of two strings
equ	checks (optionally "caseless") equality of two strings
isLowerCase	true if argument is a lowercase letter ("a" through "z")
isUpperCase	true if argument is an uppercase letter ("A" through "Z")
isAlpha	true if argument is a letter ("A" through "Z" or "a" through "z")

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

nextAlpha	alphabetically next character after argument character
prevAlpha	alphabetically previous character before argument character
isNul	true if argument is a "null" character
copy	copy a record, array, memory, or characters
clear	clear a record, array, memory, or characters
newUpperBound	adjust the upper bound of a one-dimensional array
\$adrOfFirstElement	get the address of the first element of an array
new	allocate a record, array, or data section
\$newRecords	allocate multiple records
dispose	dispose of a record, array, data section, or module
bind	bind a module
unBind	unbind a module
\$canFindModule	whether a module can be allocated without error
\$isBound	whether a module is already bound
\$invokeModule	invoke a module the way MAINEX does
\$useProgramInterface	true if bound because an interface procedure called
\$programName	name under which MAINSAIL was invoked
\$getCommandLine	get program arguments
\$setCommandLine	set program arguments

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

<code>thisDataSection</code>	return pointer to current data section
<code>\$moduleName</code>	return name of module, given data section pointer
<code>\$searchCallChain</code>	find caller from particular module
<code>\$writeCalls</code>	show call stack of coroutine
<code>\$fieldInfo</code>	return information about a record or data section field
<code>\$className</code>	return name of class of a pointer
<code>\$classInfo</code>	return names and types of record or data section fields
<code>\$dscrPtr</code>	class descriptor for pointer
<code>\$classDscrFor</code>	class descriptor for a given class
<code>\$isArray</code>	true if pointer points to an array
<code>\$createClassDscr</code>	create a new class at runtime
<code>\$createRecord</code>	create a record given a class descriptor
<code>openLibrary</code>	open a module library file
<code>closeLibrary</code>	close a module library file
<code>setModName</code>	set a module name association
<code>relModName</code>	release a module name association
<code>setFileName</code>	set a module file name association
<code>relFileName</code>	release a module file name association
<code>exit</code>	orderly exit from MAINSAIL
<code>fastExit</code>	fast exit from MAINSAIL
<code>\$setExitCode</code>	set exit code for operating system

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

floor	largest (long) integer not exceeding a (long) real
ceiling	smallest (long) integer not exceeded by a (long) real
truncate	truncate a (long) real to a (long) integer
abs	absolute value of a (long) integer or (long) real
bMask	form a bits mask (sequence of 1-bits)
lbMask	form a long bits mask (sequence of 1-bits)
sin	(long) real sine
cos	(long) real cosine
tan	(long) real tangent
\$cot	(long) real cotangent
aSin	(long) real arcsine
aCos	(long) real arccosine
aTan	(long) real arctangent
\$atan2	(long) real two-argument arctangent
sinh	(long) real hyperbolic sine
cosh	(long) real hyperbolic cosine
tanh	(long) real hyperbolic tangent
exp	(long) real exponential
ln	(long) real natural logarithm
log	(long) real base-10 logarithm
sqrt	(long) real square root
\$log2	truncated base 2 logarithm of constant
\$hash	compute hash code
size	size of a class or data type
\$ioSize	size of data type when written to file
\$bitsPerStorageUnit	bits in a storage unit
\$bitsPerChar	bits in a character unit
\$typeName	name of a type, given type code

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

displace	displace a pointer, address, or charadr
displacement, lDisplacement	distance from one address or charadr to another
eol	end-of-line string
eop	end-of-page string
tab	tab string
\$nulChar	null character
\$pageSize	storage units per page
\$charsPerPage	character units per page
\$charsPerStorageUnit	character units per storage unit
(x)Load	load a value (of type x) from an address
cLoad	load a character from a charadr
store	store a value into an address or charadr
newString	make a string from a charadr and an integer (length)
\$getToTop	put a string at top of string space
\$getInArea	put a string in an area's string space
newPage	get some pages
pageDispose	dispose of pages
newScratch	get some scratch space
\$newScratchChars	get some scratch space measured in chars
scratchDispose	dispose of scratch space
\$date	get the date
\$time	get the time
\$dateAndTime	get the date and time simultaneously
\$setTheDate	set the date, if necessary

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

\$assembleDate	convert year-month-date combination into standard representation
\$assembleTime	convert hour-minute-second combination into standard representation
\$assembleDateAndTime	combined \$assembleDate and \$assembleTime
\$disassembleDate	convert standard representation into year-month-date combination
\$disassembleTime	convert standard representation into hour-minute-second combination
\$disassembleDateAndTime	\$disassembleDate and \$disassembleTime
\$dateToStr	convert date representation to string
\$timeToStr	convert time representation to string
\$dateAndTimeToStr	combined \$dateToStr and \$timeToStr
\$strToDate	convert string to date representation
\$strToTime	convert string to time representation
\$strToDateAndTime	combined \$strToDate and \$strToTime
\$removeDateAndTime	parse date and time string
\$addToDateAndTime	add two dates and times
\$dateAndTimeDifference	subtract two dates and times
\$dateAndTimeCompare	compare two dates and times
\$dateFormat	whether date is GMT, local, or difference
\$timeFormat	whether time is GMT, local, or difference
\$convertDateAndTime	convert GMT time to local or vice versa
\$timeSubcommandsSet	whether GMT conversion info available

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

\$cpuTime	get system-dependent CPU time for current program
\$cpuTimeResolution	number of \$cpuTime units per second
\$timeout	pause for specified period
\$userID	return the system-dependent user ID, if available
\$cpuID	return the system-dependent CPU ID, if available
\$currentDirectory	name of system-dependent current working or connected directory or catalog
\$homeDirectory	home directory or catalog of current user
\$directory	list files in a directory
\$fileInfo	return information about a file
\$moduleInfo	information about objmod
\$collect	perform a garbage collection
\$checkConsistency	verify that MAINSAIL data structures are in order
\$addMemMngModule	specify module to invoke before memory management operations
\$removeMemMngModule	undo \$addMemMngModule
\$collectLock	used to prevent/permit garbage collections
\$overheadPercentExitValue	used to prevent thrashing

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

\$areaOf	determine area of pointer or string
\$clearArea	empty an area
\$clearStrSpc	empty an area's string space
\$defaultArea	default area
\$disposeArea	reclaim an area
\$disposeDataSecsInArea	dispose only data sections in area
\$findArea	find area with given title
\$inArea	determine if pointer or string in given area
\$newArea	allocate area
\$createCoroutine	create a coroutine
\$resumeCoroutine	continue or start execution in a coroutine
\$killCoroutine	get rid of a coroutine
\$skilledCoroutine	determine whether a coroutine has been killed
\$moveCoroutine	move coroutine to another point in tree
\$findCoroutine	return a pointer to a coroutine record, given its name
\$thisCoroutine	current coroutine
\$majorVersion, \$minorVersion	get MAINSAIL version number
\$maxChar	maximum character code
\$maxInteger	maximum integer
\$maxLongInteger	maximum long integer
\$minInteger	minimum integer
\$minLongInteger	minimum long integer

Table 20.1-1. System Procedures, Macros, and Variables Summary (continued)

\$platformNameAbbreviation, \$platformNameFull, \$platformNumber	identify target platform
\$systemNameAbbreviation, \$systemNameFull, \$systemNumber	identify target operating system
\$processorNameAbbreviation, \$processorNameFull, \$processorNumber	identify target processor
\$attributes	attributes of target system
\$charSet	character set of target operating system
\$preferredRadix	"natural" radix for addresses, etc.
\$compileTimeValue	information about current compilation
\$thisFileName	file name currently being compiled
\$clrConfigurationBit	clear bit governing runtime system
\$clrSystemBit	clear bit governing runtime system
\$setConfigurationBit	set bit governing runtime system
\$setSystemBit	set bit governing runtime system
\$tstConfigurationBit	examine bit governing runtime system
\$tstSystemBit	examine bit governing runtime system

Table 20.1-1. System Procedures, Macros, and Variables Summary (end)

21. Sample MAINSAIL Code

The code for a subset of the MAINSAIL utility HSHMOD is shown in Example 21-2. A program uses HSHMOD by creating a separate HSHMOD data section for each hash table. The header declarations for this subset of HSHMOD are shown in Example 21-1. The declarations for the full HSHMOD are stored in the MAINSAIL system source library, and may be picked up by a user program with:

```
REDEFINE $scanName = "hshHdr"; SOURCEFILE "(system library)";
```

```
# prefix class for hashed records
CLASS hashedRecord (
    STRING key;
    POINTER(hashedRecord) link;
);

# explicit class so user can classify pointers to it
CLASS hshCls (
    PROCEDURE hashInit (OPTIONAL INTEGER tableSize);
    PROCEDURE hashEnter (POINTER(hashedRecord) p);
    POINTER(hashedRecord)
    PROCEDURE hashLookUp (STRING key);
    POINTER(hashedRecord)
    PROCEDURE hashRemove (STRING key);
    POINTER(hashedRecord)
    PROCEDURE hashNext (POINTER(hashedRecord) p);
);

MODULE (hshCls) hshMod;
```

Example 21-1. HSHMOD Declarations

```

BEGIN "hshMod"

# this module maintains a general-purpose hash table

REDEFINE $scanName = "hshHdr"; # Pick up interface
SOURCEFILE "(system library)"; # declarations

DEFINE
    numCharsToHash      =      4,
    defaultTableSize    =      131;

INTEGER numberOfHashLists;

POINTER(hashRecord) ARRAY(0 TO *) hashList;

#      +-----+
#      0 |       | ----> linked list of all records
#      +-----+           whose keys hash to 0
#      1 |       | ----> linked list of all records
#      +-----+           whose keys hash to 1
#      2 |       | ----> linked list of all records
#      +-----+           whose keys hash to 2
#      3 |       | ----> linked list of all records
#      +-----+           whose keys hash to 3
#      |       |
#      |       |
#      |       |
#      +-----+

PROCEDURE hashInit (OPTIONAL INTEGER tableSize);
BEGIN
IF tableSize LEQ 0 THEN tableSize := defaultTableSize;
new(hashList,0,tableSize - 1);
numberOfHashLists := tableSize;
END;

```

Example 21-2. HSHMOD Source Text (continued)

```

INTEGER PROCEDURE hash (STRING key);
BEGIN
INTEGER h,i,j;

# s hashes to
#   (length(s) + 3 * char1 + 5 * char2 +
#     7 * char3 + 9 * char4)
#     MOD numberOfHashLists
#
# where chari represents ith character of s

h := length(key); i := h MIN numCharsToHash; j := 1;
WHILE i .- 1 GEQ 0 DO h .+ cRead(key) * (j .+ 2);
RETURN(h MOD numberOfHashLists) END;

POINTER(hashRecord) PROCEDURE search
  (STRING key;
   PRODUCES OPTIONAL INTEGER hashValue;
   PRODUCES OPTIONAL POINTER(hashRecord)
   beforeTarget);
BEGIN
POINTER(hashRecord) target;

# general-purpose search procedure

IF NOT hashList THEN hashInit; # automatic initialization
hashValue := hash(key);
beforeTarget := NULLPOINTER;
target := hashList[hashValue];
WHILE target AND target.key NEQ key DOB
  beforeTarget := target; target := target.link END;
RETURN(target) END;

```

Example 21-2. HSHMOD Source Text (continued)

```

PROCEDURE hashEnter (POINTER(hashRecord) p);
BEGIN # enter p at front of its hash list
INTEGER h;
IF NOT hashList THEN hashInit;
IF p THENB
    h := hash(p.key); p.link := hashList[h];
    hashList[h] := p END
EL errMsg("hashEnter: argument is NULLPOINTER") END;

POINTER(hashRecord) PROCEDURE hashLookUp (STRING key);
RETURN(search(key)); # return record with given key
                    # (Zero if not found)

POINTER(hashRecord) PROCEDURE hashRemove (STRING key);
BEGIN # remove record with given key
INTEGER          h;
POINTER(hashRecord) target,beforeTarget;
IF target := search(key,h,beforeTarget) THEN
    IF beforeTarget THEN beforeTarget.link := target.link
    EL hashList[h] := target.link;
RETURN(target) END;

POINTER(hashRecord) PROCEDURE hashNext
    (POINTER(hashRecord) p);
BEGIN
OWN INTEGER h;
POINTER(hashRecord) q;

# generate next record in hashList (successive calls
# starting with p = NULLPOINTER will generate all records,
# then NULLPOINTER)

```

Example 21-2. HSHMOD Source Text (continued)

```
IF NOT p THEN h := -1
EF q := p.link THEN RETURN(q)
EL h := hash(p.key);
DOB IF h .+ 1 GEQ numberOfHashLists THEN
    RETURN(NULLPTR);
    IF p := hashList[h] THEN RETURN(p) END END;

END "hshMod"
```

Example 21-2. HSHMOD Source Text (end)

The MAINSAIL Environment

22. The MAINSAIL Compiler

The MAINSAIL compiler translates MAINSAIL source text into ready-to-run object modules. The compiler has a full set of subcommands governing characteristics of the output object module. Compiler subcommands are listed in Table 22-1.

22.1. Code Generators

The MAINSAIL compiler uses a separate code generator for each hardware architecture. Each code generator module is of the same class and is dynamically selected and bound during the execution of the compiler. Since each code generator is written in MAINSAIL, any code generator can be used on any computing system. This provides for universal cross-compilation.

The platforms on which MAINSAIL runs at the time of this writing are shown in Table 22.1-1. XIDAK is constantly adding new systems to the list.

22.2. Disassemblers

Corresponding to each code generator, XIDAK supplies a disassembler capable of producing a text file that shows the original MAINSAIL source text interspersed with a mnemonic listing of the machine code generated by the compiler. This permits the user to evaluate the quality of the code emitted by the compiler and to compare the relative efficiency of different constructs.

22.3. Foreign Language Interface

MAINSAIL programmers can easily interface to code written in other programming languages through the facility called the Foreign Language Interface (FLI). The programmer supplies the FLI with a description of the foreign entry points, and the FLI automatically generates interface code that is linked with a MAINSAIL bootstrap and the foreign object module.

<u>Subcommand</u>	<u>Description</u>
ABORT	Abort this compilation
ACheck	Set default to emit code to catch arithmetic overflow, etc.
ACHECKALL	ACHECK unconditionally
ALIST	Allow disassembly
Check	Set default to emit code to catch subscript errors, etc.
CHECKALL	CHECK unconditionally
CMDLINE s	Add s to the end of the cmdLine list (nonsticky)
DEBUG	Make this module debuggable, and turn on INCREMENTAL
FLDXREF {f}	Write field cross reference (to file f (nonsticky))
FLI s	Generate code for foreign interface s
GENcode	Generate code
GENINLINES	Generate bodies for inline procs
INCREMENTAL	Allow output to be incrementally recompiled
ININTLIB f	Input intmod is in library f
INOBJFILE f	Input objmod is in file f (nonsticky)
INOBJLIB f	Input objmod is in library f
ITFXREF {f}	Write interface cross reference (to file f (nonsticky))
LOG	Show log info
MODTIME	Measure time spent in this module
MONITOR	Turn on PER{MOD,PROC,STMT} and {MOD,PROC}TIME
Optimize	Set default to optimize all procs
Optimize p	Optimize procs p = p1 p2 ... pn (nonsticky)
OPTIMIZEALL	Optimize all procs

Table 22-1. MAINSAIL Compiler Subcommands (continued)

OUTINTFILE f	Output intmod to file f (nonsticky)
OUTINTLIB f	Output intmod to library file f
OUTOBJFILE f	Output objmod to file f (nonsticky)
OUTOBJLIB f	Output objmod to library file f
PERMOD	Count total statements executed in the module
PERPROC	Count total statements executed in each proc
PERSTMT	Count times each statement is executed
PROCS	Show names of procs as they are parsed and generated
PROCTIME	Measure time spent in each proc
RECOMPILE p	Recompile procs p = p1 p2 ... pn (nonsticky)
REDEFINE x y	Do \$GLOBALREDEFINE x = [y];
RESPONSE	Get user response to error messages
RPC {C}	generate code for remote procedure call {in C}
SAVEON {f}	Create intmod containing all compiler info {save on file f}
SLIST {f}	Write source listing {to file f (nonsticky)}
SUBCOMMAND s	Execute MAINEX subcommand s
TARGET s	Generate for target system s
UNBOUND	Nonbound-invocation module
# s	A comment (s is ignored)

Table 22-1. MAINSAIL Compiler Subcommands (continued)

NOACheck	Turn off ACHECK
NOACHECKALL	Turn off ACHECKALL
NOALIST	Turn off ALIST
NOCCheck	Turn off CHECK
NOCHECKALL	Turn off CHECKALL
NODEBUG	Turn off DEBUG and INCREMENTAL
NOFLDXREF	Turn off FLDXREF
NOGENcode	Turn off GENCODE
NOGENINLINES	Turn off GENINLINES
NOINCREMENTAL	Turn off INCREMENTAL
NOININTLIB	Turn off ININTLIB
NOINOBJLIB	Turn off INOBJLIB
NOITFXREF	Turn off ITFXREF
NOLOG	Turn off LOG
NOMONITOR	Turn off PER{MOD,PROC,STMT} and {MOD,PROC}TIME
NOOptimize	Turn off OPTIMIZE
NOOptimize p	Do not optimize proc(s) p, where p = p1 ... pn
NOOPTIMIZEALL	Turn off NOOPTIMIZEALL
NOOUTINTLIB	Turn off OUTINTLIB
NOOUTOBJLIB	Turn off OUTOBJLIB
NOPROCS	Turn off PROCS
NOREDEFINE	Remove all global definitions
NOREDEFINE x	Remove global definition(s) of x, where x = x1 ... xn
NORESPONSE	Turn off RESPONSE
NORPC	Turn off RPC
NOSAVEON	Turn off SAVEON
NOSLIST	Turn off SLIST
NOUNBOUND	turn off UNBOUND

For backward compatibility:

LIBRARY f	Same as GENCODE, OUTOBJLIB f
OUTPUT {f}	Same as GENCODE {, OUTOBJFILE f}
NOLIBRARY	Same as NOOUTOBJLIB
NOOUTPUT	Same as NOGENCODE

Table 22-1. MAINSAIL Compiler Subcommands (end)

<u>Platform Abbrev.</u>	<u>Processor</u>	<u>Platform Name</u>
aeg	M68000	Apollo's Aegis on Motorola M68000
aix	System/370	IBM's AIX on IBM System/370
alnt	M68000	Alliant's CONCENTRIX on Motorola M68000
cms	System/370	IBM's VM/SP CMS on IBM System/370
hp20	MC68020/ MC68881	HP's HP-UX on Motorola MC68020/MC68881
hp38	80386	SCO's XENIX on HP Vectra with Intel 80386
hpux	M68000	HP's HP-UX on Motorola M68000
ip32c	Interpro 32C	Intergraph's System V UNIX on Interpro 32C
ipsc2	80386	Intel's iPSC/2 System V UNIX on Intel 80386
ix20	MC68020/ MC68881	Apollo's DOMAIN/IX on Motorola MC68020/MC68881
ixfpa	MC68020/ Weitek	Apollo's DOMAIN/IX on Motorola MC68020/Weitek FPA
ixpri	PRISM	Apollo's DOMAIN/IX on Apollo PRISM
sun2	M68000	Sun Microsystems' SunOS on Motorola M68000
sun3	MC68020/ MC68881	Sun Microsystems' SunOS on Motorola MC68020/MC68881
sun38	80386	Sun Microsystems' SunOS on Intel 80386
sun4	SPARC	Sun Microsystems' SunOS on SPARC
ultrix	VAX-11	DEC's ULTRIX-32 on VAX-11
vms	VAX-11	DEC's VAX/VMS on VAX-11
xcms	System/370 XA	IBM's VM/XA SP CMS on IBM System/370

Table 22.1-1. Computer Systems on Which MAINSAIL Is Supported

23. MAINDEBUG, the MAINSAIL Debugger

MAINDEBUG is a source-level debugger for MAINSAIL programs. The programmer interacts with the MAINSAIL source text by using a cursor to indicate the location of breakpoints.

MAINDEBUG can operate in either line-oriented or display-oriented mode. The display-oriented mode is integrated with MAINEDIT, so that single-keystroke commands move the terminal cursor over source text statements as they are executed. The display-oriented debugger keeps the source text in one or more windows and displays program output in a separate window.

When a program generates an error (e.g., an array subscript error), the debugger can be dynamically invoked to point at the offending statement. It can then be used to examine the call stack and the values of variables so that the cause of the error can be determined.

MAINDEBUG's command processor is highly flexible because it invokes the MAINSAIL compiler to process all expressions specified in commands. The debugger is also able to interpret MAINSAIL statements on the fly; this is useful, for example, to examine the effect of a procedure called with a certain set of arguments. A breakpoint can be placed at the beginning of the procedure, and then the procedure can be called by interpreting the call from the debugger. The user can then step through the procedure.

A summary of MAINDEBUG commands is shown in Table 23-1.

