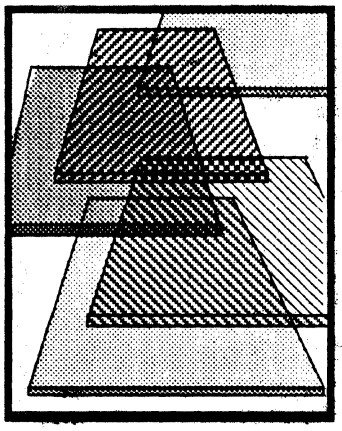
XEROX

Xerox Development Environment



Mesa Programmer's Manual

XDE3.0-4001
Version 3.0
November 1984

PRELIMINARY

Office Systems Division
Xerox Corporation
3450 Hillview Avenue
Palo Alto, California 94304

Notice

This manual is the current release of the Xerox Development Environment (XDE) and may be revised by Xerox without notice. No representations or warranties of any kind are made relative to this manual and use thereof, including implied warranties of merchantability and fitness for a particular purpose or that any utilization thereof will be free from the proprietary rights of a third party. Xerox does not assume any responsibility or liability for any errors or inaccuracies that may be contained in the manual or have any liabilities or obligations for any damages, including but not limited to special, indirect or consequential damages, arising out of or in connection with the use of this manual or products or programs developed from its use. No part of this manual, either in whole or part, may be reproduced or transmitted mechanically or electronically without the written permission of Xerox Corporation.

Preface

This document is one of a series of manuals written to aid in programming and operating the Xerox Development Environment (XDE).

Comments and suggestions on this document and its use are encouraged. The form at the back of this document has been prepared for this purpose. Please send your comments to

Xerox Corporation
Office Systems Division
XDE Technical Documentation, M/S 37-18
3450 Hillview Avenue
Palo Alto, California 94304

Table of contents

14 MsgSW

15 ScratchSW

16 StringSW

17 TextSW

18 TTYSW

19 Put

25 Caret

26 Cursor

27 Menu

28 Scrollbar

29 Selection

30 ToolFont

31 WindowFont

32 AsciiSink

33 BlockSource

34 DiskSource

35 PieceSource

36 ScratchSource

37 StringSource

38 TextData

39 TextSink

40 TextSource

IV User input and events

41 Event

42 EventTypes

43 TIP

44 UserInput

V File management

49 MLoader

50 MSegment

51 MStream

52 MVolume

VI Sorting and searching

53 BTree

Appendices

A Example Tool

B References B-1

C Listing of Public Symbols C-1

Index

Illustrations

I

General environment

The Xerox Development Environment provides interfaces for building tools and whole systems from start to finish. Interfaces suitable for use by programmers at differing levels of ability and familiarity with XDE are available for many tasks.

The interfaces in this section are all basic and should be studied both for content and to get a feel for the XDE paradigm of "tools and interfaces." The simplest interfaces are **Atom** and **Date**, followed by **Token**, **Executive**, **Expand**, **HeraldWindow**, and **Profile** interfaces. The most important interfaces in this group are **AddressTranslation**, along with **CmFile** and the **ToolDriver**.

In general, a programmer new to the Xerox Development Environment can get started building tools by looking at the interfaces in this and the next (Tool building) sections and then studying the Example Tool in Appendix A for specifics. The interfaces discussed later in this manual can be added to the programmer's repertoire as needed.

I.1 Files

Most facilities described in this manual are implemented by boot files. Some of the facilities are provided by packages that can be loaded in the boot files.

This manual does not explicitly mention the location of files. This information is in the documentation issued with each release of Mesa.

I.2 Philosophy and conventions

The development environment assumes that programs that run in it are friendly and are not trying to circumvent or sabotage the system. The system does not enforce many of the conventions described here, but it assumes that tool writers will adhere to them voluntarily. As with rules of etiquette, if these conventions are not followed, communication and sharing can break down: the development environment may degrade or break down altogether.

I.2.1 Users and clients

Throughout, this manual refers to *users* and to *clients*. These terms are not interchangeable, but refer to very different things.

A *user* is human being sitting at a workstation, typing keys, pressing buttons, and moving the mouse. User actions are not predictable or controllable by programs. Users never invoke program interfaces; they interact with facilities of the development environment in ways described in the *XDE User's Guide*.

A *client* is a program that invokes the facilities of the development environment. The client may act as a result of some *user* action, but the client's behavior is the result of a program and under the control of its implementor.

Tajo refers to the piece of the development environment that implements the user interface facilities.

I.2.2 Tools philosophy

The most important principle in the development environment is that users should have complete control over their environment. In particular, clients should not pre-empt users. A user should never be forced by a client into a situation where the only thing that can be done is to interact with one tool. Furthermore, the client should avoid falling into a particular "mode" when interacting with the user. The tool should avoid imposing unnecessary restrictions on the permitted sequencing of user actions.

This goal of user control has important implications for tool design. A client should never seize control of the processor while getting user input. This tends to happen when the client wants to use the "get a command from the user and execute it" mode of operation. Instead, a tool should arrange for Tajo to notify it when the user wishes to communicate some event to the tool. This is known as the "don't call us, we'll call you" principle.

The user owns the window layout on the screen. Although the client can rearrange the windows, this is discouraged. Users have particular and differing tastes in the way they wish to lay out windows on the display; it is not the client's role to override the user's decisions. In particular, clients should avoid making windows jump up and down to capture the user's attention. If the user has put a window off to the side, he does not want to be bothered by it.

The facilities provided by the development environment are designed to facilitate this same program writing style. In particular, the **Tool** interface makes it easy for a programmer to write a program that interacts with the user in this way. The development environment manages the details of user interaction so that tools are presented with a sequence of discrete commands or actions. Programmers should study the Example Tool in Appendix A for an example of how to use these facilities.

I.2.3 Notifier

Tajo sends most user input actions to the window that has set itself to be the focus for user input; the rest of the actions are directed to the window containing the cursor. (See the **TIP** interface for details on how the decision is made where to send these actions.) A process in Tajo notes all user input actions and determines which window should receive each. A

client is concerned only with the actions that are directed to its window; it need not concern itself with determing which actions are intended for it.

Two processes are involved in user input management. One is a high-priority process that queues user actions as they happen. The first process is called the **Interrupt Level**, the **Stimulus Level**, or the **StimLev**. The other is a normal-priority process that processes the user actions. This second process is called the **Notifier**, **Processing Level**, or the **Matcher**.

The **Notifier** informs a tool of a user action directed to it by calling a tool-supplied **NotifyProc** procedure. The standard tool facilities provide appropriate NotifyProcs so that tool writers need not worry about providing their own. Ambitious tool writers can, of course provide their own; see the **TIP** and **UserInput** interfaces. Tool writers creating their own subwindow types will probably have to do this.

It is important to realize that most tools operate from the Notifier process. The **Notifier** waits until a **NotifyProc** finishes for one user action before processing the next user action. The procedures associated with form subwindow commands, for instance, are executed in the Notifier process.

One implication of this use of the Notifier process is that a **NotifyProc** that requires a lot of computing or communicating will delay the processing of all other user actions until it completes. As a result, it is considered very impolite to "steal the Notifier" for any great length of time, thus preventing the user from using other tools. It is the Tajo philosophy that tools should never pre-empt the machine. Tool writers should **FORK** any command that will take more than three to five seconds to complete. Of course, the tool writer must take great care when stepping into this world of parallel processing. See the Example Tool in Appendix A for one method of protecting the tool when **FORK**ing.

Another implication of this use of the Notifier process is that the **Notifier** can be used to obtain mutual exclusion for processing user actions. This is desirable when a client wants to make a "background" request, one that will only receive machine resources if the **Notifier** is otherwise unoccupied. It is also desirable when a client wants to stop as much activity as possible, such as when a world swap is about to take place (see the section on Stopping tools).

Some facilities require that their procedures be invoked "from the Notifier." To allow non-Notifier processes to invoke these functions, Tajo provides a mechanism called a Periodic Notifier. The blinking caret in Tajo uses a Periodic Notifier. The **UserInput** interface describes Periodic Notifiers.

I.2.4 Multiple processes, multiple instances

Tajo supports many programs running simultaneously. The designer of a package should bear in mind that his package may be invoked by several different asynchronous clients. One implication of this constraint is that a package should be monitored.

The simplest design is to have a single entry procedure that all clients must call. While one client is using the package, all other clients will block on the monitor lock. Of course, no state should be maintained internally between successive calls to the package, since there is no guarantee that the same client is calling each time.

This simple approach has the disadvantage that clients are stopped for what may be a long time, with no option of taking alternate action. The restriction can be eased by having the entry procedure check a "busy" bit in the package. If the package is busy, the procedure can return this result to the client. The client can then decide whether to give up, try something else, or try again. This flexibility is less likely to tie up a tool for a long period, and the user can use the tool for other purposes.

If the package is providing a collection of procedures and cannot conform to the constraint that it provide its services in a single procedure, the package and its clients must pass state back and forth in the form of an object. For instance, the **FileTransfer** package implements a *Connection* object that holds information about a client's remote connection. The package can either use a single monitor on its code to protect the object or provide a monitor as part of each object. If it does the latter, several clients can be executing safely at the same time.

Some packages require that a client provide procedures to be called by the package. The designer of such a package should have these client-provided procedure take an extra parameter, a long pointer to client-instance data. When the client provides the package with the procedures, it also provides the instance data to pass to the procedures when they are called. This instance data can then be used by the client to distinguish between several different instances of itself that are sharing the same code.

As an example, the **FTP** program uses the **FileTransfer** facility to move files. For each file to be transferred, **FileTransfer** calls a procedure provided by **FTP** that decides whether the transfer should take place. This procedure uses the value of some switches to make this decision. **FTP** cannot keep these switches in global variables, since there may be several clients using its facilities at the same time. Instead, **FTP** passes the switch values to **FileTransfer** as its instance data, and **FileTransfer** passes the switches back to **FTP**'s procedure when it is called.

I.2.5 Resource management

Programs in the development environment must explicitly manage resources. For example, memory is explicitly allocated and deallocated by programs; there is no garbage collector to reclaim unused memory. All programs share the same pool of resources, and there is no scheduler watching for programs using more than their share of execution time, memory, or any other resource.

Programs must manage resources carefully. If a program does not return a resource when it is done with it, that resource will never become available to any other program and the performance of the environment will degrade. The most common resource, and one of the more difficult to manage, is memory.

When interfaces exchange resources, clients must be very careful about who is responsible for the resource. The program responsible for deallocating a resource is the *owner* of that resource. One example of a resource is a file handle. If a program passes a file handle to another program, both programs must agree about who owns that file handle. Did the caller transfer ownership by passing the file handle or is it retaining ownership and only letting the called procedure use the file handle? If the two programs disagree, the file will be released either twice or not at all. All interfaces involving resources must state explicitly whether ownership is transferred. To ease the problem of memory management when the ownership of memory can change, a common heap, called the *system heap*, is

used in Tajo. If a piece of memory can have its ownership transferred, it is either allocated from the system heap or a deallocation procedure must be provided for it. The **Storage** interface is useful for allocating and deallocating objects from the system heap.

The most common resource appearing in interfaces is a **STRING** or **LONG STRING**. There must be agreement about which program is responsible for deallocating the string body. Typically, a string passed as an input parameter does not carry ownership with it; implementors of such procedures should not deallocate or change the string. If the implementor must modify the string or use it after the procedure returns, it should first be copied. Tool writers should be particularly careful when a procedure returns a string to note whether ownership has come with it.

I.2.6 Tool state conventions

Tools can be in one of three states, *active*, *tiny*, or *inactive*. If a tool is active, the user has access to its full functionality and interface. If a tool is tiny, its window is displayed as a small icon, but its functions are still available. (Of course, the user may not be able to invoke them directly because the window is small.) If a tool is inactive, the tool window does not appear on the display and the tool is not functional. The tool appears on a menu of inactive tools.

A user makes a tool active when he wishes to use it. He makes a tool tiny when he expects to use it in the near future, but needs its space on the display for some other use. A user makes a tool inactive if he does not expect to use it at least for a while. An inactive tool might never be reactivated by the user.

Tool writers are responsible for supporting these definitions of tool state. Tajo provides the window management for these transitions, deallocating its resources as a tool is deactivated, and reallocating them when it is activated again. However, the tool writer must manage the resources the tool uses by providing a transition procedure that is called as the tool changes from one state to another. When a tool becomes tiny, its state is close to that of an active tool. However, it should not consume resources only needed for the display of the window, since the window is represented by an icon. When a tool becomes inactive, it should release all of its resources (free all streams, turn off all communications packages, deallocate all storage from the system heap, and so forth).

I.2.7 Program invocation

The development environment provides two styles of tool invocation, an interactive style and a batch style. The interactive style is supported by the tool window paradigm: users communicate with tools via a window and interact frequently with the program. A tool writer typically provides an interactive interface by creating a form subwindow with command items for each procedure. The batch style is supported by the Executive: users invoke programs via a command line and have very little interaction with the program while it is running. A tool writer typically provides a batch interface by writing one or more **Exec.ExecProc**s that can be called from the Executive.

It is usually desirable to be able to access the facilities of a package in either style as well as to access them from other programs. By taking care in the package design, a tool writer can make supporting these different invocation methods straightforward.

The tool writer should provide an interface that defines the function provided by his package. This functional interface can be called directly from programs, making it possible for client programs to use the package directly. The tool writer can then write two interface packages that invoke the functions of the package through the functional interface. One interface package implements an **ExecProc**; the other implements a tool window.

A few requirements must be satisfied by the functional interface to make it possible to write both interface packages. The functional interface should make no assumptions about where its input comes from or where its output goes. If the package must interact with the user, it requires interface packages for the interaction. It must not assume that it has a window it can communicate through. Also, the package should not assume it knows the location of input parameters. All input should be passed to the package explicitly by the interface packages, even if the input is just in the form of a command line that must be parsed. An output procedure should be provided by the caller.

I.2.8 Stopping tools

The development environment consists of cooperating processes. There are no facilities for cleanly terminating an arbitrary collection of processes. It is assumed that tool writers will be good citizens and design their tools to stop voluntarily when asked to stop.

A tool should stop if the user aborts it. The **UserInput** interface contains procedures that check whether a user has aborted a tool with the **ABORT** key in the tool's window. A tool should check for a user abort at frequent intervals and be prepared both to stop executing and to clean up after itself. Because the tool controls when it checks, it can check at points in its execution when its state is easy to clean up. Packages that can be called from several programs should take a procedure parameter that can be called to see whether the user has aborted.

There is another reason that a tool might be asked to stop: when it is running in CoPilot and CoPilot is about to return to the debuggee. CoPilot must take a snapshot of the state of the world; it requires that all processes stop so that the snapshot it takes corresponds to the state of the world when it does the core swap. CoPilot guarantees that the Notifier is not running, so tools that execute in the Notifier are automatically stopped. However, other programs must watch for the Supervisor event **Event.aboutToSwap**. If a program is notified about the swap while it has a process running, it must either stop the process or abort the world swap by raising the signal **Supervisor.EnumerationAborted** from within its agent procedure.

I.3 Interface abstracts

AddressTranslation translates between various elements in the internal form of network addresses and Ascii strings. Address translation is involved in any tool built for network activities.

Atom provides the mechanism for making **TIP** Atoms.

CmFile provides a simple set of procedures for processing "`User.cm`" format files. `User.cm` contains information for tailoring the environment to a user's taste.

Date converts dates and their string representations.

Exec supports program loading and running in the batch Executive. It includes operations for command line access and manipulation.

Expand provides facilities for the Executive-style expansion of lines containing expansion characters.

HeraldWindow implements routines for providing feedback to the user and for booting files and volumes.

Profile provides an interface to commonly accessed user and system data such as passwords, domains, and names.

Token provides a general text scanning facility, including several standard scanning procedures such as those for parsing numbers and booleans. It also permits clients to define their own entities.

ToolDriver allows a tool to inform the **ToolDriver** package of its existence and of the existence of its subwindows. The **ToolDriver** package can thus use a tool's functions on behalf of a user communicating with the package via a script file.

1

AddressTranslation

The AddressTranslation interface translates strings into the internal representation of network addresses. If a string cannot be translated locally, the Clearinghouse service will be consulted. Use the **Format** interface to convert network addresses from internal representation to text.

1.1 Types

```
AddressTranslation.NetworkAddress: TYPE = System.NetworkAddress;
```

1.2 Constants and data objects

None.

1.3 Signals and errors

```
AddressTranslation.Error: ERROR [errorRecord: AddressTranslation.ErrorRecord];
AddressTranslation.ErrorRecord: TYPE = RECORD [
SELECT errorType: AddressTranslation.ErrorType FROM
  scanError => [position: CARDINAL],
  badSyntax => [field: AddressTranslation.Field],
  chLookupProblem => [rc: CH.ReturnCode],
  otherCHProblem => [reason: AddressTranslation.Reason],
  ENDCASE];
AddressTranslation.Field: TYPE = {net, host, socket, ambiguous};
AddressTranslation.ErrorType: TYPE = {
  scanError, badSyntax, chLookupProblem, otherCHProblem};
```

scanError is raised if the input string contains illegal characters; **position** is the position of the offending character.

badSyntax is raised if the string to be parsed does not have the proper syntax; **field** identifies the incorrect field.

chLookupProblem	is raised if a Clearinghouse service could not find the name; **rc** gives details of the failure.
otherCHProblem	is raised if a name or value was not parseable by the Clearinghouse code or if the Clearinghouse service could not provide the address; **reason** gives more information on the failure.

```
AddressTranslation.Reason: TYPE = {
  noUsefulProperties, ambiguousSeparators, tooManySeparators, authentication,
  invalidName, invalidPassword, couldntDetermineAddress, spare1, spare2, spare3};
```

noUsefulProperties	the name was found, but did not have any of the desired properties associated with it (i.e., it did not have a network address).
ambiguousSeparators	the input string contained both ': and '@ separators.
tooManySeparators	the input string had more than two separators.
authentication	a problem occurred with the authentication servers.
invalidName	the user was logged in with an invalid name.
invalidPassword	the user was logged in with an invalid password.
couldntDetermineAddress	the string given to AddressTranslation was not found in the Clearinghouse service.

1.4 Procedures

```
AddressTranslation.StringToNetworkAddress: PROCEDURE [
  s: LONG STRING, id: Auth.IdentityHandle NIL,
  distingName: LONG STRING ← NIL]
  RETURNS [
  addr: AddressTranslation.NetworkAddress, chUsed: BOOLEAN];
```

The **StringToNetworkAddress** procedure parses **s** and returns a network address. When contacting the Clearinghouse service, **AddressTranslation** will look for a network address; **id** is the **Auth** identity that is used to contact the Clearinghouse service. If defaulted to **NIL**, one will be created from the Profile Tool. **distingName**, if not **NIL**, will be filled in with the actual distinguished name used in the Clearinghouse lookup; that is, the name obtained after dereferencing all aliases. **chUsed** will be **TRUE** if the Clearinghouse swas contacted. This procedure can raise the error **Error**.

```
AddressTranslation.StringToHostNumber: PROCEDURE [
  LONG STRING] RETURNS [System.HostNumber];
```

The **StringToHostNumber** procedure parses the **LONG STRING** and returns a **System.HostNumber**. This procedure only translates numeric strings; the Clearinghouse service will not be contacted. This procedure can raise the error **Error**.

```
AddressTranslation.StringToNetworkNumber: PROCEDURE [
  LONG STRING] RETURNS [System.NetworkNumber];
```

The **StringToNetworkNumber** procedure parses the **LONG STRING** and returns a **System.NetworkNumber**. This procedure only translates numeric strings; the Clearinghouse service will not be contacted. This procedure can raise the error **Error**.

```
AddressTranslation.PrintError: PROCEDURE [
  error: AddressTranslation.ErrorRecord, proc: Format.StringProc, clientData: LONG POINTER
NIL];
```

The **PrintError** procedure prints an error message to the **proc** provided by the client. **clientData** will be passed to the client's **proc**.

1.5 Examples

The standard format for network addresses is **hostNumber** or **netNumber.hostNumber.socketNumber**. For compatibility, '# may be used to delimit the parts of an address, but '. is preferred.

hostNumber can have four forms:

- An octal number optionally followed by a 'B or 'b
- A Clearinghouse name
- The special string "*"
- The special string "ME"

Clearinghouse names are strings of the form **local:domain:organization**. The local part of the name must start with an alphabetic character; the lengths of the parts of a name may not exceed **CH.maxLocalNameLength**, **CH.maxDomainNameLength**, and **CH.maxOrgNameLength** characters, respectively. Clearinghouse names are looked up in the Clearinghouse database using types from the unordered set {workstation, fileserver, printserver, mailserver, router, nsAddress, its, gws, ciu, ecs} until a match is found. The special string * gets the broadcast host number. The special string ME will not call Clearinghouse functions, but will get the host number of the machine that it is running on. For compatibility, '@ may be used to separate the parts of a Clearinghouse name, but ': is preferred.

netNumber and **socketNumber**, if used, can only be a octal number optionally followed by a 'B or 'b. Both **netNumber** and **socketNumber** can be defaulted. **netNumber** defaults to the caller's local network number; **socketNumber** defaults to **System.nullSocket**.

The translation routine will translate any string that is well formed and unambiguous. Examples:

```
74B.25200000016.2

Lassen

Lassen:OSBU North:Xerox

25200000016

74.Lassen

*

74.*.

.Lassen.2B

.25200000016b.2
```

74.2 is ambiguous because it could mean net.host or host.socket.

If the domain or organization fields are omitted, the default values are obtained from the **Profile** interface.

AddressTranslation has been extended to handle more types of numeric input. The three fields of a network address (net, host, and socket) may be specified in any of your favorite numeric bases including octal, decimal, hex, and even the baroque "product format."

Parsing rules are as follows:

- The possible bases are defined by the following ordered enumeration: {octal, decimal, hex, clearinghouse}.
- The character '- is ignored when determining the base, and ignored again when determining the value of a numeric specificaton.
- All fields are assumed to be octal. The assumption holds as long as no characters are encountered outside the range ['0 .. '7]. The last character of the field may be a 'B or 'b, which affirms the octal assertion.
- If a character in the range ['8 .. '9] is encountered, the assumed base is assigned the MAX[decimal, current base]. The last character of the field may be a 'D or 'd, which affirms the decimal assertion.
- If a character in the range ['A .. 'F] is encountered, the assumed base is assigned the MAX[hex, current base]. The last character of the field may be an 'H or 'h, which affirms the hex assertion.
- If the first character of a field is an alpha, the field is assumed to be a Clearinghouse name. This leads to the rule that hex specifications must begin with a number.

Examples:

14InchesBaby is a clearinghouse specification.

BEADFACE is a clearinghouse specification.

OBEADFACE is a hex numeric specification.

2

Atom

The **Atom** interface provides the definitions and procedures to create and manipulate atoms (unique objects; in this case text strings, something like Lisp Atoms).

2.1 Types

```
Atom.ATOM: TYPE = LONG STRING ← NIL;
```

An **Atom.ATOM** is a **LONG POINTER TO StringBody** that is guaranteed to be equal to any other **ATOM** with an equal **StringBody**. That is, **String.EqualStrings[atom1, atom2, FALSE]** if **atom1 = atom2**.

```
Atom.AList: TYPE = LONG POINTER TO DPCell ← NIL;
```

This type is not used by the Atom implementation.

```
Atom.DPCell: TYPE = RECORD [first: LONG STRING, rest: AList];
```

This type is not used by the Atom implementation.

2.2 Constants and data objects

None.

2.3 Signals and errors

None.

2.4 Procedures

```
Atom.MakeAtom: PROCEDURE [ref: LONG STRING] RETURNS [Atom.ATOM];
```

MakeAtom returns the **ATOM** corresponding to **ref**, creating one if necessary.

```
Atom.GetPName: PROCEDURE [atom: Atom.ATOM] RETURNS [pName: LONG STRING];
```

GetPName returns the **STRING** corresponding to **atom**, returning **NIL** if **atom** is unknown (not an **ATOM**).

3

CmFile

This interface provides a simple set of procedures for processing `User.cm` format files. See also the **Token** interface, since it is assumed that clients will use **Token** to parse the contents of Cm files.

A Cm file is a sequence of *sections*. A section is a title line followed by zero or more name-value pairs. A section may not have embedded blank lines because a blank line is considered to terminate a section. The title line begins with a [and the section title is defined to terminate with the first succeeding]. The section title may be optionally preceded by a logical volume name and a colon, with no embedded spaces. An example of this would be [Tajo:System]. If the section title is preceded by a logical volume name, the lines in that section will be recognized only on the named volume and will override specific lines in sections by the same name with no volume qualification. Each name-value pair is on a separate line; the name must be followed by :. Both the name and the value can be preceded by white space. The value field is terminated by the first carriage return. A comment line is a line beginning with--; it may appear anywhere within a section.

3.1 Types

```
CmFile.Handle: TYPE = Token.Handle;
```

A **CmFile.Handle** can be used with any of the routines in the **Token** interface for parsing. Many of the procedures in this interface take a **Handle** parameter and provide standard routines for parsing Cm files.

3.2 Constants and data objects

```
CmFile.noMatch: CARDINAL = StringLookUp.noMatch;
```

3.3 Signals and errors

```
CmFile.Error: SIGNAL [code: CmFile.ErrorCode];

CmFile.ErrorCode: TYPE = {fileNotFound, invalidHandle, other};
```

fileNotFound	the file to be processed could not be acquired for reading.
invalidHandle	a **Cmfile** procedure has been called with a **Token.Handle** that was not created by **CmFile**.

```
CmFile.TableError: SIGNAL [h: CmFile.Handle, name: LONG STRING];
```

Within the procedure **NextValue**, a name was encountered in the **CmSection** that was not in the table of names expected. **h** is the handle that was used in the Cm file parsing, and **name** is the unrecognized name. If this signal is resumed, the name/value pair is ignored and processing continues. **h** is positioned to the beginning of the value field of the item. The client may read from **h** up through but not past the closing carriage return while in the catch frame without interfering with further processing.

3.4 Procedures

```
CmFile.Close: PROCEDURE [h: CmFile.Handle] RETURNS [nil: CmFile.Handle];
```

The **Close** procedure frees the **Handle** and returns **NIL**. If an illegal **CmFile.Handle** is supplied, **CmFile.Error[invalidHandle]** is raised.

```
CmFile.FindItem:PROCEDURE [h:CmFile.Handle, title, name: LONG STRING] RETURNS [found: BOOLEAN];
```

FindItem searches for the entry **name** in section **title** in the file on which the Handle **h** was opened. If the search is successful, **FindItem** returns **TRUE**. Otherwise, it returns **FALSE**. If the search is successful, the Handle **h** will be positioned to the beginning of the value for **name**. Procedures in the Token interface can then be used to parse the value field; e.g., Token.Boolean can be used to parse a boolean value. If an illegal **CmFile.Handle** is supplied, **CmFile.Error[invalidHandle]** is raised.

```
CmFile.FindSection: PROCEDURE [
  h: CmFile.Handle, title: LONG STRING] RETURNS [opened: BOOLEAN];
```

The **FindSection** procedure searches for the section named **title** in the file on which the **Handle** was opened. If it finds the section, it returns **TRUE** and positions the **Handle** to parse that section. If an illegal **CmFile.Handle** is supplied, **CmFile.Error[invalidHandle]** is raised.

```
CmFile.FreeString:PROCEDURE [LONG STRING] RETURNS [nil:LONG STRING];
```

FreeString deallocates strings returned from other procedures in **CmFile**. It returns **NIL**.

```
CmFile.Line: PROCEDURE [
  fileName, title, name: LONG STRING] RETURNS [LONG STRING];
```

The **Line** procedure returns the value for **name** from section **title** in the file on which the handle **h** was opened. It returns **NIL** if the file, section, or the named entry cannot be found. It is the caller's responsibility to deallocate the **LONG STRING** returned from **Line** using **FreeString**. If the file named **fileName** is not found or cannot be acquired for reading, **CmFile.Error[fileNotFound]** is raised.

```
CmFile.NextItem: PROCEDURE [h: CmFile.Handle] RETURNS [name, value: LONG STRING];
```

The **NextItem** procedure is used for enumerating the entries in a section. To start the enumeration, position the **Handle** by calling **FindSection**. When name is **NIL**, the end of the section has been encountered. **name** is the name partion of an item; that is, the part preceding the colon. **value** is the rest of the line with leading white space suppressed. It is the caller's responsibility to deallocate the **LONG STRING**s returned from **NextItem** using **CmFile.FreeString**.

```
CmFile.NextValue: PROCEDURE [h: CmFile.Handle, table: StringLookUp.TableDesc] RETURNS
[index: CARDINAL];
```

The **NextValue** procedure is used for enumerating the entries in a section. To start the enumeration, position the **Handle** by calling **FindSection**. The name of the next item in the section is looked up in **table**, and the index of the item is returned. Standard **Token** procedures can then be used to parse the value of the entry. When **index** is **CmFile.noMatch**, the end of the section has been encountered. If an item that is not in the table is found, the resumable **SIGNAL CmFile.TableError** is raised. If **TableError** is resumed, the value is skipped and the scan continues.

```
CmFile.Open: PROCEDURE [fileName: LONG STRING] RETURNS [h: CmFile.Handle];
```

The **Open** procedure returns a **CmFile.Handle** on the file **fileName**. This handle is then used by other **CmFile** or **Token** routines for processing the file. If the file does not exist or cannot be acquired for reading, **CmFile.Error[fileNotFound]** is raised. If **CmFile.Error[fileNotFound]** is resumed, **NIL** is returned.

```
CmFile.ReadLineOrToken: PROCEDURE [
  h: Token.Handle, buffer: LONG STRING, terminator: CHARACTER];
```

ReadLineOrToken reads from **h** until **terminator** is found, unless either end-of-line or end-of-stream is encountered. The resulting line or token is returned via **buffer** and the break character is retained in the **Token.Object** pointed to by **h**. If **buffer** is too short, **ReadLineOrToken** quits and the break character is the character that was being processed when the **buffer** overflowed.

```
CmFile.TitleMatch: PROCEDURE [
  buffer, title: LONG STRING] RETURNS [matches: BOOLEAN];
```

TitleMatch returns **TRUE** if and only if the contents of **buffer** is in the right format to be the start of the section specified by **title**. For example, if **buffer** were [Id] and **title** were Id, **TitleMatch** would return **TRUE**.

```
CmFile.UserDotCmLine: PROCEDURE [title, name: LONG STRING] RETURNS [LONG STRING];
```

The **UserDotCmLine** procedure performs a **Line** operation on the file named `User.cm`.

```
CmFile.UserDotCmOpen: PROCEDURE RETURNS [h: CmFile.Handle];
```

The **UserDotCmOpen** procedure performs an **Open** on the file named `User.cm`.