A ary,l1,u1,l2,u2,l3,u3	show array slice ary[l1 TO u1,...]
{+}B{@{m.}i}{[cond]}{:cmds}	set break at cursor (or mod m, iUnit i)
{n}{.i}C	continue {at iUnit i}, till nth break
{n}.C	continue at cursor, till nth break
.D dl;...;dn	compile defs or dcls dl;...;dn
E {m}	execute MAINSAIL exec {or module m}
.F p,f1,f2,...	V p.f1,p.f2,... (p can be unclassified)
H e1,e2,...	hex values of e1,e2,...
.H p1,p2,...	hex values of objects at p1,p2,...
{1}I	display {abbreviated} debug info
.I i1,i2,...	display info about identifiers
{n}J	step n times, jump into procs
K n	break when count = n
M	set to break context
M m	open module m (m can be a file name)
-M m	close m's intmod and dispose m's objmod
.M p	open module with data section p
{n}N	move to nth caller from current proc

Table 23-1. Debugging Command Summary (continued)

{n}-N	move to nth callee from current proc
{n}.N	move to where exception was raised
{n}-.N	undo .N command n times
O n	cursor to iUnit n, current module
OC s	open coroutine s
OI s	open intmod library s
OL s	open objmod library s
{-}OP s	set {clear} options s
Q	quit (exit the program)
+Q	exit MAINDEBUG
R	remove break at cursor
R@m.i	remove break at module m, iUnit i
R@@	remove all breaks
{n}S	step n times, do not enter procs
{+}T{@{m.}i}{[cond]}{:cmds}	same as B, except set temp break
V e1,e2,...	values of e1,e2,...
.V p1,p2,...	values of objects at p1,p2,...
XM a	examine memory at address a
XS s1;...;sn	execute statements s1;...;sn
<ECM>	enter MAINEDIT command mode

Table 23-1. Debugging Command Summary (end)

24. MAINEDIT, a Portable Text Editor

MAINEDIT is a portable, display-oriented text editor that supports simultaneous editing of multiple files in multiple windows. It supports different keystroke interpreters, or front ends, including emulators for the popular vi and EDT text editors. MAINEDIT uses a set of "display modules", each of the same class, to support a number of different display terminals. XIDAK can implement a display module for a new terminal easily and rapidly.

The first MAINEDIT front end was MAINED, which includes commands that take full advantage of MAINEDIT's multiple-buffer, multiple-window capabilities. The front ends that emulate other text editors lack some these commands, since they were not present in the original text editors; however, a MAINEDIT user can invoke any front end's commands from any other front end, so the MAINED buffer and window commands may be used from the other editor emulators.

Some of the other features of MAINED include:

- An "Again" command, which repeats the last command.
- Deletion, recovery, and copying of characters, words, lines, or pages.
- Execution of any MAINSAIL program within a MAINEDIT window.
- An "Undo" command, which undoes the last command.
- An "Abort" key, which aborts the current command or macro execution.
- A keyboard macro facility, which allows the user to invoke a series of commands with a single command character or macro name.

MAINEDIT display modules are currently available for the display terminals listed in Table 24-1.

<u>Module</u>	<u>Terminal(s)</u>
AM48	48-line Ann Arbor Ambassador
AM60	60-line Ann Arbor Ambassador
AT386	IBM PC/AT and compatibles
BIGSUN	Sun Microsystems workstation, arbitrary number of lines
BORRO	Apollo Computer workstation
D400	Data General DASHER
D460	Data General D410/460
D460C	Data General D410/460, 132 columns
DATAME	Datamedia 3000, Telemedia
EWY100	ELXSI-modified Wyse 100
FBORRO	Apollo Computer workstation
FRAME	Apollo Computer workstation
FFRAME	Apollo Computer workstation
HEATH	Heath (or Zenith) H-19
HP300H	large-screen Hewlett-Packard terminal
HPTERM	Hewlett-Packard terminals
LINDPY	any terminal; line-oriented
SUN	Sun Microsystems workstation
SUN3	Sun Microsystems window
SUN46	Sun Microsystems workstation, 46 lines
TELEVI	Teletext (except model 950)
TVI950	Teletext model 950
TRMCAP	any terminal for which information is available in a UNIX-style database
VIS550	Visual 550
VT100	VT100
VT102	VT102 (VT100 with insert and delete)
VT102M	VT102 (imperfect emulators)
WY43	Wyse WY-60
WY50	Wyse WY-50
WY5043	Wyse WY-50, 43-line mode
WY75	Wyse WY-75

Table 24-1. Available Display Modules

A complete summary of MAINED commands follows:

Command Mode

nA do last command again, count = n, original modifier
QA do last command again, original count and modifier
+nA do last command again, count = n, "+" modifier
Q+A do last command again, original count, "+" modifier
-nA do last command again, count = n, "-" modifier
-QA do last command again, original count, "-" modifier
("-" direction is towards beginning of file)
("+ direction is towards end of file)
.A anchor current window
+.A anchor at bottom
-.A anchor at top
n.A anchor, change size to n rows
+n.A anchor at bottom, change size to n rows
-n.A anchor at top, change size to n rows
..A unanchor current window
Q..A unanchor all windows
Q+..A unanchor all windows at bottom of screen
Q-..A unanchor all windows at top of screen

B break line, remove spaces
 nB break line, indent n spaces
 QB break line, leave spaces
 -B break line, remove spaces, leave cursor
 -nB break line, indent n spaces, leave cursor
 -QB break line, leave spaces, leave cursor
 .Bs edit buffer s, use current window if not on screen
 +.Bs edit buffer s, new window at bottom if not on screen
 n.Bs same as ".Bs", except n-row window
 +n.Bs same as "+.Bs", except n-row window
 -.Bs edit buffer s, new window at top if not on screen
 -n.Bs same as "-.Bs", except n-row window
 --{n}.Bs
 overlay (n-row) window at top
 ++{n}.Bs
 overlay (n-row) window at bottom
 ..{n}Bs
 edit s, making window 1/mth of screen, where m is the
 number of windows; but no window is allowed to be
 smaller than n lines
 Q.Bs change bufferName of current buffer to s
 Q..Bs change command front end to s
 +Q..Bs change command and view front ends to s
 -Q..Bs kill front end s

nC[C|W|L|P] copy n objects at and after
 QC[C|W|L|P] copy all objects at and after
 -nC[C|W|L|P] copy n objects before
 -QC[C|W|L|P] copy all objects before
 n.C center n lines at and after
 Q.C center all lines at and after
 CM push savedMode onto mode stack

nD[C|W|L|P] delete n objects at and after
 QD[C|W|L|P] delete all objects at and after
 -nD[C|W|L|P] delete n objects before
 -QD[C|W|L|P] delete all objects before
 ("D" copies text into delete buffer, but does not delete it;
 ".D" deletes text, but does not copy it into the delte buffer)

E escape to caller, if any
 QEs execute module s (dispose-bind-unbind)
 .E show name of currently executing module
 Q.E show names of all executing modules

F prompt to save altered buffers, then continue
 +F change autoSaveLimit (0 means no autoSave reminder)
 QF If a program invoked with "QE" is running, raise the
 exception \$abortProgramExcpt; otherwise, exit from
 MAINEDIT, continuation not allowed
 -QF prompt to save altered buffers, then pause
 .Fs edit file s, use current window if not on screen
 +.Fs edit file s, new window at bottom if not on screen
 n.Fs same as ".Fs", except n-row window
 +n.Fs same as "+.Fs", except n-row window
 -.Fs edit file s, new window at top if not on screen
 -n.Fs same as "-.Fs", except n-row window
 --{n}.Fs
 overlay (n-row) window at top
 ++{n}.Fs
 overlay (n-row) window at bottom
 ..{n}Fs edit s, making window 1/mth of screen, where m is the
 number of windows; but no window is allowed to be
 smaller than n lines
 Q.Fs change fileName of current buffer to s

 G go to first line of next page
 -G go to first line of previous page
 p.lG go to page p, line l
 .G go to first line of current page
 .lG go to line l of current page
 pG go to first line of page p
 +nG first line of current page + n
 -nG first line of current page - n
 (start with "Q" to set mark ("@" command) before going)

 nH undo previous n changes
 QH undo all changes on current line
 -nH redo next n changes
 -QH redo all changes on current line

 I enter insert mode
 lIB insert a buffer (name is asked)
 nICc insert n c's (n required)
 QICc insert c's to right margin
 lIF insert a file (name is asked)
 nIL insert n blank lines (n required)
 QIL insert blank lines to end of window
 .I insert blank line, enter insert mode
 n.I insert blank line, indent n spaces, enter insert mode

nJ join next to current line, n separating spaces
QJ join next to current line, leave spaces
-nJ join current to previous line, n separating spaces
-QJ join current to previous line, leave spaces
.J fill current paragraph to right margin of
window
n.J fill n lines
+.J fill and justify to right margin of window
-.J fill starting at cursor column
Q.J fill all remaining paragraphs in buffer
.mJ fill to right margin (justify) n column m
nQ.J fill next n paragraphs
(All modifiers may be combined; i.e., nQ+-.m.J means fill next n
paragraphs from cursor column, justifying to column m).

nK delete (kill) n characters at and to right
QK delete all characters at and to right
-nK delete n characters to left
-QK delete all characters to left
.K prompt to kill each buffer (prompts to save)
Q.K kill one buffer (prompts for name)
..K kill a character without copying into delete buffer

nL[C|W|L] make n objects lower case
QL[C|W|L] make all objects lower case

nM[C|W|L|P] move current object n further
QM[C|W|L|P] move current object to end
-nM[C|W|L|P] move current object n before
-QM[C|W|L|P] move current object to start
.M[C|W|L] mark the appropriate delete buffer

0N refresh message line
nN refresh n lines at and below in current window
-nN refresh n lines above in current window
QN refresh entire screen
Q.N refresh current window

O enter overstrike mode
nOCC overstrike n c's (n required)
QOCC overstrike c's to right margin
.O set editor option
-.O clear editor option
.O? show option settings

P insert page mark above

Q emphasize the command

R[C|W|L|P] recall and insert group of objects
nR[C|W|L|P] recall and insert n objects
QR[C|W|L|P] recall and insert all objects to mark
(".R" means leave in delete buffer)
RM pop mode stack into curMode
.RM set curMode to top of mode stack (not popped)
QRM set curMode to savedMode

nSc skip right to nth occurrence of character "c"
+Sc skip over c's
-nSc skip left to nth occurrence of character "c"
-+Sc skip left over c's
QnSc skip down to nth "c"-line
Q+Sc skip down to next line not starting with c
-QnSc skip up to nth "c"-line
Q+Sc skip up to next line not starting with c
(a "c"-line is a line with first visible char equal to "c".
A <sp>-line is one with no visible characters.)

Ts<eol> search right and all lines down for s ...
T<eol> search right and all lines down for last target(s)
nTs<eol> search right and n-1 lines down for s ...
QTr<eol>s...<eol><eol>
search right and all lines down for r or s or ...
QnTr<eol>s...<eol><eol>
search right and n-1 lines down for r or s or ...
(-T searches left and up)
(qualifying with "+" wraps around buffer beginning or end)
(qualifying with "QQ" makes into a line search)
("{-}.T" is an "identifier" search, i.e. the target cannot be
bordered by an alphabetic or digit)

U same as "L", except convert to upper case

V give character position, prompt for new one

Q..Vs change view front end to s

W scroll up 4/5 of a window
nW scroll up n lines
QW scroll up all lines
-W scroll down 4/5 of a window
-nW scroll down n lines
-QW scroll down all lines
n.W move current line to line n from top of window
-n.W move current line to line n from bottom of window

nX move to column n of window (x-coordinate)
-nX move to window width - n + 1
QnX put right margin at column n (from line origin)
-QnX put left margin at column n (from line origin)

nY move to row n of window (y-coordinate)
-nY move to row n of window, count from bottom up
QY set current window to maximum size
QnY set number of window rows to n (n = 0 kills window)
+QY expand window to bottom of screen
+QnY expand window n rows
-QY synonym for Q0Y
-QnY contract window n rows
n.Y move cursor to nth next window on screen
-n.Y move cursor to nth previous window on screen
Q.Y move cursor to bottommost window on screen
-Q.Y move cursor to topmost window on screen

Z same as "S", except delete skipped objects
(".Z" means do not delete, but put into delete buffer;
"..Z" means delete, but do not put into delete buffer)

n<bs> move left n columns
Q<bs> move to left margin

n<tab> move cursor to nth next tab stop
-n<tab> move cursor to nth previous tab stop
Q<tab> set tab stops

n<lf> move down n rows
Q<lf> move to last row

<abort> abort current command, enter command mode

n<eol> move to left margin of nth next line
-n<eol> move to left margin of nth previous line

```

n<sp>  move right n columns
Q<sp>  move to last column
(equivalent to ">")

'n      insert character with decimal code "n"
'Bn     insert character with binary code "n"
'Hn     insert character with hexadecimal code "n"
'On     insert character with octal code "n"

n(      move left n words
n.(     move to one past end of nth word to left
Q(      move to first visible character of line

n)      move right n words
n.)     move to one past end of (n-1)st word to right
Q)      move to after end of line

,       invoke named macro

/x.../  define x to be ..., where x is a macro name
(... is carried out as it is typed in)

n<      move left n columns
Q<      move to first column

=       show line info
Q=      show buffer info
+Q=     show buffer info with front end info

n>      move right n columns
Q>      move to last column

.@      set mark to current location
@       go to marked location
Q@      set mark, go to previously marked location

n\      move down n rows
Q\      move to bottom row of window

n^      move up n rows
Q^      move to top row of window

n<del>  move left n columns
Q<del>  move to left margin

```

n.+v<eol>	An := An + v
n.-v<eol>	An := An - v
-n.-v<eol>	An := v - An
n.*v<eol>	An := An * v
n./v<eol>	An := An / v
-n./v<eol>	An := v / An
n.^v<eol>	An := An ^ v (raise An to the power v)
-n.^v<eol>	An := v ^ An (raise v to the power in An)
n._v<eol>	An := v (and set An's format to v's)
n.=	Display value of accumulator n
Q.=	Display value of all active accumulators

Overstrike Mode

<bs>, 	move left 1 column, except end of previous line if at left margin
<tab>	overstrike spaces to next tab stop
<lf>	move down 1 row
<eol>	move to left margin of next line

Insert Mode

<bs>	move left 1 column
<tab>	insert spaces to next tab stop
<lf>	move down 1 row
<eol>	break line, move cursor to start of new line
	delete character to left, except join current line to previous line if at left margin (like "-QJ" command)

25. MAINPM, the MAINSAIL Performance Monitor

MAINPM is a performance monitor for MAINSAIL programs. Once a MAINSAIL program is functionally correct, MAINPM can be used to isolate performance problems. MAINPM lets the programmer examine the time used by a program in different degrees of granularity, based on module, procedure, or statement. Program execution can also be sampled with periodic interrupts. Consumption of string and chunk (array, record, and data section) space can be monitored. MAINPM can produce both deep and shallow information as well as a source listing annotated with statement counts.

Sample MAINPM output is shown in Examples 25-1 and 25-2.

NAME (mod or mod.proc)	SHALLOW TIME		DEEP TIME		STMT COUNT
-----	(seconds)	(%)	(seconds)	(%)	-----
NUMBER.FIBONACCI	45.057	99.64	45.057	99.64	485570
NUMBER.F	.003	.01	.003	.01	22
NUMBER.INITPROC	.002	.00	45.169	99.89	9
NUMBER.IFACTORIAL	.000	.00	.000	.00	12

Total execution time: 45.221 seconds

Example 25-1. Timing and Statement Counts Table

STATEMENT COUNTS

SOURCE FILE: number

```
BEGIN "number"

LONG INTEGER PROCEDURE iFactorial (INTEGER n);
BEGIN
LONG INTEGER total;
INTEGER i;
1 total := 1L;
1 FOR i := 2 UPTO n DO
9         total .* cvli(i);
1 RETURN(total)
END;

LONG INTEGER PROCEDURE f (INTEGER n);
# Return n factorial.
11 IF n = 0 THEN
1         RETURN(1L)
10 ELSE RETURN(cvli(n) * f(n - 1));

LONG INTEGER PROCEDURE fibonacci (INTEGER n);
242785 IF n LEQ 1 THEN
121393         RETURN(cvli(n))
121392 ELSE RETURN(fibonacci(n - 2) + fibonacci(n - 1));

INITIAL PROCEDURE;
BEGIN
1 ttyWrite("10 factorial computed recursively is ",
f(10),eol);
1 ttyWrite("10 factorial computed iteratively is ",
iFactorial(10),eol);
1 ttyWrite("The 25th Fibonacci number is ",
fibonacci(25),eol)
END;

END "number"
```

Example 25-2. Source Text with Statement Counts

26. The MAINSAIL Structure Blaster

The MAINSAIL Structure Blaster allows an arbitrary MAINSAIL data structure to be written to or read from a file with a single procedure call. The file I/O is performed as efficiently as the underlying operating system permits.

The Structure Blaster may be used to "checkpoint" a data structure at a given point in a program's execution, or may function as a data base primitive. In addition, since a facility is provided to translate a structure from one machine's format to another, a data structure built on one machine may be shipped to another (presumably faster) machine for processing by another MAINSAIL program, then shipped back to the originating machine.

The headers of some of the Structure Blaster procedures are shown in Table 26-1. \$structureWrite writes a structure to a file; \$structureRead reads a structure from a file; \$structureDispose frees up the memory occupied by a structure; and \$structureCopy makes a copy of a structure. In each case, the parameter "root" is a pointer to the (arbitrary) structure on which the operation is to be performed.

Facilities are also provided to write a human-readable form of a structure to a text file (or to allow a user to enter a structure as text and have it "compiled" by the Structure Blaster).

```

LONG INTEGER
PROCEDURE    $structureWrite
              (POINTER(dataFile) f;
              POINTER root;
              OPTIONAL LONG INTEGER
                startPageOrPos;
              MODIFIES OPTIONAL
                POINTER($strucInfo)
                strucInfo;
              OPTIONAL BITS ctrlBits);

POINTER
PROCEDURE    $structureRead
              (POINTER(dataFile) f;
              OPTIONAL LONG INTEGER
                startPageOrPos,
                numPagesOrSize;
              PRODUCES OPTIONAL LONG INTEGER
                actualNumPagesOrSize;
              OPTIONAL BITS ctrlBits;
              OPTIONAL POINTER($area) area);

PROCEDURE    $structureDispose
              (MODIFIES POINTER root;
              OPTIONAL BITS ctrlBits);

POINTER
PROCEDURE    $structureCopy
              (POINTER root;
              OPTIONAL BITS ctrlBits;
              OPTIONAL POINTER($area) area);

```

Table 26-1. Selected Structure Blaster Procedure Headers

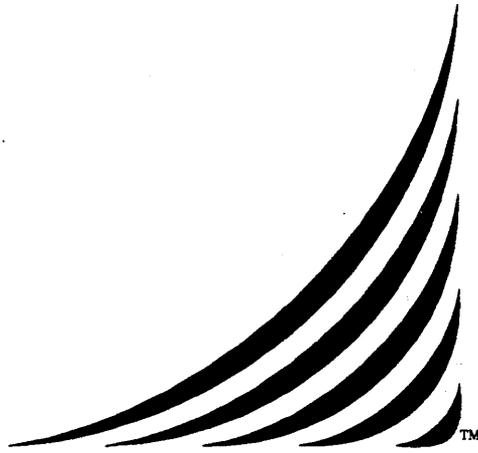
27. MAINSAIL STREAMS

STREAMS is a collection of facilities for distributed applications, process and device control, and enhancements to the functionality of MAINSAIL coroutines. At present, STREAMS is still under development, so not all facilities are implemented on all systems where they could be, and some interface changes may still be made.

The main high-level function provided by STREAMS is the RPC (remote procedure call) package. RPC allows interprocess communication to look like calls to an ordinary MAINSAIL module. It requires the programmer to write an "RPC server", a module that provides a set of functions (each implemented as an interface procedure), and compile it with a special compiler subcommand to produce two modules that are both compiled with the regular compiler. The two modules transmit and receive the server interface procedure arguments between processes; one runs in the server process, and the other in the process (the "RPC client") that invokes the server functions.

The STREAMS package includes the Scheduler, which allows one coroutine to run while another blocks. Each coroutine looks like an ordinary sequential MAINSAIL application; coroutines are rescheduled automatically when they perform I/O operations. In conjunction with RPC, the Scheduler allows a MAINSAIL application to be distributed among a number of processes to achieve coarse-grain parallel processing; this allows programmers to take advantage of multi-processor systems and high-speed networks to speed up their applications.

STREAMS also provides a large set of procedures for low-level I/O and server management. Typically, programmers use this level of STREAMS only to control special devices (like terminals), since the RPC mechanism is a better way of performing most server functions than low-level STREAMS facilities.



Master Index

24 March 1989

xidakTM

28. Master Index

The MAINSAIL master index combines the separate indices in each MAINSAIL document into one large index. It lists an abbreviation for the relevant document name as well as the page in the document with each index entry. At the beginning of the index is the list of documents covered, including title, abbreviation, and date of issue.

Because MAINSAIL documents are updated from time to time, the page number in the master index may not coincide exactly with the page number in the document if the index and document were issued on different dates. The date of any document may be found on its cover page.

If you find that a topic you wish to look up is not listed in the master index, you may send a "User Change Request" (UCR) form to XIDAK asking that the topic be covered in the documentation. UCR forms are available from XIDAK upon request. For more urgent problems, XIDAK's customer service personnel may be contacted by telephone.

Master Index

Abbreviations:

KSTRMU = STREAMS and MAINKERMIT User's Guides
(24 March 1989)
M1 = MAINSAIL Language Manual Part I
(24 March 1989)
M2 = MAINSAIL Language Manual Part II
(24 March 1989)
MEDTU = MAINEDIT User's Guide
(24 March 1989)
MTUT1 = MAINSAIL Tutorial, Part I
(24 March 1989)
MTUT2 = MAINSAIL Tutorial, Part II
(24 March 1989)
TOOLU = MAINSAIL Tools User's Guides
(24 March 1989)

! M1 34, 37; MTUT1 98

"

command TOOLU 92
searching in debugger MTUT2 46

M1 8; MTUT1 34; TOOLU 32, 301
#DOWN TOOLU 82
#LEFT TOOLU 82
#NEXT TOOLU 82
#PREV TOOLU 82
#RIGHT TOOLU 82
#UP TOOLU 82

\$ M1 9, 120
prefix in identifiers MTUT1 10

& M2 60; MTUT2 5, 19
(string concatenation) MTUT1 15

' command MEDTU 40; TOOLU 92

(M1 31, 39
 and) MTUT1 19
 command MEDTU 23; TOOLU 92
 commands MEDTU 30
 (service protocol table) KSTRMU 64, 115

) M1 31, 39
 command MEDTU 23; TOOLU 92
 commands MEDTU 30

* M1 34, 61; MEDTU 8, 10, 25; MTUT1 19, 98; TOOLU 12
 prompt TOOLU 279

** M1 240

+ M1 34; MEDTU 28; MTUT1 19, 98
 +F command MEDTU 73
 +Q command TOOLU 63

, command MEDTU 75

- M1 33, 34; MEDTU 28; MTUT1 19, 97, 98
 -M command TOOLU 80

. M1 76
 in dotted operations M1 40

.! MTUT1 128

.& MTUT1 128

.* MTUT1 128
 command MEDTU 79

+. MTUT1 128
 command MEDTU 79

.- MTUT1 128
 command MEDTU 79

./ MTUT1 128
 command MEDTU 79

.= command MEDTU 79

.^ MTUT1 128
 command MEDTU 79

._ command MEDTU 79

.A commands MEDTU 58

.B commands MEDTU 59, 62

.C commands MEDTU 46

.CLR MTUT1 128

.D command TOOLU 74

.DIV MTUT1 128

.F commands MEDTU 59, 62
.H command TOOLU 77
.IOR MTUT1 128
.J commands MEDTU 50
.K commands MEDTU 62
.M command TOOLU 79
.MAX MTUT1 128
.MIN MTUT1 128
.MOD MTUT1 128
.MSK MTUT1 128
.N command TOOLU 81
.O commands MEDTU 82
.SHL MTUT1 128
.SHR MTUT1 128
.V command TOOLU 77, 87
.XOR MTUT1 128

/ M1 34; MTUT1 98
command MEDTU 74; TOOLU 64

: M1 78; MEDTU 25, 84
:= M1 43, 80; MTUT1 28, 122

; M1 10

< M1 34; MTUT1 98
(less than) MTUT1 31
and > in syntax descriptions KSTRMU 1; M1 4; TOOLU 1, 52, 122, 150, 188
command MEDTU 23; TOOLU 92
commands MEDTU 30
for strings M1 33
<< M1 240
<= M1 240
<> M1 240
<abort> MEDTU 2, 69; TOOLU 56
<bs> MEDTU 2, 30
as macro ID MEDTU 75
 MEDTU 2, 30
<DELETEDPAGES> MEDTU 42
<ecm> MEDTU 2, 22; TOOLU 57
<eol> MEDTU 2, 30
as macro ID MEDTU 75
to line-oriented debugger TOOLU 64
<esc> MEDTU 2
<lf> MEDTU 2, 30

as macro ID MEDTU 75
<sp> MEDTU 2, 30
<suppress-output> key MEDTU 71
<tab> MEDTU 2, 30, 72, 83
as macro ID MEDTU 75

= M1 34, 67, 132; MTUT1 16, 30, 98
command MEDTU 10, 71; TOOLU 65
for strings M1 33

> M1 34; MTUT1 98
(greater than) MTUT1 31
command MEDTU 23; TOOLU 92
commands MEDTU 30
for strings M1 33
in stream names KSTRMU 19
prompt TOOLU 281

>= M1 240

>> M1 240

? command TOOLU 64

@
command TOOLU 63
commands MEDTU 33

[M1 48, 150
and] in command descriptions MEDTU 3
and] in macro text MTUT2 47
and] in syntax descriptions KSTRMU 2; M1 4; TOOLU 2, 52, 122, 150, 188

\ TOOLU 12, 59, 285
command MEDTU 23
commands MEDTU 30
in text forms TOOLU 157

] M1 48, 150

^ M1 34
command MEDTU 23
commands MEDTU 30

_ M1 240
_final procedure KSTRMU 46
_init procedure KSTRMU 45

{ M1 240; TOOLU 62
and } in command descriptions MEDTU 3
and } in syntax descriptions KSTRMU 2; M1 4; TOOLU 2, 52, 122, 150, 188

|
in command descriptions MEDTU 3
in syntax descriptions KSTRMU 2; M1 4; TOOLU 2, 52, 122, 150, 188

} M1 240; TOOLU 62

1I command TOOLU 78
1IB command MEDTU 39
1IF command MEDTU 39

A

command TOOLU 66
commands MEDTU 69
\$a20 M1 221
abbreviation
of buffer names MEDTU 26
of keyword MTUT1 32
platform name M1 222; M2 199
processor name M1 221; M2 201
system name M1 223; M2 258
ABORT compiler subcommand TOOLU 17
abort MEDTU 2
aborting
a command MEDTU 69
terminal output MEDTU 71
\$abortProcedureExcpt M1 173, 224; M2 16; MTUT1 273, 292
in dying coroutines M2 156
\$abortProgramExcpt M1 175, 224; M2 16; MTUT1 273; TOOLU 63
abs M2 17
absolute value (abs) M2 17
\$acceptClient KSTRMU 60
accumulators MEDTU 79
ACHECK
"\$DIRECTIVE" directive M1 168
compiler subcommand TOOLU 17
ACHECKALL
"\$DIRECTIVE" directive M1 168
compiler subcommand TOOLU 17
ACKER example module MTUT1 84
Ackermann's function MTUT1 83
aCos M2 17

adaptable RPC clients KSTRMU 36
 ADD
 INTLIB command TOOLU 234
 MODLIB command TOOLU 311
 ADDDATA LIB command TOOLU 263
 addition of date and time M2 20
 \$addMemMngModule M2 18, 221
 ADDNUM example module MTUT1 314
 ADDRESS M1 22
 address MTUT1 284; MTUT2 1
 classified M1 74
 examining contents of TOOLU 88
 invalid M1 22
 of an array M2 21
 RPC parameter KSTRMU 26, 27
 space MTUT1 292
 unaligned M1 22
 unclassified M1 75
 addressCode M1 220
 ADDTEXT LIB command TOOLU 263
 \$addToDateAndTime M2 20; MTUT2 46
 \$adrOfFirstElement M2 21
 advanced STREAMS KSTRMU 6, 10
 \$aeg M1 222, 223
 again command MEDTU 69
 \$aix M1 222
 aligned address M1 22
 alignment
 of addresses MTUT1 284, 289; MTUT2 45
 of chunks M1 81
 of storage units M1 22
 of Structure Blaster structures TOOLU 152
 allocation
 efficient M1 206
 of areas M2 177
 of array M1 62; M2 174; MTUT1 159
 of data section M1 111; M2 174
 of module M1 111; M2 33, 153, 174
 of multiple records M2 180
 of record M1 74; M2 71, 174; MTUT1 149
 \$allYearDigits M2 21
 bit M2 100
 \$almostOutOfMemoryExcp M1 224; M2 22
 \$alnt M1 222
 aLoad M2 161
 \$altErase KSTRMU 109
 altering an existing file MTUT1 118

alterOK M2 23; MTUT1 118
 bit KSTRMU 51; M2 192, 224
\$ALWAYS M1 98; MTUT2 6
\$ALWAYSINLINE M1 98
AM48 module MEDTU 134
AM60 module MEDTU 134
Ambassador MEDTU 134
ancestry of coroutines M1 181
anchoring a window MEDTU 56
AND M1 34; MTUT1 31, 98, 123
angle of ray with x-axis M2 28
\$aos M1 223
Apollo Computer, display module for MEDTU 135
append M2 23
 bit M2 231
arbitrary characters, inserting MEDTU 40
arccosine (aCos) M2 17
arcsine (aSin) M2 24
arctangent
 aTan M2 28
 \$atan2 M2 28
\$area M1 208; M2 2
area M1 206
 allocation M2 177
 clearing M2 43
 clearing string space of M2 43
 disposing M2 112
 finding M2 129
 pointer or string in M2 23, 148
\$areaAttr TOOLU 180
\$areaOf M1 207; M2 23
\$arg M1 138; MTUT1 190
argument
 macro M1 137
 optional M1 90
 order of evaluation M1 93
 procedure M1 86, 87, 89, 90, 93
 repeatable M1 90
 to FLI procedures TOOLU 47
arguments, command line M2 137; MEDTU 7
arithmetic
 checking M1 168
 error MTUT1 273
 functions MEDTU 79
 operators MTUT1 19
 overflow M2 24, 125
\$arithmeticChecked bit M2 171

\$arithmeticExcp M1 224; M2 24
 ARRAY M1 61
 array MTUT1 158
 address of first element M2 21
 allocation M1 62; M2 174; MTUT1 159; MTUT2 47
 assignment M1 66
 bounds M1 61, 69
 changing size of MTUT1 172
 clearing M1 66; M2 41
 comparison M1 67
 conversion M2 75
 copying M2 62
 declaration M1 61
 dimensions M1 61
 disposal M1 62; M2 111
 element access M1 65
 examining slices of TOOLU 66
 initialization M1 62, 66; MTUT1 167
 long MTUT1 172
 name M1 69
 new upper bound M2 183
 parameter M1 94
 pseudo-fields M1 69
 reordering by parallel index array TOOLU 349
 reversing TOOLU 349
 size of M2 247
 slice TOOLU 66
 subscript error M2 257
 unit TOOLU 160
 variable-bounded M1 61; MTUT1 170
 \$arrayType M1 70
 arrow keys MEDTU 133
 ary_alloc_status KSTRMU 44
 ary_dims KSTRMU 44
 ary_first_elem KSTRMU 44
 ary_lb1 KSTRMU 44
 ary_lb2 KSTRMU 44
 ary_lb3 KSTRMU 44
 ary_type KSTRMU 44
 ary_ub1 KSTRMU 44
 ary_ub2 KSTRMU 44
 ary_ub3 KSTRMU 44
 ASCII MTUT2 40
 \$ascii M1 226
 aSin M2 24
 \$assembleDate M2 25
 \$assembleDateAndTime M2 26

\$assembleTime M2 27
Assignment
 Expression M1 31; MTUT1 122
 Statement M1 43; MTUT1 28
assignment M1 80
 compatibility M1 42; MTUT1 154
asynchronous
 execution simulation KSTRMU 76
 interrupt catching KSTRMU 90
AT386 module MEDTU 137
aTan M2 28
\$atan2 M2 28
attribute in text form TOOLU 158
\$attributes M2 29
attributes, system M2 29
automatic sourcefile M1 158
autosave limit MEDTU 73

B

 command TOOLU 68
 commands MEDTU 48
B@ command TOOLU 73
back end MEDTU 4
back end, specifying MEDTU 64
BACKEND keyword MEDTU 16
backspace KSTRMU 105
BANDB example module MTUT1 102
base
 2 logarithm M2 165
 file (LIB) TOOLU 244
 of bits constant M1 18
basic
 command MEDTU 29
 STREAMS KSTRMU 6
batch editing MEDTU 148
baud
 rate MEDTU 6
 TTY KSTRMU 8, 105
BAUDRATE keyword MEDTU 14
\$becomeServer KSTRMU 24
BEGIN M1 45
Begin Statement M1 45; MTUT1 29
\$BEGINC M1 152
BEGINSCAN M1 156; MTUT2 54
BIGSTR example module MTUT1 29

BIGSUN module MEDTU 138
 binary M1 18; M2 29
 bit M2 76, 82, 90, 219
 search for bugs MTUT2 36
 tree MTUT1 154
 bind M1 111; M2 29
 bindable module M2 33
 binding a module MTUT1 198
 \$bindService KSTRMU 60, 74
 BINTR2 example module MTUT1 293
 BINTRE example module MTUT1 155
 bit
 clearing (CLR) M1 34, 37; MTUT1 98
 masking (MSK) M1 34, 37; MTUT1 98
 vectors MTUT1 91
 bit mask (bMask, lbMask) M2 31, 158
 bit shifting operators (SHL, SHR) M1 34; MTUT1 98
 bit testing operators (TST, TSTA, NTST, NTSTA) M1 34; MTUT1 98
 BITS M1 17
 bits MTUT1 91
 guaranteed range MTUT2 45
 string parse M2 219
 bitsCode M1 220
 \$bitsPerChar M2 30; MTUT1 285
 \$bitsPerStorageUnit M1 11; M2 31; MTUT1 131, 285
 bitwise operations (IOR, XOR, MSK, CLR) M1 37
 BKWRD2 example module MTUT1 319
 BKWRDS example module MTUT1 72
 blank space, removing M2 221, 223
 bLoad M2 161
 \$block KSTRMU 16, 63, 92, 103
 block copying M2 62
 blocking
 I/O KSTRMU 12
 in RPC calls KSTRMU 24
 on I/O MTUT1 319
 bMask M2 31
 body, procedure M1 84
 boLoad M2 161
 BOOLEAN M1 15
 boolean MTUT1 30
 booleanCode M1 220
 BOOTFILENAME CONF command TOOLU 197
 bootstraps, making TOOLU 193
 border MEDTU 84
 BORRO module MEDTU 135
 bound

- data section M1 106; MTUT1 196
- modules M2 153
- modules (as invoked from MAINEX) TOOLU 279
- bounds of array M1 69
- bracketed
 - text M1 134; MTUT1 178
 - text macro MTUT2 47
- break M2 32
 - bit M2 231
 - character (scan) M2 228
 - on procedure count TOOLU 78
 - TTY KSTRMU 8, 108
- breaking
 - line in MAINEX subcommand TOOLU 285
 - lines MEDTU 48
- breakpoint TOOLU 55
 - at specified offset TOOLU 73
 - commands TOOLU 68, 86
 - conditions TOOLU 68
 - continuing from TOOLU 73
 - in invoked procedure TOOLU 83
 - permanent TOOLU 68
 - removing TOOLU 84
 - removing at specified offset TOOLU 84
 - temporary TOOLU 86
 - temporary, at specified offset TOOLU 86
 - where to set TOOLU 69
- breakpoints
 - displaying list of TOOLU 77
 - removing all TOOLU 85
- \$briefFormat M2 33, 264
 - bit M2 99
- BS (backspace) key MEDTU 2
- \$bsFormat M2 129
- BSMNT example module MTUT1 236
- bubble sort TOOLU 341
- bucket, hash MTUT1 133
- buffer MEDTU 3
 - delete MEDTU 42
 - deleting MEDTU 62
 - end MEDTU 25
 - inserting MEDTU 39
 - listing status of MEDTU 63
 - RPC parameter KSTRMU 26, 27
- buffer names
 - abbreviated MEDTU 26
 - changing MEDTU 62

buffers, moving among MEDTU 59
bugs
 binary search for MTUT2 36
 tracking collection-influenced MTUT2 34
\$BUILTIN M2 2
byte M1 11; MTUT1 284

C

calling (from) TOOLU 19, 40
command TOOLU 73
commands MEDTU 46
LIB mode TOOLU 254
RPC KSTRMU 40
cache
 (of files) MTUT2 12; TOOLU 213
 of files M1 195
CALC example module MTUT1 77
call
 chain TOOLU 190
 macro M1 137
 procedure M1 86, 87, 90, 99
 stack TOOLU 80
call chain, searching M2 235
CALLS module TOOLU 54, 190
\$canFindModule M2 33
\$cannotFallOut M2 33
 bit M2 206
\$cannotReturn M2 34
 bit M2 118, 206
\$canonicalHostName KSTRMU 60
carriage return KSTRMU 99, 106
CASE M1 48
Case Statement M1 48; MTUT1 129
case
 conversion commands MEDTU 49
 sensitivity in file name MTUT2 42
 sensitivity in file names M1 225
 upper and lower M1 3, 6; MEDTU 20
Case Statement, breakpoint on TOOLU 70
\$CASEC M1 149
\$caseIndexExcpt M1 224; M2 34
caseless string comparison M2 56, 117
catch-all
 selector ([]) M1 48
 selector in \$CASEC M1 150

CD LIB command TOOLU 255
 ceiling M2 34
 centering lines MEDTU 46
 chain, comparison M1 38
 chained comparison MTUT1 183
 character M1 6
 address (charadr) M1 23
 and string system procedures MTUT1 59
 as integer constant (e.g., 'A') M1 15
 code MTUT1 14, 54
 search TOOLU 92
 set M2 36
 set assumptions MTUT2 40
 set guarantees M1 6
 set translation M1 214
 string MTUT1 14
 unit MTUT1 106, 284
 units of scratch space M2 182
 units per page M2 37
 units per storage unit M2 37
 character code, maximum M2 167
 character unit, size in bits M2 30
 ScharacterRead M2 35; MTUT1 295; MTUT2 8
 characters
 reading from a file M2 35
 writing to a file M2 36
 ScharacterWrite M2 36; MTUT1 295; MTUT2 8
 CHARADR M1 23
 charadr MTUT1 284; MTUT2 1
 examining contents of TOOLU 88
 RPC parameter KSTRMU 26, 27
 charadrCode M1 220
 \$charSet M2 36
 \$charsInArea bit TOOLU 168, 173
 \$charsPerPage M2 37; MTUT1 291
 \$charsPerStorageUnit M2 37; MTUT1 285
 CHECK M1 164
 "\$DIRECTIVE" directive M1 165
 compiler subcommand TOOLU 18
 CHECKALL
 "\$DIRECTIVE" directive M1 165
 compiler subcommand TOOLU 18
 CHECKCONSISTENCY MAINEX subcommand TOOLU 288
 \$checkConsistency M2 37
 \$checked bit M2 171
 CHECKING M1 164
 child process KSTRMU 5, 9, 65, 67

- chunk M1 206
 - alignment M1 81
 - information about TOOLU 304
 - unit TOOLU 158
- CLASS M1 73
- class MTUT1 144
 - declaration M1 73
 - descriptor M2 114
 - explicit specification MTUT1 223
 - explicitly specified in field variable M1 78
 - forward M1 74
 - information (field names and types) M2 38
 - name M2 39
 - prefix M1 78
 - related M1 80
 - unit TOOLU 159
 - with procedure fields M1 110
- class creation, dynamic M2 69
- class descriptor, creation M2 69
- \$classCode M1 154, 220
- \$classDscr M2 38, 69, 114
- \$classDscrFor M2 38
- classified
 - address M1 74
 - pointer M1 74; MTUT1 145
- \$classInfo M2 38; MTUT1 298
- \$className M2 39; MTUT1 298
- \$CLASSOF M1 155
- cleaning
 - up after a procedure M1 174; M2 16
 - up after a procedure or program MTUT1 273
- clear M2 41; MTUT1 289
 - pending I/O KSTRMU 8
- \$clearArea M1 207; M2 43
- \$clearFileCache M2 45
- \$clearStream KSTRMU 101
- \$clearStrSpc M1 207; M2 43
- client KSTRMU 58
 - process KSTRMU 5
 - RPC KSTRMU 21, 32
- client/server rendezvous KSTRMU 74
- cLoad M2 45, 161
- close M2 46
- closed procedure call M1 98
- \$closedFile M2 47
- CLOSEEXELIB MAINEX subcommand TOOLU 288
- CLOSEF module TOOLU 191

CLOSEINTLIB MAINEX subcommand TOOLU 288
 closeLibrary M2 47
 CLOSEOBLIB MAINEX subcommand TOOLU 288
 \$closeStream KSTRMU 18
 closing a file M1 191; M2 46, 47; MTUT1 107
 \$clp M1 221
 CLR M1 34, 37; MTUT1 98
 \$clrConfigurationBit M2 48
 \$clrSystemBit M2 49
 CM command MEDTU 77
 CMDFILE MAINEX subcommand TOOLU 289
 cmdFile M1 193; M2 49; MTUT1 107
 and STREAMS KSTRMU 12
 redirection MTUT2 44
 redirection from MAINEX TOOLU 289
 \$cmdFileEofExcp M1 193, 224; M2 49; MTUT1 273
 CMDLINE compiler subcommand TOOLU 18
 CMDLOG (special buffer) MEDTU 86
 cmdMatch M2 50; MTUT1 168
 CMPSTR example module MTUT1 49
 \$cms M1 222, 223
 CMSBITS CONF command TOOLU 202
 \$collect M2 54; MTUT1 297
 \$collectableChkSpc M2 54
 bit M1 209; M2 178
 \$collectableStrSpc M2 55
 bit M1 209; M2 178
 \$collectLock M2 55; MTUT1 297
 COLLECTMEMORYPERCENT MTUT2 12
 CONF command TOOLU 197
 COLORS example module MTUT1 35
 command
 front end MEDTU 17
 line M2 137, 236
 line arguments MEDTU 7
 mode MEDTU 21
 summary TOOLU 64
 command syntax, of MAINSAIL utilities TOOLU 188
 commands, MAINPM TOOLU 125
 COMMANDSTRING CONF command TOOLU 197
 comment MTUT1 34
 in compiler subcommands TOOLU 32
 in MAINEX subcommands TOOLU 301
 comments M1 8
 common data representation among machines M1 213; TOOLU 321
 \$compactableChkSpc M2 56
 bit M1 209; M2 178

- compare M2 56; MTUT1 125; MTUT2 4
- \$compareIntmods M2 57; TOOLU 229
- \$compareObjmods M2 57; TOOLU 320
- comparing data structures TOOLU 167
- comparison
 - chain MTUT1 183
 - chains M1 38
 - of date and time M2 95
 - with Zero MTUT2 7
- comparison operators (=, NEQ, <, LEQ, >, GEQ) M1 34; MTUT1 98
- compatibility of data types or classes MTUT1 154
- COMPIL MTUT1 6
 - module TOOLU 6
- compilation
 - date and time M2 59
 - date of module M2 170
 - selective MTUT2 54
- \$compile M2 57; TOOLU 49
- compiler
 - directives M1 143
 - errors M2 58
 - invoking from a program TOOLU 49
 - subcommands TOOLU 12
- compiler subcommand, DEBUG TOOLU 53
- COMPILETIME M2 2
- compiletime
 - evaluation M1 10; M2 2; MTUT2 6
 - evaluation of operators M1 32
 - libraries MTUT1 83
 - library MTUT2 50
 - pseudo-procedures M1 32
- \$compileTimeValue M2 57
- compound identifier M1 120
- \$compressed TOOLU 166
 - bit TOOLU 169, 179
- compressed
 - text form of structure TOOLU 150
 - text forms TOOLU 156, 166
- \$concat M2 60
- concatenation M1 19, 34; MTUT1 98; MTUT2 5
 - macro MTUT1 189
 - of strings M2 60, 115; MTUT1 15
- CONCHK MTUT1 4
 - module TOOLU 192
 - using to track bugs MTUT2 37
- conditional
 - commands to MAINEX TOOLU 225