3.5 Example

The following examples are based on the `User.cm` processing done by the Print program. It uses facilities of both the **CmFile**, **StringLookUp**, and the **Token** interfaces. The type field in the `User.cm` section corresponds to an enumerated type. The Interpress file corresponds to a name (string) that may be a quoted string containing spaces. **SetupOptions** returns the values found in the `User.cm` or the default values of the items if they are not present in the `User.cm`. The first example is more straightforward than the second, but it involves more string copying.

```
SetupOptions: PROCEDURE RETURNS [
  type: PrintOps.FileFormat ← OldPress, interpressPrinter LONG STRING ← NIL] =
  BEGIN
  Option: TYPE = MACHINE DEPENDENT{
    preferredFormat(0), interpress(1), noMatch(StringLookUp.noMatch)};
  DefinedOption: TYPE = Option [preferredFormat..interpress];
  optionTable: ARRAY DefinedOption OF LONG STRING ← [
    preferredFormat: "PreferredFormat"L, interpress: "Interpress"L];
  userCm: CmFile.Handle ← NIL;
  i: Option;
  entry, value: LONG STRING ← NIL;
  thisOption: Option;
  userCm ← CmFile.UserDotCmOpen[ ! CmFile.Error = > CONTINUE];
  IF userCm # NIL AND CmFile.FindSection[userCm, "HardCopy"L] THEN
    DO ENABLE UNWIND = > {
        entry ← CmFile.FreeString[entry]; value ← CmFile.FreeString[value]};
      [entry, value] ← CmFile.NextItem[userCm];
      IF entry = NIL THEN EXIT;
      thisOption ← StringLookUp.InTable[
        key: entry, table: DESCRIPTOR[BASE[optionTable], LENGTH[optionTable]]];
      SELECT thisOption FROM
        preferredFormat = >
          BEGIN
          parseHandle: Token.Handle ← Token.StringToHandle[value];
          parseValue:LONG STRING ← Token.Item[parseHandle];
          IF String.EquivalentStrings[parseValue, "Interpress"L] THEN
            type ← Interpress;
          [] ← Token.FreeTokenString[parseValue];
          [] ← Token.FreeStringHandle[parseHandle];
          END;
        interpress = >
          BEGIN
          parseHandle: Token.Handle ← Token.StringToHandle[value];
          InterpressPrinter ← Token.FreeTokenString[InterpressPrinter ];
          InterpressPrinter ← Token.MaybeQuoted[
            h: parseHandle, data: NIL, filter: Token.NonWhiteSpace,
            isQuote: Token.Quote, skip: whiteSpace,
            -- allocate minimum space for the string, since we will use it directly
            temporary: FALSE];
          [] ← Token.FreeStringHandle[parseHandle];
          END;
```

```
        ENDCASE;
      entry ← CmFile.FreeString[entry]; value ← CmFile.FreeString[value];
      ENDLOOP;
  IF userCm # NIL THEN [] ← CmFile.Close[userCm];
  END;

SetupOptions: PROCEDURE RETURNS [
  type: PrintOps.FileFormat ← OldPress, interpressPrinter: LONG STRING ← NIL] =
  BEGIN
  Option: TYPE = MACHINE DEPENDENT{
    preferredFormat(0), interpress(1), noMatch(StringLookUp.noMatch)};
  DefinedOption: TYPE = Option [preferredFormat..interpress];
  optionTable: ARRAY DefinedOption OF LONG STRING ← [
    preferredFormat: "PreferredFormat"L, interpress: "Interpress"L];
  userCm: CmFile.Handle ← NIL; i: Option;
  -- the following declaration exists to make the LOOPHOLE in MyNextValue safe.
  -- If CmFile.NextValue changes type, the compiler will flag the following as an error.
  CheckType: PROCEDURE [h: CmFile.Handle, table: StringLookUp.TableDesc]
    RETURNS [index: CARDINAL] = CmFile.NextValue;
  -- loophole CheckType into the type expected by StringLookUp
  MyNextValue: PROCEDURE [
    th: CmFile.Handle,
    table: LONG DESCRIPTOR FOR ARRAY DefinedOption OF LONG STRING]
    RETURNS [index: Option] = LOOPHOLE[CheckType];
  userCm ← CmFile.UserDotCmOpen[ ! CmFile.Error = > CONTINUE];
  IF userCm # NIL AND CmFile.FindSection[userCm, "HardCopy"L] THEN
    DO
      SELECT
      (i ← MyNextValue[h: userCm, table: DESCRIPTOR[optionTable]
        ! CmFile.TableError = > RESUME ]) FROM
        noMatch = > EXIT;
        preferredFormat = >
          BEGIN
          value: LONG STRING = Token.Item[userCm];
          IF String.EquivalentStrings[value, "Interpress"L] THEN
            type ← Interpress;
          [] ← Token.FreeTokenString[value];
          END;
        interpress = >
          BEGIN
          value: LONG STRING = Token.MaybeQuoted[
            h: userCm, data: NIL, filter: Token.NonWhiteSpace,
            isQuote: Token.Quote, skip: whiteSpace,
            -- allocate minimum space for the string, since we will use it as the value
            temporary: FALSE];
            InterpressPrinter ← value;
          END;
        ENDCASE;
      ENDLOOP;
  IF userCm # NIL THEN [] ← CmFile.Close[userCm];
  END;
```

4

Date

The **Date** interface provides for a conversion between dates and their string representations. (Also see **Time** in the *Pilot Programmer's Manual*).

4.1 Types

```
Date.Packed: TYPE = Time.Packed;
```

Packed is copied from the **Time** interface.

```
Date.Notes: TYPE = {normal, noZone, zoneGuessed, noTime, timeAndZoneGuessed};
```

Notes is used as one of the return values from the call on **StringToPacked**. **normal** means the value returned is unambiguous; **noZone** means that a time-of-day was present, but without a time zone indication. (The local time zone as provided by **System.LocalTimeParameters** is assumed.) **zoneGuessed** is returned instead of **noZone** if local time parameters are not available, and the time zone is assumed to be Pacific Time (standard or daylight time is determined by the date). **noTime** and **timeAndZoneGuessed** are equivalent to **noZone** and **zoneGuessed**, respectively, where the time is assumed to be 00:00:00 (local midnight).

4.2 Constants and data objects

None.

4.3 Signals and errors

```
Date.Unintelligible: ERROR [vicinity: NATURAL];
```

If **StringToPacked** cannot reasonably interpret its input as a date, **Unintelligible** is raised; **vicinity** gives the approximate index in the input string where the parser gave up.

4.4 Procedures

```
Date.PackedToLongString: PROCEDURE [Date.Packed] RETURNS [LONG STRING];
```

The **PackedToLongString** procedure converts the date to a **LONG STRING** that is allocated from the system heap. The format is identical to that obtained by a call on **Time.Append**.

```
Date.PackedToString: PROCEDURE [Date.Packed] RETURNS [STRING];
```

The **PackedToString** procedure converts the date to a **STRING** that is allocated from the system MDS heap.

```
Date.StringToPacked: PROCEDURE [LONG STRING]
 RETURNS [dt:Date.Packed, notes:Date.Notes, length:NATURAL];
```

The **StringToPacked** procedure parses the string and returns a GMT time according to the Pilot standard. The date is generally assumed to precede the time, although if the time precedes the date it will usually be properly recognized. The date syntax is a somewhat less restrictive version of RFC733; full RFC733 is recognized, plus forms like "month day, year," "mm/dd/yy," and variations with Roman numerals used for the month. The form "year month day" is also accepted if the year is a full 4-digit quantity. Forms with "-" instead of significant space are also acceptable, as well as forms in which a delimiter (space or "-") can be elided without confusion. The time is generally assumed to be in RFC733 format, optionally including a time zone specification. In addition, "am" or "pm" may optionally appear following the time (but preceding the time zone, if any). **notes** is interpreted as described above. **length** indicates the number of characters consumed by the parser; that is, it is the index of the first character of the argument that was not examined by the parser. This procedure can raise the error **Date.Unintelligible**.

5

Exec

The **Exec** interface supports program loading and running as well as command line access and manipulation. The paradigm for programs running from the Executive is that they will register with the Executive one or more command names and a corresponding procedure to be called for each command.

5.1 Types

```
Exec.CheckAbortProc: TYPE = PROCEDURE [h: Handle] RETURNS [abort: BOOLEAN];
```

A **CheckAbortProc** procedure is used to check if a subsystem has been aborted by the user. **CheckAbortProc** procedures are used by **Run** and **ProcessCommandLine**.

```
Exec.ExecProc:TYPE = PROCEDURE [h: Handle, clientData: LONG POINTER ← NIL]
  RETURNS[outcome: Outcome ← normal];
```

An **ExecProc** procedure is the type of procedure a subsystem registers with the Executive so that its facilities can be invoked. The Executive calls the procedure with a **Handle** that can be used for input and output, as well as a **LONG POINTER**, **clientData**, which can be used for optional instance data. The subsystem returns an **outcome** that the Executive uses to decide whether to continue with the current command line. If the result is normal, the Executive continues; if it is any other value, the Executive skips the remainder of the current command line and prompts the user for more commands.

```
Exec.GetCharProc: TYPE = PROCEDURE [h: Handle,] RETURNS [char: CHARACTER];
```

GetCharProc is the type declaration for the Executive procedure that returns the next character on the command line (see **Exec.GetChar**).

```
Exec.Handle: TYPE = LONG POINTER TO Exec.Object;
```

When the Executive calls one of its registered procedures, it passes it a **Handle** that the subsystem can use to obtain the Executive's facilities.

```
Exec.Object: TYPE = ...;

Exec.Outcome: TYPE = MACHINE DEPENDENT{
  normal(0), warning, error, abort, spare1, spare2, spare3, last(LAST[CARDINAL])};
```

Outcome is returned by an **ExecProc** to indicate the status of the operation.

normal	the procedure was completed successfully.
warning	the procedure wishes to warn you about suspicious results.
error	the procedure was not able to be completed successfully.
abort	the procedure was aborted by the user.

All outcomes except **normal** cause the Executive to abort the rest of the current command line.

```
Exec.RemovedStatus: TYPE = {ok, noCommand, noProgram};
```

RemovedStatus is used by the **Unload** command to indicate its success.

ok	the program associated with the command was successfully unloaded.
noCommand	a command of the requested name was not found.
noProgram	the requested command was found, but the program that implements the command could not be located.

5.2 Constants and data objects

None.

5.3 Signals and errors

None.

5.4 Procedures

```
Exec.Abort: PROCEDURE RETURNS [error: ERROR];
```

The **Abort** procedure returns the error that subsystems should raise to abort processing.

```
Exec.AddCommand: PROCEDURE [
  name: LONG STRING, proc: Exec.ExecProc, help, unload: Exec.ExecProc ← NIL];
  unload: ExecProc ← DefaultUnloadProc, clientData: LONG POINTER ← NIL];
```

The Executive maintains a list of commands that are invoked by typing their name into the Executive window. Each command has an associated procedure that implements its functions, as well as a help procedure, a cleanup procedure, and optional client-instance data. The **AddCommand** procedure adds **name** to the Executive's list of commands and associates **proc** with it as the procedure to call when the command is invoked. Even

though by convention all command names in the Executive terminate with .~, these characters are not automatically appended to **name**, but instead are the client's responsibility. If there is already a command by the same name, **AddCommand** overrides the old entry.

In addition to the **name** parameter, **AddCommand** takes three other parameters, **help**, **unload**, and **clientData**. The **help** procedure is run whenever you ask for help on the corresponding registered command. The **unload** procedure is called when you wish to remove a command from the command list and unload its corresponding procedure. Unloading an Executive command consists of two steps: first, all commands added by the module being unloaded must be removed from the Executive's list of commands; and second, one of the procedures associated with any command added by the module that implements the subsystem must be unloaded. It is sufficient to unload only one procedure in the implementing module because unloading any procedure causes the entire module to be unloaded. The first step, that of removing commands from the command list, is done in the **unload** procedure. That is, the client's **unload** procedure must initiate a **RemoveCommand** on itself and all other commands registered by that module, as well as perform any other cleanup necessary before being unloaded. **UnloadCommand** will call **unload** and then automatically perform the second step, which is to actually unload an associated procedure (there are restrictions on the client unload procedure; see **UnloadCommand** and **RemoveCommand** for details). Usually, the command being unloaded is the only one registered by its containing module, and there are no other cleanup functions to perform. In this case, the client need not have its own **unload** procedure but instead may use **DefaultUnloadProc**, since **DefaultUnloadProc** removes the command for the subsystem it is associated with and then unloads the corresponding procedure (see **DefaultUnloadProc**).

```
Exec.AliasCommand:PROCEDURE[old, new: LONG STRING] RETURNS[ok: BOOLEAN];
```

AliasCommand allows youto associate the same procedure with more than one command, after the original command has already been registered. **old** is the name of the command originally added with **AddCommand**, and **new** is the name of the command to associate with the same procedure as **old**. Any number of commands can be aliases of an original command, and any number of aliases can have aliases also.

```
Exec.AppendCommands: PROCEDURE [h: Exec.Handle, command: LONG STRING];
```

The **AppendCommands** procedure appends the parameter **command** to the current command line. The effect is as if you had typed the contents of **command** after the current command line. Note that it is processed before any commands that have been typed ahead to the Executive.

```
Exec.CheckForAbort: CheckAbortProc;
```

The **CheckForAbort** procedure indicates whether the subsystem should abort.

```
Exec.Confirm: PROCEDURE [h: Exec.Handle] RETURNS [yes: BOOLEAN];
```

The **Confirm** procedure asks you for confirmation.

```
Exec.DefaultUnloadProc: ExecProc;
```

DefaultUnloadProc is a default value for the **unload** procedure; it is specified at the time a command is registered with the Executive (see **AddCommand**). It can be used in cases when the subsystem registers only one command, itself, and no other cleanup is to be done upon being unloaded.

```
Exec.EndOfCommandLine: PROCEDURE [h: Exec.Handle] RETURNS [BOOLEAN];
```

The **EndOfCommandLine** procedure indicates whether there are any more characters for this subsystem on the command line.

```
Exec.EnumerateCommands: PROCEDURE [
  userProc: PROCEDURE [
  name: LONG STRING, proc, help, unload: Exec.ExecProc,
  clientData: LONG POINTER]RETURNS [stop: BOOLEAN]];
```

The **EnumerateCommands** procedure enumerates the commands currently registered with the Executive. It calls the procedure **userProc** on the data for each command. **name** belongs to the Executive and should not be deallocated by the client. If **stop** is **TRUE**, the enumeration will halt.

```
Exec.FeedbackProc: PROCEDURE [h: Exec.Handle] RETURNS [proc: Format.StringProc];
```

FeedbackProc provides a way for clients to differentiate between feedback, which reports the current status during processing, and output, which can be thought of as the results of executing the command.

```
Exec.FreeTokenString: PROCEDURE [s: LONG STRING] RETURNS [NIL: LONG STRING];
```

The **FreeTokenString** procedure frees strings that were obtained via **Exec.GetToken**. It returns **NIL**.

```
Exec.GetChar: GetCharProc;
```

The **GetChar** procedure returns the next character from the command line. Note that the portion of the command line seen by a subsystem starts immediately after the name of the command. When the command line is exhausted, **GetChar** will return **Ascii.NUL**. (See also **EndOfCommandLine**).

```
Exec.GetNameandPassword: PROCEDURE [
  h: Handle name, password: LONG STRING, prompt: LONG STRING ← NIL];
```

The **GetNameandPassword** procedure prompts you for a name and password. If the prompt parameter is **NIL**, the name prompt is "User: ". If the prompt parameter is not **NIL**, it will be used as the name prompt.

```
Exec.GetToken: PROCEDURE [h: Exec.Handle] RETURNS [token, switches: LONG STRING];
```

The **GetToken** procedure obtains the next token and its switches from the command line; leading white space is skipped. A token is defined to be the contents of a quoted string (e.g., "This is a token") or the smallest sequence of characters containing no white-space characters (SP, TAB, or CR) and no slash character (/). If the character immediately

following the token is a slash, all characters up to the next white-space character or slash character are read as switches. Note that the token string or switches string may be NIL. The strings returned from this procedure should be freed by the client using FreeTokenString.

```
Exec.GetTTY: PROCEDURE [h: Exec.Handle] RETURNS [tty: TTY.Handle];
```

The functions provided by the Executive for interacting with the user (as opposed to interacting with the command line) are quite limited (Confirm and GetNameandPassword). Subsystems that require more extensive interaction with the user can obtain a TTY.Handle from the Executive with GetTTY. The procedures in Pilot's TTY interface can then be used with this Handle for interaction with the user. ReleaseTTY is used to free the TTY.Handle when the subsystem is finished with it. In general, a subsystem interacting heavily with users should create its own tool window instead of interacting in a TTY style.

```
Exec.Load: PROCEDURE [
  write: Format.StringProc, name: LONG STRING, codeLinks: BOOLEAN ← FALSE,
  RETURNS [handle: MLoader.Handle];
```

The Load procedure loads a program specified by name. write is used by the Load procedure for all its output to the user. codeLinks indicates whether code links should be used in loading. The handle returned by Load can be passed to the Start procedure to start the program.

```
Exec.Login: PROCEDURE [h: Exec.Handle, name, password: LONG STRING];
```

The Login procedure is equivalent to calling GetNameandPassword with a prompt of NIL.

```
Exec.LookUpCommand: PROCEDURE [command: LONG STRING] RETURNS [
  name: LONG STRING, proc, help, unload: Exec.ExecProc, didExpand: BOOLEAN],
  clientData: LONG POINTER];
```

The LookUpCommand procedure permits a client to look up a specific command. The didExpand result of LookupCommand indicates whether the command parameter was an exact match of name or whether it was a unique prefix of name and had to be expanded to match. name is owned by the Executive, and should neither be changed nor deallocated.

```
Exec.MatchPattern: PROCEDURE [string, pattern: LONG STRING]
  RETURNS [matched: BOOLEAN]
```

The MatchPattern procedure is provided for clients that need to match names against patterns containing * and #. * matches zero or more characters and # matches exactly one character.

```
Exec.OutputProc: PROCEDURE [h: Exec.Handle] RETURNS [proc: Format.StringProc]
```

The OutputProc procedure returns a Format.StringProc that can be used with the Format interface for output. This procedure directs output to the Executive that called the ExecProc.

```
Exec.PrependCommands: PROCEDURE [h: Exec.Handle, command: LONG STRING]
```

The **PrependCommands** procedure inserts the parameter **command** in the front of the command line. It will be executed as soon as the current command completes.

```
Exec.ProcessCommandLine: PROCEDURE [
    cmd: LONG STRING, write: Format.StringProc,
    checkAbort: Exec.CheckAbortProc] RETURNS[outcome: Outcome];
```

It is possible for a program to invoke the Executive facilities without having an **Exec.Handle**; that is, without being in the process of executing an **Exec.ExecProc**. (If it does have an **Exec.Handle**, it can invoke the facilities via **PrependCommands** or **AppendCommands** by calling **ProcessCommandLine**.) The subsystem must not only provide the command line to be executed but must also provide the output and **checkAbort** procedures that would normally be supplied by an Executive window.

```
Exec.PutChar: PROCEDURE [h: Exec.Handle, c: CHARACTER]
```

The **PutChar** procedure outputs a single character to the Executive.

```
Exec.ReleaseTTY: PROCEDURE [tty: TTY.Handle]
```

The **ReleaseTTY** procedure is used to free the **TTY.Handle** obtained via **GetTTY** when the subsystem is finished with it. If **tty** was not created by **GetTTY**, the procedure does nothing.

```
Exec.RemoveCommand: PROCEDURE [h: Exec.Handle, name: LONG STRING] ;
```

The **RemoveCommand** procedure removes a command from the list of commands registered with the Executive; it is used in conjunction with unloading a subsystem (see **UnloadCommand** and **AddCommand**). To successfully unload a particular subsystem, all commands registered by the module that implements the subsystem must be removed using **RemoveCommand**.

```
Exec.RenameCommand: PROCEDURE [
    old, new: LONG STRING] RETURNS[ok: BOOLEAN];
```

The **RenameCommand** provides a way for you to change the name of a command registered with the Executive.

```
Exec.Run: PROCEDURE [
    h: Token.Handle, write: Format.StringProc,
    checkAbort: PROCEDURE RETURNS[abort: BOOLEAN], codeLinks: BOOLEAN ← FALSE];
```

The **Run** procedure reads a command line by asking the **Token** facility for the next line in **h**. **Run** then runs the programs listed on the command line. **write** is the output procedure to be used to report to the client. **codeLinks** indicates whether **codeLinks** should be used in loading.

```
Exec.Start: PROCEDURE [handle: MLoader.Handle];
```

The **Start** procedure starts a program that has been loaded by **Load**.

```
Exec.Unload: PROCEDURE [handle: MLoader.Handle];
```

The **Unload** procedure unloads a program that has been loaded by **Load**.

```
Exec.UnloadCommand: PROCEDURE [
    h: Handle, name: LONG STRING] RETURNS[RemovedStatus];
```

UnloadCommand invokes the **unload** procedure associated **name**. If **name** has been changed using **RenameCommand** or **AliasCommand**, **UnloadCommand** finds the correct unload procedure, regardless of whether **name** represents the original command or an aliased/renamed command. Because of the way the Executive is monitored, the client's **unload** procedure may not contain calls to **AddCommand**, **AliasCommand**, **EnumerateCommands**, **LookupCommand**, or **RenameCommand**. (See also **AddCommand**.)

5.5 Examples

The following example registers the procedure **DoIt** under the command name **MyCommand**. The procedure takes a sequence of tokens with switches from the command line and processes them. It checks at regular intervals to see if you have aborted it. **Write** is used within the procedure for output. **DoIt** is an entry procedure that protects any global data it might use from being accessed by several concurrent calls on **DoIt**.

```
DoIt: ENTRY Exec.ExecProc =
    BEGIN
    name, switches: LONG STRING ← NIL;
    Write: Format.StringProc = Exec.OutputProc[h];
    outcome: Exec.Outcome ← normal;
    DO
        ENABLE UNWind => {
            name ← Exec.FreeTokenString[name];
            switches ← Exec.FreeTokenString[switches]};
        IF Exec.CheckForAbort[h] THEN {outcome ← abort; EXIT};
        [name, switches] ← Exec.GetToken[h];
        IF name = NIL AND switches = NIL THEN EXIT;
        --- perform function --
        name ← Exec.FreeTokenString[name];
        switches ← Exec.FreeTokenString[switches];
        ENDLOOP;
    RETURN[outcome];
    END;

Exec.AddCommand["MyCommand"L, DoIt];
```

The following is an example of how to define an **unload** procedure. The **DefaultUnloadProc** is not sufficient in this case: first, there is global cleanup to perform; second, the program registers more than one command with the Executive.

```
...

Test: PROGRAM =
  ...
  BEGIN
  message: LONG STRING;
  Test1: Exec.ExecProc =
    BEGIN
    ...
    END;

  Test2: Exec.ExecProc =
    BEGIN
    ...
    END;

  Test: Exec.ExecProc =
    BEGIN
    ...
    END;

  MyUnload: Exec.ExecProc =
    BEGIN
    Heap.systemZone.FREE[@message];
    -- Order of command removal doesn't matter
    Exec.RemoveCommand[h, "Test1"L];
    Exec.RemoveCommand[h, "Test2"L];
    Exec.RemoveCommand[h, "Test"L];
    END;

  message ← String.CopyToNewString["Output message: "L, Heap.systemZone];
  Exec.AddCommand[name: "Test1"L, proc: Test1, unload: MyUnload];
  Exec.AddCommand[name: "Test2"L, proc: Test2, unload: MyUnload];
  Exec.AddCommand[name: "Test"L, proc: Test, unload: MyUnload];
  END;
```

6

Expand

The **Expand** interface provides facilities for the Executive-style expansion of lines containing *, @, ↑ or '. Expansion is done within a local directory, if one is specified; otherwise, it is done within the current search path. The expansion characters to be used are specified in a mask and have the following meanings:

star	matches zero or more characters.
atSign	the following token is a file name and should be replaced by the contents of that file.
upArrow	ignore the up arrow character and the one immediately following it.
quote	do not treat the next character as an expansion character.

6.1 Types

```
Expand.AbortProcType: TYPE = PROCEDURE RETURNS [BOOLEAN];
```

To permit a client to abort the expansion, a procedure of type **AbortProcType** must be provided to the Expand package. It is called at intervals during an expansion.

```
Expand.ExpandQ: TYPE [1];
```

This is a private type, included for use by the Executive.

```
Expand.Mask: TYPE = RECORD [
  star, atSign, quote: BOOLEAN,
  upArrow: Expand.UpArrowAction],
  localDirectory: LONG STRING];
```

If **star**, **atSign**, or **quote** are **TRUE**, the procedures expand according to the description above. If **localDirectory** is specified, expansion is done only within that directory. If an incomplete directory name is provided (that is, if **localDirectory** does not begin with '<), it is assumed to be directly beneath the volume root directory.

```
Expand.UpArrowAction: TYPE = {skip, remove, none};
```

skip skips the up arrow and succeeding character but leaves them in the expanded string.

remove skips the up arrow and succeeding character and removes them from the expanded string.

none treats the up arrow as a regular character.

6.2 Constants and data objects

```
Expand.defaultMask: Expand.Mask = [
  star: TRUE, atSign: TRUE, quote: TRUE, upArrow: remove,
  localDirectory: NIL];
```

6.3 Signals and errors

```
Expand.UnknownCommandFile: SIGNAL [name: LONG STRING] RETURNS [LONG STRING];
```

A call to **Expand.ExpandString** raises the signal **UnknownCommandFile** if an @ is encountered and the corresponding command file cannot be found. **name** is the name of the missing file; the client can catch this signal and resume with a string containing the contents of the missing command file. See the example below.

6.4 Procedures

```
Expand.ExpandQueues: PROCEDURE [toQ, fromQ: Expand.ExpandQ, all: BOOLEAN ← FALSE,
isAborted: Expand.AbortProcType ← NIL, mask: Expand.Mask ← Expand.defaultMask];
```

The **ExpandQueues** procedure is a private procedure for use by the Executive.

```
Expand.ExpandString: PROCEDURE [cmdLine: LONG STRING,
  isAborted: Expand.AbortProcType ← NIL, mask: Mask ← Expand.defaultMask] RETURNS
  [LONG STRING];
```

The **ExpandString** procedure expands the command line according to its mask and return the expanded line. The string it returns is allocated from the system heap; it is the client's responsibility to free it. If **cmdLine** is **NIL**, then no actions are performed. If an unknown command file is encountered, the signal **UnknownCommandFile** is raised.

```
Expand.ExpandToTokens: PROCEDURE [cmdLine: LONG STRING, proc: PROCEDURE [LONG STRING]
RETURNS [BOOLEAN],
  isAborted: Expand.AbortProcType ← NIL, mask: Mask ← Expand.defaultMask];
```

The **ExpandToTokens** procedure expands **cmdLine** according to its mask, parses it into **Token.Items**, and calls the client's procedure **proc** once on each token until the command line is exhausted or **proc** returns **TRUE**. The client need not be concerned with allocation and deallocation of the **Token.Items** created by this procedure. If **cmdLine** is **NIL**, no actions are performed.

6.5 Example

The following is an example of a tool that runs in the Executive. It attempts to expand **commandLine**; if it encounters an unknown file, you are prompted to type the contents of the file. The contents of the file are then returned to the Expand package, which continues processing **commandLine**.

```
DIRECTORY
...
Example: PROGRAM IMPORTS ... =
  BEGIN
  ...
  MainBody: Exec.ExecProc = BEGIN
    ...
    GetCommandFileFromUser: PROCEDURE [ h: Exec.Handle, name: LONG STRING]
      RETURNS [result: LONG STRING] =
      BEGIN
      tty: TTY.Handle ← Exec.GetTTY [h];
      result ← Storage.String[100];
      TTY.PutCR[tty];
      TTY.PutString[tty, "File name """L];
      TTY.PutString[tty, name];
      TTY.PutString[
        tty, """ unknown. Type what it would contain."L];
      TTY.PutCR[tty];
      TTY.GetLine[tty, result];
      Exec.ReleaseTTY[tty];
      END;
    ...
    Expand.ExpandString[commandLine, abortProc, mask !
      Expand.UnknownCommandFile = >
        RESUME[GetCommandFileFromUser[h, name]];
    ...
    END;

  -- mainline
  ...
  Exec.AddCommand["Example.~", MainBody];
  ...
  END.
```

7

HeraldWindow

The HeraldWindow interface provides two functions to the client: feedback and booting. It also allows the client to access some of the global state maintained by the tool that implements the **HeraldWindow** interface.

7.1 Types

```
HeraldWindow.ConfirmProcType: TYPE = PROCEDURE [
  post: Format.StringProc, cleanup: BOOLEAN ← TRUE] RETURNS [okay: BOOLEAN];
```

Feedback and confirmation of booting are provided through a **Format.StringProc** and **ConfirmProcType**. If **cleanup** is **TRUE**, the Supervisor notifies subsystems of the event.

```
HeraldWindow.CursorState: TYPE = {invert, negative, positive};
```

```
HeraldWindow.Slot: TYPE = LONG POINTER TO HeraldWindow.SlotObject;
```

```
HeraldWindow.SlotObject: TYPE = ...;
```

Multiple cursor-sized feedback regions are supported in the HeraldWindow.

7.2 Constants and data objects

```
HeraldWindow.displayedPages: READONLY LONG CARDINAL;
```

While the Herald Window is not inactive, **displayedPages** contains the number of free pages on the system volume.

```
HeraldWindow.switches: READONLY System.Switches;
```

switches contains the current booting switches, to be used as the default switches by booting commands unless explicitly overwritten.

```
HeraldWindow.window: READONLY Window.Handle;
```

window is the handle for the HeraldWindow's window.

7.3 Signals and errors

```
HeraldWindow.InvalidSwitches: SIGNAL;
```

InvalidSwitches is raised by **ScanSwitches** if the \ character has been used with an invalid following character. It can be resumed to ignore the illegal characters.