- compilation M1 143, 148; MTUT1 184
- CONF
 - module M1 13; TOOLU 193
 - multiline commands MTUT2 47
- configuration
 - bit M2 236
 - parameters TOOLU 193
- CONFIGURATIONBITS CONF command TOOLU 38, 198
- CONFIRM LIB command TOOLU 266
- confirm M2 60; MTUT1 133
- CONNECT LIB command TOOLU 255
- connection KSTRMU 58
- conserver KSTRMU 45
- consistency
 - of memory M2 37; TOOLU 192
 - of module interfaces M1 113; TOOLU 288
- constant
 - definition M1 27
 - macro MTUT1 16, 177
- consumer-producer problems MTUT1 299
- CONTEXT keyword MEDTU 12
- context
 - debugger TOOLU 54
 - debugger, coroutine TOOLU 81
 - debugger, current breakpoint TOOLU 79
 - debugger, displaying TOOLU 77, 78
 - debugger, exception TOOLU 81
 - debugger, iUnit TOOLU 81
 - debugger, module TOOLU 79
 - debugger, procedure TOOLU 80
- continuation line in MAINEX subcommand TOOLU 285
- CONTINUE M1 54
- Continue Statement M1 54; MTUT1 71
- \$CONTINUEC M1 152
- continuing from a breakpoint TOOLU 73
- CONTROL key MEDTU 3
- control
 - characters MEDTU 10, 40
 - characters (on message line) MEDTU 8
 - section M1 106, 116; M2 29, 111, 174, 271; MTUT2 11
 - sequence MEDTU 74
- CONTROLINFO MAINEX subcommand TOOLU 289
- \$controlInfoBit M2 246
- controlling terminal KSTRMU 104
- conversion
 - between data types MTUT1 95
 - in general M1 14, 25

- of character to string (cvcs) M2 78
- of date and time to string M2 97
- of date to string M2 98
- of string to date M2 254
- of string to date and time M2 255
- of string to time M2 256
- of time to string M2 263
- to address (cva) M2 73
- to array (cvAry) M2 75
- to bits (cvb) M2 75
- to boolean (\$cvbo) M2 77
- to charadr (cvc) M2 77
- to integer (cvi) M2 79
- to long bits (cvlb) M2 81
- to long integer (cvli) M2 83
- to long real (cvlr) M2 85
- to lower case (cvl) M2 80
- to pointer (cvp) M2 86
- to real (cvr) M2 87
- to string (cvs) M2 88
- to upper case (cvu) M2 91
- conversions, table of allowed M1 26
- \$convertDateAndTime M1 198; M2 61
- converting
 - data image to text form TOOLU 169, 184
 - text form to data image TOOLU 184
 - text form to structure image TOOLU 176
 - to upper or lower case MEDTU 49
- cooperating
 - child process KSTRMU 9, 16, 68
 - child test KSTRMU 110
- Coordinated Universal Time M1 198; M2 26, 61, 97, 106, 255
- COPFIL example module MTUT1 109
- COPIER MTUT1 117
 - module TOOLU 203
- COPY
 - INTLIB command TOOLU 234
 - LIB command TOOLU 263
 - MODLIB command TOOLU 311
- copy MTUT1 289
 - of memory or record or array elements M2 62
- \$copyFile M2 64
- copying
 - a data structure TOOLU 168
 - files M2 64
 - text MEDTU 46
- COROUT example module MTUT1 305

coroutine M1 179; MTUT1 299
 ancestry M1 181
 as treated by errMsg M2 118
 call chain of current TOOLU 190
 creation M1 180; M2 70
 current M2 260
 exception in M1 183; MTUT1 346
 finding given a name M2 130
 killing M1 180; M2 156, 157; TOOLU 204
 most recent resumer M1 181
 moving in tree M2 173
 resuming M1 180; M2 225; TOOLU 204
 scheduled KSTRMU 75
 scheduler MTUT1 314
 service KSTRMU 25
 stack MTUT1 305
 starting M2 225
 that raised current exception M2 121
 utilities TOOLU 204
 \$coroutineExcp M1 224; M2 64
 coroutines
 and the "I" command TOOLU 77
 and the "J" command TOOLU 78
 and the "S" command TOOLU 85
 opening with the "OC" command TOOLU 81
 sharing data KSTRMU 76
 cos M2 65
 cosh M2 65
 cosine (cos) M2 65
 \$cot M2 66
 cotangent M2 66
 count break TOOLU 78
 \$countingPerModule bit M2 171
 \$countingPerProc bit M2 171
 \$countingPerStmt bit M2 171
 COUNTR example module MTUT1 130
 COUNTS command TOOLU 128
 counts MEDTU 3, 28
 in debugger commands TOOLU 64
 CP LIB command TOOLU 263
 CPU
 ID M2 66
 time M2 67
 CPU ID, in object module TOOLU 350
 \$cpuID M2 66
 \$cpuTime M2 67
 \$cpuTimeResolution M2 67

cRead M2 68; MTUT1 58, 288; MTUT2 15
 \$cReadStream KSTRMU 100
 CREATE
 INTLIB command TOOLU 235
 LIB command TOOLU 260
 MODLIB command TOOLU 311
 create M2 69
 bit KSTRMU 51; M2 192
 \$createClassDscr M2 69; MTUT1 298
 \$createCoroutine M2 70; MTUT1 299
 \$createDate field of \$fileInfoCls M2 127
 \$createRecord M2 71; MTUT1 298
 \$createRendezvousName KSTRMU 73
 \$createTime field of \$fileInfoCls M2 127
 \$createUniqueFile M2 72
 creating a text file MEDTU 5
 creation
 of coroutine M1 180; M2 70
 of record of unknown class M2 71
 cross-compilation M1 7; TOOLU 31
 cross-CONF TOOLU 200
 cross-INTLIB TOOLU 238
 cross-MODLIB TOOLU 315
 cross-reference listing TOOLU 23, 369
 cross-reference listings, merging TOOLU 36
 CRTHDR example module MTUT1 323
 CSUBCOMMANDS MAINEX subcommand TOOLU 289
 CTRL key MEDTU 3
 CTRL-C KSTRMU 9, 87, 105
 CTRL-D KSTRMU 94, 105
 CTRL-Q KSTRMU 105
 CTRL-S KSTRMU 105
 CTRL-Z KSTRMU 94, 105
 current
 coroutine M2 260
 exception M2 121, 122, 123
 file (of debugger context) TOOLU 57
 file name M2 59, 262
 line number M2 59
 module name M2 59
 page number M2 59
 procedure name M2 59
 current character, word, line, and page MEDTU 24
 current coroutine, call chain of TOOLU 190
 current exception, information about MTUT1 281
 current window, status line MEDTU 9
 \$currentDirectory KSTRMU 65; M2 73

- cursor MEDTU 10, 24
 - movement MEDTU 23, 30; TOOLU 90
 - movement with arrow keys MEDTU 133
- cursor movement, among buffers MEDTU 59
- cva M2 73; MTUT1 95
- cvAry M2 75
- cvb M2 75; MTUT1 95
- \$cvbo M2 77
- cvc M2 77; MTUT1 95, 289
- cvcs M2 78; MTUT1 58; MTUT2 18
- cvi M2 79; MTUT1 95
- cvl M2 80; MTUT1 58; MTUT2 18
- cvlb M2 81; MTUT1 95
- cvli M2 83; MTUT1 95
- cvlr M2 85; MTUT1 95
- cvp M2 86; MTUT1 95
- cvr M2 87; MTUT1 95
- cvs M2 88; MTUT1 95; MTUT2 18
 - length of resulting string M2 159
- cvu M2 91; MTUT1 58; MTUT2 18
- cWrite M2 92; MTUT1 60, 288; MTUT2 19
- \$cWriteStream KSTRMU 100

D

- command TOOLU 90
- commands MEDTU 24, 42
- D400 module MEDTU 141
- D460 module MEDTU 142
- D460C module MEDTU 142
- DARWIN example module MTUT1 327
- data
 - file M1 189; MTUT1 106
 - portable format (PDF) M1 213; TOOLU 321
 - section M1 105, 106, 110
 - section allocation M1 111; M2 174
 - section disposal M1 111; M2 111
 - section of current module M2 260
 - sections and the Structure Blaster TOOLU 154
 - sink TOOLU 303
 - structure image TOOLU 150
 - type code M1 11
 - type conversion M1 25
 - types M1 14
- data file
 - editing MEDTU 131

- viewing TOOLU 210
- data image, converting to text form TOOLU 169, 184
- data section
 - bound M1 108
 - examining TOOLU 87
 - module name for M2 172
 - size of M2 247
- data structure
 - examining or editing TOOLU 156
 - manipulating arbitrary TOOLU 150
- data-type-aligned address M1 22
- dataFile, predeclared class M1 187
- \$dataImage TOOLU 171
- DATAME module MEDTU 143
- Datamedia 3000 MEDTU 143
- dataSec unit TOOLU 162
- \$date M2 94; TOOLU 171
- date M1 198; M2 94
 - and time M2 94
 - and time addition M2 20
 - and time arithmetic MTUT2 46
 - and time difference M2 96
 - conversion from string M2 254, 255
 - conversion to string M2 97, 98
 - of file modification M2 127
 - of module compilation M2 170
 - of structure creation TOOLU 171
 - standard representation M2 94
- date and time
 - compilation M2 59
 - removing from string M2 222
- \$dateAndTime M2 94
- \$dateAndTimeCompare M2 95
- \$dateAndTimeDifference M2 96; MTUT2 46
- \$dateAndTimeToStr M2 97
- \$dateFormat M2 98
- \$dateToStr M2 98
- DATMGR back end MEDTU 4, 131
- DCL M1 154
 - of forward class M1 74
- deadlock, semaphore KSTRMU 80
- deallocation, efficient M1 206
- DEBUG
 - (module) TOOLU 53
 - compiler subcommand TOOLU 18, 53
- \$debugBit bit M2 171
- \$debugExec M2 100; TOOLU 114

- debuggable module TOOLU 53
- debugger
 - and source text confusion MTUT2 33
 - command syntax TOOLU 57
 - context TOOLU 54
 - displaying structure MTUT2 39
 - invoking from a program TOOLU 114
 - options TOOLU 83
 - variable TOOLU 74
- debugger context
 - coroutine TOOLU 81
 - current breakpoint TOOLU 79
 - displaying TOOLU 77, 78
 - exception TOOLU 81
 - iUnit TOOLU 81
 - module TOOLU 79
 - procedure TOOLU 80
- debugger example
 - display interface TOOLU 102
 - line-oriented interface TOOLU 94
- debugging
 - an RPC module KSTRMU 22
 - collection-influenced bugs MTUT2 34
- declaration M1 57
 - array M1 61
 - class M1 73
 - generic procedure M1 99
 - module M1 107
 - of debugger variables TOOLU 74
 - outer M1 105, 107
 - procedure M1 84
 - qualifiers M1 59
 - simple variable M1 59
 - variable MTUT1 17
- declarations file (for use with SOURCEFILE directive) MTUT1 209
- deep usage TOOLU 122
- \$def M1 159
- default
 - baud rate MEDTU 14
 - data section M1 108
 - display module MEDTU 13
 - file MEDTU 12
 - page and line MEDTU 12
 - window width MEDTU 15
- DEFAULTACHECK "\$DIRECTIVE" directive M1 168
- \$defaultArea M1 208; M2 2, 100
- DEFAULTPROTOCOL service protocol table entry KSTRMU 117

DEFINE M1 132; MTUT1 16
 LIB command TOOLU 256
 DEFINETIMEZONE MAINEX subcommand M1 201; TOOLU 289
 defining
 a macro MEDTU 74
 as consecutive integers M1 159
 deinitialize PDFMOD TOOLU 325
 delayed recompilation of erroneous procedure TOOLU 8
 DELETE
 INTLIB command TOOLU 235
 key MEDTU 2
 LIB command TOOLU 264
 MODLIB command TOOLU 312
 \$delete M2 103
 delete KSTRMU 105; M2 102
 bit KSTRMU 51, 77; M2 46, 192, 226; TOOLU 168, 179
 buffer MEDTU 42
 DELETEDPAGES MEDTU 42
 deleting MEDTU 24, 42
 a file M2 103
 buffer MEDTU 62
 files TOOLU 206
 pages MEDTU 42
 DELFIL module TOOLU 206
 DELTA example module MTUT1 30
 DELTA2 example module MTUT1 42
 \$deregisterException M2 103
 \$descendantKilledExcp M1 184, 224; M2 104, 156; MTUT2 49
 descendants, waiting for KSTRMU 78
 detail of MAINPM statistics TOOLU 129
 device
 module M1 185, 192
 modules TOOLU 303
 prefix M1 192
 device module, LIB TOOLU 242
 \$devModBrk M2 104
 \$devModBrkStr M2 104
 DEVOWL example module MTUT1 290
 \$dgux M1 222
 difference of date and time M2 96
 differences
 between files (utility) TOOLU 276, 368
 between times MTUT2 46
 \$dimension M1 70
 dimension M1 61
 \$disposeDataSecsInArea M1 207
 direct

- access to data section MTUT1 213
- access to interface field M1 110
- access to modules M1 107
- arguments TOOLU 59
- direction of MAINED commands MEDTU 28
- \$DIRECTIVE M1 144
- directive
 - BEGINSCAN M1 156
 - CHECK M1 164
 - CHECKING M1 164
 - DCL M1 154
 - \$DIRECTIVE M1 144
 - DONESCAN M1 156
 - DSP M1 147
 - ENCODE M1 145; TOOLU 45
 - \$LEGALNOTICE M1 148
 - MESSAGE M1 143
 - NOCHECK M1 164
 - RESTOREFROM M1 122
 - SAVEON M1 122
 - SKIPSCAN M1 156
 - SOURCEFILE M1 143
- directives, compiler M1 143
- DIRECTORY
 - INTLIB command TOOLU 235
 - LIB command TOOLU 257
 - MODLIB command TOOLU 312
- \$directory M2 105
- directory
 - current M2 73
 - files in M2 105
 - home M2 147
 - whether a file is M2 129
- directory structure, LIB TOOLU 245
- DIRMOD example module MTUT1 218
- \$disableInterrupts KSTRMU 87
- DISASM module TOOLU 33
- \$disassembleDate M2 106
- \$disassembleDateAndTime M2 106
- \$disassembleTime M2 107
- disassembly listing TOOLU 33
- discard M2 108
 - bit M2 219, 221, 222, 231
- \$discardOutput KSTRMU 109
- displace M2 108
- displacement
 - between addresses or charadrs M2 110, 158

- to a field of a class or module (DSP) M1 147
- display module MEDTU 6, 133; MTUT1 248
- display module, default MEDTU 13
- display-oriented
 - debugger interface TOOLU 56
 - interface TOOLU 102
- display-oriented debugger interface, switching to TOOLU 63
- displaying
 - arrays TOOLU 66
 - contents of address TOOLU 88
 - memory locations TOOLU 88
 - objects TOOLU 87
 - source text TOOLU 91
 - structure in debugger MTUT2 39
 - values TOOLU 86
- DISPLAYMODULE keyword MEDTU 13
- disposal
 - efficient M1 206
 - of array M1 62; M2 111
 - of data section M1 111; M2 111
 - of module M1 111; M2 111
 - of record M1 74; M2 111
- dispose M2 111; MTUT1 292; MTUT2 2
 - improper use MTUT2 34
 - of bound data sections TOOLU 209
- dispose_array KSTRMU 46
- \$disposeArea M1 207; M2 112
- \$disposeDataSecsInArea M2 112
- \$disposedDataSecExcp M1 111, 224; M2 113
- \$disposeSemaphore KSTRMU 78
- DISPSE module TOOLU 209
- distributed applications KSTRMU 4
- DIV M1 34; MTUT1 19, 98
- division by zero M2 24; MTUT1 273
- division operators (/ , DIV, MOD) M1 34; MTUT1 98
- DL command MEDTU 25
- DNGN example module MTUT1 229
- DO M1 51; MTUT1 36
- DOB M1 51; MTUT1 32
- \$DOC M1 152
- doCommandsInFile TOOLU 251
- doCommandsInString TOOLU 251
- DONE M1 54
- Done Statement M1 54; MTUT1 71
- \$DONEC M1 152
- DONESCAN M1 156
 - compiler directive MTUT1 209

\$doNotClear M2 113, 179
\$doNotIncludeTimeZone M2 113
 bit M2 99
\$doNotMatch M2 114
 bit M2 216
DONOTPROMPTFORBAUDRATE keyword MEDTU 14
DONOTPROMPTFORDISPLAYMODULE keyword MEDTU 13
\$doNotRaise M2 114
 bit M2 119
DONOTUPDATEEPARMS keyword MEDTU 15
dotted
 operator MTUT1 128
 operators M1 40, 43
DOWNT0 M1 51
DROP LIB command TOOLU 264
\$dscrPtr M2 114; MTUT1 298
DSP M1 147; M2 115
DSPBUG example module MTUT2 36
DSTCONNECT LIB command TOOLU 255
DSTENDRULE MAINEX subcommand M1 201; TOOLU 291
DSTNAME MAINEX subcommand M1 201; TOOLU 300
DSTOFFSET MAINEX subcommand M1 201; TOOLU 291
DSTSTARTRULE MAINEX subcommand M1 201; TOOLU 291
dungeon game example MTUT1 225
\$dup M2 115
duplicating text MEDTU 46
DVIEW module TOOLU 210
dynamic creation of classes M2 69
dynamically sized arrays M1 61; MTUT1 170

E

 command TOOLU 74
 commands MEDTU 87
 in margin of window MEDTU 25
EATER example module MTUT1 343
EB M1 46; MTUT1 32
EBCDIC MTUT2 40
\$ebcdic M1 226
echo
 of cmdFile and logFile M1 194
 of TTY KSTRMU 6, 105
ECHOCMDFILE MAINEX subcommand TOOLU 292
\$echoCmdFile M2 237; TOOLU 198
ECHOIFREDIRECTED MAINEX subcommand TOOLU 293
\$echoIfRedirected M2 237; TOOLU 198

ecological simulation MTUT1 322
 EDIT module MEDTU 5
 editing
 a data file MEDTU 131
 a data structure TOOLU 156
 a text file MEDTU 5
 commands (in debugger) TOOLU 90
 in batch MEDTU 148
 line KSTRMU 105
 editor MEDTU 1; MTUT1 4
 EDITORPARMSFILE keyword MEDTU 12
 EF M1 29, 46; MTUT1 32
 \$EFC M1 148; MTUT1 184
 efficiency MTUT2 1
 determining with MAINPM TOOLU 122
 RPC KSTRMU 39
 efficient
 allocation and deallocation M1 206
 allocation of records M2 180
 file I/O M2 197, 198
 I/O of structures TOOLU 151
 string comparison MTUT1 125; MTUT2 4
 EL M1 29, 46; MTUT1 32
 element of array MTUT1 158
 ELSE M1 29, 46; MTUT1 31
 ELSEC M1 148; MTUT1 184
 ELSEX module TOOLU 225
 \$elx M1 221
 ELXSI System 6400 procedure calling standard, calling (from) TOOLU 19
 ELXSI System 6400 procedures, calling (from) TOOLU 40
 \$emb M1 222, 223
 emphasis MEDTU 29
 Empty Statement M1 55; MTUT1 130
 empty area M1 208
 \$enableInterrupt KSTRMU 89
 \$enableInterrupts KSTRMU 9, 87
 ENCODE M1 145; TOOLU 45
 END M1 45, 48
 breakpoint on TOOLU 70, 71
 end
 of buffer MEDTU 25
 of file (eof) M2 116
 end-of-file M1 191; MTUT1 107; MTUT2 41
 on stream KSTRMU 94
 on TTY KSTRMU 105; M1 192
 end-of-line M1 10; MTUT1 14
 and stream KSTRMU 99

and TTY KSTRMU 106
ENDC M1 148, 149, 152; MTUT1 184
ENDX module TOOLU 225
ENTER
 key MEDTU 2
 MAINEX subcommand MTUT2 45; TOOLU 293
enter command mode MEDTU 2, 22
enterLogicalName M2 115; MTUT1 111; MTUT2 45
eof M1 191; M2 116; MTUT2 41
\$eofIndicator KSTRMU 109
eol M1 7; M2 116; MTUT1 14
 and TTY KSTRMU 106
eop M1 7; M2 117; MTUT1 15
\$eos KSTRMU 93, 103
 on socket KSTRMU 56
 on TTY KSTRMU 105
eparms
 file MEDTU 5, 12, 20
 file for MAINEDIT MTUT1 114
eparms file, changing MEDTU 16
equ M2 117; MTUT1 125; MTUT2 4
equate, macro M1 132, 135
\$erase KSTRMU 109
ERREXC example module MTUT1 282
errMsg M2 118; MTUT1 133
 and exceptions MTUT1 281
 and scheduling KSTRMU 76
 registered exceptions M2 103, 215
 response abbreviations M1 176
Error response: M2 118; MTUT1 8
\$error KSTRMU 63, 93, 103
error
 definition M1 3
 STREAMS KSTRMU 16
errorOK M2 120
 bit KSTRMU 19, 25, 51, 73, 79, 102, 108; M2 20, 25, 26, 27, 31, 46, 52, 62, 69, 71, 73, 94,
 95, 96, 99, 103, 105, 106, 107, 129, 147, 156, 170, 173, 179, 181, 192, 193, 197, 204,
 224, 226, 235, 242, 254, 255, 256, 261, 264, 266, 271, 274; TOOLU 168, 169, 170,
 171, 173, 175, 176, 179
errors MTUT1 8
 compiler M2 58
ESCAPE key MEDTU 2
escape mode MEDTU 21, 22, 87; TOOLU 57
evaluation
 completetime MTUT2 6
 of procedure arguments MTUT1 49
EWY100 module MEDTU 144