7.4 Procedures

```
HeraldWindow.AlwaysConfirm: HeraldWindow.ConfirmProcType;
```

The **AlwaysConfirm** procedure does not wait for confirmation but simply notifies subsystems that booting is about to take place (the parameter **cleanup** is defaulted to **TRUE**).

```
HeraldWindow.AppendBrokenMessage: PROCEDURE [
  msg1, msg2, msg3: LONG STRING ← NIL, newLine, clearOld: BOOLEAN ← TRUE];
```

The **AppendBrokenMessage** procedure provides a mechanism for client programs to provide textual feedback in a standard location to the user. The HeraldWindow has room for two lines of text; old messages are automatically erased after 30 seconds and new ones are placed in a queue. If **newLine** is **TRUE**, this message starts a new line on the display. If **clearOld** is **TRUE**, all old messages are deleted. **AppendBrokenMessage** permits the display of messages that are a combination of several strings. (See also **AppendMessage**.)

```
HeraldWindow.AppendLogicalVolumeName: PROCEDURE [
  s: LONG STRING, id: Volume.ID ← Volume.systemID];
```

The **AppendLogicalVolumeName** procedure appends the name of the logical volume **id** onto the client-owned string **s**. If **s** is not large enough, **String.StringBoundsFault** is raised.

```
HeraldWindow.AppendMessage: PROCEDURE [
  msg: LONG STRING ← NIL, newLine, clearOld: BOOLEAN ← TRUE];
```

The **AppendMessage** procedure is just like **AppendBrokenMessage** except that it accepts only a single string parameter.

```
HeraldWindow.AppendPhysicalVolumeName: PROCEDURE [s: LONG STRING];
```

The **AppendLogicalVolumeName** procedure appends the name of the physical volume onto the client-owned string **s**. If **s** is not large enough, **String.StringBoundsFault** is raised.

```
HeraldWindow.AppendSwitches: PROCEDURE [s: LONG STRING];
```

The **AppendSwitches** procedure appends the current booting switches to the client-owned string **s**. If **s** is not large enough, **String.StringBoundsFault** is raised.

```
HeraldWindow.BootFromFile: PROCEDURE [
  name: LONG STRING, bootSwitches: System.Switches ← switches,
```

```
  postProc: Format.StringProc ← DefaultPost,
  confirmProc: HeraldWindow.ConfirmProcType ← HeraldWindow.DefaultConfirm];
```

The **BootFromFile** procedure boots a file in the local directory (appending the extension ".boot" if necessary). **bootSwitches** are the Pilot switches to be used when booting. The procedure scans the string **name** for any switches. These optional switches appear after the file name, separated from it by a slash (/). They obey escape procedures described in the discussion of **ScanSwitches** and are used in preference to the **bootSwitches** parameter. **confirmProc** is called to confirm that the boot should really be performed. This procedure should always be called from within the Notifier process. The file will be locked for read access if **confirmProc** returns **FALSE**.

```
HeraldWindow.BootFromVolumeID: PROCEDURE [
  id: Volume.ID, bootSwitches: System.Switches ← switches,
  postProc: Format.StringProc ← HeraldWindow.DefaultPost,
  confirmProc: HeraldWindow.ConfirmProcType ← HeraldWindow.DefaultConfirm];
```

The **BootFromVolumeID** procedure boots the logical volume specified by **id**. **bootSwitches** are the Pilot switches to be used when booting. **confirmProc** is called to confirm that the boot should really be performed. This procedure should always be called from within the Notifier process.

```
HeraldWindow.BootFromVolumeName: PROCEDURE [
  name: LONG STRING, bootSwitches: System.Switches ← switches,
  postProc: Format.StringProc ← HeraldWindow.DefaultPost,
  confirmProc: HeraldWindow.ConfirmProcType ← HeraldWindow.DefaultConfirm];
```

The **BootFromVolumeName** procedure boots the logical volume specified by **name**. **bootSwitches** are the Pilot switches to be used when booting. The procedure scans the string **name** for any switches. These optional switches appear after the file name, separated from it by a slash (/). They obey escape procedures described in the discussion of **ScanSwitches** and are used in preference to the **bootSwitches** parameter. **confirmProc** is called to confirm that the boot should really be performed. This procedure should always be called from within the Notifier process. If **name** does not match a logical volume name, the system volume is booted with no switches.

```
HeraldWindow.DefaultConfirm: HeraldWindow.ConfirmProcType;
```

The **DefaultConfirm** procedure waits for you to confirm the boot by waiting for confirmation with **POINT** or denial with **EXTEND**, while displaying a **mouseRed** cursor (see the **Cursor** interface). If you confirm the boot, the Supervisor notifies subsystems of the event (**cleanup** is **TRUE**).

```
HeraldWindow.DefaultPost: Format.StringProc;
```

The **DefaultPost** procedure sends output to whichever window is taking indirect type-out. If there is no such window, the output is discarded.

```
HeraldWindow.FreeCursorSlot: PROCEDURE [
  slot: HeraldWindow.Slot] RETURNS [nil: HeraldWindow.Slot]
```

The **FreeCursorSlot** procedure frees one of the cursor slots allocated by the HeraldWindow.

```
HeraldWindow.GetCursorSlot: PROCEDURE RETURNS [slot: HeraldWindow.Slot]
```

The **GetCursorSlot** procedure allocates a cursor slot in the HeraldWindow. If it cannot find a slot, **NIL** is returned.

```
HeraldWindow.ScanSwitches: PROCEDURE [s: LONG STRING, defaultSwitches: System.Switches ←
System.defaultSwitches] RETURNS [switches: System.Switches]
```

The **ScanSwitches** procedure returns the **defaultSwitches**, modified by the switches in the strings. The scanner recognizes the following syntax: The characters ~ and - change the sense of the following switch. Each character of the string is the character representation of the switch. **ScanSwitches** supports a slightly expanded version of the Mesa compiler escape convention, with \ as the escape character:

Code	Interpretation	
\n, \N, \r, \R	Ascii.CR	
\t, \T	Ascii.TAB	
\b, \B	Ascii.BS	
\f, \F	Ascii.FF	
\l, \L	Ascii.LF	-- *note that* \n *is* LF *in C.*
ddd	*ddd*C	-- *where d is an octal digit, ddd ≤ 377B*
\\	\	
\'	'	
\"	"	
\~	~	-- *not recognized by the Compiler*
\-	-	-- *not recognized by the Compiler*

Any other character following \ causes the signal **InvalidSwitches** to be raised. This signal can be resumed to ignore the switch character.

```
HeraldWindow.SetCursor: PROCEDURE [slot: HeraldWindow.Slot, cursor: Cursor.Defined];
```

The **SetCursor** procedure displays a cursor at a previously acquired cursor slot. The cursor is one of those that are predefined by the **Cursor** interface.

```
HeraldWindow.SetCursorState: PROCEDURE [
  slot: HeraldWindow.Slot, state: HeraldWindow.CursorState];
```

The **SetCursorState** procedure modifies (e.g., inverts) the display state of the indicated cursor.

```
HeraldWindow.SetSwitches: PROCEDURE [new: System.Switches];
```

The **SetSwitches** procedure changes the switches used during booting.

```
HeraldWindow.StoreCursor: PROCEDURE [
  slot: HeraldWindow.Slot, cursor: LONG POINTER TO UserTerminal.CursorArray];
```

The **StoreCursor** procedure displays a cursor at a previously acquired cursor slot.

8

Profile

Profile provides an interface to a number of commonly accessed user and system data items. All these items are read-only. Changes to the variables defined below are monitored by the Pilot Supervisor notification facility. See **Events** and **EventTypes** for more discussion of the Supervisor.

This interface supports non-product protocols for Pup-based file servers and Grapevine registries. Support for these protocols will be removed in a future release. Clients are encouraged to remove dependencies on these protocols.

8.1 Types

Profile.BalanceBeamChoice: TYPE = {never, notForCharacter, always};

BalanceBeamChoice	determines where the insertion point is placed when a selection is made.
never	the insertion point is always at the end of the selection.
notForCharacter	the insertion point is always at the end of a character selection but uses a balance beam algorithm for word or line selections.
always	the balance beam algorithm is always used.

Profile.FileServerProtocol: TYPE = {PUP, ns};

FileServerProtocol	determines the type of protocol used to communicate with file servers. Support for the Pup file server protocol will be removed in a future release.
PUP	communicates with Pup-based servers
ns	communicates with the product-based Network Services.

```
Profile.Place: TYPE = MACHINE DEPENDENT {
  unknown(0), tajo, copilot, last(LAST[CARDINAL])};
```

Place distinguishes between Tajo, CoPilot, or some other boot file. Clients may depend on particular facilities in Tajo or CoPilot.

```
Profile.Qualification: TYPE = {registry, clearinghouse, none};
```

Qualification is a parameter to **Profile.Qualify**. An unqualified token is qualified by appending the qualifing name(s) to the token, separated by the necessary punctuation. Note: **registry** qualification appends a "." followed by the registry; e.g., Jones.PA. **clearinghouse** qualification appends the domain and organization using ":" as the punctuation; e.g., Jones:OSBU North:Xerox.

```
Profile.String: TYPE = LONG STRING;
```

String is the type of all string variables. It will be changed to **LONG POINTER TO READONLY StringBody** when other definitions can be changed as well.

8.2 Constants and data objects

```
Profile.balanceBeamChoice: READONLY BalanceBeamChoice;
```

balanceBeamChoice is the current setting of the balance beam algorithm. Changes to this variable notify the subsystem **Event.tajoDefaults** and the event **EventTypes.debugging**.

```
Profile.debugging: READONLY BOOLEAN;
```

Used internally by Tajo to decide whether to attempt error recovery or call the debugger. If **debugging** is **TRUE**, the debugger will be called. If Tajo invokes the debugger, it may not be possible to continue the session. Changes to this variable are monitored by subsystem **Event.tajoDefaults** and the event **EventTypes.debugging**.

```
Profile.defaultFileServerProtocol: READONLY FileServerProtocol;
```

defaultFileServerProtocol is the default file server protocol. Changes to this variable notify the subsystem **Event.tajoDefaults** and the event **EventTypes.FileServerProtocol**.

```
Profile.initialToolStateDefault: READONLY ToolWindow.State;
```

This is the state in which a tool is created if it does not override the default provided in the **Tool.Create** call.

```
Profile.noChange: LONG STRING = LOOPHOLE[LAST[LONG CARDINAL]];
```

This is the default string value used in **Profile** procedures to indicate that a string variable should not be changed.

```
Profile.place: READONLY Profile.Place;
```

This is the type of boot file running (e.g , Tajo or CoPilot).

```
Profile.swapCtrlAndCommand: READONLY BOOLEAN;
```

swapCtrlAndCommand is TRUE if the mapping of the CONTROL key and COMMAND key should be swapped.

8.3 Signals and errors

None.

8.4 Procedures

```
Profile.GetDefaultDomain: PROCEDURE [PROCEDURE [String]];
```

The **GetDefaultDomain** procedure calls the procedure parameter with the default Clearinghouse domain. The call is made from with the Profile machinery's monitor lock.

```
Profile.GetDefaultOrganization: PROCEDURE [PROCEDURE [String]];
```

The **GetDefaultOrganization** procedure calls the procedure parameter with the default Clearinghouse organization. The call is made from within the Profile machinery's monitor lock.

```
Profile.GetDefaultRegistry: PROCEDURE [PROCEDURE [String]];
```

The **GetDefaultRegistry** procedure calls the procedure parameter with the default Grapevine registry. The call is made from within the Profile machinery's monitor lock. Support for Grapevine will be removed in a future release.

```
Profile.GetID: PROCEDURE [
  flavor, Auth.Flavor ← simple, proc: PROCEDURE [id: Auth.IdentityHandle]];
```

The **GetID** procedure calls the procedure parameter with the user identity corresponding to the current user name and password and having the specified authentication flavor. **id** is not authenticated. The call is made from within the Profile machinery's monitor lock.

```
Profile.GetLibrarian: PROCEDURE [PROCEDURE [String]];
```

The **GetLibrarian** procedure calls the procedure parameter with the name of the default librarian server used in librarian transactions. The call is made from within the Profile machinery's monitor lock.

```
Profile.GetLibrarianNames: PROCEDURE [PROCEDURE [prefix, suffix: String]];
```

The **GetLibrarianNames** procedure calls the procedure parameter with the default name prefix and suffix to be used when nameing libjects. The call is made from within the Profile machinery's monitor lock.

```
Profile.GetUser: PROCEDURE [
  proc: PROCEDURE [name, password: String], qualification: Qualification ← none];
```

The **GetUser** procedure calls the procedure parameter with the user name and password. The call is made from within the Profile machinery's monitor lock. If the current user

name is already qualified with an appropriate qualification, it is not changed. Otherwise, any qualification is stripped from the token. **Note:** If qualification is **registry**, the Grapevine registry is used to qualify the name. If qualification is **clearinghouse**, the Clearinghouse domain and organization are used to qualify the name.

```
Profile.Qualify: PROCEDURE [
  token, newToken: LONG STRING, qualification: Profile.Qualification];
```

The **Qualify** procedure produces the requested qualification for a token. If the token is already qualified with an appropriate qualification, it is not changed. Otherwise, any qualification is stripped from the token. **Note:** If qualification is **registry**, the Grapevine registry is used to qualify the token. If qualification is **clearinghouse**, the Clearinghouse domain and organization are used to qualify the token. **newToken** contains the qualified token. This procedure may raise **String.StringBoundsFault** if **newToken** is not long enough.

```
Profile.SetBalanceBeamChoice: PROCEDURE [BalanceBeamChoice];
```

The **SetBalanceBeamChoice** procedure changes the variable **Profile.balanceBeamChoice**.

```
Profile.SetDebugging: PROCEDURE [BOOLEAN];
```

The **SetDebugging** procedure changes the variable **Profile.debugging**. This procedure notifies the subsystem **Event.tajoDefaults** with the event **EventTypes.debugging**.

```
Profile.SetDefaultDomain: PROCEDURE [domain: String];
```

The **SetDefaultDomain** procedure changes the default Clearinghouse domain. The parameter string is copied. This procedure notifies the subsystem **Event.tajoDefaults** with the event **EventTypes.domain**.

```
Profile.SetDefaultOrganization: PROCEDURE [organization: String];
```

The **SetDefaultOrganization** procedure changes the default Clearinghouse organizataion. The parameter string is copied. This procedure notifies the subsystem **Event.tajoDefaults** with the event **EventTypes.organization**.

```
Profile.SetDefaultRegistry: PROCEDURE [registry: LONG STRING];
```

The **SetDefaultRegistry** procedure changes the default Grapevine registry. The parameter string is copied. This procedure notifies the subsystem **Event.tajoDefaults** with the event **EventTypes.registry**.

```
Profile.SetFileServerProtocol: PROCEDURE [FileServerProtocol];
```

The **SetFileServerProtocol** procedure changes the variable **Profile.defaultFileServerProtocol**. This procedure notifies the subsystem **Event.tajoDefaults** with the event **EventTypes.fileServerProtocol**.

```
Profile.SetLibrarian: PROCEDURE [
  name, prefix, suffix: LONG STRING ← Profile.noChange];
```

The **SetLibrarian** procedure changes the default librarian name prefix, the default librarian name suffix and the default librarian server name. Parameters that are defaulted are not changed. The parameter strings are copied. This procedure notifies the subsystem **Event.tajoDefaults** with the event **EventTypes.librarian**.

```
Profile.SetSwapCtrlAndCommand: PROCEDURE [BOOLEAN];
```

The **SetSwapCtrlAndCommand** procedure sets the variable **Profile.swapCtrlAndCommand**.

```
Profile.SetUser: PROCEDURE [name, password: String ← Profile.noChange];
```

The **SetUser** procedure changes the user name and password. Parameters that are defaulted are not changed. The parameter strings are copied. This procedure notifies the subsystem **Event.primaryCredentials** with the event **EventTypes.primaryCredentials**.

9

Token

The Token interface provides general scanning and simple parsing facilities for any source of characters. The interface supports client-defined filters; some standard token filters are also provided.

9.1 Types

```
Token.FilterProcType: TYPE = PROCEDURE [
  c: CHARACTER, data: Token.FilterState] RETURNS [inClass: BOOLEAN];
```

A **FilterProcType** is the mechanism by which a client defines a class of tokens. Procedures that use filters call them once for each candidate character. Instance data permits the client to maintain the state of the parse. If a client tries to access instance data but none was passed in, the signal **NilData** should be raised. The **FilterProcType** returns a boolean indicating whether the character is part of the token.

```
Token.FilterState: TYPE = LONG POINTER TO StandardFilterState;
```

A **FilterState** is a **LONG POINTER** to client instance data that is passed to a client's **FilterProcType** procedure. A client may **LOOPHOLE** the **FilterState** to a more convenient type. The system-provided filters that require a non-**NIL** **FilterState** (such as **Delimited**) use the first two words of data.

```
Token.GetCharProcType: TYPE = PROCEDURE [
  h: Token.Handle] RETURNS [c: CHARACTER];
```

A **GetCharProcType** provides a stream of characters to be parsed. When a **GetCharProcType** procedure returns **Ascii.NUL**, the **Token** package assumes that the source has been exhausted. The **Handle** is passed into the **GetCharProcType** so that a client can hide instance data in its object. Although there is not an instance data field in **Object**, the client could **LOOPHOLE** a pointer to a larger record that contained its data.

```
Token.Handle: TYPE = LONG POINTER TO Token.Object;
```

```
Token.NetFormat: TYPE = Format.NetFormat;
```

```
Token.Object: TYPE = MACHINE DEPENDENT RECORD [
  getChar(0): Token.GetCharProcType, break(1): CHARACTER ← Ascii.NUL];
```

The **Object** encapsulates the source of characters to be parsed. The **Token** package uses the **getChar** field of the **Handle** to obtain the stream of characters. It assumes that the source has been exhausted when **getChar** returns **Ascii.NUL**. **Token** uses the break field to record the final character that it reads. It records the final character because there is no way to put back a character into the character source. It must read one character beyond the token it is parsing to ensure that it has reached the end. If it simply returned the token, this character would be lost. Since the **Token** package stores the last character in the **Object**, that character is available to the client. The client can ignore it, inspect it to decide what to parse next, or put it back into the character source. Note that when a client attempts to parse past the end of the input, the break character contains **Ascii.NUL**.

```
Token.QuoteProcType: TYPE = PROCEDURE [
  c: CHARACTER] RETURNS [closing: CHARACTER];
```

The **QuoteProcType** is used to recognize quoted tokens. If **c** is a quote character recognized by the **QuoteProcType**, **closing** is the matching character that closes the quotation. If **closing** is **Token.nonQuote**, **c** was not a quote character.

```
Token.SkipMode: TYPE = {none, whiteSpace, nonToken};
```

The **SkipMode** controls what characters a procedure will skip before collecting a token.

none no characters should be skipped and the token should start with the next character.

whiteSpace white-space characters (space, carriage return, and tab) should be skipped before collecting the token.

nonToken any characters that are not legal token characters should be skipped before collecting the token.

```
Token.StandardFilterState: TYPE = ARRAY [0..2) OF UNSPECIFIED;
```

The **StandardFilterState** is client data that is passed to a client's **FilterProcType** procedure. A client that uses instance data can use a **StandardFilterState** for storing two words of state data.

9.2 Constants and data objects

```
Token.nonQuote: CHARACTER = ...;
```

The **nonQuote** character is returned from a **QuoteProcType** to indicate that the character passed to it is not a quote character.

9.3 Signals and errors

```
Token.NilData: SIGNAL;
```

Procedures that take a **FilterProcType** argument also take an argument that is a pointer to client instance data. If the client has no need for instance data, it can pass a **NIL** as the instance data pointer. If a **FilterProcType** attempts to access the client instance data, but the client passed in **NIL** instead of a pointer to instance data, the signal **NilData** should be raised. Implementors of **FilterProcTypes** are strongly encouraged to check for **NIL** and raise this condition if they use client instance data.

```
Token.SyntaxError: SIGNAL [s: LONG STRING];
```

The resumable **SIGNAL SyntaxError** can be raised if incorrect syntax is encountered by **Boolean**, **Decimal**, **HostNumber**, **LongNumber**, **LongDecimal**, **NetworkAddress**, **NetworkNumber**, **Octal**, or **SocketNumber**. In each case, resuming the signal causes the procedure to return a default value (described in the discussion of the various procedures).

```
Token.UnterminatedQuote:SIGNAL
```

The resumeable **SIGNAL UnterminatedQuote** is raised from **MaybeQuoted** if the **getChar** procedure of the **Handle** returns **Ascii.NUL** before the terminating quote character has been read. If the signal is resumed, **MaybeQuoted** will return as if it had read a closing-quote character.

9.4 Procedures

```
Token.Alphabetic: Token.FilterProcType;
```

Alphabetic can be used to collect tokens composed of alphabetic characters; that is, the characters **'a** through **'z** and **'A** through **'Z**. This procedure requires no client data (**data** may be **NIL**).

```
Token.AlphaNumeric: Token.FilterProcType;
```

AlphaNumeric can be used to collect tokens composed of alphanumeric characters; that is, the characters **'a** through **'z**, **'A** through **'Z**, and **'0** through **'9**. This procedure requires no client data (**data** may be **NIL**).

```
Token.Boolean: PROCEDURE [
  h: Token.Handle, signalOnError: BOOLEAN ← TRUE] RETURNS [true: BOOLEAN];
```

The **Boolean** procedure parses the next characters of the source as a boolean constant. Valid Boolean values are "TRUE" or "FALSE," but unlike the Mesa language, case does not matter ("true" and "false" are also acceptable). In case of a syntax error, the signal **SyntaxError** is optionally raised. If **signalOnError** is **FALSE**, or **SyntaxError** is resumed, then **FALSE** is returned for a syntax error. This procedure skips leading white space.

```
Token.Brackets: Token.QuoteProcType;
```

Brackets recognizes the following sets of matching open/close-quote pairs: ( ), [ ], { }, and < >.

```
Token.Decimal: PROCEDURE [
  h: Token.Handle, signalOnError: BOOLEAN ← TRUE] RETURNS [i: INTEGER];
```

The **Decimal** procedure parses the next characters of the source as a decimal constant. Decimals have the format as specified in the *Mesa Language Manual*. In case of a syntax error, the signal **SyntaxError** is optionally raised. If **signalOnError** is **FALSE** or **SyntaxError** is resumed, then zero is returned for a syntax error. This procedure skips leading white space.

```
Token.Delimited: Token.FilterProcType;
```

When **Delimited** is passed to a procedure such as **Filtered**, the value of **skip** passed along with it must be **nonToken**. It skips leading white space, then defines the first character of the token to be both the opening-quote character and the closing-quote character, returning all characters occurring between the first and second appearance of that character. As an example, **Delimited** would return the token "XXX" from either of the following input strings: " YXXXY" and "/XXX/". **Delimited** requires a non-**NIL data**.

```
Token.FileName: Token.FilterProcType;
```

The **FileName FilterProcType** can be used to collect tokens composed of file name characters; that is, '[, '], '<, '>, '*, '!, ';, '#, '-, '., '$, '+, or **AlphaNumeric** characters. Note that the filter does not guarantee that the token forms a valid file name, only that the token contains only these characters. This procedure requires no client data (**data** may be **NIL**).

```
Token.Filtered: PROCEDURE [
  h: Token.Handle, data: Token.FilterState, filter: Token.FilterProcType, skip:
  Token.SkipMode ← whiteSpace, temporary: BOOLEAN ← TRUE]
  RETURNS [value: LONG STRING];
```

The **Filtered** procedure collects the token string defined by the client's filter. If the client-instance data parameter **data** is not **NIL**, the first two words of **data** are set to zero before any calls are made to **filter**. **filter** is called with **data** once on each character until it returns **FALSE**. The string returned, which may be **NIL**, must be freed by calling **FreeTokenString**. Leading characters are skipped according to the value of **skip**. If **temporary** is **TRUE**, it is assumed that the string will be freed shortly and no effort is made to use the minimum storage for it. If **temporary** is **FALSE**, the minimum amount of storage is used. **filter** may raise **NilData**.

```
Token.FreeStringHandle: PROCEDURE [h: Token.Handle] RETURNS [nil: Token.Handle];
```

The **FreeStringHandle** procedure destroys a **Token.Handle** created by **StringToHandle**. It does not destroy the underlying string. It returns **NIL**.

```
Token.FreeTokenString: PROCEDURE [s: LONG STRING] RETURNS [nil: LONG STRING ← NIL];
```

The **FreeTokenString** procedure frees a string allocated by **Token**. It returns **NIL**. All such strings are allocated from the system heap.

```
Token.HostNumber: PROCEDURE [
  h: Token.Handle, format: NetFormat ← octal, signalOnError: BOOLEAN ← TRUE]
  RETURNS [host: System.HostNumber];
```

The **HostNumber** procedure parses the next characters of the source as a host number in format **format**. See the **Format** interface for a description of host numbers. In case of a syntax error, the signal **SyntaxError** is optionally raised. If **signalOnError** is **FALSE**, or **SyntaxError** is resumed, then **System.nullHostNumber** is returned for a syntax error. This procedure skips leading white space.

```
Token.Item: PROCEDURE [
  h: Token.Handle, temporary: BOOLEAN ← TRUE] RETURNS [value: LONG STRING];
```

The **Item** procedure returns the next token delimited by white space. Leading white space is skipped and the characters are collected until another white-space character is encountered. The string returned must be freed by calling **FreeTokenString**. If **temporary** is **TRUE**, it is assumed that the string will be freed shortly and no effort is made to use the minimum storage for it. If **temporary** is **FALSE**, only as much storage is used for the string as needed.

```
Token.Line: Token.FilterProcType;
```

The **Line FilterProcType** can be used to collect a line. It collects characters until it encounters a carriage return. This procedure requires no client data (**data** may be **NIL**).

```
Token.LongNumber: PROCEDURE [
  h: Token.Handle, radix: CARDINAL, signalOnError: BOOLEAN ← TRUE]
  RETURNS [u: LONG UNSPECIFIED];
```

The **LongNumber** procedure parses the next characters of the source as a long number in radix **radix**. Numbers have the format specified in the *Mesa Language Manual*. In case of a syntax error, the signal **SyntaxError** is optionally raised. If **signalOnError** is **FALSE** or **SyntaxError** is resumed, then zero is returned for a syntax error. This procedure skips leading white space.

```
Token.LongDecimal: PROCEDURE [
  h: Token.Handle, signalOnError: BOOLEAN ← TRUE] RETURNS [i: LONG INTEGER];
```

LongDecimal is just like **LongNumber**, but with a **radix** of 10.

```
Token.LongOctal: PROCEDURE [
  h: Token.Handle, signalOnError: BOOLEAN ← TRUE] RETURNS [c: LONG CARDINAL];
```

LongOctal is just like **LongNumber**, but with a **radix** of 8.

```
Token.MaybeQuoted: PROCEDURE [
  h: Token.Handle, data: Token.FilterState, filter: Token.FilterProcType ←
  Token.NonWhiteSpace, isQuote: Token.QuoteprocType ← Token.Quote, skip:
  Token.SkipMode ← whiteSpace, temporary: BOOLEAN ← TRUE];
```

The **MaybeQuoted** procedure permits the client to scan for one of two kinds of token. The first candidate character is passed to **isQuote**, which either returns **Token.nonQuote** or the closing-quote character. If a closing-quote character other than **Token.nonQuote** is returned, characters are collected in the token until the closing quote is encountered. If the input is exhausted before the closing quote is encountered, the signal **UnterminatedQuote** is raised. If it is resumed, **MayBeQuoted** returns the token collected up to that point. The closing-quote character may be included in the token by including two instances of the character in the input; that is, if **MaybeQuoted** encounters two closing-quote characters in a row, it will insert one closing-quote character in the token rather than terminating the token on the first closing quote. The outer quote characters are not part of the token and are discarded. If **Token.nonQuote** is returned from the **isQuote** procedure, the filter is used to collect characters the same way as in **Filtered: filter** is called with the client-instance data parameter **data** once on each character until it returns **FALSE**. In either case (quoted or filtered), the break character returned in the **Handle** will be the character following the token.

Leading characters are skipped according to the value of **skip**.

If **temporary** is **TRUE**, it is assumed that the string will be freed shortly and no effort is made to use the minimum storage for it. If **temporary** is **FALSE**, only as much storage is used for the string as is needed. The string returned must be freed by calling **FreeTokenString**.

```
Token.NetworkAddress: PROCEDURE [
  h: Token.Handle, format: NetFormat ← octal, signalOnError: BOOLEAN ← TRUE]
  RETURNS [address: System.NetworkAddress];
```

The **NetworkAddress** procedure parses the next characters of the source as a network address in format **format**. (See the **Format** interface for a description of network addresses.) In case of a syntax error, the signal **SyntaxError** is optionally raised. If **signalOnError** is **FALSE** or **SyntaxError** is resumed, then **System.nullNetworkAddress** is returned for a syntax error. This procedure skips leading white space.

```
Token.NetworkNumber: PROCEDURE [
  h: Token.Handle, format: NetFormat ← octal, signalOnError: BOOLEAN ← TRUE]
  RETURNS [networkNumber: System.NetworkNumber];
```

The **NetworkNumber** procedure parses the next characters of the source as a network number in format **format**. (See the **Format** interface for a description of network numbers.) In case of a syntax error, the signal **SyntaxError** is optionally raised. If **signalOnError** is **FALSE** or **SyntaxError** is resumed, then **System.nullNetworkNumber** is returned for a syntax error. This procedure skips leading white space.

```
Token.NonWhiteSpace: FilterProcType;
```

The **NonWhiteSpace FilterProc** defines all characters that are not white space; that is, **WhiteSpace[char]** = ~**NonWhiteSpace[char]**. This procedure requires no client data (**data** may be **NIL**).

```
Token.Number: PROCEDURE [
  h: Token.Handle, radix: CARDINAL, signalOnError: BOOLEAN ← TRUE]
  RETURNS [u: UNSPECIFIED];
```

The **Number** procedure parses the next characters of the source as a number in radix **radix**. Numbers have the format specified in the *Mesa Language Manual*. In case of a syntax error, the signal **SyntaxError** is optionally raised. If **signalOnError** is FALSE or **SyntaxError** is resumed, then zero is returned for a syntax error. This procedure skips leading white space.

```
Token.Numeric: Token.FilterProcType;
```

The **Numeric** FilterProcType can be used to collect a composed of digits; that is, the characters '0 through '9. This procedure requires no client data (**data** may be NIL).

```
Token.Octal: PROCEDURE [
  h: Token.Handle, signalOnError: BOOLEAN ← TRUE] RETURNS [c: CARDINAL];
```

Octal is just like **Number**, but with **radix** = 8.

```
Token.Quote: Token.QuoteProcType;
```

The **Quote QuoteProcType** recognizes the single quote and double quote as quotation characters and looks for another instance of the open-quote character to close the quotation.

```
Token.Skip: PROCEDURE [
  h: Token.Handle, data: Token.FilterState, filter: Token.FilterProcType,
  skipInClass: BOOLEAN ← TRUE];
```

The **Skip** procedure is used to skip over characters. A filter is provided to define the class of characters, and the boolean **skipInClass** indicates whether the characters to be skipped are those accepted or rejected by the filter. If the client-instance data parameter **data** is not NIL, the first two words of **data** are set to zero before any calls are made to **filter**. If **data** is NIL and **filter** references data, the signal **NilData** should be raised.

```
Token.SocketNumber: PROCEDURE [
  h: Token.Handle, format: NetFormat ← octal, signalOnError: BOOLEAN ← TRUE]
  RETURNS [socketNumber: System.SocketNumber];
```

The **SocketNumber** procedure parses the next characters of the source as a socket number in format **format**. (See the **Format** interface for a description of socket numbers.) In case of a syntax error, the signal **SyntaxError** is optionally raised. If **signalOnError** is FALSE, or **SyntaxError** is resumed, then **System.nullSocketNumber** is returned for a syntax error. This procedure skips leading white space.

```
Token.StringToHandle: PROCEDURE [s: LONG STRING, offset: CARDINAL ← 0]
  RETURNS [h: Token.Handle];
```

The **StringToHandle** procedure creates a **Token.Handle** whose source is a string. **offset** is the index into the string that marks the beginning of the characters to be parsed. The string is not copied, so clients are responsible for synchronizing access to the string with the **Token** package.

```
Token.Switches: Token.FilterProcType;
```

The **Switches FilterProcType** can be used to collect switch characters. It accepts the characters '~, '-, and **AlphaNumeric** characters. This procedure requires no client data (**data** may be **NIL**).

```
Token.WhiteSpace: Token.FilterProcType;
```

The **WhiteSpace FilterProcType** defines the white-space characters. This filter is used by **Token** for skipping white space. This procedure requires no client data (**data** may be **NIL**).

```
Token.WindowBox: PROCEDURE [h: Token.Handle] RETURNS [Window.Box];
```

The **WindowBox** procedure parses the next data in the **Handle** as a window box and returns the corresponding **Window.Box**. The syntax of the entry for a window boxes is as follows:

```
WindowBox: [x: number, y: number, w: number, h: number]
```

White space is ignored and the keywords **x**, **y** ,**w** and **h** may appear in any order or case. It is not necessary to have all four values present. If a value is to be omitted, its keyword must also be omitted. The result is initialized to **Window.NullBox**, so omitted values remain unchanged from this initialization. The values for the numbers refer to absolute screen coordinates and should obey the syntax for **Token.Decimal**. If an invalid **Token.Handle** is supplied, the results are undefined.

9.5 Discussion and examples

An example of the **Token** interface in parsing `User.cm` entries can be found at the end of the **CmFile** chapter.

The following example demonstrates how the **Token** interface could be used to parse command line input into "tokens," optionally followed by switches. In this context, tokens and switches are defined to be any sequence of non-white-space characters not including the slash character (/).

```
GetToken: PROCEDURE [h: Exec.Handle] RETURNS [token, switches: LONG STRING] =
  BEGIN
  get: PROCEDURE [Token.Handle] RETURNS [c: CHARACTER] = {
    RETURN[Exec.GetChar[h]]};
  getToken: Token.Object ← [getChar: get, break: Ascii.NUL];

  tokenFilter: Token.FilterProcType = {
    RETURN[SELECT TRUE FROM
      Token.WhiteSpace[c, data], c = Ascii.NUL => FALSE,
      c = '/ => FALSE,
      ENDCASE => TRUE]};
  token ← Token.Filtered[@getToken, NIL, tokenFilter];
  switches ← IF getToken.break = '/ THEN
    Token.Filtered[@getToken, NIL, tokenFilter]
  ELSE NIL;
  END;
```

We can extend this example so that the token is defined to be either a sequence of non-white-space characters or a sequence of characters (possibly containing white-space characters) between double quotes.

```
GetToken: PROCEDURE [h: Exec.Handle] RETURNS [token, switches: LONG STRING] =
  BEGIN
  get: PROCEDURE [Token.Handle] RETURNS [c: CHARACTER] = {
    RETURN[Exec.GetChar[h]]};
  getToken: Token.Object ← [getChar: get, break: Ascii.NUL];
  isQuote: Token.QuoteProcType = {
    RETURN[IF c = '" THEN c ELSE Token.nonQuote]};
  tokenFilter: Token.FilterProcType ← {
    RETURN[SELECT TRUE FROM
      Token.WhiteSpace[c, data], c = Ascii.NUL = > FALSE,
      c = '/ = > FALSE,
      ENDCASE = > TRUE]};
  token ← Token.MaybeQuoted[@getToken, NIL, tokenFilter, isQuote];
  switches ← IF getToken.break = '/ THEN
    Token.Filtered[@getToken, NIL, tokenFilter]
  ELSE NIL;
  END;
```

10

ToolDriver

The **ToolDriver** interface allows a tool to inform the **ToolDriver** package of its existence and of the existence of its subwindows. The **ToolDriver** package can thus use the tool's functions on behalf of a user communicating with the package via a script file. Every tool that provides some generally useful function should use the **ToolDriver** facilities. Although the **ToolDriver** is an add-on package (not built into the regular Tajo), the interface routines are available in Tajo even without the ToolDriver so that the tool being STARTed need not concern itself with unbound procedures. For details on running the **ToolDriver** itself, see the *XDE User's Guide*.

10.1 Types

```
ToolDriver.Address: TYPE = RECORD [name: LONG STRING, sw: Window.Handle];
```

Address is an element of the array passed to **NoteSWs** to describe the relationship between a subwindow of a tool and its name.

```
ToolDriver.AddressDescriptor: TYPE =
  LONG DESCRIPTOR FOR ARRAY OF ToolDriver.Address;
```

AddressDescriptor is the array passed to **NoteSWs** describing the subwindows of a tool.

```
ToolDriver.FindDataProcType: TYPE = PROCEDURE [
  toolID: ToolDriver.ToolID] RETURNS [LONG POINTER];
```

The **FindDataProcType** procedure is obsolete.

```
ToolDriver.NoteDataProcType: TYPE = PROCEDURE [
  toolID: ToolDriver.ToolID, data: LONG POINTER];
```

The **NoteDataProcType** procedure is obsolete.

```
ToolDriver.NoteSWsProcType: TYPE = PROCEDURE [
  tool: LONG STRING, subwindows: ToolDriver.AddressDescriptor];
```

NoteSWsProcType is the type of the **NoteSWs** procedure.

```
ToolDriver.RemoveDataProcType: TYPE = PROCEDURE [toolID: ToolDriver.ToolID];
```

The **RemoveDataProcType** type is obsolete.

```
ToolDriver.RemoveSWsProcType: TYPE = PROCEDURE [tool: LONG STRING];
```

RemoveSWsProcType is the type of the **RemoveSWs** procedure.

```
ToolDriver.ToolID: TYPE = CARDINAL [0..1024);
```

ToolID is private and should not be used.

10.2 Constants and data objects

None.

10.3 Signals and errors

None.

10.4 Procedures

```
ToolDriver.FindData: ToolDriver.FindDataProcType;
```

The **FindData** procedure is obsolete and not implemented.

```
ToolDriver.NoteData: ToolDriver.NoteDataProcType;
```

The **NoteData** procedure is obsolete and not implemented.

```
ToolDriver.NoteSWs: ToolDriver.NoteSWsProcType;
```

The **NoteSWs** procedure is used by a tool to announce its existence. **tool** is whatever name the tool wishes to go by for purposes of the ToolDriver. It need not be the same as the name displayed in the herald of the tool's window; in general, it will be different because the ToolDriver restricts the **tool** to contain only alphanumerics. **subwindows** is a list of subwindows that the tool wishes to make available to the ToolDriver. The **name** for each of these must also contain only alphanumerics. Tools that register with the **ToolDriver** interface should have unique names in each of the menus used by the tool so as not to be ambiguous to the **ToolDriver** package.

```
ToolDriver.RemoveData: ToolDriver.RemoveDataProcType;
```

The **RemoveData** procedure is obsolete and not implemented.

```
ToolDriver.RemoveSWs: ToolDriver.RemoveSWsProcType;
```

The **RemoveSWs** procedure should be called when a tool goes inactive, unless it is prepared to be called by the ToolDriver while inactive.

```
ToolDriver.SetDataProcs: PROCEDURE [
  findData: ToolDriver.FindDataProcType, noteData: ToolDriver.NoteDataProcType,
  removeData: ToolDriver.RemoveDataProcType];
```

The **SetDataProcs** procedure is obsolete and not implemented.

```
ToolDriver.SetSWsProcs: PROCEDURE [
  noteSWsProc: ToolDriver.NoteSWsProcType,
  removeSWsProc: ToolDriver.RemoveSWsProcType];
```

The **SetSWsProcs** procedure is obsolete and not implemented.

10.5 Example

The following example registers a tool and its subwindows when the subwindows are created, which happens whenever a tool becomes active.

```
MakeSWs: Tool.MakeSWsProc =
    BEGIN
    addresses: ARRAY [0..3) OF ToolDriver.Address;
    ...
    msgSW ← Tool.MakeMsgSW[ ... ];
    formSW ← Tool.MakeFormSW[ ... ];
    fileSW ← Tool.MakeFileSW[ ... ];
    ...
    address ← [
        [name: "MsgSW"L, sw: msgSW];
        [name: "FormSW"L, sw: formSW];
        [name: "FileSW"L, sw: fileSW]];
    ToolDriver.NoteSWs[tool: "Sample"L, subwindows: DESCRIPTOR[addresses]];
    END;
```

II

Tool building

These interfaces support most tool builders, who need only prepackaged parts. The subwindow types given here can easily be combined into tools. The Example Tool, discussed in Appendix A, shows how to put these pieces together and how to use them with other interfaces such as file management interfaces (see the File Management section).

If you require significantly more or different functionality for a new tool, use the interfaces described in the next major section of this document (Window and Subwindow Building). It is not recommended, however, that you use the lower-level interfaces unless you have tool-building experience. Those interfaces require much greater attention to detail to apply them properly, especially when integrating them into the system.

II.1 Interface abstracts

FileSW provides the definitions and procedures for creating text subwindows whose backing storage is a disk file, plus procedures that are specific to file subwindows.

FormSW implements a form subwindow, which is a. mechanism for invoking commands and specifying command parameters. This type of subwindow is standard for invoking tools.

MsgSW implements message subwindows, which provide a simple way of posting messages to the user.

ScratchSW creates a subwindow backed by a scratch source; that is, by a piece of virtual memory.

StringSW provides the definitions and procedures for creating and manipulating text subwindows whose backing store is a **LONG STRING**.

TextSW defines extensive facilities for viewing text independent of its source.

TTYSW implements a TTY subwindow, which emulates a teletype.

Put provides procedures for converting data types to formatted text and outputting that text to windows.

Tool provides facilities for building an interactive tool. It is designed to make the writing of tools with a standard user interface as easy as possible, by allowing the client to avoid many of Tajo's low-level facilities at the cost of some loss in flexibility.

ToolWindow provides facilities for constructing subwindows in a tool window. Many standard subwindow types are provided by the development environment; normally only clients that wish to make complex tools need this interface.

11

FileSW

The **FileSW** interface provides the definitions and procedures for creating text subwindows whose backing storage is a disk file. It also provides procedures that are specific to file-type subwindows. All non-file subwindow-specific manipulations are contained in the interface **TextSW**.

11.1 Types

```
FileSW.Access: TYPE = TextSource.Access;

FileSW.EnumerateProcType: TYPE = PROCEDURE[
  sw: Window.Handle, name: LONG STRING, access: FileSW.Access]
  RETURNS [done: BOOLEAN];

FileSW.Options: TYPE = TextSW.Options;
```

11.2 Constants and data objects

```
FileSW.defaultOptions: FileSW.Options = [
  access: read, menu: TRUE, split: TRUE, wrap: TRUE, scrollbar: TRUE,
  flushTop: FALSE, flushBottom: FALSE];
```

11.3 Signals and errors

```
FileSW.Error: SIGNAL [code: FileSW.ErrorCode];

FileSW.ErrorCode: TYPE = {
  notAFileSW, isAFileSW, notEditable, isEditable, accessDenied, other};
```

11.4 Procedures

```
FileSW.Create: PROCEDURE [
  sw: Window.Handle, name: LONG STRING,
  options: FileSW.Options ← FileSW.defaultOptions,
```

```
    s: Stream.Handle ← NIL, position: TextSource.Position ← 0,
    allowTypeIn: BOOLEAN ← TRUE, resetLengthOnNewSession: BOOLEAN ← FALSE];
```

The **Create** procedure creates a disk source and then creates a text subwindow using that disk source. The name **Create** is something of a misnomer, since the subwindow must already have been created by a call on **ToolWindow.Create** or **ToolWindow.CreateSubwindow**; the call on **Create** is actually a differentiation process. If **s** is **NIL**, a stream is automatically opened on the file **name**. If **s** is not **NIL**, **name** must be the name of the file to which **s** is attached. Note that if **s** is not **NIL**, the file subwindow owns the stream and will destroy it when the window is **Destroy**ed. The text is positioned so that the character specified by **position** is displayed on the first line of **sw**. If **options.access** is **read** and the file can't be found, **TextSource.Error[fileNameError]** is raised. The parameter **allowTypeIn** controls whether the window permits user type-in. The parameter **resetLengthOnNewSession**, which controls whether the file length is reset to zero on a new session, is probably of interest only to the implementation of CoPilot or tools that run in CoPilot. Subwindows created by **FileSW.Create** should always be destroyed by **FileSW.Destroy**, not by **TextSW.Destroy**.

```
FileSW.Destroy: PROCEDURE [sw: Window.Handle];
```

The **Destroy** procedure destroys a file subwindow created by **FileSW.Create** and deletes the stream backing the window.

```
FileSW.Enumerate: PROCEDURE [proc: FileSW.EnumerateProcType];
```

The **Enumerate** procedure enumerates all the current file subwindows, including file subwindows that are not in the window tree and file subwindows that are part of inactive tools.

```
FileSW.GetFile: PROCEDURE [
    sw: Window.Handle] RETURNS [name: LONG STRING, s: Stream.Handle];
```

The **GetFile** procedure returns the file name and stream that are currently attached to a file subwindow. The string returned by **GetFile** is owned by Tajo and must not be freed by the client.

```
FileSW.IsEditable: PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
```

The **IsEditable** procedure returns **TRUE** if and only if a window is currently editable.

```
FileSW.IsIt: PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
```

The **IsIt** procedure returns **TRUE** if and only if a window is a file subwindow.

```
FileSW.LoadMCR: Menu.MCRType;
```

The **LoadMCR** procedure is a menu command routine that does the standard load operation using the current selection as the file-name argument. Clients that construct their own menus may call it.

```
FileSW.MakeEditable: PROCEDURE [sw: Window.Handle] RETURNS [ok: BOOLEAN];
```

The **MakeEditable** procedure makes a file subwindow editable. It returns an indication of success.

```
FileSW.PutEditableFile: PROCEDURE [
  sw: Window.Handle, name: LONG STRING] RETURNS [ok: BOOLEAN];
```

The **PutEditableFile** procedure stores the edited file on the new file **name**. If **name** = NIL, the old version of the file is saved as **"currentName$"** and the edited file is output to **currentName**. It returns an indication of success.

```
FileSW.ResetEditableFile: PROCEDURE [sw: Window.Handle];
```

The **ResetEditableFile** procedure resets an edited file to its original state. The file subwindow is not editable after the call.

```
FileSW.SetFile: PROCEDURE [
  sw: Window.Handle, name: LONG STRING, s: Stream.Handle ← NIL,
  position: TextSource.Position ← 0];
```

The **SetFile** procedure loads a new file into a file subwindow. Note that if s is not NIL, the file subwindow owns the stream **s** and will destroy it when the window is **Destroy**ed.

12

FileWindow

The **FileWindow** interface provides facilities for manipulating file windows. It also maintains a mapping between file windows and the files that are loaded into them. A *file window* is a tool containing a text subwindow for manipulating and displaying text. All of the **FileWindow** procedures that have **Window.Handle** parameters or results deal with the text subwindow in the FileWindow. Some procedures also accept the tool window or even the clipping window for the FileWindow. The text subwindow is either an editable or non-editable file subwindow (see **FileSW**), or a scratch subwindow (see **ScratchSW**).

12.1 Types

```
FileWindow.ContinueStop: TYPE = {continue, stop};

FileWindow.EnumerateProcType: TYPE = PROC[
  sw: Window.Handle] RETURNS [continue: FileWindow.ContinueStop];
```

12.2 Constants and data objects

```
FileWindow.defaultOptions: TextSW.Options = [
  access: read, menu: TRUE, split: TRUE, wrap: TRUE, scrollbar: TRUE,
  flushTop: FALSE, flushBottom: FALSE];
```

12.3 Signals and errors

None.

12.4 Procedures

```
FileWindow.Create: PROC [
  box: Window.Box, options: TextSW.Options ← FileWindow.defaultOptions,
  initialState: ToolWindow.State ← active]
  RETURNS [sw: Window.Handle];
```

Create creates an empty file window. It takes the dimensions of the desired window and a set of options for the state of the window. **Create** returns the text subwindow for the file window. The **options** parameter is ignored.

```
FileWindow.CreateMCR: Menu.MCRType;
```

CreateMCR is the FileWindow package's implementation of the Create menu operation. It is defined in the **FileWindow** interface so that clients can create their own menus with this procedure implementing one of the operations.

```
FileWindow.Destroy: PROC [sw: Window.Handle];
```

Destroy destroys a file window. The parameter may be either the tool window, the clipping window, or the text subwindow for a file window.

```
FileWindow.DestroyMCR: Menu.MCRType;
```

DestroyMCR implements the Destroy menu operation. If it would reduce the number of file windows below the minimum, the display blinks. Otherwise, the user is asked to confirm destruction of the window by clicking the **POINT** mouse button. **DestroyMCR** is defined in the **FileWindow** interface so that clients can create their own menus with this procedure implementing one of the operations.

```
FileWindow.Enumerate: PROC [proc: FileWindow.EnumerateProcType];
```

Enumerate calls **proc** with the text subwindow for each file window until **proc** returns **stop** or all file windows have been enumerated.

```
FileWindow.FileInWindow: PROC [
  sw: Window.Handle] RETURNS [fileName: LONG STRING, s: Stream.Handle];
```

The **FileInWindow** procedure returns the file name and stream that back the window. The **sw** parameter is expected to be the text subwindow for the file window, or **NIL**. If it is **NIL**, a file window is selected using the same heuristics as **WindowForFile** and the results for that window are returned. If **WindowForFile** fails, **[NIL, NIL]** is returned. The results do not belong to the user and should be treated as read-only. They are potentially dangling references, since the file in the window may change. If needed, the string should be copied immediately. Even this is not 100% safe.

```
FileWindow.GetInfo: PROC RETURNS [
  ext: LONG STRING, fileMenu, sourceMenu: Menu.Handle,
  minimumWindows: CARDINAL];
```

The **GetInfo** procedure returns the global data maintained by the FileWindow package. This data is set by **SetExtension**, **SetSourceMenu**, and **SetMinimumWindows**.

```
FileWindow.IsIt: PROC [sw: Window.Handle] RETURNS [BOOLEAN];
```

IsIt returns **TRUE** if the window is a file window and **FALSE** otherwise. The parameter may be either the tool window, the clipping window, or the text subwindow for a file window.

```
FileWindow.LoadWindow: PROC [
  fileName: LONG STRING, position: LONG CARDINAL ← 0, s: Stream.Handle ← NIL,
  loadIFSame: BOOLEAN ← FALSE, sw: Window.Handle ← NIL];
```

LoadWindow loads a file into a file window. **s** must be an **MStream.Handle**. If **s** is not **NIL**, it is assumed to be a stream on file **fileName** and is used as the backing stream. The file is positioned in the window at **position**; that is, the top line in the window contains the character in that position. **loadIFSame** controls whether to reload the requested file if it is already loaded in the window. If **sw** is not **NIL**, it is the text subwindow of a file window in which to load the file. If it is **NIL**, the file window package searches for a suitable window to load the file into, using the same heuristics as **WindowForFile**. If **sw** is **NIL** and **WindowForFile** fails, then either an unnamed **ERROR** or an address fault results.

```
FileWindow.Position: PROC [sw: Window.Handle, position: LONG CARDINAL];
```

Position sets the position of the file in the window so that the top line in the window contains the character at that position. If the position is out of range for the file, no action is taken. The **sw** parameter is expected to be the text subwindow for the file window, or **NIL**.

```
FileWindow.SetExtension: PROC [ext: LONG STRING];
```

In loading a window, the **FileWindow** package first attempts to find a file with the specified name. If that fails, it tries three different extensions, in turn, to the name in an attempt to find a file to load. These extensions are ".mesa" (initially), ".config", and ".cm". The first extension can be modified by a client using **SetExtension**. **SetExtension** will copy the contents of **ext**.

```
FileWindow.SetMinimumWindows: PROC [keep: CARDINAL];
```

SetMinimumWindows permits the client to set the minimum number of file windows that must exist at all times. **Destroy** operations that would take the number of windows below this minimum will fail to destroy any window.

```
FileWindow.SetSize: PROC [sw: Window.Handle, box: Window.Box];
```

SetSize changes the size of the file window. The parameter may be either the tool window, the clipping window, or the text subwindow for a file window.

```
FileWindow.SetSourceMenu: PROC [menu: Menu.Handle];
```

SetSourceMenu associates a menu with all file windows.

```
FileWindow.WindowForFile: PROC [fileName: LONG STRING] RETURNS [Window.Handle];
```

WindowForFile searches for a file window into which the file can be loaded. It returns the first non-editable file window containing a file whose full name or simple name matches **fileName**. If a non-editable file window already contains the file, that window is returned. If no such window is found, the file window package searches all file windows that are either non-editable windows or nearly empty scratch subwindows. In order of preference,

WindowForFile tries to find either an empty active, a full active, an empty tiny, or a full tiny file window. If it cannot find a suitable file window, it returns **NIL**.

13

FormSW

The **FormSW** interface is used in building tools that interact with the user via the window user interface. A *form subwindow* is a mechanism for invoking commands and specifying the command parameters. A form subwindow consists of *form items*, which are rectangular regions in the subwindow, similar to ruled-off areas on a preprinted form.

A form item can be one of the following types. *Command items* correspond to the operations a tool can perform. A command item appears in the form subwindow as a keyword followed by an "!". *String items* are strings filled in by the user that serve as parameters to command items. A ": " is appended to string item keywords. *Enumerated items* are lists of string items.

These items may be displayed in two ways: "keyword: {a, b, c,...}" or "keyword: {a}." In both cases, choosing is done via *menu prompts*. Menu prompts are always available for enumerated items and sometimes for string items. When you press the menu button over the keyword for an enumerated field, a menu of allowed values is displayed. Choosing one of the values from the menu sets the enumerated item to that value. Similarly, when you press the menu button over the keyword for a string item, a menu of character strings is displayed. Choosing one of the items (strings) from the menu appends the menu string at the current position of the type-in point. Enumerated items may also be chosen by bringing up a menu and selecting the desired item with the mouse. In the first example above, the current value becomes highlighted. In the other example, only the current value is displayed.

Boolean items are form items that can have one of two values, either **TRUE** or **FALSE**. The boolean state of the item is indicated by highlighting. Highlighted means **TRUE**. *Numeric items* are like string items, except that only strings representing numbers are permitted. A "= " is appended to numeric item keywords. *Tag items* are used to clarify an otherwise complicated form subwindow by separating the items along logical divisions and labeling them as such. The labels, which are tag items, do not correspond to any user-input actions, but instead serve to annotate the form.

The **ItemObject** is the fundamental data structure of the form subwindow interface; **TYPE**s and **PROCEDURE**s in **FormSW** provide mechanisms for defining and manipulating them. Readers not familiar with the form subwindow interface are advised first to carefully

study ExampleTool.bcd, found in Appendix A, and then to examine FormSW.ItemObject before learning about other TYPEs and PROCEDUREs in this interface.

The client constructs a form subwindow by specifying an array of *form-item handles*. Each handle points to an item; each item is a variant record containing a pointer to the tool's internal data that will be displayed and altered. The elements of the item handle array point to objects that contain information about how and where the corresponding form item should be displayed in the form subwindow. An item object may also contain *notification procedures* that are called by the form subwindow interface to inform the client of events affecting that item.

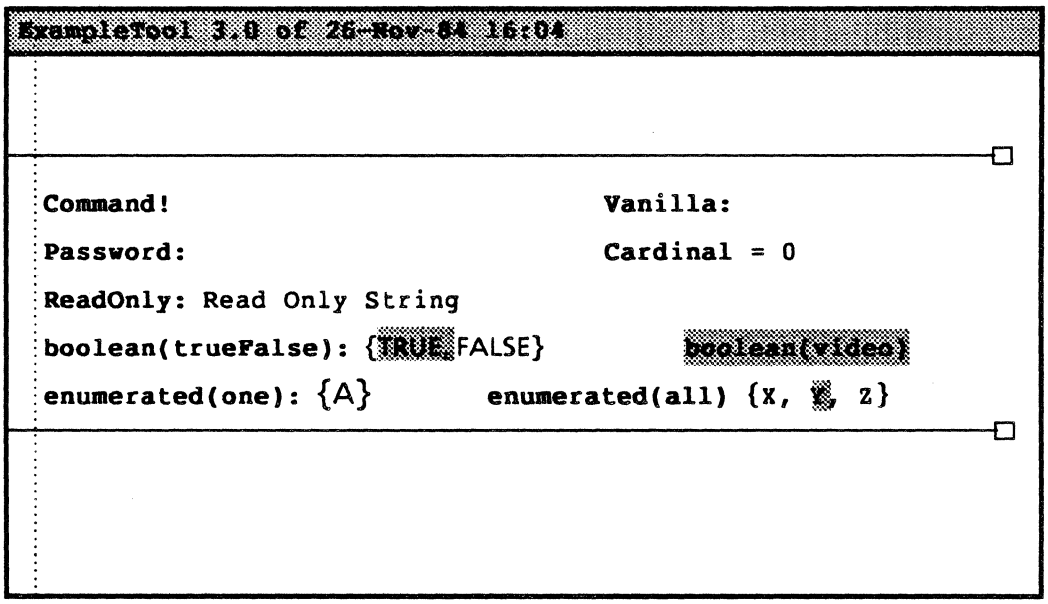


Figure 13.1: Example Tool

The client's items are displayed in a subwindow; the user can alter them at any time unless explicitly prohibited by the client. The form subwindow interface supplies procedures to display, select, or alter any of these items.

Clients of this interface should keep in mind that forms cannot be arbitrarily large because of sizable storage requirements. The fixed overhead in heap usage per form item is 23 words (broken down as follows: 4 words for the item record, 2 words for the handle, 8 words for the item's TextSource plus 1 word for heap overhead, and 9 words for the item's TextDisplay Object). The variable overhead is due to the STRINGs associated with an item (the tag, for example), line tables associated with multi-line items, and the variant part of the item record.

It is important to distinguish between the user actions of *choice* and *selection*: the user is said to select an item (or part of an item) if that action changes the current selection; otherwise the user is said to make a choice of (or in) the item. It is not always possible to distinguish between the two cases by simply looking at the display-marking actions.