- examining
 - a data structure TOOLU 156
 - arrays TOOLU 66
 - contents of address TOOLU 88
 - memory locations TOOLU 88
 - objects TOOLU 87
 - structure in debugger MTUT2 39
 - values TOOLU 86
- exception M1 170
 - cannot fall out M2 206
 - cannot return M2 206
 - current M1 173
 - current, information about MTUT1 281
 - during another exception M1 173
 - efficient recognition of MTUT2 8
 - for operation M1 41
 - in coroutine M1 183; MTUT1 346
 - information about current M2 121, 122, 123
 - must propagate M2 206
 - naming M1 174; M2 178
 - nested MTUT1 273
 - predefined M1 224
 - raised by errMsg MTUT1 281
 - raising M2 205; MTUT1 269
 - registering M1 178
 - return if no handler M2 206
 - returning from M1 172; M2 207
 - stack MTUT1 276
 - tracking infinite loop with MTUT2 38
- \$exceptionBits M1 173; M2 121; MTUT1 281
- \$exceptionCoroutine M2 121
- \$exceptionName M1 173; M2 122; MTUT1 281
- \$exceptionPointerArg M1 173; M2 122; MTUT1 281; MTUT2 8
- exceptions MTUT1 266; TOOLU 63, 81
 - and the FLI TOOLU 47
 - automatic and predefined MTUT1 273
 - implementation MTUT2 23
- \$exceptionStringArg1 M1 173; M2 122; MTUT1 281
- \$exceptionStringArg2 M1 173; M2 123; MTUT1 281
- \$excludeSeconds M2 123
 - bit M2 264
- exclusive or (XOR) M1 34, 37; MTUT1 98
- EXCPT1 sample module MTUT1 270
- EXCPT2 sample module MTUT1 271
- \$executableBootName KSTRMU 67
- EXECUTE
 - command TOOLU 127

- LIB command TOOLU 268
- \$executeIntlibCommands M2 123; TOOLU 241
- \$executeModlibCommands M2 123; TOOLU 318
- \$executeStampCommands M2 124; TOOLU 361
- executing
 - modules from MAINED MEDTU 86
 - statements TOOLU 88
- execution
 - access TOOLU 307
 - of specified module TOOLU 74
- \$exeFileFirst M2 237
- exeList M1 129; MTUT1 246
- exeSearch M1 128
- exhausting memory M2 22
- EXIT LIB command TOOLU 268
- exit M2 124
- exiting TOOLU 63
 - CONF TOOLU 201
 - MAINED MEDTU 21
 - MAINEDIT MEDTU 11, 67
- exp M2 124
- expiration date, of object module TOOLU 350
- explicit
 - class M1 78
 - class specification MTUT1 223
 - module pointers M1 110
- exponent M1 16; M2 125
 - bit M2 89
- \$exponentExcpt M1 224; M2 125
- exponential (exp) M2 124
- exponentiation M1 34; MTUT1 98
- Expression Statement M1 43; MTUT1 127
- expression, displaying value of TOOLU 86
- expressions M1 27
- EXPUNGE LIB command TOOLU 264
- extension of generic procedures M1 103
- EXTRACT
 - INTLIB command TOOLU 237
 - LIB command TOOLU 263
 - MODLIB command TOOLU 312

F

- command TOOLU 90
- commands MEDTU 67

falling

- out of a handler M1 172; MTUT1 269, 272
- out of a typed procedure MTUT1 273
- FALSE M1 15; MTUT1 30
- fast I/O of structures TOOLU 151
- fastExit M2 125
- fatal M2 126
 - bit M2 119, 121
- FBORRO module MEDTU 135
- FCC TOOLU 38, 199
 - example TOOLU 40
- featherweight process KSTRMU 5
- FFRAME module MEDTU 135
- field MTUT1 144
 - class M1 73
 - information procedure (\$fieldInfo) M2 126
 - interface M1 105, 107, 108, 110
 - read from PDF source TOOLU 325
 - record M1 71
 - variable M1 71, 76, 78, 79, 108, 110
- field base
 - most recently used TOOLU 66
 - next to most recently used TOOLU 66
- field variables, generic procedures M1 115
- \$fieldInfo M2 126
- fields of records and data sections, examining TOOLU 87
- file M1 185
 - access M1 189
 - association with buffer MEDTU 3
 - base (LIB) TOOLU 244
 - cache M1 195; MTUT2 12; TOOLU 213
 - closing M1 191; M2 46, 47
 - closing utility TOOLU 191
 - comparison utility TOOLU 276, 368
 - containing multiple modules M1 116
 - data M1 189
 - data type size M2 152
 - deleting TOOLU 206
 - editing MEDTU 5
 - formats TOOLU 203
 - host TOOLU 244
 - I/O MTUT1 106
 - information M2 127
 - input and output M1 189
 - inserting MEDTU 39
 - library (LIB) TOOLU 244
 - maintaining in memory TOOLU 303
 - merging TOOLU 334

- name M1 185; MEDTU 2; MTUT1 111
- name syntax MTUT2 41
- opening M1 190; M2 190
- organization M1 189
- position MTUT1 131
- predeclared class M1 187
- random access MTUT2 42
- remote KSTRMU 50
- renaming TOOLU 339
- RPC KSTRMU 50
- searchpath M2 244
- selection MEDTU 59
- simultaneous open MTUT2 43
- system MTUT1 106
- text M1 188
- version TOOLU 246
- whether is directory M2 129
- file handles, limit MTUT2 43
- file name
 - case MTUT2 42
 - case sensitivity M1 225
 - changing MEDTU 62
 - current M2 59, 262
 - intmod and objmod M1 127
 - logical MTUT2 45
 - unique M2 72
- file positioning (getPos, relPos, setPos) M2 141, 218, 242
- \$fileAttr field of \$fileInfoCls M2 129
- FILEINFO MAINEX subcommand TOOLU 294
- \$fileInfo M2 127
- \$fileInfoBit M2 246
- \$fileNamesAreCaseSensitive M1 225
- files
 - multiple MEDTU 59
 - multiple opens MEDTU 93
- \$fillBuffer bit KSTRMU 96
- filling
 - paragraphs M2 135
 - text MEDTU 50
- FILMRG module TOOLU 216
- filter, character stream MTUT1 300
- FINAL M1 114
- final
 - procedure M1 114
 - procedure caveat MTUT2 48
- \$findArea M1 207; M2 129
- \$findCoroutine M2 130

- first M2 130; MTUT1 60; MTUT2 15
- fixed M2 131
 - bit M2 89
- \$fixFormat M2 129
- flag MTUT1 91
- fldRead M2 131; MTUT2 18
- fldWrite M2 132
- FLDXREF compiler subcommand TOOLU 19
- FLI M1 145; TOOLU 38
 - and exceptions TOOLU 47
 - and garbage collection TOOLU 40, 47
 - compiler subcommand TOOLU 19
 - parameters and data types TOOLU 47
 - target M2 58
- FLI example
 - FCC TOOLU 40
 - MEC TOOLU 44
- floating point number M1 16
- floor M2 133
- \$flow KSTRMU 107
- flow control KSTRMU 105
- \$flushStream KSTRMU 97, 101
- FOR
 - in Iterative Statement M1 51
 - in substring specification M1 28
- FOR-clause MTUT1 27
- \$FORC M1 152
- Foreign Language Interface M1 145; TOOLU 19, 38, 199
- foreign
 - module MTUT1 247
 - strings MTUT1 297
- \$foreignCodeStartsExecution TOOLU 38, 198
- FOREIGNMODULES CONF command TOOLU 38, 199
- format
 - of a file M2 129
 - of date M2 98
 - of structure TOOLU 171
 - of time M2 262
- formatted M2 134
 - bit M2 90, 219
- formatting paragraphs M2 135
- \$formParagraph M2 135
- FORTRAN, calling (from) TOOLU 19, 40
- FORTRAN77, calling (from) TOOLU 19, 40
- FORWARD M1 96
- forward
 - class M1 74

procedures MTUT1 76
FRAME module MEDTU 135
frame pointer MTUT2 24
free list MTUT2 2
front end MEDTU 4
front end, specifying MEDTU 64
FRONTEND keyword MEDTU 4, 16
full platform name M2 199
full-duplex TTY KSTRMU 6, 104, 111
\$fullPathName field of \$fileInfoCls M2 127
\$fullPathNames M2 136
bit M2 105
fully qualified LIB file name TOOLU 245
FVIEW example module M1 217

G

command MEDTU 25; TOOLU 90
commands MEDTU 30
gaps in memory TOOLU 306
garbage
collection M1 12, 20, 22, 23, 41, 65, 74, 206; M2 2, 37, 250; MTUT1 144, 292, 297;
MTUT2 1, 2, 12, 13, 46; TOOLU 197, 304
collection (of data sections) M2 30
collection (tracking) TOOLU 296
collection and \$noCollectablePtrs M2 184
collection and \$noCollectableSirs M2 185
collection and \$noCompactablePtrs M2 185
collection and \$storageUnit I/O MTUT1 296
collection and FLI TOOLU 40
collection and the FLI TOOLU 47
collector errors MTUT2 34
garbage collection
controlling M1 13
inducing M2 54
inhibiting M2 55
interception M2 18, 221
statistics TOOLU 306
string MTUT2 21
gateway, stream KSTRMU 128
GCCHP module TOOLU 213
GENCODE compiler subcommand TOOLU 21
generateMultipleQuickSort M2 137; TOOLU 346
generateQuickSort M2 137; TOOLU 345
GENERIC M1 99
generic

- procedure extension M1 103
- procedures MTUT1 85
- procedures as field variables M1 115
- GENINLINES compiler subcommand TOOLU 21
- gensrv installation KSTRMU 115
- GEQ M1 34; MTUT1 98
 - (greater or equal) MTUT1 31
 - for strings M1 33
- \$getBaudRate KSTRMU 108
- \$getCommandLine M2 137
- \$getEofPos M2 140
- \$getHosts KSTRMU 60
- \$getInArea M1 20, 207; M2 140; MTUT1 297
- getPos M2 141; MTUT1 132
- \$getProtocols KSTRMU 60
- \$getSubcommands M2 141; TOOLU 302
- \$getToTop M2 143
- global
 - cache (of files) TOOLU 213
 - symbol table M2 143
- \$globalEnter M2 143
- globalLeftBorder MEDTU 84
- \$globalLookup M2 143
- \$globallyCached bit M2 205, 238
- \$GLOBALREDEFINE M1 123, 146
- \$globalRemove M2 143
- globalRightBorder MEDTU 84
- \$globalSymbol M2 143
- GMT M1 198; M2 26, 61, 97, 106, 255
- \$gmt M2 144
 - bit M2 25, 26, 27, 94, 95, 97, 107, 129, 254, 255, 256, 261
- GMTOFFSET MAINEX subcommand M1 201; TOOLU 294
- \$GMTtoLocalTime M2 145
 - bit M2 26, 255
- \$gotValue M1 191; M2 145; MTUT1 107; MTUT2 41
- Greenwich Mean Time M1 198; M2 26, 61, 97, 106, 255
- GPL example module MTUT1 236
- guaranteed range MTUT1 19

H

- command TOOLU 77
- commands MEDTU 70
- half-duplex TTY KSTRMU 6, 17, 104, 111
- \$halfDuplex M1 225
- \$HANDLE M1 171; MTUT1 268

Handle Statement M1 171; MTUT1 268
handle limit on files MTUT2 43
Handle Statement, code for MTUT2 23
Handle Statements, nested MTUT1 276
\$HANDLEB M1 171
handler
 falling out of M1 172; MTUT1 272
 nested MTUT1 273
 search for MTUT2 26
hard deletion of LIB file TOOLU 247
hardcopy terminals and MAINEDIT MEDTU 148
HARDDELETE LIB command TOOLU 265
HASBODY, \$compileTimeValue argument M2 59
\$hasFileVersions M1 225
\$hasFinalProc bit M2 171
\$hash M2 146
hash
 code MTUT1 133
 table MTUT1 163, 213
 table utility TOOLU 219
hashEnter M2 147; TOOLU 220, 221
hashInit M2 147; TOOLU 220, 221
hashLoad M2 147; TOOLU 220, 223
hashLookup M2 147; TOOLU 220, 221
hashLookupNext M2 147; TOOLU 220, 223
hashLookupNextInit M2 147; TOOLU 220, 223
hashNext M2 147; TOOLU 220, 222
hashRemove M2 147; TOOLU 220, 222
hashRemoveRecord M2 147; TOOLU 220, 223
hashStore M2 147; TOOLU 220, 223
\$hasInitialProc bit M2 171
HEADER LIB command TOOLU 269
header, procedure M1 84; MTUT1 41
HEATH module MEDTU 145
Heath H-19 MEDTU 145
help for debugger commands TOOLU 64
hex M2 146
 bit M2 76, 82, 90, 200, 219
hexadecimal M1 18
 object values TOOLU 77
 values TOOLU 77
\$hisHostName KSTRMU 57
\$homeDirectory M2 147
Hoops! MEDTU 70
host machine TOOLU 4
HOSTNAME service protocol table entry KSTRMU 116
\$hostName KSTRMU 63

HOSTPROTOCOL service protocol table entry KSTRMU 116
\$hp20 M1 222
HP300H module MEDTU 147
\$hp38 M1 222
HPTERM module MEDTU 147
\$hpux M1 222
HSHMOD MTUT1 213
 module M2 147; TOOLU 219
HSHMOD module, source text MTUT1 215
hyperbolic
 cosine (cosh) M2 65
 sine (sinh) M2 247
 tangent (tanh) M2 260
\$hyphenateDate M2 147
 bit M2 99

I

 command TOOLU 77
 commands MEDTU 38
I/O MTUT1 106
 clearing KSTRMU 8
 low-level stream KSTRMU 92
 scheduled KSTRMU 75
\$i38 M1 221
\$ibm M1 221
identifier M1 233; MTUT1 9
 definition M1 9
 qualified M1 120
 reserved M1 9
 scope M1 58
 search MEDTU 34
 visibility M1 121
IF M1 29, 46
If
 Expression M1 29; MTUT1 121
 Statement M1 46; MTUT1 31
IFC M1 148; MTUT1 184
IFFY example module MTUT1 32
IFX module TOOLU 225
ignore-count breakpoint TOOLU 69, 73
\$ignoreMe bit M2 280
illegal, definition M1 3
iLoad M2 161
\$image bit KSTRMU 96, 99
\$imageType TOOLU 171, 174

- implicit module pointer M1 106, 109
- in-place sort TOOLU 341
- inaccessible data structure M1 12
- \$inArea M1 207; M2 148
- \$includeTimeZone M2 148
 - bit M2 99
- \$includeWeekday M2 149
 - bit M2 99
- inclusive or (!, IOR) M1 34, 37; MTUT1 98
- INCREMENTAL compiler subcommand TOOLU 21
- incremental recompilation TOOLU 10
- index
 - array M1 65
 - sort TOOLU 342
- indirect
 - access to data section MTUT1 196
 - access to interface field M1 108
 - access to modules M1 107
 - arguments TOOLU 60
- INF M1 29; MTUT1 124
- infinite loop, tracking MTUT2 38
- infix order MTUT1 154
- INFO command TOOLU 130
- information MEDTU 63
 - about memory usage TOOLU 304
- inherited fields M1 78
- ININTLIB compiler subcommand TOOLU 22
- INIT M1 62
- Init Statement M1 62; MTUT1 167
- INITIAL M1 113
- initial procedure M1 113; MTUT1 10
- initialization
 - of areas M2 177
 - of arrays M1 62, 66
 - of local variables M1 85
- INITIALIZE (special macro) MEDTU 77
- initialize PDFMOD TOOLU 326
- \$initializeStreams KSTRMU 15
- INITIALSTATICPOOLSIZE MTUT2 12
 - CONF command TOOLU 200
- \$initRand M2 149; TOOLU 336
- \$initsRand M2 149; TOOLU 336
- INLINE M1 98; MTUT2 6
- inline procedures MTUT2 6
- \$inlinesHaveBodies bit M2 171
- INOBJFILE compiler subcommand TOOLU 22
- INOBJLIB compiler subcommand TOOLU 22

- input M2 149; MTUT1 106
 - bit KSTRMU 19, 51, 101; M2 192
 - from a file M1 189
- insert mode MEDTU 21, 22, 38
- inserting
 - blank lines MEDTU 38
 - characters (I commands) MEDTU 38
 - files and buffers (I commands) MEDTU 39
- \$insertLeft M2 150, 173
- insertModeDefault MEDTU 85
- \$insertRight M2 150, 173
- installation, server KSTRMU 115
- instance procedures MTUT1 85
- INTCOM module TOOLU 228
- INTEGER M1 15
- integer MTUT1 19
 - maximum M2 167
 - square root example MTUT1 37
 - string parse M2 220
- integerCode M1 220
- interactive
 - child KSTRMU 70
 - macro equate M1 135; MTUT1 185
- interface
 - consistency checking M1 113; TOOLU 288
 - field M1 105, 107; MTUT1 196
 - field access M1 108, 110
 - procedure access M1 111
 - RPC module KSTRMU 26
 - variable M1 108, 110
 - variable access M1 111
- intermodule call MTUT2 8
- interpreting statements TOOLU 88
- interprocess
 - communication KSTRMU 4
 - communication stream KSTRMU 11
 - procedure calls KSTRMU 21
- \$interrupt KSTRMU 109
- interrupt KSTRMU 9, 87
 - character KSTRMU 105
- interrupt character, reading KSTRMU 7
- \$interruptsEnabled KSTRMU 87
- \$intFileFirst M2 237
- INTLIB
 - module TOOLU 230
 - program interface TOOLU 241
- intList M1 129

intmod M1 119, 127; MTUT1 350; TOOLU 53
close TOOLU 80
comparison TOOLU 228
converting source library to MTUT2 51
file name M1 127
information about M2 151
libraries and debugger TOOLU 82
library M1 127; TOOLU 230
search rules M1 128
STREAMS KSTRMU 15
supporting M1 120
intmod library, opening with the "OI" command TOOLU 82
\$intmodInfo M2 151
intSearch M1 128
invalid
address M1 22; MTUT1 289; MTUT2 45
exponent M2 125
\$invokeModule M1 117; M2 151; TOOLU 279
invoking
a module M2 151; TOOLU 74, 279
module with arguments M2 137
modules from MAINED MEDTU 86
IOR M1 34, 37; MTUT1 98
\$ioSize M1 216; M2 152; MTUT1 142
\$ip32c M1 222
IPC stream KSTRMU 11
\$ipsc2 M1 222
isAlpha M2 152; MTUT1 61
\$isArray M2 153
\$isaTty KSTRMU 16, 110
\$isBound M2 153
\$isDirectory M2 129
\$isHalfDuplex KSTRMU 16
\$isLocked KSTRMU 78
isLowerCase M2 154; MTUT1 61
isNul M2 154
\$isScheduled KSTRMU 8, 16
isUpperCase M2 155; MTUT1 61
Iterative Statement M1 51; MTUT1 26, 36
iterative variable MTUT1 27
Iterative Statement, breakpoint on TOOLU 70
ITF1 example module MTUT1 199
ITF2 example module MTUT1 200
ITFXREF compiler subcommand TOOLU 23
iUnit TOOLU 54, 81
breakpoint at TOOLU 73
\$ix20 M1 222

\$ixfpa M1 222
\$ixpri M1 222

J

command TOOLU 78
commands MEDTU 48
joining lines MEDTU 48
jumping into procedures TOOLU 78
justifying text MEDTU 50

K

command MEDTU 24; TOOLU 78
commands MEDTU 42
KEEP LIB command TOOLU 266
keepNul M2 155
bit KSTRMU 51; M2 64, 132, 192; TOOLU 173, 176
Kermit terminal emulator KSTRMU 80
KERMODNAME CONF command TOOLU 200
kernel, MAINSAIL module TOOLU 200
KEY example module MTUT1 236
keyboard
interrupt KSTRMU 9, 87
macros MEDTU 74
\$keyboardInterruptExcp KSTRMU 9, 87
keys, special MEDTU 2
keyword M1 9, 123, 233; MTUT1 9
abbreviation MTUT1 32
kill command on UNIX MTUT2 38
KILLCO module TOOLU 204
\$killCoroutine M2 156; MTUT1 299
\$skilledCoroutine M2 157
killing
a coroutine M1 180; M2 156, 157
text MEDTU 42
\$killServerExcp KSTRMU 37