12.1 Types

FormSW.BooleanHandle: TYPE = LONG POINTER TO boolean FormSW.ItemObject;

See the description of ItemObject for the definitions of the common fields of the ItemObject record. There is no special trailer appended to the tag for boolean parameter items . When

the user chooses a boolean parameter item, the tag is inverted on the display, the sense of the BOOLEAN pointed to by **switch** is inverted, and then the supplied client **proc** is invoked. (See also **NopNotifyProc** and **BooleanChoices**.)

switch is a LONG POINTER TO BOOLEAN provided so that the client can access the BOOLEAN without necessarily accessing the **ItemObject**. The BOOLEAN must occupy its own word in memory. This can be achieved by allocating the BOOLEAN in the client's global frame (but not in a RECORD in the global frame unless it is a MACHINE DEPENDENT RECORD and the BOOLEAN is specified to occupy a word) or by using the overlaid variant FormSW.**WordBoolean**. Using the overlaid variant is clumsy and should be avoided.

proc is called every time the user changes the boolean.

```
FormSW.ClientItemsProcType: TYPE = PROCEDURE [
  sw: Window.Handle] RETURNS [items: FormSW.ItemDescriptor, freeDesc: BOOLEAN];
```

The **ClientItemsProcType** procedure is called when the form subwindow package needs to create the form subwindow, usually when the enclosing tool window is created or made active. The procedure returns the **ItemDescriptor** that describes the contents of the form subwindow. If **freeDesc** is TRUE, then the **ItemDescriptor** has been allocated from the system heap and the form subwindow package frees it when the subwindow is destroyed (usually when the enclosing tool window is deactivated). If **freeDesc** is FALSE, the **ItemDescriptor** is not deallocated, and the management of its storage is the client's responsibility.

```
FormSW.CommandHandle: TYPE = LONG POINTER TO command FormSW.ItemObject;
```

See the description of **ItemObject** for the definitions of the common fields of the **ItemObject** record. For **command** parameter items the character "!" is appended to the **tag** to remind the user that this is a command item. User choice of this type of parameter item causes invocation of the supplied client **proc** to be invoked like menu-command choice. (See also **NopNotifyProc**.)

```
FormSW.Enumerated: TYPE = RECORD [string: LONG STRING, value: UNSPECIFIED];
```

This type is used to specify the representation of an element of an enumerated item in an **enumerated ItemObject**. The element displayed as **string** has the value **value** associated with it. (See also **EnumeratedHandle** and **EnumeratedDescriptor**.)

```
FormSW.EnumeratedDescriptor: TYPE =
  LONG DESCRIPTOR FOR ARRAY OF FormSW.Enumerated;
```

An **EnumeratedDescriptor** lists the possible values of an enumerated item. (See also **EnumeratedHandle** and **EnumeratedDescriptor**.)

```
FormSW.EnumeratedFeedback: TYPE = {all, one};
```

This type specifies whether to display all or one of the **enumerated ItemObjects**. (See also **EnumeratedHandle**.) Examples of the two forms of feedback are:

all The item displays as "tag: {a, b, c}". Choosing an item within the curly brackets video-inverts that item.

one The item displays as "tag: {a}". Depressing the menu mouse button displays the set of strings available for choice. Choosing an item causes it to be displayed.

```
FormSW.EnumeratedHandle: TYPE = LONG POINTER TO enumerated FormSW.ItemObject;
```

See the description of **ItemObject** for the definitions of the common fields of the **ItemObject** record. For **enumerated** parameter items the special trailer ": {" is appended to the **tag**. In addition, a "}" is appended at the end of the item's display representation. When the user modifies this type of parameter item, the display is updated according to the value of **feedback**, the **UNSPECIFIED** pointed to by **value** is updated to match the display, and then the supplied client **proc** is invoked. (See also **NopEnumeratedNotifyProc**, **BooleanChoices**. and **nullEnumeratedValue**.)

choices For both forms of feedback, **all** or **one**, the items available for choice are those **STRING**s supplied by the client in the **choices**. When the **string** from one of the **choices** is chosen, the corresponding **value** from the **Enumerated** is stored into **ItemObject.value**↑. Depressing the menu mouse button displays the set of strings available for choice.

value This field is a **POINTER TO UNSPECIFIED** so that the client need not have access to the **ItemObject** in order to have access to the **UNSPECIFIED**. This introduces the same word-alignment problems that occur with the **boolean ItemObject**'s **switch**, and the same solutions and caveats apply here. **value** points to an **UNSPECIFIED** so that its possible values can be from any type (usually an enumeration).

proc This field is a **PROCEDURE** that is called whenever the user changes **value**. (See also **NopEnumeratedNotifyProc**.)

copyChoices This field indicates whether the client's **choices** were copied into the system heap and can be freed to the system heap automatically by **FormSW**.

```
FormSW.EnumeratedNotifyProcType: TYPE = PROCEDURE [
  sw: Window.Handle ← NIL, item: FormSW.ItemHandle ← NIL,
  index: CARDINAL ← FormSW.nullIndex, oldValue:
  UNSPECIFIED ← FormSW.nullEnumeratedValue];
```

A **EnumeratedNotifyProcType** is called whenever the user changes the corresponding enumerated item in a form subwindow. **sw** is the subwindow containing the item. **item** is the **ItemHandle** of the enumerated item. **index** is the index of the item in the **ItemDescriptor** for the subwindow. **oldValue** is the value of the enumerated item before the user changed it. (See also **EnumeratedHandle**.)

```
FormSW.FilterProcType: TYPE = PROCEDURE[
  sw: Window.Handle, item: FormSW.ItemHandle, insert: CARDINAL,
  string: LONG STRING];
```

A **FilterProcType** is called to permit a client to perform editing operations on a **string ItemObject**. **sw** is the subwindow containing the item. **item** is the **ItemHandle** of the enumerated item. **string**, which may be **NIL**, contains the characters to edit into the backing-store string at position **insert**. The zero position is defined to be the left of the first character of the string. (See also **StringHandle**.)

```
FormSW.Flag: TYPE = {clientOwnsItem, drawBox, hasContext, invisible, readOnly};
```

Flag defines the types of state bits maintained for a form subwindow item. (See also **ItemFlags**.)

```
FormSW.FreeHintsProcType: TYPE = PROCEDURE[FormSW.Hints];
```

A **FreeHintsProcType** is called to free the **Hints** in a **string ItemObject**, allowing the **Hints** to be somewhere other than in the client's global frame.

```
FormSW.Hints: TYPE = LONG DESCRIPTOR FOR ARRAY OF LONG STRING;
```

Hints is a set of strings that is available to the user in a menu to suggest possible strings for use when editing a **string ItemObject**. (See also **StringHandle**.)

```
FormSW.ItemFlags: TYPE = RECORD [
  readOnly: BOOLEAN ← FALSE,
  invisible: BOOLEAN ← FALSE,
  drawBox: BOOLEAN ← FALSE,
  hasContext: BOOLEAN ← FALSE,
  clientOwnsItem: BOOLEAN ← FALSE,
  modified: BOOLEAN ← FALSE];
```

ItemFlags is a **RECORD** of state bits for an item in a form subwindow. The meaning of the flags is as follows:

readOnly — If this flag is **TRUE**, the user cannot modify this parameter. If any modification is attempted, the **readOnlyNotifyProc** for this subwindow is called.

invisible — If this flag is **TRUE**, the item is not displayed in the subwindow, and it is treated by form subwindows exactly as if it were not present, except that it is occupying an index slot.

drawBox — If this flag is **TRUE**, the item is displayed enclosed within a box that is one bit thick.

hasContext — If this flag is **TRUE**, a client context two words long is associated with the item. This context serves the same function as a client context associated with a subwindow. However, unlike **Context**, **FormSW** returns a pointer to the client data words, not the value of the data words. (See also **ContextFromItem**.)

clientOwnsItem If this flag is TRUE, the form subwindow will not try to de-allocate the item if the subwindow is destroyed. This flag is usually FALSE, meaning that the client does not need to be concerned with storage allocation and de-allocation. Instead, the form subwindow "owns" the storage and is responsible for maintaining it.

modified The **modified** flag is set when an item on the form subwindow has been modified. See the **FormSW.SetModifyNotification** procedure for setting a notificiation procedure on this flag.

```
FormSW.ItemHandle: TYPE = LONG POINTER TO FormSW.ItemObject;

FormSW.ItemObject: TYPE = RECORD [
    tag: LONG STRING,
    place: Window.Place,
    flags: FormSW.ItemFlags,
    body: SELECT type: FormSW.ItemType FROM
    boolean => [
        switch: LONG POINTER TO BOOLEAN,
        proc: FormSW.NotifyProcType],
    command => [proc: FormSW.ProcType],
    enumerated => [
        feedback: FormSW.EnumeratedFeedback,
        copyChoices: BOOLEAN,
        value: LONG POINTER TO UNSPECIFIED,
        proc: FormSW.EnumeratedNotifyProcType,
        choices: FormSW.EnumeratedDescriptor],
    longNumber => [
        signed, notNegative: BOOLEAN,
        radix: FormSW.Radix,
        boxWidth: CARDINAL [0..256),
        proc: FormSW.LongNumberNotifyProcType,
        default: LONG UNSPECIFIED,
        value: LONG POINTER TO LONG UNSPECIFIED,
        string: LONG STRING, bias: INTEGER],
    number => [
        signed, notNegative: BOOLEAN,
        radix: FormSW.Radix,
        boxWidth: CARDINAL [0..128),
        proc: FormSW.NumberNotifyProcType,
        default: UNSPECIFIED,
        value: LONG POINTER TO UNSPECIFIED,
        string: LONG STRING, bias: INTEGER],
    source => [
        source: TextSource.Handle,
        boxWidth: CARDINAL,
        filterProc: FormSW.FilterProcType,
        menuProc: FormSW.MenuProcType],
    string => [
        feedback: FormSW.StringFeedback,
        inHeap: BOOLEAN,
        string: LONG POINTER TO LONG STRING,
```

```
    boxWidth: CARDINAL,
    filterProc: FormSW.FilterProcType,
    menuProc: FormSW.MenuProcType],
  tagOnly => [sw: Window.Handle, otherItem: CARDINAL],
ENDCASE];
```

The **ItemObject** is complex so that it can provide sufficient flexibility for the tool writer who wants fine control over displaying and altering items. Most clients should not explicitly construct an **ItemObject**, but should instead use the procedures that allocate an **ItemObject** and take advantage of default values. In **FormSW** procedure types, the argument is called **item** if it is an **ItemHandle** and **items** if it is an **ItemDescriptor**. Note that **DESCRIPTOR FOR ARRAY** is implicitly a **DESCRIPTOR FOR ARRAY** *(0..0)*. Trying to index an **ItemDescriptor** by an enumerated type results in a compilation error. Instead of indexing by an enumerated type, the procedure **FindIndex** should be used to get the desired index.

Only the common fields of the **ItemObject** are described here. For a description of the fields of each variant part, see the descriptions of the corresponding handles (**BooleanHandle**, **CommandHandle**, **EnumeratedHandle**, **LabelHandle**, **LongNumberHandle**, **NumberHandle**, **SourceHandle**, **StringHandle**, and **TagOnlyHandle**).

tag is a client-supplied string that is displayed immediately preceding the data associated with the parameter (e.g., "tag: string"). It may be **NIL**, in which case any trailer characters that are usually displayed after the tag will be suppressed (e.g., ": ").

place is used only if the **type** field of the subwindow **option** has the value **fixed**; otherwise it is ignored. **place** is the **x,y** position (subwindow relative) where the tag and data are to be displayed. The array of item pointers is required to have the places in ascending (English-reading) order; i.e., left to right, top to bottom. If the position is negative, it is treated as a relative offset, where the magnitude of **x** specifies the number of bits to leave between the end of the preceding item and the start of the tag for this item. The use of a negative **x** following a string or number item that uses **defaultBoxWidth** results in the **ERROR ItemError[illegalCoordinate, i]**, where **i** is the index of the offending item. Negative **y** positions are also interpreted specially. They are line positions; i.e., they specify position as a multiple of the line height for the subwindow. The constants **line0** through **line9** can be used as **y** values to specify that the item should be on the zero through ninth lines in the subwindow. (See also the procedure **LineHeight**, **LineN**, **SetTagPlaces** and the constants **newLine**, **nextLine**, **nextPlace**, and **sameLine**.)

flags is a **RECORD** of state bits for the item. (See **ItemFlags** for the meaning of the flags.)

```
FormSW.ItemType: TYPE = {
  boolean, command, enumerated, longNumber, number, source, string, tagOnly};
```

ItemType defines the different types of form subwindow items supported by **FormSW**.

```
FormSW.LabelHandle: TYPE = FormSW.TagOnlyHandle;
```

One use of a **tagOnly** item type is to act as a label for some part of the form. For example, a form might consist of two parts, one for specifying input parameters and the other for output parameters. The client could distinguish the individual items by prefixing their

tags with "Input-" or "Output-", or two sets of items could have the same tags but be preceded by a labeling line consisting of an item whose tag was "Input parameters" or "Output parameters." (See also TagOnlyHandle.)

sw — This is the form subwindow that contains the item. It is automatically set by Create; clients should ignore it.

otherItem — This is the index of the other item for which this item is acting as a tag. For labels, otherItem should be nullIndex.

```
FormSW.LongNumberHandle: TYPE =
  LONG POINTER TO longNumber FormSW.ItemObject;
```

The number and longNumber item types are for specifying numeric form items and are very similar, with only a few exceptions. See the description of ItemObject for the definitions of the common fields of the ItemObject record. The longNumber parameter item differs in the following ways: boxWidth must be larger; value points to a LONG UNSPECIFIED instead of an UNSPECIFIED; default is a LONG UNSPECIFIED instead of an UNSPECIFIED; and proc takes a LONG UNSPECIFIED instead of an UNSPECIFIED for the old value. Refer to FormSW.NumberHandle for an explanation of the fields in the longNumber variant. (See also NopLongNumberNotifyProc.)

```
FormSW.LongNumberNotifyProcType: TYPE = PROCEDURE [
  sw: Window.Handle ← NIL, item: FormSW.ItemHandle ← NIL,
  index: CARDINAL ← FormSW.nullIndex, oldValue: LONG UNSPECIFIED ← LAST[INTEGER]];
```

A LongNumberNotifyProcType is called each time the user edits a longNumber ItemObject. sw is the subwindow containing the item. item is the ItemHandle of the longNumber item. index is the index of the item in the ItemDescriptor for the subwindow. oldValue is the value of the longNumber item before it was changed by the user. (See also LongNumberHandle.)

```
FormSW.MenuProcType: TYPE = PROCEDURE [sw: Window.Handle, index: CARDINAL]
    RETURNS [hints: FormSW.Hints, freeHintsProc: FormSW.FreeHintsProcType,
    replace: BOOLEAN];
```

A MenuProcType procedure is associated with a string ItemObject. It is called whenever the user selects the string item with the menu button. This gives the client the opportunity to supply a list of strings to be displayed in a menu. sw is the subwindow containing the item. item is the ItemHandle of the string item. The MenuProcType procedure returns the information needed for the menu. If replace is FALSE, the menu item will be inserted into the item's string when the user chooses it, just as if the user had typed the menu string. If BASE[hints] = NIL, no prompt menu will be available.freeHintsProc is called to free the hints, allowing the hints to be somewhere other than in the client's global frame. (See also InHeapFreeHintsProc, NopFreeHintsProc, VanillaMenuProc and StringHandle.)

```
FormSW.NotifyProcType: TYPE = FormSW.ProcType;
```

A **NotifyProcType** procedure is called whenever a client changes a boolean item. **sw** is the subwindow containing the item. **item** is the **ItemHandle** of the boolean item. **index** is the index of the item in the **ItemDescriptor** for the subwindow.

```
FormSW.NumberHandle: TYPE = LONG POINTER TO number FormSW.ItemObject;
```

See the description of **ItemObject** for the definitions of the common fields of the **ItemObject** record. For **number** (and **longNumber**) parameter items the special trailer "= " is appended to the **tag**. The user can select and edit a **number** (or **longNumber**) item just like a **string** item, and the client can also exercise control over its alteration and display.

signed — **FormSW** needs to know whether to treat the value as a signed number (i.e., **INTEGER**). It is treated as a **CARDINAL** if **signed** is **FALSE**.

notNegative — The user is permitted to enter negative values if **notNegative** is **FALSE**.

radix — If the user does not provide a specific radix ('D for decimal or 'B for octal) when he enters or modifies the item, then the radix is assumed to be 10 if **radix** is **decimal**, 8 if **radix** is **octal**.

boxWidth — This is added to the **tag**'s width (including the supplied trailer) to determine the width of the box in which the number is displayed. If the special value **defaultBoxWidth** is used, then the box will extend to the right edge of the subwindow or to the next item, whichever is closer.

proc — The client's **proc** is called after each user edit to the item. (See also **NumberNotifyProc** and **NopNumberNotifyProc**).

default — The user might not want to enter any value for the item. In this case, the value is forced to **default**.

value — is a **LONG POINTER TO UNSPECIFIED** so that the client need not have access to the **ItemObject** in order to have access to the **UNSPECIFIED**. **FormSW** assumes that the **UNSPECIFIED** occupies a full word; hence it should not be declared by the client to be a subrange of **CARDINAL** or **INTEGER**. **value** points to an **UNSPECIFIED** so that it can be either a **CARDINAL** or an **INTEGER**.

string — is the string representation of **value**↑. **string** is always convertible to **value**↑ unless it is empty, in which case **value**↑ will be **default**.

bias — is the difference between the displayed number and **value**↑. (Displayed number + **bias** = **value**↑.)

```
FormSW.Options: TYPE = RECORD [
  type: FormSW.Type ← fixed,
  boldTags: BOOLEAN ← TRUE,
  autoScroll: BOOLEAN ← TRUE,
  scrollVertical: BOOLEAN ← TRUE];
```

Options are associated with a form subwindow to control certain formatting aspects of the window.

type If **type** is **fixed**, then the client specifies the layout of items in the window; that is, the **place** field of each **ItemObject** specifies the location of the item in the window. If **type** is **relative**, then the **place** field of the **ItemObjects** is ignored and **FormSW** decides where to locate each item in the window.

boldTags If **boldTags** is TRUE, then all tags aredisplayed in a bold font. If **boldTags** is FALSE, all tags are displayed normally.

autoScroll If **autoScroll** is TRUE, then when editing an item would cause it to disappear from the bottom of the window, the window is automatically scrolled so that the item remains visible. If **autoScroll** is FALSE, no such automatic scrolling is done.

scrollVertical If **scrollVertical** is TRUE, then the user is permitted to scroll the subwindow. If **scrollVertical** is FALSE, the user is not permitted to scroll it.

```
FormSW.ProcType: TYPE = PROCEDURE [
  sw: Window.Handle ← NIL, item: FormSW.ItemHandle ← NIL,
  index: CARDINAL ← FormSW.nullIndex];
```

A **ProcType** procedure is called whenever a client issues a command. **sw** is the subwindow containing the item. **item** is the **ItemHandle** of the command item. **index** is the index of the item in the **ItemDescriptor** for the subwindow.

```
FormSW.Radix: TYPE = {decimal, octal};
```

In **number ItemObjects** and **longNumber ItemObjects**, if the user does not provide a specific radix ('D for decimal or 'B for octal) when he enters or modifies the item, then the radix is assumed to be 10 if **radix** is **decimal**, 8 if **radix** is **octal**.

```
FormSW.ReadOnlyProcType: TYPE = FormSW.ProcType;
```

A **ReadOnlyProcType** procedure is called whenever a client tries to edit a read-only item. **sw** is the subwindow containing the item. **item** is the **ItemHandle** of the item. **index** is the index of the item in the **ItemDescriptor** for the subwindow.

```
FormSW.SourceHandle: TYPE = LONG POINTER TO source FormSW.ItemObject;
```

Not implemented.

```
FormSW.StringFeedback: TYPE = {normal, password};
```

This type controls the style of feedback for **string ItemObjects**.

normal the characters themselves are to be displayed.

password a "*" is displayed in place of each character.

```
FormSW.StringHandle: TYPE = LONG POINTER TO string FormSW.ItemObject;
```

See the description of **ItemObject** for the definitions of the common fields of the **ItemObject** record. For **string** parameter items, the characters ": " are appended to the **tag** to indicatie

that this is a string item. String items give the tool writer explicit control over the alteration of the supplied string and how it is to be displayed. The tool-supplied procedures are called whenever characters are to be added to the **string**.

inHeap If this **BOOLEAN** is **TRUE**, the Tajo **StringEditProc** dynamically allocates and deallocates the backing string from the system heap.

string This is a **LONG POINTER TO LONG STRING** that contains the characters entered by the user. The level of indirection is provided so that the original string may be replaced.

feedback The characters of **string** are displayed on the screen as text unless **feedback** is **password**, in which case a "*" is printed in place of each character of **string**.

boxWidth This is added to the **tag**'s width (including the supplied trailer) to determine the width of the box in which the **LONG STRING** is displayed. If the special value **defaultBoxWidth** is used, then the box extends to the right edge of the subwindow or to the next item, whichever is closer.

filterProc The client's **filterProc** is called whenever the user inputs characters to a selected string item. **string**, which may be **NIL**, contains the characters to edit into the backing-store string at position **insert**. The backing-store modification is performed by calling **StringEditProc**. By interposing a **filterProc** between the user and **StringEditProc**, **FormSW** can optimize the display updating and maintain the consistency of selection and insert. (See also **StringEditProc**.)

menuProc The client's **menuProc** is called whenever the user selects the string item with the menu button. This gives the client the opportunity to supply a list of strings to be displayed in a menu. (See also **MenuProcType** and **VanillaMenuProc**.)

```
FormSW.TagOnlyHandle: TYPE = LONG POINTER TO tagOnly FormSW.ItemObject;
```

See the description of **ItemObject** for the definitions of the common fields of the **ItemObject** record. One use of a **tagOnly ItemObject** is to substitute for the **tag** of a **string** item. This is useful when the client wishes to present the illusion that the **tag** for an item is not on the same line as the item's body. (See also **LabelHandle**.)

sw This is the form subwindow that contains the item. It is automatically set by **Create**; clients should ignore it.

otherItem This is the index of the other item for which this item is acting as a tag. For a **TagOnlyHandle**, it must be the index of a **string** item (otherwise the **ERROR ItemError[notStringOtherItem, i]** will be generated by **Create**, where **i** is the index of the **tagOnly** item).

To allow a **tagOnly** to act as a substitute tag, no special trailer is appended to the tag. When a **tagOnly** item is used as a substitute tag, all of the user actions directed at its tag are redirected by **FormSW** to the **otherItem**. Because of this redirection, the notification procedures of the target **string** item are called with arguments identical to those provided by **FormSW** when the **string** item's tag is operated on by the user.

```
FormSW.Type: TYPE = {fixed, relative};
```

Type indicates whether the client controls the formatting of a form subwindow or whether **FormSW** automatically formats the window.

fixed The client specifies the location of each item in the form subwindow by specifying the **place** field of the **ItemObjects**.

relative **FormSW** arranges the items in the window automatically.

```
FormSW.WordBoolean: TYPE = RECORD [SELECT OVERLAID * FROM
  f1 => [b: BOOLEAN],
  f2 => [w: WORD],
  ENDCASE];
```

WordBoolean is an overlaid variant record provided for forcing a boolean to occupy its own word in memory. This is a requirement of any boolean to be used with a **boolean ItemObject**.

13.2 Constants and data objects

```
FormSW.defaultBoxWidth: CARDINAL = 0;
```

defaultBoxWidth indicates that the display box of an item should extend to the right edge of a subwindow or to the next item, whichever is closer.

```
FormSW.lineDiff: PRIVATE INTEGER = -1;

FormSW.line0: INTEGER = -3;

FormSW.line1: INTEGER = FormSW.line0 + FormSW.lineDiff;

FormSW.line2: INTEGER = FormSW.line1 + FormSW.lineDiff;

FormSW.line3: INTEGER = FormSW.line2 + FormSW.lineDiff;

FormSW.line4: INTEGER = FormSW.line3 + FormSW.lineDiff;

FormSW.line5: INTEGER = FormSW.line4 + FormSW.lineDiff;

FormSW.line6: INTEGER = FormSW.line5 + FormSW.lineDiff;

FormSW.line7: INTEGER = FormSW.line6 + FormSW.lineDiff;

FormSW.line8: INTEGER = FormSW.line7 + FormSW.lineDiff;

FormSW.line9: INTEGER = FormSW.line8 + FormSW.lineDiff;

FormSW.newLine: Window.Place = [0, FormSW.nextLine];
```

newLine specifies that this item should start on the next line down from the preceding item. It works even if there is no preceding item.

```
FormSW.nextLine: INTEGER = -2;
```

nextLine specifies that the **y** position for an item should be the next line after the **y** position of the preceding item.

```
FormSW.nextPlace: Window.Place = [-10, FormSW.sameLine];
```

nextPlace specifies that this item should be on the same line as the preceding one, and should start a little past where the previous one left off. This is subject to all of the caveats mentioned for negative **x**'s in the discussion of **places**.

```
FormSW.nullEnumeratedValue: UNSPECIFIED = LAST[CARDINAL];
```

An enumerated value can never have an unknown value (unless the client is not playing by the rules). The value given to an enumerated value when no value is chosen is **nullEnumeratedValue**. If an enumerated value has **nullEnumeratedValue**, the display of the item has nothing between the braces (for **one** feedback) or nothing selected (for **all** feedback). (See also **EnumeratedHandle**.)

```
FormSW.nullIndex: CARDINAL = LAST[CARDINAL];
```

nullIndex is used as an index in **SetSelection** or **SetTypeIn** when the client wants nothing selected or wants no insert point.

```
FormSW.nullItems: FormSW.ItemDescriptor = DESCRIPTOR[LONG[NIL], 0];
```

```
FormSW.sameLine: INTEGER = -1;
```

sameLine specifies that the **y** position for this item should be the same as the **y** position for the preceding item. If this is the first item, the ERROR **ItemError[illegalCoordinate**, ----] results.

13.3 Signals and errors

```
FormSW.Error: SIGNAL [code: FormSW.ErrorCode];
```

```
FormSW.ErrorCode: TYPE = {alreadyAFormSW, notAFormSW, other};
```

alreadyAFormSW a client has passed a form subwindow to the **Create** procedure.

notAFormSW a client has passed a subwindow that is not a form subwindow to the **Destroy** procedure.

other should never be raised.

```
FormSW.ItemError: SIGNAL [code: FormSW.ItemErrorCode, index: CARDINAL];
```

The **index** argument to **ItemError** is the index of the item that **FormSW** was processing when it discovered the error condition.

```
FormSW.ItemErrorCode: TYPE = {
  illegalCoordinate, notStringOtherItem, nilBackingStore, other};
```

illegalCoordinate	the client has made a error in specifying the layout of items in the form subwindow, such as not presenting the items in ascending order. Either an index has been skipped or the items are not ordered left to right, top to bottom. Another layout error is specifying a relative position for the first visible item in the subwindow, either using **sameLine** or a relative (negative) **x** value. Another layout error is specifying a relative (negative) **x** value for the item after an item that uses **defaultBoxWidth**.
notStringOtherItem	is raised if a **tagOnly** item refers to an item that is not a string item.
nilBackingStore	is raised if **NIL** has been passed as the pointer to the backing object for a **boolean, enumerated, longNumber, number**, or **string** item.
other	should never be raised.

13.4 Procedures

```
FormSW.Adjust: ToolWindow.AdjustProcType;
```

The **Adjust** procedure adjusts a subwindow if it is necessary to move the subwindow within the parent window or to change its size.

```
FormSW.AllocateItemDescriptor: PROCEDURE [
nItems: CARDINAL, z:UNCOUNTED ZONES ←NIL]
   RETURNS [FormSW.ItemDescriptor];
```

The **AllocateItemDescriptor** procedure allocates an ItemDescriptor for the **nItem** number of items from **z**. **z** is defaulted to the system heap.

```
FormSW.BooleanChoices: PROCEDURE RETURNS [FormSW.EnumeratedDescriptor];
```

The procedure **BooleanChoices** permits a tool to display a **BOOLEAN** choice without using the **boolean ItemObject**'s display conventions. It provides the **EnumeratedDescriptor** to be used in an **enumerated ItemObject** to display the enumerated values **TRUE** and **FALSE**.

```
FormSW.BooleanItem: PROCEDURE [
   tag: LONG STRING ← NIL,
   readOnly, invisible, drawBox, hasContext: BOOLEAN ← FALSE,
   place: Window.Place ← FormSW.nextPlace,
   proc: FormSW.NotifyProcType ← FormSW.NopNotifyProc,
   switch: LONG POINTER TO BOOLEAN,
   z: UNCOUNTED ZONES ←NIL]
   RETURNS [FormSW.BooleanHandle];
```

The procedure **BooleanItem** allocates a record of type **boolean ItemObject** from **z**. **z** is defaulted to the system heap. Such an item has a **FALSE clientOwnsItem**. It occupies a node large enough only for a **boolean ItemObject**, not for any **ItemObject**. For a discussion of the parameters, see **BooleanHandle**.

```
FormSW.CommandItem: PROCEDURE [
  tag: LONG STRING ← NIL,
  readOnly, invisible, drawBox, hasContext: BOOLEAN ← FALSE,
  place: Window.Place ← FormSW.nextPlace, proc: FormSW.ProcType,
  z: UNCOUNTED ZONES ←NIL]
  RETURNS [FormSW.CommandHandle];
```

The procedure **CommandItem** allocates a record of type **command ItemObject** from **z**. **z** is defaulted to the system heap. Such an item has a **FALSE clientOwnsItem**. It occupies a node large enough only for a **command ItemObject**, not for any **ItemObject**. For a discussion of the parameters, see **CommandHandle**.

```
FormSW.ContextFromItem: PROCEDURE [FormSW.ItemHandle] RETURNS [LONG POINTER];
```

The procedure **ContextFromItem** returns a pointer to the client data associated with an item.

```
FormSW.Create: PROCEDURE [
  sw: Window.Handle, clientItemsProc: FormSW.ClientItemsProcType,
  readOnlyNotifyProc: FormSW.ReadOnlyProcType ← FormSW.IgnoreReadOnlyProc,
  options: FormSW.Options ← [],
  initialState: ToolWindow.State ← active];
  zone: UNCOUNTED ZONE ← Heap.systemZone ];
```

The procedure **Create** creates a form subwindow. It can raise the errors **Error[alreadyAFormSW]**, and **ItemError[..., nilBackingStore, illegalCoordinate, notStringOtherItem, ...]**.

sw is the subwindow that is transformed into a form subwindow. If the subwindow is already a form subwindow, the **ERROR Error[alreadyAFormSW]** results.

clientItemsProc is called to get the **items**. If the **ItemDescriptor** was manufactured from the system heap, which can be done by calling **AllocateItemDescriptor**, then the client can have **FormSW** free it by returning a **TRUE freeDesc**.

readOnlyNotifyProc is called whenever the user attempts to modify an item with a **TRUE readOnly** flag. (See also **IgnoreReadOnlyProc** and **NopReadOnlyProc**).

options If a **type** = **relative**, then where and how the items and their associated data are displayed is automatically determined by the form subwindow.. If the client specifies a **type** of **fixed**, it must designate a subwindow place for each item to be displayed. It is the client's responsibility to avoid overlapping or overwriting items and their data. If **scrollVertical** is **TRUE**, a vertical scrollbar is provided. [Note: In the **relative** case the parameter items are simply displayed one per line. This implies that the height of a subwindow that would contain all of your parameters is = **n*LineHeight[]**.]

initialState	determines whether the form subwindow is awake when created. If **initialState** is not **active**, then the form subwindow is asleep. If **initialState** is **active**, then the **clientItemsProc** is called while still in **Create**.
zone	A heap can be passed to the **Create** procedure, from which storage will be allocated. The default heap is the system heap.

```
FormSW.Destroy: PROCEDURE [Window.Handle];
```

The **Destroy** procedure transforms a form subwindow back into an undifferentiated subwindow. If it is not currently a form subwindow, the ERROR **Error[notAFormSW]** results. (See also **IsIt**.)

```
FormSW.Display: PROCEDURE [w: Window.Handle, yOffset: CARDINAL ← 0];
```

The **Display** procedure allows a tool to redisplay the contents of the subwindow. Note that **Display** allows the tool to scroll, or unscroll, the items before the redisplay via the **yOffset**, which specifies the number of bits to offset the items upward.

```
FormSW.DisplayItem: PROCEDURE [sw: Window.Handle, index: CARDINAL];
```

The **DisplayItem** procedure is provided to allow a tool to redisplay the contents of an individual item. Redisplaying a single item may cause other items to also be redisplayed. **DisplayItem** must be called immediately if the client changes any of the **flags** that affect the way the item is displayed or if the client changes the backing store for the item. Such changes are not safe in an arbitrary pre-emption environment, as there is a potential race condition. (See also **ModifyEditable** and **ToggleVisibility**.)

```
FormSW.EnumeratedItem: PROCEDURE [
  tag: LONG STRING ← NIL,
  readOnly, invisible, drawBox, hasContext: BOOLEAN ← FALSE,
  place: Window.Place ← FormSW.nextPlace,
  feedback: FormSW.EnumeratedFeedback ← one,
  proc: FormSW.EnumeratedNotifyProcType ← FormSW.NopEnumeratedNotifyProc,
  copyChoices: BOOLEAN ← TRUE, choices: FormSW.EnumeratedDescriptor,
  value: LONG POINTER TO UNSPECIFIED,
  z: UNCOUNTED ZONES ← NIL]
  RETURNS [FormSW.EnumeratedHandle];
```

The procedure **EnumeratedItem** allocates a record of type **enumerated ItemObject** from **z**. **z** is defaulted to the system heap. Such an item has a FALSE **clientOwnsItem**. It occupies a node large enough only for an **enumerated ItemObject**, not for any **ItemObject**. The parameters are used to initialize the **ItemObject**. For a discussion of their meaning, see **EnumeratedHandle**.

```
FormSW.FindIndex: PROCEDURE [sw: Window.Handle, item: FormSW.ItemHandle] RETURNS
[CARDINAL];
```

FormSW assumes that there is a unique mapping between an item and an index into the **ItemDescriptor** for each subwindow. Given an item, the procedure **FindIndex** finds its index. (See also **FindItem**.)

```
FormSW.FindItem: PROCEDURE [
  sw: Window.Handle, index: CARDINAL] RETURNS [FormSW.ItemHandle];
```

FormSW assumes that there is a unique mapping between an item and an index into the **ItemDescriptor** for each subwindow. Given an index into an **ItemDescriptor**, the procedure **FindItem** finds its item. If **index** is too large (that is, does not correspond to an item in the subwindow's **ItemDescriptor**), **FindItem** returns **NIL**. (See also **FindIndex**.)

```
FormSW.FreeAllItems: PROCEDURE [sw: Window.Handle];
```

The procedure **FreeAllItems** deallocates all the items in a form subwindow. (See **FreeItem** for the semantics of deallocating an item.) Items are freed from the **UNCOUNTED ZONE** passed to the **Create** procedure.

```
FormSW.FreeItem: PROCEDURE [
  item: FormSW.ItemHandle, z: UNCOUNTED ZONE ← NIL] RETURNS [FormSW.ItemHandle];
```

The procedure **FreeItem** deallocates from **z** an item allocated by **FormSW** by one of the procedures **BooleanItem**, **CommandItem**, **EnumeratedItem**, **LabelItem**, **LongNumberItem**, **NumberItem**, **StringItem**, or **TagOnlyItem**. **z** is defaulted to the system heap.

If **item.clientOwnsItem** is **TRUE**, then for each item type, the following actions are taken:

enumerated	If **copyChoices** is **TRUE**, the **choices** are freed.
longNumber, number	The **ItemObject.string** is freed.
string	If **inHeap** is **TRUE**, the **ItemObject.string** is freed.
All other types	Nothing is freed.

The client must be very careful when using this procedure. It may deallocate the item that contains either the selection or insertion, in which case the client must guarantee there will be no references to either. It is considerably safer to deallocate all of the items at once. (See **FreeAllItems**.)

```
FormSW.GetSelection: PROCEDURE [
  Window.Handle] RETURNS [index: CARDINAL, first, last: CARDINAL];
```

The **GetSelection** procedure allows a tool to get the currently selected item. **index** is the index of the form item containing the current selection. If **nullIndex** is returned, then there is no current selection. The current selection is described using the character positions **first** and **last**. These positions are relative to a zero origin, which is to the left of the first character of the tag (or main body of the item, if there is no tag). The interval is half open (i.e., **first** = **last** = 0 is an empty selection, and **first** = 0, **last** = 1 is a selection containing the first character in the item).

```
FormSW.GetTypeIn: PROCEDURE [
  Window.Handle] RETURNS [index: CARDINAL, position: CARDINAL];
```

The **GetTypeIn** procedure allows a tool to get the item containing the insert point. **position** indicates the number of characters to the left of the insertion point. This position is relative to a zero origin, which is to the left of the first character of the tag (or main body of

the item, if there is no tag). **index** is the index of the form item containing the insertion point. If **nullindex** is returned, then there is no insertion point.

```
FormSW.IgnoreReadOnlyProc: FormSW.ReadOnlyProcType;
```

The **IgnoreReadOnlyProc** procedure blinks the display when called.

```
FormSW.IndexFromEnumeratedValue: PROCEDURE [
  FormSW.EnumeratedHandle] RETURNS [CARDINAL];
```

The **IndexFromEnumeratedValue** procedure returns the index into **choices** of the current value of **enumerated ItemObject**.

```
FormSW.InHeapFreeHintsProc: FormSW.FreeHintsProcType;
```

The **InHeapFreeHintsProc** procedure is a **FreeHintsProcType** that assumes the **hints** are from the system heap and returns them there. If the **hints** are not from the system heap, then the client should supply its own **FreeHintsProc**.

```
FormSW.IsIt: PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
```

The **IsIt** procedure returns **TRUE** if **sw** is a form subwindow and **FALSE** otherwise.

```
FormSW.IsItemInverted: PROCEDURE [
  sw: Window.Handle, index: CARDINAL] RETURNS [yes: BOOLEAN];
```

This procedure is not currently implemented.

```
FormSW.LabelItem: PROCEDURE [
  tag: LONG STRING ← NIL,
  readOnly, invisible, drawBox, hasContext: BOOLEAN ← FALSE,
  place: Window.Place ← FormSW.nextPlace,
  z: UNCOUNTED ZONE ← NIL ]
  RETURNS [FormSW.LabelHandle];
```

The procedure **LabelItem** allocates a record of type **tagOnly ItemObject** from **z**. **z** is defaulted to the system heap with has a **FALSE clientOwnsItem**. It occupies a node large enough only for a **boolean ItemObject**, not for any **ItemObject**. For a discussion of the parameters, see **LabelHandle**.

```
FormSW.LineHeight: PROCEDURE [sw: Window.Handle ← NIL] RETURNS [CARDINAL];
```

The height of a line can be determined by calling **LineHeight**, which accounts for all fudge factors added to the **fontHeight**. The parameter **sw** is ignored.

```
FormSW.LineN: PROCEDURE [n: CARDINAL] RETURNS [INTEGER];
```

The procedure **LineN** takes a line number and returns the appropriate negative **y** for use as a **place** parameter. This is helpful for calculating where the next item should be positioned in the form subwindow.

```
FormSW.LongNumberItem: PROCEDURE [
  tag: LONG STRING ← NIL,
  readOnly, invisible, drawBox, hasContext: BOOLEAN ← FALSE,
  place: Window.Place ← FormSW.nextPlace,
  signed: BOOLEAN ← TRUE, notNegative: BOOLEAN ← FALSE,
  radix: FormSW.Radix ← decimal, boxWidth: CARDINAL [0..256) ← 64,
  proc: FormSW.LongNumberNotifyProcType ←
   FormSW.NopLongNumberNotifyProc,
  default: LONG UNSPECIFIED ← LAST[LONG INTEGER],
  value: LONG POINTER TO LONG UNSPECIFIED,
  bias: INTEGER ← 0, z: UNCOUNTED ZONE ← NIL]
  RETURNS [FormSW.LongNumberHandle];
```

The procedure **LongNumberItem** allocates a record of type **longNumber ItemObject** from **z**. **z** is defaulted to the system heap. **clientOwnsItem** is defaulted to **FALSE**. It occupies a node large enough only for a **longNumber ItemObject**. (For a discussion of the parameters, see **LongNumberHandle**.) **bias** is the difference between what **value** points to and what is displayed. (Displayed number + **bias** = **value**↑.)

```
FormSW.MarkItem: PROCEDURE [
  sw: Window.Handle, index: CARDINAL, action: TextData.MarkingAction,
  mode: TextData.SelectionMode];
```

This procedure is not currently implemented.

```
FormSW.MinHeight: PROCEDURE [
  items: FormSW.ItemDescriptor, type: FormSW.Type] RETURNS [CARDINAL];
```

The procedure **MinHeight** returns the minimum height a form subwindow would need to display **items**. The form subwindow that displays **items** need not exist when this procedure is called. (See also **NeededHeight**).

```
FormSW.ModifyBoolean: PROCEDURE [
  sw: Window.Handle, index: CARDINAL, mark: BOOLEAN, notify: BOOLEAN];
```

This procedure is not currently implemented.

```
FormSW.ModifyCommand: PROCEDURE [
  sw: Window.Handle, index: CARDINAL, mark: BOOLEAN, notify: BOOLEAN];
```

This procedure is not currently implemented.

```
FormSW.ModifyEditable: PROCEDURE [
  sw: Window.Handle, index: CARDINAL, position, length: CARDINAL,
  new: LONG STRING ← NIL, keepTrash: BOOLEAN ← FALSE];
```

The best way to modify the backing store of an editable item (i.e., one of type **string**, **number**, or **longNumber**) is to call **ModifyEditable**, which changes the backing store and the display as little and as quickly as possible. **position** is the left end of the text in the item's body that is to be changed. The zero position is to the left of the first character of the main body of the item. If **new** is **NIL**, then the modification is a deletion; otherwise, if **length** is 0, it is an insertion. If **length** is non-zero, the modification is a replacement. In all

cases, the removed characters are discarded unless **keepTrash** is TRUE, in which case they become the current contents of the global trash bin. (See the **Selection** interface for a discussion of the trash bin.) The item to be modified cannot be **readOnly**.

```
FormSW.ModifyEnumerated: PROCEDURE [
  sw: Window.Handle, index: CARDINAL, mark: BOOLEAN, notify: BOOLEAN, newValue:
  UNSPECIFIED];
```

This procedure is not currently implemented.

```
FormSW.NeededHeight: PROCEDURE [
  Window.Handle] RETURNS [min, current: CARDINAL];
```

A tool often needs to know how high a form subwindow should be to display all items. There are two heights of interest: the minimum height for the subwindow is the height if none of the textual item types (i.e., **longNumber**, **number**, **string**, **source**) overflow a single line; the current height is the true height of the subwindow, accounting for overflowing items. These are returned by **NeededHeight** as **min** and **current**, respectively. **NeededHeight** requires that the form subwindow already exist. (See also **MinHeight**.)

```
FormSW.NopEnumeratedNotifyProc: FormSW.EnumeratedNotifyProcType;
```

NopNotifyProc is a **EnumeratedNotifyProcType** that does nothing when called.

```
FormSW.NopFreeHintsProc: FormSW.FreeHintsProcType;
```

NopFreeHintsProc is a **FreeHintsProcType** that does nothing when called. It is appropriate if the **hints** are in the client's global frame.

```
FormSW.NopLongNumberNotifyProc: FormSW.LongNumberNotifyProcType;
```

NopLongNumberNotifyProc is a **LongNumberNotifyProcType** that does nothing when called.

```
FormSW.NopNotifyProc: FormSW.NotifyProcType;
```

NopNotifyProc is a **NotifyProcType** that does nothing when called.

```
FormSW.NopNumberNotifyProc: FormSW.NumberNotifyProcType;
```

NopNumberNotifyProc is a **NumberNotifyProcType** that does nothing when called.

```
FormSW.NopReadOnlyProc: FormSW.ReadOnlyProcType;
```

NopReadOnlyProc is a **ReadOnlyProcType** that does nothing when called.

```
FormSW.NumberItem: PROCEDURE [
  tag: LONG STRING ← NIL,
  readOnly, invisible, drawBox, hasContext: BOOLEAN ← FALSE,
  place: Window.Place ← FormSW.nextPlace, signed: BOOLEAN ← TRUE,
  notNegative: BOOLEAN ← FALSE, radix: FormSW.Radix ← decimal,
  boxWidth: CARDINAL [0..128) ← 64,
  proc: FormSW.NumberNotifyProcType ← FormSW.NopNumberNotifyProc,
```

```
    default: UNSPECIFIED ← LAST[INTEGER], value: LONG POINTER TO UNSPECIFIED,
    bias : INTEGER ← 0, z: UNCOUNTED ZONE ← NIL]
    RETURNS [FormSW.NumberHandle];
```

The procedure **NumberItem** allocates a record of type **number ItemObject** from **z**. **z** is defaulted to the system heap. **clientOwnsItem** is set to **FALSE**. It occupies a node large enough only for a **number ItemObject**. (For a discussion of the parameters, see **NumberHandle**.) **bias** is the difference between what **value** points to and what is displayed. (Displayed number + **bias** = **value** ↑ .)

```
FormSW.RedisplayItem: PROCEDURE [
    sw: Window.Handle, index: CARDINAL, sameSize: BOOLEAN];
```

This procedure is not currently implemented.

```
FormSW.SetCurrent: PROCEDURE [sw: Window.Handle, index: CARDINAL];
```

The **SetCurrent** procedure is equivalent to **SetSelection**, with **first** and **last** selecting the non-tag and trailer portion of the item. It also places the insert point at the item's end.

```
FormSW.SetModifyNotificationProc: PROCEDURE [
    sw: Window.Handle, proc: FormSw.ProcType];
```

The **SetModifyNotificationProc** allows the client to have a procedure that is called when the form subwindow has been modified. The procedure **Proc** should reset the **modified** bit by calling **FormSW.ToggleFlag [modified]**.

```
FormSW.SetOptions: PROCEDURE [sw: Window.Handle, options: FormSW.Options];
```

The procedure **SetOptions** changes the current **Options** for the subwindow **sw**.

```
FormSW.SetSelection: PROCEDURE [
    sw: Window.Handle, index: CARDINAL, first, last: CARDINAL];
```

The procedure **SetSelection** allows a tool to set the current selection to one of the items in the form subwindow. (See the **Selection** interface for a discussion of the current selection.) This procedure should be used judiciously to avoid pre-empting the user. **index** is the index of the form item containing the current selection. **nullIndex** is used as an index when the client wants "nothing" selected. The new selection is delimited by the character positions **first** and **last**. These positions are relative to a zero origin, which is to the left of the first character of the tag (or main body of the item, if there is no tag). The interval is half open, (i.e., **first** = **last** = 0 is an empty selection, and **first** = 0, **last** = 1 is a selection containing the first character in the item).

```
FormSW.SetTagPlaces: PROCEDURE [
    items: FormSW.ItemDescriptor,
    tabStops: LONG DESCRIPTOR FOR ARRAY OF CARDINAL, bitTabs: BOOLEAN];
```

It is often desirable for items on different lines to have the same horizontal positions. The **SetTagPlaces** procedure simplifies this task . The **tabStops** are in raster points if **bitTabs** is **TRUE**; otherwise, they are multiplied by the width of the digit 0. A positive **x** is used as a zero-origin index into the **tabStops** array. If the **place** is **nextPlace**, it means "move to the

next tab stop". Negative **x**'s are ignored. This routine is a pre-processor that changes the items' places; it should be called before giving the items to the **FormSW** package.

```
FormSW.SetTypeIn: PROCEDURE [
  sw: Window.Handle, index: CARDINAL, position: CARDINAL];
```

The procedure **SetTypeIn** allows a tool to set the insert point of the window to a location in an item. (See the **Selection** interface for a discussion of the insert point.) It should be used judiciously to avoid pre-empting the user. **index** is the index of the form item containing the insertion point. **nullIndex** is used as an index when the client wants no insert point. **position** indicates the number of characters to the left of the new insertion point. The zero position is to the left of the first character of the tag (or main body of the item, if there is no tag).

```
FormSW.SkipToNext: PROCEDURE [sw: Window.Handle];
```

SkipToNext implements the **Next** function. If a client notification procedure wants to implement a synonym for the **Next** function, it should call **SkipToNext**.

```
FormSW.Sleep: PROCEDURE [Window.Handle];
```

If a tool window is being made tiny, its subwindows do not need to keep state information for display. A form subwindow can be told to discard such state data by calling **Sleep**. This is done automatically if using the **Tool** interface. (See also **Wakeup**.)

```
FormSW.SourceEditProc: FormSW.FilterProcType;
```

This procedure is not currently implemented.

```
FormSW.SourceItem: PROCEDURE [
  tag: LONG STRING ← NIL,
  readOnly, invisible, drawBox, hasContext, inHeap: BOOLEAN ← FALSE,
  place: Window.Place ← FormSW.nextPlace,
  boxWidth: CARDINAL ← FormSW.defaultBoxWidth,
  filterProc: FormSW.FilterProcType ← FormSW.SourceEditProc,
  menuProc: FormSW.MenuProcType ← FormSW.VanillaMenuProc,
  source: TextSource.Handle, z: UNCOUNTED ZONE ← NIL]
  RETURNS [FormSW.SourceHandle];
```

This procedure is not currently implemented.

```
FormSW.StringEditProc: FormSW.FilterProcType;
```

The **StringEditProc** procedure is the standard editing procedure provided by Tajo for editing **string ItemObject**.

```
FormSW.StringItem: PROCEDURE [
  tag: LONG STRING ← NIL,
  readOnly, invisible, drawBox, hasContext, inHeap: BOOLEAN ← FALSE,
  place: Window.Place ← FormSW.nextPlace,
  feedback: FormSW.StringFeedback ← normal,
  boxWidth: CARDINAL ← FormSW.defaultBoxWidth,
  filterProc: FormSW.FilterProcType ← FormSW.StringEditProc,
```

```
menuProc: FormSW.MenuProcType ← FormSW.VanillaMenuProc,
string: LONG POINTER TO LONG STRING, z: UNCOUNTED ZONE ← NIL]
RETURNS [FormSW.StringHandle];
```

The procedure **StringItem** allocates a record of type **string ItemObject** from **z**. **z** is defaulted to the system heap. Such an item has a **FALSE clientOwnsItem**. It occupies a node large enough only for a **string ItemObject**, not for any **ItemObject**. (For a discussion of the parameters, see **StringHandle**.)

```
FormSW.TagOnlyItem: PROCEDURE [
  tag: LONG STRING ← NIL,
  readOnly, invisible, drawBox, hasContext: BOOLEAN ← FALSE,
  place: Window.Place ← FormSW.nextPlace,
  otherItem: CARDINAL ← FormSW.nullIndex, z: UNCOUNTED ZONE ← NIL]
  RETURNS [FormSW.TagOnlyHandle];
```

The procedure **TagOnlyItem** allocates a record of type **tagOnly ItemObject** from **z**. **z** is defaulted to the system heap. Such an item has a **FALSE clientOwnsItem**. It occupies a node large enough only for a **tagOnly ItemObject**, not for any **ItemObject**. (For a discussion of the parameters, see **TagOnlyHandle**.)

```
FormSW.ToggleFlag: PROCEDURE [
    sw: Window.Handle, index: CARDINAL,
    flag: FormSW.Flag];
```

The procedure **ToggleFlag** toggles the **flag** of **index**'s item.

```
FormSW.ToggleVisibility: PROCEDURE [sw: Window.Handle, index: CARDINAL];
```

The **ToggleVisibility** procedure changes the visibility of an item from visible to invisible. It minimizes the necessary repainting when the item's visibility is changed. In addition, the procedure deals properly with making the item invisible when it contains the current selection or insertion point. (See the **Selection** interface for a discussion of the current selection and insertion point.) **sw** is the form subwindow containing the item, and **index** is the index of the item in the subwindow's **ItemDescriptor**.

```
FormSW.VanillaMenuProc: FormSW.MenuProcType;
```

The **VanillaMenuProc** procedure is a **MenuProcType** for which **BASE**[**hints**] = **NIL**, implying that no prompt menu will be available to the user.

```
FormSW.Wakeup: PROCEDURE [Window.Handle];
```

If the tool window is being made tiny, its subwindows do not need to keep state information for display. A form subwindow can be told to recreate the display state (when the window becomes big) by calling **Wakeup**. This is done automatically if using the **Tool** interface. (See also **Sleep**.)

14

MsgSW

The **MsgSW** interface implements message subwindows. Message subwindows provide a simple way of posting messages to the user. Typical tools have a message subwindow as their first subwindow. See `ExampleTool.mesa` in Appendix A. A Message subwindow is built on a String subwindow (see **StringSW**).

14.1 Types

```
MsgSW.Severity: TYPE = {info, warning, fatal};
```

Every message subwindow has a **Severity** associated with it, which is the **Severity** of the latest message sent to it by **MsgSW.Post** or **MsgSW.PostAndLog** if **prefix** is **TRUE**. Messages of severity **warning** are prefaced by "Warning: ", and messages of severity **fatal** are prefaced by "Fatal Error: ".

14.2 Constants and data objects

```
MsgSW.defaultOptions: TextSW.Options = [
  access: append, menu: TRUE, split: TRUE, wrap: TRUE, scrollbar: TRUE,
  flushTop: FALSE, flushBottom: FALSE];
```

defaultOptions are the default window options used in creating a message subwindow.

14.3 Signals and errors

```
MsgSW.Error: SIGNAL [code: MsgSW.ErrorCode];
```

```
MsgSW.ErrorCode: TYPE = {appendOnly, notAMsgSW, other};
```

appendOnly is raised by **MsgSW.Create** if **options.access** is not append.

notAMsgSW is raised when a client performs a **MsgSW** operation on a window that is not a message subwindow.

14.4 Procedures

```
MsgSW.AppendString: UserInput.StringProcType;
```

The **AppendString** procedure appends the parameter **string** onto the latest message. This is the procedure used for **UserInput.StringOut**. The **Severity** associated with **sw** is set to **info**. This procedure can raise **Error[notAMsgSW]**.

```
MsgSW.Clear: PROCEDURE [sw: Window.Handle];
```

The **Clear** procedure erases the contents of the message subwindow. The **Severity** associated with **sw** is set to **info**. This procedure can raise **Error[notAMsgSW]**.

```
MsgSW.Create: PROCEDURE [
  sw: Window.Handle, lines: CARDINAL ← 1,
  options: TextSW.Options ← MsgSW.defaultOptions];
```

The **Create** procedure creates a message subwindow from an ordinary subwindow. The **lines** parameter specifies the minimum number of lines that the subwindow will keep in its backing store before discarding the oldest line. The subwindow height controls how many lines will be visible. If the number of lines visible to the user is greater than **lines**, then all the visible lines are kept in the backing store. When the **options.access** parameter is anything but **append**, an **Error** is raised with a code of **appendOnly**. Subwindows created by **MsgSW.Create** should be destroyed by **MsgSW.Destroy**, not by **TextSW.Destroy**.

```
MsgSW.Destroy: PROCEDURE [sw: Window.Handle];
```

The **Destroy** procedure destroys the backing store and transforms the message subwindow into an ordinary subwindow. This procedure can raise **Error[notAMsgSW]**.

```
MsgSW.GetSeverity: PROCEDURE [w: Window.Handle] RETURNS [severity: MsgSW.Severity];
```

The **GetSeverity** procedure returns the severity associated with the message subwindow **sw**. This is either the severity of the last message sent to the subwindow or the severity set by **SetSeverity**, whichever happened last. This procedure can raise **Error[notAMsgSW]**.

```
MsgSW.IsIt: PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
```

The **IsIt** procedure returns **TRUE** if and only if **sw** is a message subwindow.

```
MsgSW.LastLine: PROCEDURE [sw: Window.Handle, ss: String.SubString];
```

The **LastLine** procedure fills in the parameter **ss** with the base, offset, and length of the current message. The client may want to copy **ss** and the string **ss.base**, because this information may change. This procedure can raise **Error[notAMsgSW]**.

```
MsgSW.Post: PROCEDURE [
  sw: Window.Handle, string: LONG STRING, severity: MsgSW.Severity ← info,
  prefix: BOOLEAN ← TRUE, endOfMsg: BOOLEAN ← TRUE];
```

The **Post** procedure appends **string** onto the latest message. The severity of the message is **severity**. If the **prefix** parameter is **TRUE** and the message is starting a new line, a short string that depends on **severity** (**info**: "", **warning**: "Warning: " or **fatal**: "Fatal Error: ")

starts the line before the client message. The **endOfMsg** parameter set to **TRUE** delimits the message without having to put an **Ascii.CR** in **string**. (See also **PostAndLog**.) This procedure can raise **Error[notAMsgSW]**.

```
MsgSW.PostAndLog: PROCEDURE [
  sw: Window.Handle, string: LONG STRING, severity: MsgSW.Severity ← info,
  prefix: BOOLEAN ← TRUE, endOfMsg: BOOLEAN ← TRUE, logSW: Window.Handle ← NIL];
```

The **PostAndLog** procedure acts like **MsgSW.Post**. In addition, the **logSW** parameter enables the same message appearing in the message subwindow to be directed to another subwindow for logging. If the value is **NIL**, the output is directed to the default **Put** window and the tool's name is prefixed to the message. (See also **Post**.) This procedure can raise **Error[notAMsgSW]**.

```
MsgSW.SetSeverity: PROCEDURE [
  sw: Window.Handle, severity: MsgSW.Severity ← info];
```

The **SetSeverity** procedure sets the severity associated with the message subwindow **sw**. This procedure can raise **Error[notAMsgSW]**.

15

ScratchSW

The **ScratchSW** interface creates a subwindow that is backed by a scratch source; that is, by a piece of virtual memory. It should be used when an editable window not backed by a file is desired. An example of the use of **ScratchSW** is for the implementation of an empty file window. (See also **ScratchSource**.)

15.1 Types

```
ScratchSW.Options: TYPE = TextSW.Options;
```

15.2 Constants and data objects

```
ScratchSW.defaultOptions: ScratchSW.Options = [
  access: edit, menu: TRUE, split: TRUE, wrap: TRUE, scrollbar: TRUE,
  flushTop: FALSE, flushBottom: FALSE];
```

15.3 Signals and errors

None.

15.4 Procedures

```
ScratchSW.Create: PROCEDURE [
  sw: Window.Handle, block: Environment.Block ← Environment.nullBlock,
  extraRoom: CARDINAL ← 0, expandable: BOOLEAN ← TRUE,
  options: ScratchSW.Options ← ScratchSW.defaultOptions];
```

The **Create** procedure creates a scratch subwindow. **sw** is the ordinary window from which the scratch subwindow is created. If **sw** is **NIL**, the signal **windowIsNil** is generated from the **Context** interface; it is not caught by **ScratchSW**. **block** is the initialized storage that is used to back the subwindow. **extraRoom** is the amount of storage beyond the end of block that the scratch subwindow can use. If **expandable** is **FALSE** and the scratch subwindow runs out of room in the block, editing operations have no effect. If **expandable** is **TRUE**, the scratch subwindow allocates another larger block when it runs out of room, copies the old block into it, and deallocates the old block (see **ScratchSource**). In this case, the block must have been allocated from **MSegment.GetPages**, and the block is dealloctated by**ScratchSW**

when the subwindow is destroyed. **options** indicates the initial value of the subwindow's **Options**. Subwindows created by **ScratchSW.Create** should be destroyed by **ScratchSW.Destroy**, not by **TextSW.Destroy**, since **TextSW.Destroy** is called from within **ScratchSW.Destroy**.

```
ScratchSW.Destroy: PROCEDURE [sw: Window.Handle];
```

The **Destroy** procedure destroys a scratch subwindow that was created by **ScratchSW.Create**, turning it back into an ordinary subwindow. If **sw** is **NIL**, then no errors or signals are generated and no actions are performed.

```
ScratchSW.Info: PROCEDURE [sw: Window.Handle]
  RETURNS [ block: Environment.Block, extraRoom: CARDINAL,
  expandable: BOOLEAN, options: ScratchSW.Options];
```

The **Info** procedure returns the block backing the scratch subwindow, how much extra room there is after the block, whether the block is expandable, and the current value of the subwindow options. If **sw** is **NIL**, then the returned values are:

```
[block: Environment.nullBlock, extraRoom: 0,
expandable: FALSE, options: ScratchSW.defaultOptions].
```

```
ScratchSW.IsIt: PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
```

The **IsIt** procedure returns **TRUE** if the **Handle** is a scratch subwindow and **FALSE** otherwise. If **sw** is **NIL**, then **ScratchSW.IsIt** returns **FALSE**.

16

StringSW

The **StringSW** interface provides the definitions and procedures to create and manipulate text subwindows whose backing store is a **LONG STRING**. (See **TextSW** for more information.)

16.1 Types

```
StringSW.Options: TYPE = TextSW.Options;
```

16.2 Constants and data objects

```
StringSW.defaultOptions: ScratchSW.Options = [
  access: edit, menu: TRUE, split: TRUE, wrap: TRUE, scrollbar: TRUE,
  flushTop: FALSE, flushBottom: FALSE];
```

defaultOptions are the default window options used in creating a string subwindow.

16.3 Signals and errors

```
StringSW.DoesNotExist: SIGNAL;
```

The signal **DoesNotExist** is never raised. It is deleted when the interface is next changed.

16.4 Procedures

```
StringSW.Create: PROCEDURE [
  sw: Window.Handle, s: LONG POINTER TO LONG STRING ← NIL,
  options: StringSW.Options ← StringSW.defaultOptions,
  expandable: BOOLEAN ← TRUE];
```

The **Create** procedure creates a string subwindow. **expandable** indicates whether the string is automatically expandable by the string window implementation. If **s** is **NIL**, **expandable** is forced to be **TRUE**. If **s↑** is **NIL** and **expandable** is **TRUE** or **s** is **NIL**, the subwindow will allocate and manage a heap string for the backing store. Expandable strings must be allocated from the system heap. If expandable is **FALSE** and **s** is not **NIL**, the client is responsible for the storage management of the string. If expandable is **FALSE** and

the string source runs out of room in the string, String.StringBoundsFault[ps] is raised. Subwindows created by StringSW.Create should be destroyed using StringSW.Destroy, not TextSW.Destroy, because StringSW.Destroy calls TextSW.Destroy.

```
StringSW.Destroy: PROCEDURE [sw: Window.Handle];
```

The **Destroy** procedure destroys a string subwindow created by StringSW.Create, turning it back into an ordinary subwindow.

```
StringSW.GetString: PROCEDURE [w: Window.Handle] RETURNS [s: LONG POINTER TO LONG STRING];
```

The **GetString** procedure returns the current backing string for a string subwindow.

```
StringSW.Info: PROCEDURE [sw: Window.Handle] RETURNS [
  s: LONG POINTER TO LONG STRING, options: StringSW.Options, expandable: BOOLEAN];
```

The **Info** procedure returns the current backing string for a string subwindow, whether the string is expandable, and the current value of the subwindow options.

```
StringSW.IsIt: PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
```

The **IsIt** procedure returns TRUE if the **Handle** is a string subwindow and FALSE otherwise.

17

TextSW

The **TextSW** interface defines a comprehensive set of facilities for viewing text independent of the source. It takes a client-created subwindow and text source, creates the necessary data structures, and then provides appropriate procedures for viewing, scrolling, and text selection. Throughout this chapter, a *display region* is either a subwindow or one of its splits. Splits are horizontal subregions created by means of the **SplitMCR**.