L

command TOOLU 91
commands MEDTU 49
LAMP example module MTUT1 236
last M2 157; MTUT1 60; MTUT2 15
\$lastError KSTRMU 63
\$lastInputError KSTRMU 16, 94
\$lastOutputError KSTRMU 16, 94
\$lbl M1 70

lb1 M1 70
 \$lb2 M1 70
 lb2 M1 70
 \$lb3 M1 70
 lb3 M1 70
 lbLoad M2 161
 lbMask M2 158; MTUT2 44
 lDisplacement M2 158
 leaving
 MAINED MEDTU 21
 MAINEDIT MEDTU 11
 left-right scrolling MEDTU 55
 legal notice directive (\$LEGALNOTICE) M1 148
 \$LEGALNOTICE M1 148
 LEGALNOTICE MODLIB command TOOLU 314
 \$length M2 159
 length M2 159; MTUT1 60
 LEQ M1 34; MTUT1 98
 (less or equal) MTUT1 31
 for strings M1 33
 LIB TOOLU 242
 LIBEX TOOLU 242
 example TOOLU 270
 LIBRARY compiler subcommand TOOLU 23
 library
 compiletime MTUT2 50
 file (LIB) TOOLU 244
 intmod MTUT1 350; TOOLU 230
 module M1 127; M2 47, 193
 module, system TOOLU 202
 objmod MTUT1 246; TOOLU 307
 objmod (opening from MAINEX) TOOLU 297
 opening with the "OL" command TOOLU 82
 liLoad M2 161
 limit on file handles MTUT2 43
 LINCOM module TOOLU 276
 LINDPY module MEDTU 148
 \$line bit KSTRMU 96, 99, 100
 line
 debugger interface TOOLU 90
 editing KSTRMU 105
 search MEDTU 34
 line number, current M2 59
 line-oriented debugger interface TOOLU 55, 94
 line-oriented debugger interface, switching to TOOLU 63
 \$lineErase KSTRMU 109
 LINEFEED key MEDTU 2

linefeed KSTRMU 99, 106
 \$linefeedKey KSTRMU 109
 \$lineOrientedDebug M2 237
 link step, absence of M1 105
 LIST
 command TOOLU 130
 example module MTUT1 150
 In M2 161
 load MTUT1 286
 a character from a charadr (cLoad) M2 45
 from an address M2 161
 local
 call MTUT2 8
 declarations M1 58
 variable M1 58, 84; MTUT1 45
 local variable, initialization M1 85
 localLeftBorder MEDTU 84
 localRightBorder MEDTU 84
 \$localTime M2 164
 bit M2 25, 26, 27, 94, 95, 97, 107, 254, 255, 256, 261
 \$localTimeToGMT M2 164
 bit M2 26, 255
 \$lock KSTRMU 78
 locking data structures KSTRMU 76, 78
 LOG
 compiler subcommand TOOLU 24
 INTLIB command TOOLU 237
 MODLIB command TOOLU 314
 log M2 165
 log file, server KSTRMU 37
 \$log2 M2 165
 logarithm
 base 2 M2 165
 base e (ln) M2 161
 base ten (log) M2 165
 LOGFILE MAINEX subcommand TOOLU 294
 logFile M1 193; M2 166; MTUT1 107
 and STREAMS KSTRMU 12
 redirecting from MAINEX TOOLU 294
 redirection MTUT2 44
 LOGIC example module MTUT1 33
 LOGIC2 example module MTUT1 43
 logical
 file name MTUT2 42, 45
 file names M1 190; M2 115, 166
 name MTUT1 111
 logical names

establishing from MAINEX TOOLU 293
looking up from MAINEX TOOLU 294
logical operators (AND, OR, NOT) M1 34; MTUT1 98
LONG

BITS M1 17
INTEGER M1 15
REAL M1 16

long

array MTUT1 172
arrays M1 61, 65
bits MTUT1 91
integer MTUT1 89
MAINEX subcommand line TOOLU 285
real MTUT1 90

long bits, guaranteed range MTUT2 45

long integer, maximum M2 168

longBitsCode M1 220

longIntegerCode M1 220

longRealCode M1 220

LOOKUP MAINEX subcommand TOOLU 294

lookUpLogicalName M2 166

lookupLogicalName MTUT1 111

LOOPS example module MTUT1 38

low-level

data types MTUT1 284; MTUT2 1
PDF procedures M2 198; TOOLU 321
stream I/O KSTRMU 92

lower case, converting to MEDTU 49

lparms TOOLU 247

lrLoad M2 161

LS LIB command TOOLU 257

M

command (with argument) TOOLU 79
command (with no argument) TOOLU 79
commands MEDTU 47

\$m20 M1 221

\$m68 M1 221

MACRO keyword MEDTU 14

macro MEDTU 74; MTUT1 16; TOOLU 64, 65

and modes MEDTU 77

argument M1 137

body M1 132, 134

bracketed text MTUT1 178; MTUT2 47

call M1 137

- common errors MTUT1 193
- concatenation MTUT1 189
- constant M1 132; MTUT1 16, 177
- declaration M1 132
- definition M1 132, 133
- equate M1 132
- ID keys MEDTU 133
- parameter M1 132, 137
- recursive MTUT1 187
- special MEDTU 77
- stored in eparms MEDTU 14
- macro equate, interactive M1 135
- macro parameter, repeatable MTUT1 190
- MAINDEBUG, invoking from a program TOOLU 114
- MAINED MEDTU 1
 - front end MEDTU 4
- MAINEDIT MEDTU 1; MTUT1 4; TOOLU 56
 - batch mode MEDTU 148
 - eparms file MTUT1 114
 - hardcopy terminals MEDTU 148
- MAINEX M2 151; MTUT1 4; TOOLU 279
 - conditional commands to TOOLU 225
 - subcommands M2 141; TOOLU 281, 302, 363
- MAINEX subcommands, reading from a file TOOLU 289, 300
- MAINKERMIT terminal emulator KSTRMU 80
- MAINPM MTUT2 1; TOOLU 24, 25, 27, 28
 - module TOOLU 123
- MAINSAIL
 - address space MTUT1 292
 - monitor TOOLU 304
- \$mainsailExec M2 166; TOOLU 301
- MAINVI MEDTU 1
 - front end MEDTU 4
- \$majorVersion M2 167
- MAKE LIB command TOOLU 262
- MAKEMODULENOTVISIBLE "\$DIRECTIVE" directive M1 120
- MAKEMODULEVISIBLE "\$DIRECTIVE" directive M1 120
- MAKENOTVISIBLE "\$DIRECTIVE" directive M1 120
- MAKEVISIBLE "\$DIRECTIVE" directive M1 120
- MAP MAINEX subcommand TOOLU 295
- map of scheduled coroutines KSTRMU 80
- \$mapAtMemInfoBit M2 246
- margins MEDTU 25
- marked location MEDTU 33
- \$markedArea TOOLU 180
- MATMUL example module MTUT1 173
- matrix multiplication MTUT1 172

MAX M1 34; MTUT1 98
 \$maxChar M1 6; M2 167; MTUT1 289
 maximum
 character code M2 167
 integer M2 167
 long integer M2 168
 \$maxInteger M2 167
 \$maxLongInteger M2 168
 MAXMEMORYSIZE MTUT2 12
 CONF command TOOLU 200
 maze program exercise MTUT1 175
 MEC TOOLU 38
 example TOOLU 44
 MEDT MEDTU 1
 front end MEDTU 4
 MEM device module TOOLU 303
 MEMINFO MAINEX subcommand TOOLU 296
 \$memInfoBit M2 246
 MEMMAP MTUT1 292
 \$memMngModule M2 18, 221
 memory
 allocation quanta TOOLU 200
 examining TOOLU 88
 exhausting M2 22
 files TOOLU 303
 information about TOOLU 304
 limit on TOOLU 200
 management M1 12, 206; MTUT1 144, 284; MTUT2 12; TOOLU 304
 maps TOOLU 295
 stream KSTRMU 11, 114
 unit M1 11
 memory management, controlling M1 13
 MEMSTR STREAMS module KSTRMU 79, 114
 merging files TOOLU 334
 MESSAGE M1 143
 compiler directive MTUT1 185
 message line MEDTU 8
 MIN M1 34; MTUT1 98
 \$minInteger M2 168
 \$minLongInteger M2 169
 \$minorVersion M2 169
 MINSIZETOALLOCATE CONF command TOOLU 200
 MKDIR LIB command TOOLU 262
 MM module TOOLU 304
 MOD M1 34; MTUT1 98
 mode MEDTU 21
 and macros MEDTU 77

- as displayed in status line MEDTU 9
- parameter M1 101
- MODIFIES M1 90; MTUT1 44
- \$modifyDate field of \$fileInfoCls M2 127
- \$modifyTime field of \$fileInfoCls M2 127
- MODLIB
 - module TOOLU 307
 - program interface TOOLU 318
- modList TOOLU 232, 309
- MODTIME compiler subcommand TOOLU 24, 123
- MODULE M1 105
 - command TOOLU 129
- module MTUT1 196, 213
 - allocation M1 111; M2 174
 - can find M2 33
 - date of compilation M2 170
 - declaration M1 107
 - disposal M1 111; M2 111
 - file name association M2 217
 - format M1 105
 - invoking M2 151
 - invoking from MAINED MEDTU 86
 - invoking from MAINEX TOOLU 279
 - libraries and debugger TOOLU 82
 - library M1 127; M2 193; MTUT1 246
 - library (closing) M2 47
 - library (opening from MAINEX) TOOLU 297
 - linkage M1 111
 - name association M2 218, 239
 - name of M2 172
 - names M1 105
 - of debugger context TOOLU 79
 - search rules M1 128; MTUT1 246
 - size M1 105; M2 170
 - source vs. object MTUT1 196
 - swapping M1 116; MTUT1 208; MTUT2 11
 - version M2 170
 - visibility M1 120
- module library
 - intrmod TOOLU 230
 - objmod TOOLU 307
 - system TOOLU 202
- module name, current M2 59
- module name association, establishing from MAINEX TOOLU 299, 300
- module pointer, implicit M1 106, 109
- module swapping, tracking TOOLU 301
- \$moduleCode M1 154, 220

\$moduleInfo M2 170
\$moduleName M2 172
MONITOR compiler subcommand TOOLU 25, 123
MONITORAREA command TOOLU 132
monitoring, performance M2 58
MONITORLIB command TOOLU 132
MONITORMODULE command TOOLU 132
most recently used field base TOOLU 66
MOVE
 INTLIB command TOOLU 238
 MODLIB command TOOLU 314
\$moveCoroutine M2 173
moving
 text MEDTU 47
 the cursor MEDTU 23
MS_ADDRESSCODE KSTRMU 43
MS_BITSCODE KSTRMU 43
MS_BOOLEANCODE KSTRMU 43
MS_CHARADRCODE KSTRMU 43
ms_exception KSTRMU 46
MS_INTEGERCODE KSTRMU 43
MS_LONGBITSCODE KSTRMU 43
MS_LONGINTEGERCODE KSTRMU 43
MS_LONGREALCODE KSTRMU 43
MS_REALCODE KSTRMU 43
MS_STRINGCODE KSTRMU 43
msgMe M2 118, 173
 bit M2 119
msgMyCaller M2 118, 174
 bit M2 119
MSK M1 34, 37; MTUT1 98
\$msTimeout KSTRMU 77
MULEXC example module MTUT1 277
multidimensional arrays MTUT1 172
multiple
 buffers into one file MEDTU 64
 file open MTUT2 43
 files MEDTU 59, 93
 modules in one source file M1 116; MTUT2 54
 opens of the same file M1 190
 sort TOOLU 346
 windows MEDTU 55
multiple processes, simulating MTUT1 299
multiple records, allocation of M2 180
multiplication operator (*) M1 34; MTUT1 98
multiprocessor program KSTRMU 81
multitasking KSTRMU 75

mutual recursion MTUT1 76
MV LIB command TOOLU 264
\$mv M1 221
\$mvux M1 222
MYHOST service protocol table entry KSTRMU 115
\$myHostName KSTRMU 57, 60

N

command TOOLU 80
commands MEDTU 72
n in command descriptions MEDTU 3, 28
\$name KSTRMU 16, 78
name M1 70
 of array M1 69
 of class M2 39
 of file M1 187
named
 Begin Statement M1 45
 Case Statement M1 49
 Iterative Statement M1 53
 macro MEDTU 74
 statement MTUT1 29
NAMEDMACRO keyword MEDTU 14
names, module M1 105
natural logarithm (ln) M2 161
NCLIB mode TOOLU 254
NEEDANYBODIES M1 158
NEEDBODY M1 158
 vs. HASBODY M2 59
NEQ M1 34, 67; MTUT1 98
 (not equal) MTUT1 31
 for strings M1 33
nested
 exception MTUT1 273
 exceptions M1 173
 Handle Statements MTUT1 276
 handler MTUT1 273
NET device module KSTRMU 130, 133
NETSTR KSTRMU 65, 128
network
 distributing application across KSTRMU 81
 protocol module KSTRMU 58
new M1 207; M2 174; MTUT1 149, 159
 array M1 62; M2 174
 data section M2 174

module M2 174
record M1 74; M2 174
\$newArea M1 207; M2 177
newestVersion KSTRMU 36
\$newException M2 178
newPage M2 179
\$newRecords M2 180; MTUT2 2
\$newRemoteModule KSTRMU 24
newScratch M2 181; MTUT1 288
\$newScratchChars M2 182; MTUT1 289
\$newSemaphore KSTRMU 78
newString M1 20; M2 182; MTUT1 297
newUpperBound M2 183; MTUT1 172
next to most recently used field base TOOLU 66
nextAlpha M2 186
NIGHTS example module MTUT1 185
NIHIL example module MTUT1 187
NOACHECK
 "\$DIRECTIVE" directive M1 168
 compiler subcommand TOOLU 17
NOACHECKALL
 "\$DIRECTIVE" directive M1 168
 compiler subcommand TOOLU 17
\$noAutoCmdFileSwitching M1 193; M2 237; TOOLU 198
NOCHECK M1 164
 "\$DIRECTIVE" directive M1 165
 compiler subcommand TOOLU 18
NOCHECKALL
 "\$DIRECTIVE" directive M1 165
 compiler subcommand TOOLU 18
NOCHECKCONSISTENCY MAINEX subcommand TOOLU 288
\$noCheckConsistency M2 246
NOCO example module MTUT1 300
\$noCollectablePtrs M2 184
 bit M1 210; M2 178
\$noCollectableStrs M2 185
 bit M1 210; M2 178
\$noCompactablePtrs M2 185
 bit M1 210; M2 178
NOCONFIRM LIB command TOOLU 266
NOCONTROLINFO MAINEX subcommand TOOLU 289
NOCOUNTS subcommand TOOLU 128
node, computer KSTRMU 4
NODEBUG compiler subcommand TOOLU 18
NOECHOCMDFILE MAINEX subcommand TOOLU 292
NOECHOIFREDIRECTED MAINEX subcommand TOOLU 293
NOFILEINFO MAINEX subcommand TOOLU 294

NOFLDXREF compiler subcommand TOOLU 19
 \$noFlow KSTRMU 107
 NOGENCODE compiler subcommand TOOLU 21
 NOGENINLINES compiler subcommand TOOLU 21
 \$noHandler M2 186
 bit M2 206
 NOINCREMENTAL compiler subcommand TOOLU 21
 NOININTLIB compiler subcommand TOOLU 22
 NOINOBJLIB compiler subcommand TOOLU 22
 \$noInterrupt KSTRMU 107
 bit KSTRMU 89
 NOITFXREF compiler subcommand TOOLU 23
 \$noLegalNotice bit M2 170
 NOLIBRARY compiler subcommand TOOLU 23
 NOLOG
 compiler subcommand TOOLU 24
 INTLIB command TOOLU 238
 MODLIB command TOOLU 314
 NOMEMINFO MAINEX subcommand TOOLU 296
 NOMONITOR compiler subcommand TOOLU 25, 123
 NOMONITORAREA command TOOLU 132
 NOMONITORLIB command TOOLU 132
 NOMONITORMODULE command TOOLU 132
 non-blocking I/O KSTRMU 12
 non-compressed text forms TOOLU 156
 non-data-type-aligned address M1 22
 non-deterministic bugs MTUT2 34
 nonbound data section M1 106, 108; MTUT1 213
 nonbound-invocation module M1 117; TOOLU 31
 \$nonPaged bit TOOLU 169, 171, 173, 175, 176, 179
 \$nonRecursive M2 187
 bit M2 156, 173, 226
 nonRecursiveDebug TOOLU 83
 nonsticky compiler subcommands TOOLU 12
 NOOPTIMIZE
 "\$DIRECTIVE" directive M1 161
 compiler subcommand TOOLU 25
 NOOPTIMIZEALL "\$DIRECTIVE" directive M1 161
 NOOUTOBJLIB compiler subcommand TOOLU 26
 NOOUTPUT compiler subcommand TOOLU 27
 NOPROC example module MTUT1 40
 NOPROCS compiler subcommand TOOLU 28
 NOREDEFINE compiler subcommand TOOLU 29
 NORESPONSE
 compiler subcommand TOOLU 30
 MAINEX subcommand TOOLU 297
 noResponse M2 187, 246

bit M2 52, 119, 121
NOSAVEON compiler subcommand TOOLU 30
NOSLIST compiler subcommand TOOLU 30
NOSWAPINFO MAINEX subcommand TOOLU 301
NOT M1 33; MTUT1 97
NOTARGET compiler subcommand TOOLU 31
NOTIMING subcommand TOOLU 128
\$noTranslate M2 187
bit M2 35, 36
NOUNBOUND compiler subcommand TOOLU 31
NOUNEXECUTED subcommand TOOLU 128
NOUPDATE
INTLIB command TOOLU 239
MODLIB command TOOLU 316
NOVERBOSE LIB command TOOLU 266
NTST M1 34; MTUT1 98
NTSTA M1 34; MTUT1 98
NTTY device module KSTRMU 12, 104
NUL device module TOOLU 303
\$nulChar M1 7; M2 188
null
character KSTRMU 108; M1 7; M2 188
string M1 19
NULLADDRESS M1 23; MTUT1 288
NULLARRAY M1 61
\$nullArrayExcpt M1 224; M2 188
\$nullCallExcpt M1 224; M2 188
NULLCHARADR M1 23; MTUT1 288
NULLPOINTER M1 22; MTUT1 146
\$nullPointerExcpt M1 224; M2 189
\$numArgs M1 138; MTUT1 190
number of PDF characters in host data TOOLU 322
numbers, manipulating MEDTU 79
numeric operators (+, -, *, /, DIV, MOD) M1 34; MTUT1 98
\$numPagesOrSize TOOLU 171
NUMS example module MTUT1 110
NV LIB mode TOOLU 254

O

command TOOLU 81
commands MEDTU 36
OBJCOM module TOOLU 319
object
module MTUT1 6, 196
of command MEDTU 29

object-oriented programming MTUT1 213
 objects, display in hexadecimal TOOLU 77
 \$objFileFirst M2 237
 objList M1 129
 objmod M1 127; MTUT1 6, 196
 comparison TOOLU 319
 dispose TOOLU 80
 file name M1 127
 information about M2 170
 library M1 127; MTUT1 246; TOOLU 307
 library (opening from MAINEX) TOOLU 297
 search rules M1 128
 objSearch M1 128
 OC command TOOLU 81
 octal M1 18; M2 189
 bit M2 76, 82, 90, 200, 219
 \$octet bit KSTRMU 96, 99, 100
 octet KSTRMU 95
 OF M1 48
 OFB M1 48
 OI command TOOLU 82
 OL command TOOLU 82
 oldestVersion KSTRMU 36
 omit M2 190
 bit M2 231
 \$onesComplement M1 225
 Oops! MEDTU 70
 OP command TOOLU 83
 open M2 190
 coroutine ("OC" command) TOOLU 81
 file MTUT1 107
 procedure call M1 98
 simultaneous MTUT2 43
 OPENEXELIB MAINEX subcommand TOOLU 297
 opening
 a file M1 190; M2 190
 a file for PDF I/O M1 215
 a Structure Blaster file TOOLU 153
 OPENINTLIB MAINEX subcommand TOOLU 297
 OPENLIBRARY MAINEX subcommand TOOLU 297
 openLibrary M2 193
 OPENMODULE "\$DIRECTIVE" directive M1 120
 OPENOBJLIB MAINEX subcommand TOOLU 297
 \$openStream KSTRMU 18
 operating
 system attributes M2 29
 system number M2 259

operating system, definition M1 3
 operating system name
 abbreviation M2 258
 full M2 258
 operators
 dotted M1 40, 43
 precedence M1 38
 tables M1 32
 optimization M1 161
 OPTIMIZE
 "\$DIRECTIVE" directive M1 161
 compiler subcommand TOOLU 25
 OPTIMIZEALL "\$DIRECTIVE" directive M1 161
 \$optimized bit M2 171
 OPTIONAL M1 90; MTUT1 55
 \$optionalFirstEol KSTRMU 103
 options MEDTU 82
 debugger TOOLU 83
 OR M1 34; MTUT1 31, 98, 123
 order
 of evaluation M1 27
 of evaluation of operands M1 38
 of evaluation of procedure arguments M1 93; MTUT1 49
 \$OSDSize field of \$fileInfoCls M2 127
 OSMEMORYPOOLSIZE CONF command TOOLU 200, 306
 out of memory exception M2 22
 outer
 declaration M1 58, 105, 107; MTUT1 61
 variable M1 58, 105
 OUTINTFILE compiler subcommand TOOLU 26
 OUTINTLIB compiler subcommand TOOLU 26
 OUTOBJFILE compiler subcommand TOOLU 26
 OUTOBJLIB compiler subcommand TOOLU 26
 OUTPUT compiler subcommand TOOLU 27
 output M2 195; MTUT1 106
 bit KSTRMU 19, 51, 101; M2 192
 to a file M1 189
 overflow M1 14, 26
 arithmetic M2 24, 125; MTUT1 273
 stack M1 103; M2 249
 \$overheadPercentExitValue M2 195
 \$overheadTooHighExcpt M1 224; M2 195
 overlays, automatic M1 116
 overstrike mode MEDTU 21, 22, 36
 overstrikeModeDefault MEDTU 85
 overstriking characters (O commands) MEDTU 36