17.1 Types

```
TextSW.Access: TYPE = TextSource.Access; -- {read, append, edit}

TextSW.Bounds: TYPE = RECORD [from, to: TextSource.Position, delta: LONG INTEGER];

TextSW.InvalidRegions: TYPE = LONG POINTER TO TextSW.InvalidList;

TextSW.InvalidList: TYPE = RECORD [
  length: CARDINAL,
  seq: SEQUENCE maxLength: CARDINAL OF TextSW.Bounds];
```

A **TextSW.InvalidRegions** is returned by the client procedure passed to **TextSW.ModifySource**. It describes the regions in the source that have been modified so that **TextSW** can update its display region accordingly. **TextSW.Bounds** describes a single region where the source was modified. **from** and **to** are the positions in the source where modifications were made, resulting in a change in length of **delta** in the source.

```
TextSW.OnOff: TYPE = {on, off};

TextSW.Options: TYPE = RECORD [
  access: TextSW.Access, menu: BOOLEAN, split: BOOLEAN, wrap: BOOLEAN,
  scrollbar: BOOLEAN, flushTop: BOOLEAN, flushBottom: BOOLEAN];
```

menu indicates whether to instantiate the standard text operations menu with the subwindow at create time. **split** indicates whether to allow the subwindow to be divided into an arbitrary number of *splits* or horizontal subregions. **wrap** indicates whether a line too long to fit across the subwindow should be broken at a word boundary and continued on the next line or be clipped at the subwindow boundary. **scrollbar** indicates whether the

subwindow should have a vertical scrollbar. **flushTop** indicates whether the standard border should be supplied at the top of the subwindow. **flushBottom** indicates whether the standard border should be supplied at the bottom of the subwindow.

```
TextSW.SplitInfoProcType: TYPE = PROCEDURE [
  first, last: TextSource.Position, nLines: CARDINAL] RETURNS [BOOLEAN];
```

17.2 Constants and data objects

```
TextSW.defaultOptions: TextSW.Options = [
  access: read, menu: TRUE, split: TRUE, wrap: TRUE, scrollbar: TRUE,
  flushTop: FALSE, flushBottom: FALSE];
```

17.3 Signals and errors

```
TextSW.DoesNotExist: SIGNAL;
```

17.4 Procedures

```
TextSW.Adjust: ToolWindow.AdjustProcType;
```

The **Adjust** procedure is called when a text subwindow is moved or sized.

```
TextSW.BlinkingCaret: PROCEDURE [sw: Window.Handle, state: TextSW.OnOff];
```

The **BlinkingCaret** procedure enables or disables the blinking caret for an append or edit text subwindow.

```
TextSW.Create:PROCEDURE [
  sw: Window.Handle, source: TextSource.Handle, sink: TextSink.Handle ← NIL,
  options: TextSW.Options ← TextSW.defaultOptions, position: TextSource.Position ← 0,
  allowTypeIn: BOOLEAN ← TRUE, resetLengthOnNewSession: BOOLEAN ← FALSE];
```

The **Create** procedure creates a text subwindow from an ordinary subwindow. **position** indicates the initial character position in **source** that should be displayed at the top of the subwindow. If **sink** is NIL, an **ASCII** sink is used as a sink.

```
TextSW.DeleteText: PROCEDURE [
  sw: Window.Handle, pos: TextSource.Position, count: LONG CARDINAL,
  keepTrash: BOOLEAN ← TRUE];
```

The **DeleteText** procedure allows the client to alter the contents of the text source currently being displayed in the text subwindow by deleting **count** positions starting at **pos**. **keepTrash** determines whether the deleted text is placed in the trashbin. (See the **Selection** interface for documentation of the trashbin.) The text subwindow and source must have either edit or append access to use this operation correctly .

```
TextSW.Destroy: PROCEDURE [sw: Window.Handle];
```

The **Destroy** procedure destroys a text subwindow, freeing all data structures. However, the client-supplied source is not destroyed. Attempting to destroy a non-text subwindow

results in no action. This procedure should not be used to destroy "differentiated" subwindows (subwindows created by interfaces such as **FileSW** or **StringSW**) because auxiliary data structures may not be recorded in the subwindow's context object and hence would be lost. An example of such a data structure is the backing string for a **StringSW** when it is allocated by Tajo rather than by the client. Such subwindows should be destroyed by calling the appropriate routine in the interface for that subwindow type.

```
TextSW.DoEditAction:PROCEDURE [
  sw: Window.Handle, action: TextSource.EditAction] RETURNS [delta: LONG INTEGER];
```

The **DoEditAction** procedure deletes characters in the source according to **action**. The characters are deleted starting at and preceding the current insertion point. **delta** is always non-negative.

```
TextSW.EnumerateSecondarySelections: PROCEDURE [
  sw: Window.Handle, proc: PROCEDURE [TextData.Selection] RETURNS [BOOLEAN]];
```

The **EnumerateSecondarySelections** procedure enumerates the secondary selections of a text subwindow, calling **proc** for each one. These will have been defined by previous **SecondarySelectionFromPosition** and **SetSecondarySelection** calls.

```
TextSW.EnumerateSplits: PROCEDURE [
  sw: Window.Handle, proc: TextSW.SplitInfoProcType];
```

The **EnumerateSplits** procedure enumerates the splits of a text subwindow. Note that a text subwindow always has at least one split.

```
TextSW.FindMCR: Menu.MCRType;
```

The **FindMCR** procedure implements the **Find** command of the **TextOps** menu. It uses the current selection as the text to find. If the current selection is contained in this display region, it searches from that position; otherwise, it uses the current top of the region. This procedure allows clients to construct their own menus.

```
TextSW.ForceOutput: PROCEDURE [sw: Window.Handle];
```

All output to text subwindows is buffered for efficiency. The **ForceOutput** procedure ensures that all pending output has made it to the source.

```
TextSW.GetEOF: PROCEDURE [sw: Window.Handle] RETURNS [TextSource.Position];
```

The **GetEOF** procedure obtains the "end-of-file" position of a text subwindow.

```
TextSW.GetInsertion: PROCEDURE [sw: Window.Handle] RETURNS [TextSource.Position];
```

The **GetInsertion** procedure obtains the insertion position of a text subwindow.

```
TextSW.GetOptions: PROCEDURE [
  sw: Window.Handle] RETURNS [options: TextSW.Options];
```

The **GetOptions** procedure returns the current options setting for a text subwindow.

```
TextSW.GetPosition: PROCEDURE [
  sw: Window.Handle, line: CARDINAL] RETURNS [TextSource.Position];
```

The **GetPosition** procedure determines the position of the first character on **line**.

```
TextSW.GetSelection: PROCEDURE [
  sw: Window.Handle] RETURNS [left, right: TextSource.Position];
```

The **GetSelection** procedure obtains the selection position of a text subwindow.

```
TextSW.GetSource: PROCEDURE [
  sw: Window.Handle] RETURNS [source: TextSource.Handle];
```

The **GetSource** procedure returns the text source backing a text subwindow.

```
TextSW.InsertBlock: PROCEDURE [
  sw: Window.Handle, block: Environment.Block,
  pos: TextSource.Position ← TextSource.nullPosition];
```

The **InsertBlock** procedure allows the client to alter the contents of the text source currently being displayed in the text subwindow by inserting the block **block** at position **pos**. If **pos** is **nullPosition**, the block is inserted at the current insertion position. The text subwindow and source must have either edit or append access to correctly use this operation.

```
TextSW.InsertChar: PROCEDURE [
  sw: Window.Handle, char: CHARACTER,
  pos: TextSource.Position ← TextSource.nullPosition];
```

The **InsertChar** procedure allows the client to alter the contents of the text source currently being displayed in the text subwindow by inserting the character **char** at position **pos**. If **pos** is **nullPosition**, the character is inserted at the current insertion position. The text subwindow and source must have either edit or append access to correctly use this operation.

```
TextSW.InsertString: PROCEDURE [
  sw: Window.Handle, s: LONG STRING,
  pos: TextSource.Position ← TextSource.nullPosition];
```

The **InsertString** procedure allows the client to alter the contents of the text source currently being displayed in the text subwindow by inserting the string **s** at position **pos**. If **pos** is **nullPosition**, the string is inserted at the current insertion position. The text subwindow and source must have either edit or append access to correctly use this operation.

```
TextSW.InsertSubString: PROCEDURE [
  sw: Window.Handle, ss: String.SubString,
  pos: TextSource.Position ← TextSource.nullPosition];
```

The **InsertSubString** procedure allows the client to alter the contents of the text source currently being displayed in the text subwindow by inserting the substring **ss** at position **pos**. If **pos** is **nullPosition**, the substring is inserted at the current insertion position. The

text subwindow and source must have either edit or append access to correctly use this operation.

```
TextSW.IsIt: PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
```

The **IsIt** procedure returns **TRUE** if the window is a text subwindow, and **FALSE** otherwise.

```
TextSW.JumpEndMCR: Menu.MCRType;
```

The **JumpEndMCR** procedure implements the **J.End** function of the **TextOps** menu. It positions the display region with the last line of the source at the top.

```
TextSW.JumpInsertionMCR: Menu.MCRType;
```

The **JumpInsertionMCR** procedure implements the **J.Insert** function of the **TextOps** menu. It positions the display region with the line containing the insertion position at the top.

```
TextSW.JumpSelectionMCR: Menu.MCRType;
```

The **JumpSelectionMCR** procedure implements the **J.Select** function of the **TextOps** menu. If the subwindow contains the current selection, it positions the display region with the line containing the current selection at the top.

```
TextSW.JumpTopMCR: Menu.MCRType;
```

The **JumpTopMCR** procedure implements the **J.First** function of the **TextOps** menu. It positions the display region with the first line of the source at the top.

```
TextSW.ModifySource:PROCEDURE [
  sw: Window.Handle,
  proc: PROCEDURE [Window.Handle, TextSource.Handle, LONG POINTER]
    RETURNS [invalidRegions: TextSW.InvalidRegions],
  data: LONG POINTER];
```

The **ModifySource** procedure is provided for clients who wish to batch several changes to a text subwindow's source. **ModifySource** acquires the **TextSW** monitor and then calls **proc** with **sw**, its source, and whatever **data** was passed in to make these changes. **proc** is expected to return a description of the regions in the source that were modified in **invalidRegions**. The text subwindow updates its display region according to this information.

```
TextSW.PositionFromPlace: PROCEDURE [
  sw: Window.Handle, place: Window.Place] RETURNS [position: TextSource.Position];
```

The **PositionFromPlace** procedure enables clients to *resolve* window coordinates to the nearest text source position. It always returns a valid position.

```
TextSW.PositionIsVisible: PROCEDURE [
  sw: Window.Handle, position: TextSource.Position] RETURNS [BOOLEAN];
```

The **PositionIsVisible** procedure returns **TRUE** if position **position** of the source is currently being displayed. It does not take other windows overlapping **sw** into account.

```
TextSW.PositionMCR: Menu.MCRType;
```

The **PositionMCR** procedure implements the **Position** function of the **TextOps** menu. It interprets the current selection as a number and positions the display region that contains the cursor with the line containing the current selection at the top.

```
TextSW.PositionToLine: PROCEDURE [sw: Window.Handle, position: TextSource.Position];
```

The **PositionToLine** procedure positions the top of a text subwindow to the first line *after* the specified position. However, if the position corresponds to the first character of a line, that line is displayed. (Compare this procedure with **SetPosition**.)

```
TextSW.RemoveAllSecondarySelections: PROCEDURE [sw: Window.Handle];
```

The **RemoveAllSecondarySelections** procedure removes each secondary selection in a text subwindow.

```
TextSW.RemoveSecondarySelection: PROCEDURE [
  sw: Window.Handle, s: TextData.Selection];
```

The **RemoveSecondarySelection** procedure removes a specified secondary selection.

```
TextSW.ReplaceText: PROCEDURE [
  sw: Window.Handle, pos: TextSource.Position, count: LONG CARDINAL,
  block: Environment.Block, keepTrash: BOOLEAN ← TRUE];
```

The **ReplaceText** procedure allows the client to alter the contents of the text source currently being displayed in **sw** by replacing the **count** characters beginning at **pos** with **block**. **keepTrash** determines whether the deleted text is placed in the trashbin. (See the **Selection** interface for documentation on the trashbin.) The text subwindow and source must have edit access to use this operation correctly .

```
TextSW.SecondarySelectionFromPosition: PROCEDURE [
  sw: Window.Handle, position: TextSource.Position] RETURNS [s: TextData.Selection];
```

The **SecondarySelectionFromPosition** procedure returns the secondary selection in the window at position **position**. If there is no secondary selection there, **NIL** is returned.

```
TextSW.SetEOF: PROCEDURE [sw: Window.Handle, eof: TextSource.Position];
```

The **SetEOF** procedure alters the "end-of-file" position of the source in the subwindow.

```
TextSW.SetInsertion: PROCEDURE [sw: Window.Handle, position: TextSource.Position];
```

The **SetInsertion** procedure alters the insertion position of the source in the subwindow.

```
TextSW.SetOptions: PROCEDURE [sw: Window.Handle, options: TextSW.Options];
```

The **SetOptions** procedure sets the current options for a text subwindow.

```
TextSW.SetPosition: PROCEDURE [sw: Window.Handle, position: TextSource.Position];
```

The **SetPosition** procedure positions the top of a text subwindow to the line containing the character at the specified position. (Compare this procedure with **PositionToLine**.) If the position is not visible because of a long wrapped line, the subwindow may scroll a line at a time until it is visible.

```
TextSW.SetSecondarySelection: PROCEDURE [
  sw: Window.Handle, left, right: TextSource.Position, mode: TextData.SelectionMode]
  RETURNS [s: TextData.Selection];
```

The **SetSecondarySelection** procedure defines a secondary selection starting at **left** and ending at **right**. The secondary selection is highlighted according to **mode**.

```
TextSW.SetSelection: PROCEDURE [
  sw: Window.Handle, left, right: TextSource.Position];
```

The **SetSelection** procedure alters the selection position of the subwindow.

```
TextSW.SetSource: PROCEDURE [
  sw: Window.Handle, source: TextSource.Handle, position: TextSource.Position ← 0,
  reset: BOOLEAN ← TRUE];
```

The **SetSource** procedure changes the text source for a text subwindow. **reset** indicates whether the current display/source correspondence is valid or should be rebuilt.

```
TextSW.Sleep: PROCEDURE [sw: Window.Handle];
```

The **Sleep** procedure requests that the text subwindow package minimize its resource requirements by destroying all state related to text display.

```
TextSW.SplitMCR: Menu.MCRType;
```

The **SplitMCR** procedure implements the **Split** function of the **TextOps** menu. It splits the display region in two.

```
TextSW.SplitView: PROCEDURE [
  sw: Window.Handle, key: TextSW.KeyName, y: INTEGER];
```

The **SplitView** procedure splits a text subwindow **y** pixels down from the top of **sw**. **key** is an ignored obsolete parameter. This procedure, used internally in building the menu and split view facilities, is potentially useful for constructing client menu routines.

```
TextSW.Update: PROCEDURE [
  sw: Window.Handle, from, to: TextSource.Position, charsDeleted: BOOLEAN ← TRUE];
```

The **Update** procedure is called when the display/source correspondence is invalid. The characters between **from** and **to** are redisplayed to reflect any changes in the source. If any characters were deleted, **charsDeleted** must be set **TRUE** because more computation may be required to reestablish the display/source correspondence. This operation, as well as the next two update procedures, are intended for more experienced **TextSW** users who wish to create their own editors.

```
TextSW.UpdateRange: PROCEDURE [
  sw: Window.Handle, from, to: TextSource.Position, delta: LONG INTEGER,
  charsDeleted: BOOLEAN ← TRUE];
```

The **UpdateRange** procedure is called to reestablish the display/source correspondence after changes have been made to the source. The modifications were between **from** and **to**, and resulted in a change **delta** in the total number of characters. If any characters were deleted, **charsDeleted** must be set **TRUE** because more computation may be required to reestablish the display/source correspondence.

```
TextSW.UpdateToEnd: PROCEDURE [
  sw: Window.Handle, from: TextSource.Position, charsDeleted: BOOLEAN ← TRUE];
```

The **UpdateToEnd** procedure is called when the display/source correspondance is invalid. The characters after **from** will be redisplayed to reflect any changes in the source. If any characters were deleted, **charsDeleted** must be set **TRUE** because more computation may be required to reestablish the display/source correspondence.

```
TextSW.Wakeup: PROCEDURE [sw: Window.Handle];
```

The **Wakeup** procedure requests that the text subwindow package recompute all its display state that it discarded when **Sleep** was called.

```
TextSW.WrapMCR: Menu.MCRType;
```

The **WrapMCR** procedure implements the **Wrap** function of the **TextOps** menu. It toggles the wrap **BOOLEAN** in the text subwindow options record.

18

TTYSW

The **TTYSW** interface allows for traditional teletype interaction. Other Tajo user-interaction facilities are based on the notification concept. Because many programs are already written using a teletype-like control structure, the teletype subwindow is available to clients for upward compatibility.

TTYSWs are built on the **TTY** abstraction that is available as a common software interface. See the **TTY** section of the *Pilot Programmer's Manual* for details on some of the following.

18.1 Types

None.

18.2 Constants and data objects

```
TTYSW.defaultOptions: TextSW.Options = [
  access: append, menu: TRUE, split: TRUE, wrap: TRUE, scrollbar: TRUE,
  flushTop: FALSE, flushBottom: FALSE];
```

18.3 Signals and errors

```
TTYSW.Error: SIGNAL [code: TTYSW.ErrorCode];

TTYSW.ErrorCode: TYPE = {notATTYSW, badTTYHandle, other};
```

notATTYSW a passed-in subwindow is not a TTY subwindow.

badTTYHandle an obsolete error code, never used.

other an obsolete error code, never used.

```
TTYSW.LineOverflow: SIGNAL [s: LONG STRING] RETURNS [ns: LONG STRING];

TTYSW.Rubout: SIGNAL;
```

The procedures below that read strings from the user are implemented by calls on similar functions from the TTY interface. If any of those routines raise **LineOverflow** or **Rubout**, that signal is mapped into the corresponding one from the TTYSW interface.

18.4 Procedures

```
TTYSW.AppendChar: PROCEDURE [sw: Window.Handle, char: CHARACTER];
```

The **AppendChar** procedure can be used for output to a teletype subwindow. (See also **AppendString** and the **Put** interface.) This procedure can raise **TTYSW.Error[notATTYSW]**.

```
TTYSW.AppendString: UserInput.StringProcType;
```

The **AppendString** procedure can be used to produce formatted output to a teletype subwindow. (See also the **Put** interface.) This procedure can raise **TTYSW.Error[notATTYSW]**.

```
TTYSW.Create: PROCEDURE [
  sw: Window.Handle, backupFile: LONG STRING, s: Stream.Handle ← NIL,
  newFile: BOOLEAN ← TRUE, options: TextSW.Options ← TTYSW.defaultOptions,
  resetLengthOnNewSession: BOOLEAN ← FALSE];
```

The **Create** procedure creates a teletype subwindow from an ordinary subwindow. The **backupFile** parameter specifies the name of the file on which the teletype subwindow writes. However, if **s** is not **NIL**, **s** is assumed to be the stream handle on the file. When **newFile** is **TRUE**, the length of the file is set to zero at create time; otherwise, the existing length is used. When the teletype subwindow is created, the client must **FORK** a process (the *input process*) that plans to do input (i.e., a procedure called directly from the Notifier cannot do input from a TTY subwindow). This process should be able to handle the signals **LineOverflow** and **Rubout** and the errors **Error**, **ABORTED**, and **String.InvalidNumber**.

```
TTYSW.Destroy: PROCEDURE [sw: Window.Handle];
```

The **Destroy** procedure destroys teletype subwindow attributes of the subwindow. However, before this procedure is called the *input process* should be aborted. (See also **DestroyFromBackgroundProcess**.)

```
TTYSW.DestroyFromBackgroundProcess: PROCEDURE [sw: Window.Handle];
```

The **DestroyFromBackgroundProcess** procedure destroys the teletype subwindow from within the *input process*. The client should call this procedure as it returns from the Input process. (See also **Destroy**.)

```
TTYSW.EndOf: PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
```

If characters have been typed in but not yet seen by the client program, **TTYSW.EndOf** returns **FALSE**; otherwise it returns **TRUE**. This is equivalent to testing that the number returned from **CharsAvailable** is 0.

```
TTYSW.GetTTYHandle: PROCEDURE [sw: Window.Handle] RETURNS [tty: TTY.Handle];
```

The **GetTTYHandle** procedure returns the **TTY.Handle** associated with **sw**. If there is no corresponding **TTY.Handle**, **TTY.nullHandle** is returned.

```
TTYSW.IsIt: PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
```

The **IsIt** procedure returns **TRUE** if the subwindow is a teletype subwindow and **FALSE** otherwise.

18.5 Procedures mapped to calls on TTY

The rest of the procedures in this interface are implemented by converting the subwindow passed into them into a **TTY.Handle** and calling the corresponding routine from the **TTY** interface. Any of them may raise **TTYSW.Error[notATTYSW]**. **TTY.Rubout** and **TTY.LineOverflow** are mapped into the corresponding signals from the **TTYSW** interface. The type of each of the following procedures can be determined by replacing the "**h: TTY.Handle**" argument with "**sw: Window.Handle**". The one exception is that the second argument of **TTYSW.PutBackChar** is "**char: CHARACTER**" and the second argument of **TTY.PutBackChar** is "**c: CHARACTER**". All of these procedures will be withdrawn in a future release. You are advised to convert your calls to be directly on the **TTY** interface.

BackingStream
CharsAvailable
GetChar
GetDecimal
GetEcho
GetEditedString
GetId
GetLine
GetLongDecimal
GetLongNumber
GetLongOctal
GetNumber
GetOctal
GetPassword
GetString
NewLine
PopAlternateInputStreams
PushAlternateInputStreams
PutBackChar
RemoveCharacter
RemoveCharacters
SetBackingSize
SetEcho

19

Put

The **Put** interface provides output procedures for windows. All the procedures in the **Put** interface take a **Window.Handle**, a piece of data to be formatted and, where appropriate, a format specification. See the documentation on the **Format** interface for comments about the actual output format of these procedures.

19.1 Types

```
Put.NetFormat: TYPE = Format.NetFormat;
```

19.2 Constants and data objects

None.

19.3 Signals and errors

None.

19.4 Procedures

In all the following procedures, the output is directed to the **UserInput.StringOut** procedure associated with the **Window.Handle**. If the **Window.Handle** is **NIL**, the output is directed to the default output sink.

```
Put.Blank, Put.Blanks: PROCEDURE [h: Window.Handle ← NIL, n: CARDINAL ← 1];
```

The **Blank** procedure invokes **Format.Blank**.

```
Put.Block: PROCEDURE [h: Window.Handle ← NIL, block: Environment.Block];
```

The **Block** procedure invokes **Format.Block**.

```
Put.Char: PROCEDURE [h: Window.Handle ← NIL, char: CHARACTER];
```

The **Char** procedure invokes **Format.Char**.

```
Put.CR: PROCEDURE [h: Window.Handle ← NIL];
```

The **CR** procedure invokes **Format.CR**.

```
Put.CurrentSelection: PROCEDURE [h: Window.Handle ← NIL];
```

The **CurrentSelection** procedure passes the string that is the current selection to the output procedure of the **UserInput.StringOut** procedure associated with the **Window.Handle**. If the **Window.Handle** is **NIL**, the output is directed to the default output sink.

```
Put.Date: PROCEDURE [
  h: Window.Handle ← NIL, pt: Time.Packed,
  format: Format.DateFormat ← noSeconds];
```

The **Date** procedure invokes **Format.Date**.

```
Put.Decimal: PROCEDURE [h: Window.Handle ← NIL, n: INTEGER];
```

The **Decimal** procedure invokes **Format.Decimal**.

```
Put.HostNumber: PROCEDURE [
  h: Window.Handle ← NIL, host: System.HostNumber, format: Format.NetFormat ← octal];
```

The **HostNumber** procedure invokes **Format.HostNumber**.

```
Put.Line: PROCEDURE [h: Window.Handle ← NIL, s: LONG STRING];
```

The **Line** procedure invokes **Format.Line**.

```
Put.LongDecimal: PROCEDURE [h: Window.Handle ← NIL, n: LONG INTEGER];
```

The **LongDecimal** procedure invokes **Format.LongDecimal**.

```
Put.LongNumber: PROCEDURE [
  h: Window.Handle ← NIL, n: LONG UNSPECIFIED, format: Format.NumberFormat];
```

The **LongNumber** procedure invokes **Format.LongNumber**.

```
Put.LongOctal: PROCEDURE [h: Window.Handle ← NIL, n: LONG UNSPECIFIED];
```

The **LongOctal** procedure invokes **Format.LongOctal**.

```
Put.LongString: PROCEDURE [h: Window.Handle ← NIL, s: LONG STRING];
```

The **LongString** procedure invokes **Format.LongString**.

```
Put.LongSubString: PROCEDURE [h: Window.Handle ← NIL, ss: String.SubString];
```

The **LongSubString** procedure invokes **Format.LongSubString**.

```
Put.NetworkAddress: PROCEDURE [
  h: Window.Handle ← NIL, address: System.NetworkAddress,
  format: Format.NetFormat ← octal];
```

The **NetworkAddress** procedure invokes **Format.NetworkAddress**.

```
Put.NetworkNumber: PROCEDURE [
  h: Window.Handle ← NIL, networkNumber: System.NetworkNumber,
  format: Format.NetFormat];
```

The **NetworkNumber** procedure invokes **Format.NetworkNumber**.

```
Put.Number: PROCEDURE [
  h: Window.Handle ← NIL, n: UNSPECIFIED, format: Format.NumberFormat];
```

The **Number** procedure invokes **Format.Number**.

```
Put.Octal: PROCEDURE [h: Window.Handle ← NIL, n: UNSPECIFIED];
```

The **Octal** procedure invokes **Format.Octal**.

```
Put.SocketNumber: PROCEDURE [
  h: Window.Handle ← NIL, socketNumber: System.SocketNumber,
  format: Format.NetFormat];
```

The **SocketNumber** procedure invokes **Format.SocketNumber**.

```
Put.SubString: PROCEDURE [h: Window.Handle ← NIL, s: String.SubString];
```

The **SubString** procedure invokes **Format.SubString**.

```
Put.Text: PROCEDURE [h: Window.Handle ← NIL, s: LONG STRING];
```

The **Text** procedure invokes **Format.Text**. [**Text** is not **String** because it causes a name conflict with the interface named String.]

20

Tool

The **Tool** interface permits tool writers to use the Tajo user interface mechanism without worrying about the details of invocation. It reduces to a minimum the knowledge the client needs of Tajo's more basic levels. Refer to the ExampleTool in Appendix A for a tool that uses the **Tool** interface.

20.1 Types

```
Tool.MakeSWsProc: TYPE = PROCEDURE [window: Window.Handle];
```

At various points, depending on the initial state of the tool and user actions, Tajo calls on the **MakeSWsProc** procedure supplied to **Create** to let the client create subwindows and menus.

```
Tool.State: TYPE = {inactive, tiny, active, default};

Tool.SWProc: TYPE = PROCEDURE [sw: Window.Handle];

Tool.SWType: TYPE = MACHINE DEPENDENT{vanilla(0), predefined(376B), last(377B)};
```

The **Tool** interface manages client-defined subwindow types just as it manages the **predefined** subwindow types: Form, File, Message, String, and TTY. If a client wants to register a subwindow type that would use the **SimpleAdjustProc**, the **NopSleepProc**, and the **NopWakeupProc,** it can instead use a **Tool.SWType** of **vanilla**.

20.2 Constants and data objects

```
Tool.DefaultHeight: INTEGER = ToolWindow.nullBox.dims.h;
```

20.3 Signals and errors

```
Tool.Error: SIGNAL [code: Tool.ErrorCode];

Tool.ErrorCode: TYPE = {
  notATool, unknownSWType, swNotFound, invalidWindow, invalidParameters,
  other};
```

invalidWindow	can be raised by any procedure that takes a **Window.Handle** argument, if the associated window is not a valid tool window.
notATool	can be raised by any procedure that takes a **Window.Handle** argument, if the associated window was not created by **Tool.Create**.
unknownSWType	can be raised by any procedure that takes a **Tool.SWType** argument.