OWN M1 60
 own variable M1 60; MTUT1 83

P command MEDTU 25; TOOLU 91

\$p1 TOOLU 66
 \$p2 TOOLU 66

\$packet bit KSTRMU 96, 99

page MTUT1 291

- map MTUT1 292
- marks MEDTU 10, 30, 39, 42, 45
- of text MEDTU 10

page number, current M2 59

page size

- in character units M2 37
- in storage units M2 197

pageDispose M2 196; MTUT1 292

PAGEMAP LIB command TOOLU 269

\$pageRead M2 197; MTUT1 295; MTUT2 8

pages, deletion and recovery MEDTU 42

\$pageSize M2 197; MTUT1 291

PAGESUMMARY LIB command TOOLU 269

\$pageWrite M2 198; MTUT1 295; MTUT2 8

paragraph, filling M2 135

parallel processing KSTRMU 81

parameter MTUT1 44

- macro M1 132, 137; MTUT1 181
- mode M1 101
- of FLI procedures TOOLU 47
- procedure M1 87, 89, 94
- qualifiers M1 89

\$parent KSTRMU 9, 15, 68, 110

parentheses M1 39; MTUT1 19

- in expressions M1 38

parse

- bits string M2 219
- integer string M2 220
- real string M2 222

\$parseHostServiceName KSTRMU 60

Pascal, calling (from) TOOLU 19, 40

passing a procedure argument MTUT1 44

passwords, in object modules TOOLU 350

path name of a file M2 127

PDF MTUT1 142

- character translation M1 214
- data in RPC call KSTRMU 27

device prefix M1 215
 file data type size M2 152
 I/O M1 214
 image KSTRMU 26
 low-level procedures M2 198; TOOLU 321
 structure image TOOLU 150
 \$pdf M2 199
 bit KSTRMU 51; M1 215; M2 192; TOOLU 153, 172, 175, 179
 PDF I/O
 opening for M1 215
 positions in file M1 216
 pdfBoRead M2 198; TOOLU 331
 pdfBoWrite M2 198; TOOLU 332
 pdfbRead M2 198; TOOLU 331
 pdfbWrite M2 198; TOOLU 332
 pdfCharRead M2 198; TOOLU 322
 pdfChars M2 198; TOOLU 322
 pdfCharWrite M2 198; TOOLU 323
 pdfcRead M2 198; TOOLU 324
 pdfcWrite M2 198; TOOLU 324
 pdfDeInit M2 198; TOOLU 325
 pdfFldRead M2 198; TOOLU 325
 \$pdfImage TOOLU 171
 pdfInit M2 198; TOOLU 326
 pdfiRead M2 198; TOOLU 331
 pdfiWrite M2 198; TOOLU 332
 pdfLbRead M2 198; TOOLU 331
 pdfLbWrite M2 198; TOOLU 332
 pdfLiRead M2 198; TOOLU 331
 pdfLiWrite M2 198; TOOLU 332
 pdfLrRead M2 198; TOOLU 331
 pdfLrWrite M2 198; TOOLU 332
 PDFMOD M2 198; TOOLU 321
 charadr read procedures TOOLU 331
 charadr write procedures TOOLU 332
 deinitializing TOOLU 325
 initializing TOOLU 326
 number of PDF characters in host data TOOLU 322
 read character from PDF source TOOLU 324
 read characters from PDF source TOOLU 322
 read field from PDF source TOOLU 325
 read value from PDF source TOOLU 326
 write character to PDF destination TOOLU 324
 write characters to PDF destination TOOLU 323
 write value to PDF destination TOOLU 329
 pdfRead M2 198; TOOLU 326
 pdfrRead M2 198; TOOLU 331

pdfWrite M2 198; TOOLU 332
pdfWrite M2 198; TOOLU 329
performance
 monitoring M2 58; TOOLU 122
 monitoring compiler subcommands TOOLU 24, 25, 27, 28
PERMOD compiler subcommand TOOLU 27, 123
PERPROC compiler subcommand TOOLU 27, 123
PERSTMT compiler subcommand TOOLU 27, 123
PLANT example module MTUT1 343
PLATFORM CONF command TOOLU 200
platform
 definition M1 3
 name abbreviation M1 222; M2 199
 number M2 200
platform name, full M2 199
\$platformNameAbbreviation M2 199
\$platformNameFull M2 199
\$platformNumber M2 200
PLAYER example module MTUT1 236
PLIST example module MTUT1 163
pLoad M2 161
PLTFRM example module MTUT1 236
PMERGE module TOOLU 334
POINTER M1 22
pointer MTUT1 144
 area of M2 23, 148
 classified M1 74
 classifying in expression MTUT1 223
 safe and unsafe assignment M1 80
 unclassified M1 75; MTUT1 244
pointerCode M1 220
\$poll KSTRMU 92, 103
pool
 OS memory TOOLU 200
 static page TOOLU 200
POPACHECK "\$DIRECTIVE" directive M1 168
POPCHECK "\$DIRECTIVE" directive M1 165
\$port KSTRMU 60, 63
port KSTRMU 58
 in rendezvous KSTRMU 74
 server KSTRMU 25
portability
 of images and text forms TOOLU 154
 of string constants M1 7
Portable
 Data Format MTUT1 142
 Data Format (PDF) M1 213; TOOLU 321

- portable structure image TOOLU 150
- porting a structure TOOLU 156
- positions in file opened for PDF I/O M1 216
- precedence
 - of assignment operator MTUT1 122
 - of operators M1 38
- predeclared identifier MTUT1 58
- predefined exception M1 224
- preferred radix TOOLU 62
- \$preferredRadix M2 200; MTUT1 288; TOOLU 62
- prefix class M1 78; MTUT1 154, 223
- prefixed class MTUT1 223
- prevAlpha M2 200
- \$pri M1 221
- primary
 - I/O KSTRMU 104
 - input and output M1 191
- \$privatelyCached bit M2 205, 238
- PRNTCO module TOOLU 204
- PROCEDURE M1 84
 - command TOOLU 129
- Procedure
 - Expression M1 28
 - Statement M1 44; MTUT1 42
- procedure MTUT1 40
 - argument M1 87, 89
 - body M1 84
 - call M1 86, 87, 90, 99
 - calls M1 28, 44
 - declaration M1 84
 - field of a class M1 110
 - generic M1 99
 - header M1 84
 - inline M1 98
 - parameter M1 87, 89, 94
 - parameter qualifier M1 89
 - qualifiers M1 95
 - recompilation of TOOLU 10
 - statement counts TOOLU 128
 - typed M1 86
 - untyped M1 86
- procedure arguments, evaluation MTUT1 49
- procedure call
 - interprocess KSTRMU 21
 - local vs. intermodule MTUT2 8
- procedure name, current M2 59
- proceed M2 201

- bit M2 231
- process KSTRMU 5
 - communication KSTRMU 21
 - control KSTRMU 69
 - rendezvous KSTRMU 5, 73
- processes, simulating multiple MTUT1 299
- processor
 - attributes M2 29
 - definition M1 3
 - name abbreviation M1 221; M2 201
 - number M2 202
- processor name, full M2 202
- \$processorNameAbbreviation M2 201
- \$processorNameFull M2 202
- \$processorNumber M2 202
- PROCS
 - compiler subcommand TOOLU 28
 - example module MTUT1 41
- PROCTIME compiler subcommand TOOLU 28, 123
- producer-consumer problems MTUT1 299
- PRODUCES M1 89; MTUT1 44
- program MTUT1 7, 196
 - arguments M2 137
 - composition M1 105
- \$programInterface M2 202
 - bit M2 31
- \$programName M2 203
- prompt M2 203
 - bit M2 192
- debugger TOOLU 83
- propagating
 - an exception MTUT1 269
 - exceptions M1 172
- propagating an exception, implementation MTUT2 29
- proportionalWindowsMode MEDTU 85
- protocol
 - for server port KSTRMU 25
 - module KSTRMU 58
 - version in RPC KSTRMU 32, 33
- \$protocolName KSTRMU 57
- pseudo-fields of arrays M1 69
- pseudo-procedures M1 32
- pseudo-random number generator M2 149, 207, 249; TOOLU 336
- pseudo-terminal KSTRMU 11, 111
- PTY stream KSTRMU 11, 111
- PTYPRO KSTRMU 10, 65
 - STREAMS module KSTRMU 111

\$tty in child KSTRMU 104
PUSHACHECK "\$DIRECTIVE" directive M1 168
PUSHCHECK "\$DIRECTIVE" directive M1 165
PWD LIB command TOOLU 255

Q

command TOOLU 63
modifier (emphasis) MEDTU 29
Q= command MEDTU 63
QDIRECTORY
INTLIB command TOOLU 238
MODLIB command TOOLU 315
QF command MEDTU 21
qualified identifier M1 120
qualifiers M1 59, 95
\$queryFileCacheParms M2 204
\$queueCoroutine KSTRMU 77
quicksort TOOLU 341
QUIT
command TOOLU 131
CONF command TOOLU 201
INTLIB command TOOLU 233
LIB command TOOLU 268
MODLIB command TOOLU 311
quitting TOOLU 63
MAINED MEDTU 21
MAINEDIT MEDTU 11
\$quoteNext KSTRMU 109
QUOTES example module MTUT1 87

R

command TOOLU 84
commands MEDTU 44
R@ command TOOLU 84
R@@ command TOOLU 85
radix, preferred M2 200
\$raise M2 205; MTUT1 269
with no arguments M1 172
\$raiseReturn M1 172; M2 207; MTUT1 269
implementation MTUT2 29
raising
an exception M2 205; MTUT1 269
coroutine M2 121
\$ranCls TOOLU 336
\$rand M2 207; TOOLU 336

random M2 207
 access to file MTUT2 42
 access to files MTUT1 131
 bit KSTRMU 51; M2 192
 file access M1 189
 number generator M2 149, 207, 249; TOOLU 336
 range, guaranteed M1 14
 \$ranMod TOOLU 336
 ray, angle with x-axis M2 28
 rcRead M2 208; MTUT1 60; MTUT2 15
 rcWrite M2 209; MTUT1 61; MTUT2 20
 \$rdg M1 221
 re-entrant procedure caveat KSTRMU 76
 READ
 INTLIB command TOOLU 238
 LIB command TOOLU 268
 MODLIB command TOOLU 315
 read M1 7; M2 210; MTUT1 26, 106, 131, 288; MTUT2 18
 a character from the end of a string (rcRead) M2 208
 a field from a file or string M2 131
 character from PDF source TOOLU 324
 characters from PDF source TOOLU 322
 field from PDF source TOOLU 325
 from "TTY" (ttyRead) M2 268
 value from PDF source TOOLU 326
 read a character from a charadr, string, or file (cRead) M2 68
 read from file, string, or memory M2 210
 read-only buffer MEDTU 84; TOOLU 57
 READER example module MTUT1 18
 reading
 characters from a file M2 35
 storage units from a file M2 250
 readOnly MEDTU 84
 \$readStream KSTRMU 95
 REAL M1 16
 real MTUT1 90
 string parse M2 222
 realCode M1 220
 rearranging text MEDTU 47
 recalling text MEDTU 42, 44
 recognition of buffer names MEDTU 26
 recompilation
 incremental TOOLU 10
 of erroneous procedure TOOLU 8
 RECOMPILE compiler subcommand TOOLU 10, 28
 record MTUT1 144
 allocation M1 74; M2 71, 174, 180; MTUT1 149

- copying M2 62
- definition M1 71
- disposal M1 74; M2 111
- examining TOOLU 87
- field access M1 76, 78, 79
- size of M2 247
- unit TOOLU 159
- records, allocation of multiple M2 180
- recovering pages MEDTU 42
- recursion MTUT1 72
- recursive
 - descent MTUT1 76
 - macro MTUT1 187
 - macros MEDTU 75
 - multiplication example MTUT1 74
 - procedure invocation M1 95, 96
- recursive descent, errors and exceptions MTUT1 266
- recursiveDebug TOOLU 83
- REDEFINE M1 133; MTUT2 54
 - compiler subcommand TOOLU 29
- redirecting cmdFile and logFile MTUT2 44
- redirection
 - of cmdFile and logFile MTUT1 115
 - of I/O in MAINEDIT MEDTU 148
 - of standard input and output M1 193, 194
- refreshing the screen MEDTU 72
- registered exceptions M2 103, 118, 215
- \$registerException M1 178; M2 215; MTUT1 281
- related classes M1 80
- release of control section M1 111; M2 111, 271
- relFileName M2 217
- relModName M2 218
- relPos M2 218; MTUT1 132
- Remote Procedure Calls KSTRMU 21
- remote
 - file KSTRMU 50
 - module execution in child KSTRMU 29
 - module interface KSTRMU 26
- \$remoteModuleCls KSTRMU 22, 26, 36
- \$remoteModuleDefaults KSTRMU 26
- \$removeBits M2 219
- \$removeBoolean M2 220
- \$removeDateAndTime M2 222
- \$removeInteger M2 220
- \$removeLeadingBlankSpace M2 221
- \$removeMemMngModule M2 18, 221
- \$removeReal M2 222

\$removeTrailingBlankSpace M2 223
 \$removeWord M2 223
 removing
 a breakpoint TOOLU 84
 a breakpoint at a specified offset TOOLU 84
 all breakpoints TOOLU 85
 RENAME LIB command TOOLU 264
 \$rename M2 223
 renaming
 a file M2 223
 files TOOLU 339
 rendezvous KSTRMU 5, 73, 114
 \$reOpen M2 224
 reorder M2 225; TOOLU 349
 REPEATABLE M1 90; MTUT1 55
 repeatable macro parameter M1 138; MTUT1 190
 repeated execution of commands MEDTU 74
 repetition of previous command MEDTU 69
 replication M1 62
 report file TOOLU 130
 report file, format TOOLU 140
 \$reportAllVersions M2 225
 bit M2 105
 repository of shared data M2 143
 \$reprint KSTRMU 109
 REPTST example module MTUT1 57
 \$reschedule KSTRMU 77
 reserved
 identifiers M1 9, 233
 word MTUT1 9
 RESPONSE
 compiler subcommand TOOLU 30
 MAINEX subcommand TOOLU 297
 RESTORE CONF command TOOLU 201
 RESTOREFROM M1 122; MTUT1 350; MTUT2 51
 whether possible M2 59
 restricted access to module TOOLU 350
 \$resumeCoroutine M2 225; MTUT1 299
 and MAINDEBUG TOOLU 85
 resumer of a coroutine, most recent M1 181
 resuming
 a coroutine M1 180; M2 225
 a handled statement MTUT1 269
 retain M2 226
 bit M2 231
 RETURN M1 45
 key MEDTU 2

Return Statement M1 45; MTUT1 48, 71
\$returnExcpt M1 224; M2 227
\$returnIfNoHandler M2 227
 bit M2 206
returning from an exception M1 172; M2 207
\$returnKey KSTRMU 109
Reverse Polish Notation MTUT1 63
reverse M2 227; TOOLU 349
\$reverseDateAndMonth M2 228
 bit M2 99
rLoad M2 161
RM
 commands MEDTU 77
 LIB command TOOLU 264
RNMFIL module TOOLU 339
\$ros M1 222
RPC KSTRMU 21
 buffer KSTRMU 26, 27
 efficiency KSTRMU 39
 execution in child KSTRMU 29
 file KSTRMU 50
 for C KSTRMU 40
 module interface KSTRMU 26
 protocol version KSTRMU 32, 33
 server/client KSTRMU 21, 32
RPC compilation, whether is M2 58
RPC module, debugging KSTRMU 22
rpc_clear_jump KSTRMU 46
rpc_register_jump KSTRMU 46
RPCSRV server module KSTRMU 32, 38
RPN example module MTUT1 64
RS-232 stream KSTRMU 11
RSMCO module TOOLU 204
runtime
 creation of classes M2 69
 efficiency MTUT2 1

S

 command TOOLU 85
 commands MEDTU 32
safe assignment of pointers M1 80
\$sArg M1 138; MTUT1 190
SAVE
 CONF command TOOLU 201
 LIB command TOOLU 266

- save reminders MEDTU 73
- SAVEON M1 122; MTUT1 350; MTUT2 51
 - compiler subcommand TOOLU 30
- saving files MEDTU 11, 21, 67
- scan M2 228; MTUT1 126; MTUT2 18
 - bits M2 233
 - fast MTUT1 289
 - integers M2 234
- scanning compiler directives M1 156
- scanRel M2 232
- \$scanSet M2 234
- scanSet M2 233
- SCHED example module MTUT1 315
- scheduled
 - coroutine KSTRMU 5, 12, 75
 - TTY KSTRMU 8
- \$scheduledCoroutineMap KSTRMU 80
- Scheduler KSTRMU 12, 75
- scheduler, coroutine MTUT1 314
- scheduling of I/O KSTRMU 12
- SCOMAP module KSTRMU 80
- scope
 - of declaration MTUT1 19, 61
 - of identifiers M1 58
- scratch space M2 181, 182
- scratchDispose M2 234; MTUT1 288
- screen
 - format MEDTU 8
 - refresh MEDTU 72
- scrolling MEDTU 23, 30, 35
 - left-right MEDTU 55
- search
 - for character TOOLU 92
 - for string TOOLU 92
 - rules for module MTUT1 246
 - rules for modules M1 128
- \$searchCallChain M2 235
- searching MEDTU 33
- SEARCHPATH MAINEX subcommand TOOLU 297
- searchpath MTUT2 42
 - for files M2 244
- security, of object modules TOOLU 350
- selecting a file MEDTU 59
- selector M1 48
 - for \$CASEC M1 150
- \$semaphore KSTRMU 78
- semaphore KSTRMU 76

- deadlock KSTRMU 80
- semicolons M1 10
- sequence, defining with \$def M1 159
- sequential file access M1 189
- server KSTRMU 58
 - generic (RPCSRV) KSTRMU 38
 - installation KSTRMU 115
 - log file KSTRMU 37
 - process KSTRMU 5
 - RPC KSTRMU 21, 32
- server.log KSTRMU 37
- server/client rendezvous KSTRMU 74
- \$serverName KSTRMU 57
- SERVICE KSTRMU 58
 - service protocol table entry KSTRMU 117
- service KSTRMU 58
 - coroutine KSTRMU 25
 - protocol table KSTRMU 64, 115
- service version, RPC KSTRMU 33
- \$serviceName KSTRMU 63
- \$setBaudRate KSTRMU 108
- \$setCommandLine M2 236
- \$setConfigurationBit M2 236
- \$setExitCode M2 237
- SETFILE MAINEX subcommand TOOLU 299
- \$setFileCacheParms M2 238
- setFileName M2 239; MTUT1 246
- setModName M2 239
- SETMODULE MAINEX subcommand TOOLU 300
- setPos M2 242; MTUT1 132
 - caveat MTUT2 44
- \$setSearchPath M2 244
- \$setSystemBit M2 245
- \$setTheDate M2 245
- setting
 - the date (if date not provided) M2 245
 - up a structure TOOLU 174
- SETUP example module MTUT1 114
- several modules in one source file M1 116
- shallow usage TOOLU 122
- shared
 - data M2 143
 - data among tasks KSTRMU 76
- \$shareStrings bit TOOLU 168, 179
- shell KSTRMU 112
 - as child process KSTRMU 10
 - starting with PTYPRO KSTRMU 66

- writing KSTRMU 90
- shift overflow MTUT2 45
- SHL M1 34; MTUT1 98
- short arrays M1 61, 65
- short-array rule M1 68
- short-circuit evaluation MTUT1 123
- SHOW CONF command TOOLU 201
- SHR M1 34; MTUT1 98
- sign bit MTUT2 44
- signal, tracking infinite loop with MTUT2 38
- SIMPLE example module MTUT1 6
- simple
 - variable M1 27
 - variable declaration M1 59
- simulation, ecological MTUT1 322
- simultaneous
 - file open MTUT2 43
 - opens of the same file M1 190
 - remote procedure calls KSTRMU 81
- sin M2 247
- SINES example module MTUT1 92
- single step TOOLU 85
- single step, on procedure return TOOLU 70
- sinh M2 247
- site.cmd KSTRMU 117
- size
 - of a class MTUT1 285
 - of a data type MTUT1 285
 - of a file M2 127
 - of data type in file M2 152
 - of module M2 170
 - of modules M1 105
 - of structure TOOLU 171
 - system procedure M2 247
- skip to character or line MEDTU 32
- SKIPSCAN M1 156; MTUT2 54
- slices of array TOOLU 66
- SLIST compiler subcommand TOOLU 30
- sLoad M2 161
- SMSQRT example module MTUT1 320
- socket stream KSTRMU 11, 56
- SOCPRO KSTRMU 65
 - stream module KSTRMU 9
 - \$tty in child KSTRMU 104
- soft deletion of LIB file TOOLU 247
- SOFTDELETE LIB command TOOLU 265
- sort M2 249; TOOLU 342

- sorting M2 137
 - package TOOLU 341
- source
 - file containing multiple modules M1 116
 - file with multiple modules MTUT2 54
 - libraries MTUT1 83
 - library MTUT2 50
 - module MTUT1 196
- SOURCEFILE M1 143
 - compiler directive MTUT1 209
- sourcefile
 - automatic M1 158
 - whether possible M2 59
- sourcefiling file name M2 59
- \$spa M1 221
- SPACE
 - bar MEDTU 2
 - command TOOLU 132
- spaces M1 10
- SPECIAL M2 2
- special keys MEDTU 2
- speed of terminal MEDTU 6
- \$spix M1 222
- sqrt M2 249
- SQRTS example module MTUT1 37
- SQUARE example module MTUT1 27
- square root (sqrt) M2 249
- \$sRand M2 249; TOOLU 336
- SRCCONNECT LIB command TOOLU 255
- SRTMOD TOOLU 341
- SRVINFL module KSTRMU 133
- stack
 - as data structure MTUT1 63
 - frames MTUT2 24
 - in coroutine MTUT1 305
 - in exception MTUT1 276
 - overflow M1 103; M2 249
 - pointer MTUT2 24
 - size MTUT2 12
 - unwinding MTUT2 24
- stacking exceptions M1 173
- \$stackOverflowExcp M1 224; M2 249
- STACKSIZE CONF command TOOLU 201
- STAMP
 - module TOOLU 350
 - program interface TOOLU 361
- standard input and output

MAINSAIL M1 193
 operating system M1 191
 START example module MTUT1 236
 starting a coroutine M2 225
 \$startOutput KSTRMU 109
 STATEMENT command TOOLU 129
 statement M1 43; MTUT1 26
 executing TOOLU 88
 static page pool TOOLU 200
 statistics file TOOLU 124, 131
 STATUS LIB command TOOLU 269
 status
 line MEDTU 9
 obtaining MEDTU 63, 71
 STDNAME MAINEX subcommand M1 201; TOOLU 300
 stepping
 into procedures TOOLU 78
 over procedures TOOLU 85
 sticky compiler subcommands TOOLU 12
 stickyTabs MEDTU 83
 \$stopOutput KSTRMU 109
 storage
 template access M1 76
 unit M1 11; M2 247; MTUT1 106, 284
 units per page M2 197
 storage unit
 in character units M2 37
 size in bits M2 31
 storage units
 reading from a file M2 250
 writing to a file M2 251
 \$storageUnitRead M2 250; MTUT1 295; MTUT2 8
 and collections MTUT2 46
 \$storageUnitWrite M2 251; MTUT1 295; MTUT2 8
 and collections MTUT2 46
 store M2 252; MTUT1 287
 \$strArea M1 208
 STRCHK module TOOLU 184
 \$stream KSTRMU 16
 stream KSTRMU 5
 clearing KSTRMU 101
 end-of-line KSTRMU 99
 flushing KSTRMU 101
 gateway KSTRMU 128
 memory KSTRMU 114
 opening and closing KSTRMU 18
 stream I/O, low-level KSTRMU 92

STREAMS KSTRMU 4; MTUT1 299
and MAINVI MEDTU 107
intmod KSTRMU 15
STRHDR KSTRMU 15
STRING M1 19; MTUT1 19
string MTUT1 14
and character system procedures MTUT1 59
area of M2 23, 148
comparison M1 33; M2 56, 117; MTUT1 125; MTUT2 4
concatenation M1 19, 34; M2 60, 115; MTUT1 15, 98
constant MTUT2 18
descriptor M1 20; MTUT2 13
implementation MTUT2 12
length M2 159
maximum length M1 19
search TOOLU 92
space M1 20, 22; MTUT1 297; MTUT2 1, 4, 13
string space
clearing M2 43
getting a string into M2 140, 143
top MTUT2 17
stringCode M1 220
strong typing MTUT1 21
\$strToDate M2 254
\$strToDateAndTime M2 255
\$strToTime M2 256
STRTXT module TOOLU 184
\$strucInfo TOOLU 174
Structure Blaster MTUT2 10; TOOLU 150
structure
comparing TOOLU 167
converting image to text form TOOLU 169, 184
converting text form to data image TOOLU 184
converting text form to image TOOLU 176
copying TOOLU 168
displaying in debugger MTUT2 39
disposing TOOLU 170
examining or editing TOOLU 156
image TOOLU 150
information TOOLU 171
manipulating arbitrary TOOLU 150
reading TOOLU 172
setting up TOOLU 174
translating or porting TOOLU 156
writing TOOLU 177
\$structureCompare M2 257; TOOLU 167
\$structureCopy M2 257; TOOLU 168

\$structureDataToText M2 257; TOOLU 169
 \$structureDispose M2 257; TOOLU 170
 \$structureInfo M2 257; TOOLU 171
 \$structureRead M2 257; TOOLU 172
 \$structureSetUp M2 257
 \$structureSetup TOOLU 174
 \$structureTextToData M2 257; TOOLU 176
 \$structureUnSetUp M2 257; TOOLU 177
 \$structureWrite M2 257; TOOLU 177
 in debugger MTUT2 39
 stub, RPC KSTRMU 22
 SUBCMD module TOOLU 363
 SUBCOMMAND compiler subcommand TOOLU 31
 SUBCOMMANDS
 CONF command TOOLU 202
 MAINEX subcommand TOOLU 300
 subcommands
 compiler TOOLU 12
 MAINEX M2 141; TOOLU 281, 302, 363
 subscript M1 65; MTUT1 159
 subscripted variable M1 65
 \$subscriptExcpt M1 224; M2 257
 SUBSTR example module MTUT1 62
 substring M1 28, 29; MTUT1 123; MTUT2 16
 subtraction of date and time M2 96
 \$success KSTRMU 93
 SUMS example module MTUT1 73
 SUN module MEDTU 150
 Sun Workstation MEDTU 150
 Sun Microsystems, display module for MEDTU 138
 \$sun2 M1 222
 SUN3 module MEDTU 150
 \$sun3 M1 222
 \$sun38 M1 222
 \$sun4 M1 222
 SUN46 module MEDTU 150
 supporting intmod M1 120
 suppressHerald bit TOOLU 251
 suppressing terminal output MEDTU 71
 \$sw38 M1 222
 \$swapBit M2 246
 SWAPINFO MAINEX subcommand TOOLU 301
 swapping
 of modules M1 116; MTUT1 208; MTUT2 11
 tracking TOOLU 301
 switch, LIB TOOLU 254
 symbol

- global M2 143
- table M1 119
- visibility M1 121
- SYMTAB example module MTUT1 135
- synchronizing scheduled coroutines KSTRMU 78
- syntax
 - MAINDEBUG commands TOOLU 57
 - of LIB file name TOOLU 245
 - of utility commands TOOLU 188
- system
 - attributes M2 29
 - bit M2 245
 - name abbreviation M1 223; M2 258
 - number M2 259
 - procedure MTUT1 58
 - procedures and macros summary M2 3
 - shell as child process KSTRMU 10
- system name, full M2 258
- \$systemExcept MTUT1 281
- \$systemExcpt M1 176, 224; M2 258
- SYSTEMLIBNAME CONF command TOOLU 202
- \$systemNameAbbreviation M2 258
- \$systemNameFull M2 258
- \$systemNumber M2 259
- \$systemSupportsScheduling KSTRMU 10, 16
- \$systemSupportsTimeout KSTRMU 11
- \$systemSupportsTimeouts KSTRMU 16
- systemWrittenOn text form attribute TOOLU 158

T

- command TOOLU 86
- commands MEDTU 33
- T@ command TOOLU 86
- TAB key MEDTU 2
- tab M1 7, 10; M2 259; MTUT1 15
 - inserting before MEDTU 83
 - special treatment of MEDTU 10
 - stops MEDTU 72
- table
 - hash MTUT1 133
 - of allowed data type conversions M1 26
- tables of operations M1 32; MTUT1 96
- tan M2 259
- tangent (tan) M2 259
- tanh M2 260

TARGET

- compiler subcommand TOOLU 31
- INTLIB command TOOLU 238
- MODLIB command TOOLU 315
- target
 - definition M1 3
 - machine TOOLU 4
- task
 - scheduled KSTRMU 75
 - scheduling KSTRMU 12
- TDB KSTRMU 133
- Telemedia MEDTU 143
- TELEVI module MEDTU 151
- Televideo MEDTU 151
- template, access M1 76
- temporary breakpoint TOOLU 86
- termcap data base MEDTU 152
- terminal
 - echo KSTRMU 105
 - emulator example KSTRMU 80
 - I/O M1 191
 - interrupt KSTRMU 9, 87
 - speed MEDTU 6
- terminal interrupt character, reading KSTRMU 7
- terminal output, suppressing MEDTU 71
- TETRA example module MTUT1 90
- \$text bit KSTRMU 96, 99, 100
- text
 - editor MEDTU 1; MTUT1 4
 - file M1 188; MTUT1 106
 - form of structure TOOLU 150, 156
 - search MEDTU 33
- text file, viewing TOOLU 365
- textFile, predeclared class M1 187
- textual substitution MTUT1 178
- THEN M1 29, 46
- THENB M1 46; MTUT1 32
- THENC M1 148; MTUT1 184
- THENX TOOLU 225
- this
 - file name M2 59
 - line number M2 59
 - module name M2 59
 - page number M2 59
 - procedure name M2 59
- \$thisCoroutine M2 260
- \$thisCoroutine.\$next MTUT2 48

thisDataSection M2 260; MTUT1 244
\$thisFileName M2 262
thrashing, preventing M2 195
\$time M2 261; TOOLU 171
time M1 198
 addition M2 20
 arithmetic MTUT2 46
 conversion from string M2 255, 256
 conversion to string M2 97, 263
 difference M2 96
 of day M2 94, 261
 of file modification M2 127
 of module compilation M2 170
 of structure creation TOOLU 171
 removing from string M2 222
 standard representation M2 94
 zone M1 198; M2 61
 zone subcommands M2 262
time-sharing, simulating MTUT1 299
\$timeDifference M2 261
 bit M2 27
\$timedOut KSTRMU 63, 93, 103
\$timeFormat M2 262
\$timeout M2 263
 and STREAMS KSTRMU 12
timeout, STREAMS KSTRMU 11, 92
\$timerDspl TOOLU 137
\$timerPtr TOOLU 137
\$timeSubcommandsSet M2 262
\$timeToStr M2 263
TIMING subcommand TOOLU 128
\$timingPerModule bit M2 171
\$timingPerProc bit M2 171
title of area M1 208
TO
 in array declaration M1 61
 in Case Statement selector M1 48
 in substring specification M1 28
top of string space MTUT2 17
\$totalPagesOrSize TOOLU 174
TOYED example module MTUT1 249
translating a structure TOOLU 156
translation to/from PDF characters M1 214
TRIANG example module MTUT1 28
triangular numbers MTUT1 28
TRMCAP module MEDTU 152
TRUE M1 15; MTUT1 30

truncate M2 265
\$truncateFile M2 265
TST M1 34; MTUT1 98
TSTA M1 34; MTUT1 98
\$stConfigurationBit M2 266
\$stSystemBit M2 267
TTY M1 191; MTUT1 117
 break KSTRMU 108
 echo KSTRMU 105
 end-of-file KSTRMU 105
 end-of-line KSTRMU 106
 file and STREAMS KSTRMU 12
 interrupt KSTRMU 9, 87
 stream KSTRMU 6, 104
\$tty KSTRMU 6, 15, 104
TTY interrupt character, reading KSTRMU 7
ttyWrite M2 267; MTUT1 117
\$ttyEofExcpt M1 192, 224; M2 267
ttyRead M1 191; M2 268; MTUT1 117; MTUT2 18
 and ttyWrite and STREAMS KSTRMU 12
TTYSTR STREAMS module KSTRMU 104
ttyWrite M1 191; M2 269; MTUT1 117
TVI950 module MEDTU 151
TVIEW module TOOLU 365
\$twelveHour M2 270
 bit M2 264
two's complement M1 225
two-argument arctangent M2 28
\$twoYearDigits M2 270
 bit M2 100
TXTMGR back end MEDTU 4
type code M1 11, 220; M2 270; MTUT1 131
typed procedure M1 86; MTUT1 48
\$typeName M2 270
\$TYPEOF M1 154

U

 command TOOLU 91
 commands MEDTU 49
\$ua20 M1 223
\$ub1 M1 70
ub1 M1 70
\$ub2 M1 70
ub2 M1 70
\$ub3 M1 70

ub3 M1 70
\$uclp M1 223
UCRSYS KSTRMU 133
\$udg M1 223
\$ui38 M1 223
\$uibm M1 223
\$ultrx M1 222
\$um20 M1 223
\$um68 M1 223
\$umv M1 223
unaligned address M1 22
unBind M2 271
unbinding a module M2 271
\$unbindService KSTRMU 60
UNBOUND
 "\$SDIRECTIVE" directive M1 117
 compiler subcommand M1 117; TOOLU 31
\$unbound bit M2 171
\$unboundModuleExcp M1 111, 224; M2 272
\$unBuffered bit TOOLU 153
\$unbuffered M2 272; MTUT1 295; MTUT2 8
 bit KSTRMU 96, 99, 100; M2 35, 36, 132, 192
unbuffered I/O for Structure Blaster TOOLU 153
\$unbufferedEol KSTRMU 109
unclassified
 pointer MTUT1 154, 244
 pointer or address M1 75
UNDEFINE LIB command TOOLU 256
undefined, definition M1 3
UNDELETE LIB command TOOLU 264
undeleting MEDTU-42
 text MEDTU 44
underflow M1 14, 26
 arithmetic M2 24; MTUT1 273
undoing commands MEDTU 70
UNEXECUTED subcommand TOOLU 128
unexecuted
 entities TOOLU 128
 procedures TOOLU 141
 statements TOOLU 142
uninitialized
 local variable MTUT2 37
 variables M1 85
unique file name M2 72
unit in text form TOOLU 158
UNIX
 termcap data base MEDTU 152

tracking infinite loops on MTUT2 38
UNIXBITS CONF command TOOLU 202
unkilling text MEDTU 42
\$unlock KSTRMU 78
\$unmarkAllAreas TOOLU 180
unqualified identifier M1 123
unsafe assignment of pointers M1 80
unspecified, definition M1 3
UNTIL M1 51
UNTIL-clause MTUT1 36
untyped procedure M1 86
unwinding the stack MTUT2 24
UPDATE
 INTLIB command TOOLU 239
 MODLIB command TOOLU 316
UPDATEBAUDRATE keyword MEDTU 14
UPDATEDISPLAYMODULE keyword MEDTU 13
upper case, converting to MEDTU 49
upperCase M2 272; MTUT1 125
 bit M2 52, 56, 117, 231
\$upri M1 223
UPTO M1 51
\$urdg M1 223
USEED example module MTUT1 264
useKeyWord M2 273
 bit M2 52, 216
useMemFiles TOOLU 252
\$useOriginalFileName M2 103, 273
 bit M2 105, 129, 192, 224
\$useProgramInterface M2 273; MTUT1 248
user identification M2 274
user-defined sort ordering TOOLU 345
\$userID KSTRMU 65; M2 274
USES M1 89; MTUT1 44
\$uspa M1 223
UTC M1 198; M2 26, 61, 97, 106, 255
\$uts5 M1 222
\$uvax M1 223
\$uw38 M1 223
\$uxa M1 223

V

 command MEDTU 31; TOOLU 77, 86
 LIB mode TOOLU 254
valid address M1 22; MTUT1 289; MTUT2 45

values, display in hexadecimal TOOLU 77
\$varFormat M2 129
variable
 debugger TOOLU 74
 declaration MTUT1 17
 definition M1 27
 field M1 71, 76, 78, 79, 108, 110
 initialization M1 85
 interface M1 108, 110
 local M1 58, 84
 outer M1 58, 105
 own M1 60
 simple M1 27
 subscripted M1 65
variable-bounded arrays M1 61; MTUT1 170
\$vax M1 221
VAX-11 Calling Standard, calling (from) TOOLU 19, 40
vector unit TOOLU 161
VERBOSE LIB command TOOLU 266
\$version TOOLU 171
version
 number M2 167, 169
 of file TOOLU 246
 of MAINSAIL M2 58
 of module M2 170
 of structure TOOLU 154, 159, 171
 \$remoteModuleCls field KSTRMU 36
 RPC protocol KSTRMU 32, 33
view front end MEDTU 17
viewing a data structure TOOLU 156
virtual code space M1 116
VIS550 display module MEDTU 153
visibility, module and identifier M1 120
visiting a file MEDTU 59
\$vms M1 222, 223
VT100 module MEDTU 154
VT102 module MEDTU 154
VT102M module MEDTU 154

W

 command TOOLU 91
 commands MEDTU 23, 35
\$w38 M1 221
\$waitForDescendants KSTRMU 78
warning M2 275

- bit M2 119, 121; TOOLU 173, 176
- WHILE M1 51
- WHILE-clause MTUT1 36
- white space, removing M2 221, 223
- window MEDTU 3, 55
 - anchoring MEDTU 56
 - size MEDTU 55
 - status line of MEDTU 9
- windowing MEDTU 23, 30, 35
- windowUserLogString TOOLU 252
- WINDOWWIDTH keyword MEDTU 15
- \$WITH M1 171; MTUT1 268
- \$WITHB M1 171
- WITHCO example module MTUT1 302
- word M1 11; MTUT1 284
 - removing from string M2 223
- \$wordErase KSTRMU 109
- wordWrap MEDTU 84
- WRDCOM module TOOLU 368
- write M2 275; MTUT1 26, 106, 131, 288; MTUT2 5, 19
 - a character to "TTY" (tycWrite) M2 267
 - a character to the front of a string (rcWrite) M2 209
 - a field to a file or string (fldWrite) M2 132
 - character to PDF destination TOOLU 324
 - characters to PDF destination TOOLU 323
 - to "TTY" (ityWrite) M2 269
 - value to PDF destination TOOLU 329
- write a character to a file, string, or memory (cWrite) M2 92
- write to a file, string, or memory address M2 275
- WRITE2 example module MTUT1 16
- WRITE3 example module MTUT1 17
- WRITE4 example module MTUT1 17
- \$writeCalls M2 279
- WRITER example module MTUT1 15
- \$writeStream KSTRMU 98
- \$writeStreamBreak KSTRMU 108, 113
- \$writeStreamInterrupt KSTRMU 113
- writing
 - characters to a file M2 36
 - storage units to a file M2 251
- WRONG example module MTUT1 96
- WY43 module MEDTU 155
- WY50 module MEDTU 155
- WY5043 module MEDTU 155
- WY75 module MEDTU 155
- Wyse
 - WY-100 MEDTU 144

WY-50 MEDTU 155
WY-60 MEDTU 155
WY-75 MEDTU 155

X commands MEDTU 30, 55

\$xa M1 221

\$xcms M1 222, 223

XM command TOOLU 88

XON/XOFF KSTRMU 105

XOR M1 34, 37; MTUT1 98

XREF module TOOLU 369

XRFMRG module TOOLU 36

XS command TOOLU 88

Y commands MEDTU 30, 55

Z commands MEDTU 43

zapping text MEDTU 43

Zenith H-19 MEDTU 145

Zero M1 14

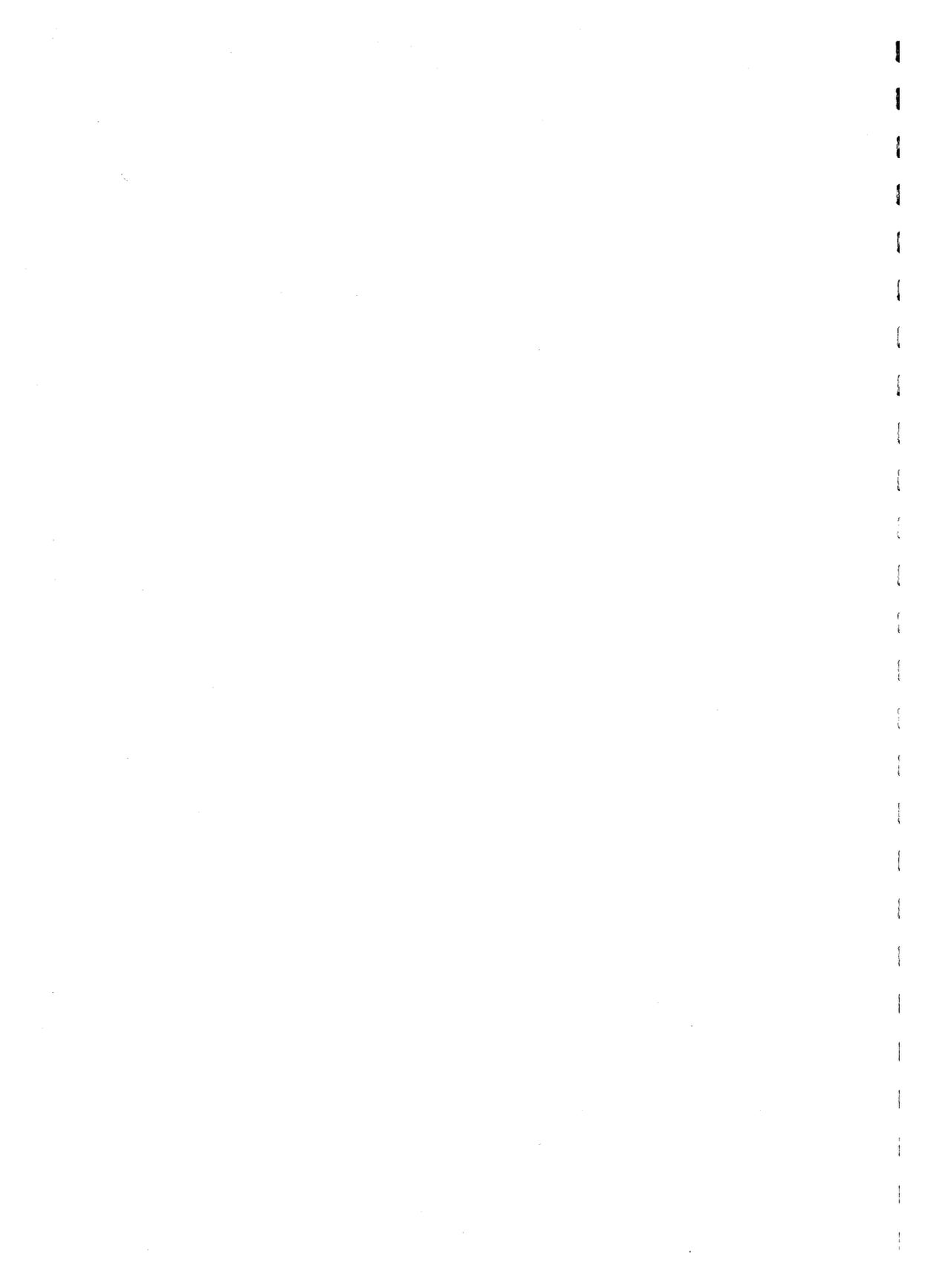
 comparison with MTUT2 7

 of a data type MTUT1 83, 93

zero, division by M2 24; MTUT1 273

zero-length files M1 190

zone, time M1 198; M2 61





XIDAK, Inc., 530 Oak Grove Avenue, M/S 101, Menlo Park, CA 94025, (415) 324-8745