20.4 Procedures

```
Tool.AddThisSW: PROCEDURE [
  window: Window.Handle, sw: Window.Handle,
  swType: Tool.SWType ← predefined, nextSW: Window.Handle ← NIL,
  h: INTEGER ← Tool.DefaultHeight];
```

The **AddThisSW** procedure allows clients that use methods other than **Tool** procedures to create subwindows for communicating these methods to the **Tool** interface. The **Tool** interface inserts **sw** above the **nextSW** subwindow, and the bottom subwindow is grown or shrunk to accommodate the new subwindow. [**Warning**: Usually the **Create** call hasn't returned when the **MakeSWsProc** procedure is called. The **Window.Handle** variable into which the client assigns the value returned from **Create** is uninitialized. Thus, the client should not reference this variable in its **MakeSWsProc** procedure. Instead, the client should use the **window** parameter passed to the **MakeSWsProc** procedure.]

```
Tool.Create: PROCEDURE [
  name: LONG STRING, makeSWsProc: Tool.MakeSWsProc,
  initialState: Tool.State ← default,
  clientTransition: ToolWindow.TransitionProcType ← NIL,
  movableBoundaries: BOOLEAN ← TRUE,
  initialBox: Window.Box ← ToolWindow.nullBox,
  cmSection, tinyName1, tinyName2: LONG STRING ← NIL,
  named: BOOLEAN ← TRUE]
  RETURNS [window: Window.Handle];
```

The **Create** procedure creates a tool. The **name** parameter is the string that appears in a tool's name stripe if the **named** parameter is **TRUE**; the string used in the inactive menu is derived from this string. The parameters **tiny1** and **tiny2** specify both parts of the tiny name used when the tool is made tiny. If these parameters are **NIL**, the tiny name is derived from the **name** parameter. **cmSection** specifies the name of the section in the `User.Cm` that contains the symbiote menu, initial state, tiny place, and initial window box. When the **initialState** is **default**, the tool assumes a predetermined state, depending on how it is created. The tool is initialized to be active when loaded while the user is in Tajo, because the user will probably want to use it right away. If the **clientTransition** procedure is not **NIL**, it is called before the tool is about to change state (e.g., before calling **MakeSWsProc**, see below) and before anything is done to the data managed by the **Tool** interface. The one exception to this ordering rule is that **FormSW.FreeAllItems** is called for each **FormSW** in the tool when the tool is going inactive before the client's transition procedure is called. It is common for a client's transition procedure to deallocate a record containing data that the **FormSW.FreeAllItems** procedure references. Thus, the data must be referenced before it goes away. [If the client doesn't like being called in this order, it could set its own procedure to be the window-transition procedure that could call **Tool.Transition**. This could be important if the client has a process that is updating things in a form subwindow.] When the **movableBoundaries** parameter is **TRUE**, the user

may select the boundary line between subwindows and reposition it. The **initialBox** parameter can be used to specify the tool box (bitmap relative). A value of **ToolWindow.nullBox** lets Tajo assign the box from the next available box slot.

```
Tool.DeleteThisSW: PROCEDURE [sw: Window.Handle];
```

The **DeleteThisSW** procedure removes the subwindow **sw** from its tool window and distributes the window space among the remaining subwindows of the tool. The subwindow will not be deleted if it is the only subwindow in the tool. Clients should first free all menus and **FormSW** items specific to **sw**. Menus should be destroyed by **Menu.Uninstantiate** followed by **Menu.Destroy**. **FormSW** items should be destroyed by **FormSW.FreeAllItems**. The space that was taken by **sw** will be given to the bottom subwindow of the tool.

```
Tool.Destroy: PROCEDURE [window: Window.Handle];
```

The **Destroy** procedure is used to destroy a tool window created by the **Tool** interface. It may also be used for removing a subwindow of the tool. This procedure also calls the client-transition procedure supplied to **Tool.Create** with a new **Tool.State** of **inactive** before the tool is destroyed. If **window** is a subwindow, its associated data structures are cleaned up as follows: normally, the client should destroy anything that it creates, such as any private data, before a tool goes inactive. The tool mechanism relieves the client from having to destroy subwindows and menus that were created in a standard way. In particular, menus should be created by a call to **Menu.Make**; **FormSW.ItemDescriptors** should be created by a call to **FormSW.AllocateItemDescriptor**; **FormSW.ItemObjects** should be created by calls to **FormSW.*Item** procedures.

```
Tool.DestroySW: PROCEDURE [window: Window.Handle];
```

The **DestroySW** procedure is not currently implemented.

```
Tool.Info: PROCEDURE [window: Window.Handle] RETURNS [
  name, cmSection: LONG STRING, makeSWsProc: Tool.MakeSWsProc,
  clientTransition: ToolWindow.TransitionProcType,
  movableBoundaries: BOOLEAN];
```

The **Info** procedure returns the values of certain parameters supplied to **Tool.Create**. The client should not modify **name** or **cmSection**, as these values may become dangling references when the tool is destroyed.

```
Tool.IsIt: PROCEDURE [window: Window.Handle] RETURNS [BOOLEAN];
```

The **IsIt** procedure returns **TRUE** if **window** was created by **Tool.Create** and **FALSE** otherwise.

```
Tool.MakeClientSW: PROCEDURE [
  window: Window.Handle, clientProc: PROCEDURE [sw: Window.Handle,
  clientData: LONG POINTER], clientData: LONG POINTER,
   swType: Tool.SWType, h: INTEGER ← Tool.DefaultHeight]
  RETURNS [sw:Window.Handle];
```

The **MakeClientSW** procedure allows clients to create their own subwindow types. The **clientProc** is the client's procedure that will create the subwindow. The client passes **clientData** to the **Tool** interface, which in turn is passed to the **clientProc** procedure. The

swType is obtained from Tool.**RegisterSWType**. The **h** parameter is the new subwindow's initial height.

```
Tool.MakeDefaultSWs: PROCEDURE [
  window: Window.Handle, messageLines: CARDINAL ← 0,
  formProc: FormSW.ClientItemsProcType ← NIL,
  formHeight: CARDINAL ← Tool.DefaultHeight, logName: LONG STRING ← NIL]
  RETURNS [msgSW, formSW, logSW: Window.Handle];
```

The **MakeDefaultSWs** procedure creates a message subwindow, a form subwindow, and a log file subwindow as subwindows of **window**. If **messageLines** is 0, there will be no message subwindow. If **formProc** is NIL, there will be no form subwindow. If **logName** is NIL, there will be no file subwindow.

```
Tool.MakeFileSW: PROCEDURE [
  window: Window.Handle, name: LONG STRING, access: FileSW.Access ← append,
  h: INTEGER ← Tool.DefaultHeight, allowTypeIn: BOOLEAN ← TRUE,
  resetLengthOnNewSession: BOOLEAN ← FALSE,
  resetLengthOnActivate: BOOLEAN ← FALSE]
  RETURNS [sw: Window.Handle];
```

The **MakeFileSW** procedure is usually called from a **MakeSWsProc** to create a file subwindow. (See the **FileSW** interface for details on file subwindows.) This procedure may raise TextSource.**Error[fileNameError]** if **access** is **read** and the file is not found, or if the file cannot be acquired. The BOOLEAN parameter **allowTypeIn** specifies whether the log accepts type-in. The BOOLEAN parameter **resetLengthOnNewSession** specifies whether the length of the file is set to zero at the start of a new debugging session. **resetLengthOnActivate** specifies whether the length of the file is set to zero when the tool is activated.

```
Tool.MakeFormSW: PROCEDURE [
  window: Window.Handle, formProc: FormSW.ClientItemsProcType,
  options: FormSW.Options ← [], h: INTEGER ← Tool.DefaultHeight,
  zone: UNCOUNTED ZONE ← NIL]
  RETURNS [sw: Window.Handle];
```

The **MakeFormSW** procedure is usually called from a **MakeSWsProc** to create a form subwindow. (See the **FormSW** interface for details on form subwindows.) To take advantage of automatic tool deallocation, FormSW.**ItemDescriptor**s should be created by a call to FormSW.**AllocateItemDescriptor** and FormSW.**ItemObject**s should be created by calls to FormSW.***Item** procedures. The **zone** parameter is passed to **FormSW** when the **FormSW** items are allocated.

```
Tool.MakeMsgSW: PROCEDURE [
  window: Window.Handle, lines: CARDINAL ← 1, h: INTEGER ← Tool.DefaultHeight] RETURNS
  [sw: Window.Handle];
```

The **MakeMsgSW** procedure is usually called from a **MakeSWsProc** to create a message subwindow. (See the **MsgSW** interface for details on message subwindows.)

```
Tool.MakeStringSW: PROCEDURE [
  window: Window.Handle, s: LONG POINTER TO LONG STRING ← NIL,
  access: TextSW.Access ← append, h: INTEGER ← Tool.DefaultHeight,
  expandable: BOOLEAN ← FALSE]
  RETURNS [sw: Window.Handle];
```

The **MakeStringSW** procedure is usually called from a **MakeSWsProc** to create a string subwindow. (See the **StringSW** interface for details on string subwindows.)

```
Tool.MakeTextSW: PROCEDURE [
  window: Window.Handle, source: TextSource.Handle, sink: TextSink.Handle ← NIL,
  options: TextSW.Options ← TextSW.defaultOptions,
  position: TextSource.Position ← 0, allowTypeIn: BOOLEAN ← TRUE]
  RETURNS [sw: Window.Handle];
```

The **MakeTextSW** procedure is usually called from a **MakeSWsProc** to create a text subwindow. (See the **TextSW** interface for details on text subwindows.)

```
Tool.MakeTTYSW: PROCEDURE [
  window: Window.Handle, name: LONG STRING, h: INTEGER ← Tool.DefaultHeight,
  resetLengthOnNewSession: BOOLEAN ← FALSE]
  RETURNS [sw: Window.Handle];
```

The **MakeTTYSW** procedure is usually called from a **MakeSWsProc** to create a TTY subwindow. (See the **TTYSW** interface for details on TTY subwindows.)

```
Tool.NopSleepProc: Tool.SWProc;
```

The **NopSleepProc** procedure is provided for those who wish to register a new **Tool.SWType**; it does nothing when called.

```
Tool.NopWakeupProc: Tool.SWProc;
```

The **NopWakeupProc** procedure is provided for those who wish to register a new **Tool.SWType**; it does nothing when called.

```
Tool.RegisterSWType: PROCEDURE [
  adjust: ToolWindow.AdjustProcType ← Tool.SimpleAdjustProc,
  sleep: ToolWindow.SWProc ← Tool.NopSleepProc,
  wakeup: ToolWindow.SWProc ← Tool.NopWakeupProc]
  RETURNS [uniqueSWType: Tool.SWType];
```

The **RegisterSWType** procedure registers a client-defined subwindow type with the **Tool** interface. The **adjust** procedure is called whenever the user moves the subwindow or changes the subwindow size. The **sleep** procedure is called whenever the window in which the subwindow lives becomes tiny. The subwindow is then expected to throw away any data that it uses only to display its contents. The **wakeup** procedure undoes what **sleep** did when the tool becomes active again. If a client wants to register a subwindow type that would use the **SimpleAdjustProc**, the **NopSleepProc**, and the **NopWakeupProc**, it can instead use a **Tool.SWType** of **vanilla**.

```
Tool.SimpleAdjustProc: ToolWindow.AdjustProcType;
```

The **SimpleAdjustProc** is a null procedure. If no **ToolWindow.AdjustProcType** is passed to **RegisterSWType**, the **SimpleAdjustProc** is used.

```
Tool.SwapSWs: PROCEDURE [
  window, oldSW, newSW: Window.Handle, newType: Tool.SWType ← predefined]
  RETURNS [oldType: Tool.SWType];
```

The **SwapSWs** procedure switches one subwindow for another subwindow in a tool. **window** is the tool window. **oldSW** identifies the currently displayed subwindow that will be replaced by **newSW**. **newSW** cannot currently be part of the tree that makes up the hierarchy of displayed windows. When this procedure has returned, **oldSW** has been removed from this tree. **Error[code: swNotFound]** may be raised from this procedure. The original **newSW** must be created with procedures other then the ones provided in the **Tool** interface; for example, you might call **ToolWindow.CreateSubwindow** followed by **FormSW.Create**. In addition, the call to **ToolWindow.CreateSubwindow** should supply **NIL** as the **parent** argument.

```
Tool.Transition: ToolWindow.TransitionProcType;
```

The **Transition** procedure is called whenever the tool changes state. In turn, it calls the client transition procedure supplied to **Tool.Create**. If the new **Tool.State** of the tool is **inactive**, the **FormSW.Items** are freed before the client transition procedure is called. The client transition procedure is called before the **Tool** interface takes any other action.

```
Tool.UnusedLogName: PROCEDURE [unused, root: LONG STRING];
```

The **UnusedLogName** procedure guarantees unique log file names among file and TTY subwindows by enumerating all the current file and TTY subwindows and checking that each name is not currently in use. If a name is in use, a derived name is generated and checked until a unique name is generated. Note that the development environment file system does not permit multiple writeable handles on a file, so this procedure should be called if there might be multiple instances of the tool. A unique name is generated by setting the length of **unused** to 0, appending the **root**, and appending a number.

21

ToolWindow

The facilities of **ToolWindow** enhance those provided by the **Window** interface. Specifically, they provide functions that implement Tajo's window illusion for tools.

21.1 Types

```
ToolWindow.AdjustProcType: TYPE = PROC [
  window: ToolWindow.Handle, box: ToolWindow.Box, when: ToolWindow.When];
```

Because users can change the location and size of windows on the display, Tajo provides the individual tools with a mechanism for knowing when one of their windows has been adjusted. Before the system adjusts a window's location or size, it calls the tool's limit procedure (see **LimitProcType**). It then uses the **box** returned by the limit procedure to call the tool's adjust procedure. The adjust procedure is called both before and after the actual adjustment is made; the **when** parameter is used by the **AdjustProcType** to indicate the difference.

```
ToolWindow.Box: TYPE = Window.Box;
```

```
ToolWindow.BoxProcType: TYPE = PROC RETURNS [box: ToolWindow.Box];
```

A **BoxProcType** is the type of the parameters passed to **SetBoxAllocator**.

```
ToolWindow.DisplayProcType: TYPE = PROC [window: ToolWindow.Handle];
```

A **DisplayProcType** is called whenever the contents of the window need to be refreshed on the display; for example, when a window previously on top of a given window is moved out of the way. For all Tajo-supplied subwindow types, display procedures are automatically supplied at create time.

```
ToolWindow.EnumerateProcType: TYPE = PROC [
  window: ToolWindow.Handle] RETURNS [done: BOOLEAN];
```

```
ToolWindow.EnumerateSWProcType: TYPE = PROC [
  window, sw: ToolWindow.Handle] RETURNS [done: BOOLEAN];
```

```
ToolWindow.Handle: TYPE = Window.Handle;

ToolWindow.LimitProcType: TYPE = PROCEDURE [
  window: ToolWindow.Handle, box: ToolWindow.Box] RETURNS [ToolWindow.Box];
```

Although the user moves windows around on the display, Tajo allows the individual tools to exercise veto or modification rights over moves. This is particularly useful for allowing a tool to prohibit, for example, its window becoming smaller than some certain size or moving completely off the visible display region. When the system adjusts the window's location or size, it first calls the limit procedure with the requested box and then passes the returned box to the tool's adjust procedure.

```
ToolWindow.OnOff: TYPE = {on, off};
```

OnOff is the type used to set and unset the tool name stripe.

```
ToolWindow.Place: TYPE = Window.Place;
```

Place is the type of the top left corner of a box.

```
ToolWindow.Size: TYPE = {tiny, normal, zoomed};
```

A tool always has one of three **Size**s.

tiny displays as a small rectangular box that contains a name for the tool.

zoomed displays as a **normal** tool, but fills the whole screen.

```
ToolWindow.State: TYPE = {inactive, tiny, active};
```

A tool is always in one of three **State**s.

inactive indicates that the user is not interested in any of the functions the tool implements, and all resources it utilizes should be freed. When a tool is inactive, a menu entry whose text is derived from its name is placed on the *Inactive* menu.

tiny the user is not interested what the tool normally displays; therefore resources associated with the display state should be freed.

```
ToolWindow.TransitionProcType: TYPE = PROC [
  window: old, new: ToolWindow.State];
```

A tool's **TransitionProcType** is called to notify a tool whenever a user action causes Tajo to change the tool's state (see **ToolWindow.State** above). **TransitionProc**s are often used to free some of a tool's resources when its state changes.

```
ToolWindow.When: TYPE = {before, after};

ToolWindow.WindowType: TYPE = {root, tool, clipping, sub, other};
```

Whereas the **Window** interface allows arbitrary window tree structures to be created, **ToolWindow** restricts the types of window trees that can be created and imposes specific

semantics on those trees. A **ToolWindow** tree consists of a root level, a tool window level, a clipping window level, and (optionally) subwindow levels.

root window is the underlying bitmap.

tool window is referred to in this document as a tool window.

clipping window is associated with each **tool** window, where the **clipping** window is the child of the **tool** window. **Clipping** windows prevent subwindows from obscuring their parents; they should be of no concern to clients.

sub windows are subwindows of **tool** windows,

other windows are all lower levels.

21.2 Constants and data objects

```
ToolWindow.nullBox: ToolWindow.Box = [[0, 0], [0, 0]];
```

21.3 Signals and errors

None.

21.4 Procedures

```
ToolWindow.Activate: PROC [window: ToolWindow.Handle];
```

Activate activates a tool; that is, changes its state to active. The tool's transition procedure is called to allow it to respond to the change in state.

```
ToolWindow.Create: PROC [
  name: LONG STRING, adjust: ToolWindow.AdjustProcType,
  transition: ToolWindow.TransitionProcType,
  box: ToolWindow.Box ← ToolWindow.nullBox,
  limit: ToolWindow.LimitProcType ← ToolWindow.StandardLimitProc,
  initialState: ToolWindow.State ← active, named: BOOLEAN ← TRUE,
  gravity: Window.Gravity ← nw]
  RETURNS [ToolWindow.Handle];
```

Create creates an empty **tool** window with the indicated **box**. If box is **nullBox**, Tajo uses the normal box allocator to assign a box to the tool. If **named** is **TRUE**, the window will have a black band across the top that displays **name**. **initialState** is the **State** with which the window is created. (See **AdjustProcType**, **TransitionProcType**, and **LimitProcType** above for explanations of the meaning of these parameters.) **gravity** is the **Window.Gravity** that Tajo should use when changing the size of the tool window.

```
ToolWindow.CreateSubwindow: PROC [
  parent: ToolWindow.Handle, display: ToolWindow.DisplayProcType ← NIL,
  box: ToolWindow.Box ← ToolWindow.nullBox, gravity: Window.Gravity ← nw]
  RETURNS [ToolWindow.Handle];
```

Subwindows are normally created by the client to simplify window manipulations. A subwindow is a box (a rectangle defined by an x, y and a width and height) *within* the the parent tool's clipping window (i.e., within that box occupied by the tool, but not including its borders or name stripe). The subwindow is clipped at its parent's clipping window so that it does not obscure the parent. However, a subwindow can extend "outside" the parent's window (it is legal for a subwindow's box to have a negative x, or a height greater than that of the window); only those bits within the parent's clipping window are displayed.

CreateSubwindow creates a new subwindow object with the indicated **box** within its window and links it into the **parent** window's chain of subwindows. The **display** procedure is called whenever the content of the window needs to be refreshed onto the bitmap display. For all Tajo-supplied subwindow types, display procedures are automatically supplied at create time. (See also **EnlinkSubwindow** and **DelinkSubwindow**.)

```
ToolWindow.Deactivate: PROC [window: ToolWindow.Handle] RETURNS [aborted: BOOLEAN];
```

Deactivate changes a tool's state to inactive. The window's transition procedure is called to respond to the state change. **Deactivate** notifies subsystems that depend on **Event.toolWindow** first. If the event is aborted, the tool is not deactivated, and **Deactivate** returns **FALSE**.

```
ToolWindow.DelinkSubwindow: PROC [child: ToolWindow.Handle];
```

DelinkSubwindow removes the subwindow and its children from the window structure. This procedure is not normally called by Tajo clients.

```
ToolWindow.Destroy: PROC [window: ToolWindow.Handle];
```

Destroy destroys both tool windows and subwindows.

```
ToolWindow.DrawNameFrame: ToolWindow.DisplayProcType;
```

DrawNameFrame draws the tool's name frame, which is the stripe containing the tool name at the top of the window.

```
ToolWindow.DrawRectangle: PROC [
  window: ToolWindow.Handle, box: ToolWindow.Box, width: CARDINAL ← 1];
```

DrawRectangle paints the outline of a rectangular box with dimensions **box**. **width** is the width (in pixels) of the rectangle's border.

```
ToolWindow.EnlinkSubwindow: PROC [parent, child, youngerSibling: ToolWindow.Handle];
```

EnlinkSubwindow links the subwindow into **parent**'s subwindow chain in the indicated position. This procedure is not normally used by Tajo clients, as subwindows are linked upon creation.

```
ToolWindow.EnumerateInactiveWindows : PROC [proc: ToolWindow.EnumerateProcType];
```

EnumerateInactiveWindows enumerates the tool windows on the Inactive menu.

```
ToolWindow.EnumerateSWs: PROC [
  window: Window.Handle, proc: ToolWindow.EnumerateSWProcType];
```

EnumerateSWs enumerates all the subwindows within a tool window.

```
ToolWindow.GetAdjustProc: PROC [
  window: ToolWindow.Handle] RETURNS [ToolWindow.AdjustProcType];
```

GetAdjustProc returns the **AdjustProcType** associated with a tool window.

```
ToolWindow.GetBox: PROC [window: ToolWindow.Handle] RETURNS [ToolWindow.Box];
```

GetBox returns the tool window's box.

```
ToolWindow.GetClippedDims: PROC [window: ToolWindow.Handle] RETURNS [Window.Dims];
```

GetClippedDims returns the dimensions of the window for the tool in its active state. The tool need not be active when this procedure is called.

```
ToolWindow.GetGravity: PROC [
  window: ToolWindow.Handle] RETURNS [gravity: Window.Gravity];
```

GetGravity returns the gravity used to change the tool window's size.

```
ToolWindow.GetInactiveName: PROC [
  window: ToolWindow.Handle] RETURNS [name: LONG STRING];
```

GetInactiveName returns the name that the tool will be given when it becomes inactive (see **SetName**). This is the name that is entered in the *inactive* menu when the tool is deactivated. It is the client's responsibility to free the string returned by this procedure to the system heap.

```
ToolWindow.GetLimitProc: PROC [
  window: ToolWindow.Handle] RETURNS [ToolWIndow.LimitProcType];
```

GetLimitProc returns the **LimitProcType** associated with the tool window.

```
ToolWindow.GetName: PROC [window: ToolWindow.Handle] RETURNS [name: LONG STRING];
```

GetName returns the name of the tool. The client must free the string returned by this procedure to the system heap.

```
ToolWindow.GetNameStripe: PROC [
  window: ToolWindow.Handle] RETURNS [ToolWindow.OnOff];
```

GetNameStripe returns the state of the name stripe, **on** or **off**.

```
ToolWindow.GetState: PROC [window: ToolWindow.Handle] RETURNS [state: ToolWindow.State];
```

GetState returns the state of a tool window.

```
ToolWindow.GetTinyName: PROC [
  window: ToolWindow.Handle] RETURNS [name, name2: LONG STRING];
```

GetTinyName copies the tiny name of the window into two strings allocated from the system heap. It is the client's responsibility to free these strings.

```
ToolWindow.GetTinyPlace: PROC [
  window: ToolWindow.Handle] RETURNS [place: ToolWindow.Place];
```

GetTinyPlace returns the place of the tool window when it is in its tiny state. The tool need not be tiny at the time this procedure is called.

```
ToolWindow.GetTransitionProc: PROC [
  window: ToolWindow.Handle] RETURNS [ToolWindow.TransitionProcType];
```

GetTransitionProc returns the **TransitionProcType** associated with the tool window.

```
ToolWindow.Hide: PROC [window: ToolWindow.Handle];
```

Hide removes **window** from the group of windows displayed on the bitmap. This procedure is not normally called by Tajo clients.

```
ToolWindow.IsPlaceInWindow: PROC [
  place: ToolWindow.Place, window: ToolWindow.Handle] RETURNS [BOOLEAN];
```

IsPlaceInWindow returns TRUE if **place** is within **window**; otherwise it returns FALSE.

```
ToolWindow.MakeSize: PROC [window: ToolWindow.Handle, size: ToolWindow.Size];
```

MakeSize changes the size of a tool window.

```
ToolWindow.SetAdjustProc: PROC [
  window: ToolWindow.Handle, proc: ToolWindow.AdjustProcType]
  RETURNS [old:ToolWindow.AdjustProcType];
```

SetAdjustProc makes **proc** the AdjustProc for a tool window and returns the old one.

```
ToolWindow.SetBox: PROC [window: ToolWindow.Handle, box: ToolWindow.Box];
```

SetBox changes the size and position of a tool window.

```
ToolWindow.SetBoxAllocator: PROC [normal, tiny: ToolWindow.BoxProcType];
```

SetBoxAllocator registers procedures that determine where to display the tool upon creation.

```
ToolWindow.SetGravity: PROC [window: ToolWindow.Handle,gravity: Window.Gravity];
```

SetGravity sets the gravity of a tool window.

```
ToolWindow.SetLimitProc: PROC [
  window: ToolWindow.Handle, proc: ToolWindow.LimitProc]
  RETURNS [old: ToolWindow.LimitProcType];
```

SetLimitProc associates **proc** with the tool window and returns the old **LimitProc**.

```
ToolWindow.SetName: PROC [window: ToolWindow.Handle, name: LONG STRING];
```

SetName procedure changes the text of the menu entry placed on the *Inactive* menu when the tool is inactive.

```
ToolWindow.SetNameStripe: PROC [window: ToolWindow.Handle, onOff: ToolWindow.OnOff];
```

SetNameStripe sets the state of the name stripe **on** or **off**.

```
ToolWindow.SetTinyName: PROC [
  window: ToolWindow.Handle, name: LONG STRING, name2: LONG STRING ← NIL];
```

SetTinyName changes the text that is displayed when the window is tiny. **name** is the first line of text and **name2** is the second.

```
ToolWindow.SetTinyPlace: PROC [window: ToolWindow.Handle, place: ToolWindow.Place];
```

SetTinyPlace dictates where the tool will be positioned when it is tiny.

```
ToolWindow.SetTransitionProc: PROC [
  window: ToolWindow.Handle, proc: ToolWindow.TransitionProcType]
  RETURNS [ToolWindow.TransitionProcType];
```

SetTransitionProc associates **proc** with the tool window and returns the old **TransitionProc**.

```
ToolWindow.Show: PROC [window: ToolWindow.Handle];
```

Show causes **window** and its subtree of windows to be displayed. It should be called after a tool window is set up.

```
ToolWindow.StandardLimitProc: ToolWindow.LimitProcType;
```

StandardLimitProc performs the normal Tajo window-limiting operations. These prevent a window from being moved off the bitmap and prevent a tool from being made smaller than a tiny window.

```
ToolWindow.Type: PROC [
  window: ToolWindow.Handle] RETURNS [type: ToolWindow.WindowType];
```

The **Type** procedure tells you the type of the window.

```
ToolWindow.WindowForSubwindow: PROC [
  sw: ToolWindow.Handle] RETURNS [window: ToolWindow.Handle];
```

WindowForSubwindow returns the tool window of a subwindow.

III

Window and subwindow building

Windows and subwindows are the most basic building blocks for tools in the XDE system. The interfaces described in this section are lower level than those described in the previous section (Tool building). In particular, those interfaces were built using these interfaces.

III.1 The window package

The window package provides procedures that enable the client to display data by whitening and blackening the bits in the window. These include procedures for painting characters and strings and blackening, whitening, or graying boxes. The window package also provides procedures for copying arrays of bits and brush-and-trajectory painting, which allows graphics curves to be easily drawn. (See the **Display** interface.)

A window is conceptually an instance of an abstract window object. The window package obtains storage for window objects from Tajo. Contact the Tajo implementors if you must allocate your own window objects.

Each window object contains a client-supplied *display procedure*, which, on demand, will repaint all or part of the window. This procedure is invoked by the window package, for instance, when a window that was obscured by an overlapping window suddenly becomes more visible. However, clients should not call their display procedure directly. Instead, they should update their data, call **InvalidateBox** to mark part or all of the window invalid, and then call **Validate** to indicate to the window package that any invalid areas should be validated by calling the window's display procedure.

The window package allows clients to supply *bitmap unders*. These are blocks of memory used to maintain the bits that would appear in the bitmap where a window is if the window and the windows covering it did not exist. The window package can then fix up the bitmap without calling the display procedure of all the windows (partially) hidden by this one when it is removed from the tree. Menus can thus appear and disappear quickly.

The window package that implements the **Window** interface is passive, responding only to calls from the client's program. It creates no processes and allocates almost no storage.

III.1.1 Windows

Windows overlap other windows and may be manipulated even when they are under other windows. Windows are contained within their parent's rectangular regions: if they would stick out of their parent, their display is trimmed at their parent's edge.

For instance, the **Window** interface defines the window management package that is used by Tajo. The **Window** interface manipulates a tree of *windows*. There is one *root window* (at level "zero") that is always equated to the visible bitmap and that supplies the background gray. Any window may have child windows contained within it. Child windows obscure their parent; that is, they are above their parent in the apparent stack of windows visible on the screen. Sibling windows may overlap: the eldest sibling--the one that appears first in the list--is the sibling on top of the stack. The **Window** interface contains routines for creating and destroying windows, for arranging them, and for displaying data within them.

Windows occupy (possibly overlapping) rectangular regions of the display. A window's location and size are defined in terms of its parent's location. The root window is always at bitmap location [0,0] even though its **box.place** may not be [0,0]. The **box.place** of **rootWindow** is the screen place of the bitmap `origin.routines`.

Arbitrary scrolling can be implemented quite simply by imbedding a window (the one that paints the data to be scrolled) within another window (the "frame") and then just altering the position (y coordinate for vertical scrolling) of the former within the latter; routines are provided that will perform the appropriate **BITBLT**s to minimize the area to be painted.

Within a window as shown on the bitmap, sections of bits may become incorrect because of external circumstances--for example, because a window that was hiding them was just deleted. The window package accumulates these *invalid areas* and then calls the client's display procedure to adjust them.

Normally, when the client is called to paint the invalid area(s), there are no bits in the area that are black but should be white (the window package has possibly cleared the area to ensure this) so the repaint procedure can use "or" functions. If the client knows that its repaint procedure always sets all the bits in the area(s), it indicates this in the window object, which may save the window package from performing unnecessary clearings.

When a window is being validated, a *bad phosphor list* is set up for it just before its display procedure is called. This list consists of the visible portions of the window's invalid areas. When there is a bad phosphor list for a window, any painting done to that window will be clipped to the list. This lets the client avoid calls to **EnumerateInvalidBoxes** to find exactly which regions need repainting. So, for example, if the window provides a gray background in a particular area, the display procedure may call **Display.Gray** for the entire region that the gray background should appear in. This guarantees that valid areas of the window will not be overwritten. **Window.FreeBadPhosphorList** causes this bad phosphor list to be ignored.

III.2 Sources and sinks

Sources and sinks are interfaces for data input and output. For instance, a source need not be dealt with as a particular structure, such as a disk file or a teletype, but can be thought of as a source of input (such as the backing store for screen display). Similarly, a sink can be thought of as a generic place to send output.

There are only two pre-defined sinks in XDE, **AsciiSink** and **TextSink**; most kinds of data can be put into those categories. **AsciiSink** is a special case of **TextSink**. There are several different sources, however. The interface **TextData** consists of data types shared by sources and sinks.

Advanced programmers may want to create sources and sinks to use as backing storage and output for their own text subwindows. For example, a source that maintains text attributes along with the text is required to display text in various fonts.

III.3 Interface abstracts

III.3.1 Windows

Context allows clients to associate data with windows. It is used by clients that implement their own window types.

Display provides facilities for display in windows.

III.3.2 Subwindows

Caret allows clients to implement and manage a blinking caret that marks the insertion point in editable windows.

Cursor manipulates the appearance of the cursor that represents the mouse position on the screen.

Menu provides the menu facility used by many tools for simple command invocation. It gives a client control over which menus the user sees and what actions an individual menu item performs.

Scrollbar provides a mechanism for specifying and invoking scroll actions, maintaining a consistent user interface for them.

Selection allows clients to manipulate the current selection; that is, the text or graphics designated by the user and highlighted on the screen.

ToolFont provides Tajo's interface to the **WindowFont** facilities, including font storage management.

WindowFont converts `.strike` fonts into a representation more convenient for the **Window** package to display characters.

III.3.3 Sources and sinks

AsciiSink implements a text sink that outputs Ascii text. (Text sinks are defined by the interface **TextSink**.)

BlockSource creates a text source backed by a block of Ascii characters.

DiskSource creates a text source backed by a stream or a file in the local file system.

PieceSource creates a text source backed by a piece table maintained on a text source.

ScratchSource creates a text source backed by a block of virtual memory containing Ascii characters.

StringSource creates a text source backed by a **LONG STRING** containing Ascii text.

TextData provides the definitions of data types that a few procedures in **TextSW** and **FormSW** need. It is not of interest to most clients.

TextSink defines a *sink* for text that is displayed in a window. Text sinks help isolate Tajo's uniform text display, selection, and editing facilities from the representation of text. It is intended for clients that have other than Ascii representation of information. The standard interface **AsciiSink** is provided for normal clients and is used as the default.

TextSource defines a *source* for text that is displayed in a window. Sources help isolate Tajo's uniform text display, selection, and editing facilities from the representation of text. It is intended for clients that wish to maintain their own data structures to be displayed in a window.

22

Context

When a tool performs various functions, it may wish to save and retrieve state from one notification to the next. This is an immediate consequence of the notification scheme, for a tool cannot keep its state in the program counter without stealing the processor after responding to an event. Thus a tool must explicitly store its state in data. Because most notification calls to a tool provide a window or subwindow handle, it is natural to associate these *contexts* with windows. The context mechanism is an alternative to the tool's having to build its own associative memory for retrieving its context, given a window handle.

22.1 Types

```
Context.Type: TYPE = MACHINE DEPENDENT{
  all(0), first(1), lastAllocated(37737B), last(37777B)};
```

Type is unique for each client of the context mechanism. An argument of this type is passed to most of the procedures in this interface so that the correct client data can be identified.

```
Context.Data: TYPE = LONG POINTER TO UNSPECIFIED;
```

Data, the value that a client may associate with each window, is typically a pointer to a record containing the client's state for some window.

```
Context.CreateProcType: TYPE =
  PROCEDURE RETURNS [Context.Data, Context.DestroyProcType];
```

CreateProcType is used by **FindOrCreate**. The procedure passed in as an argument to **FindOrCreate** is called to create a context only if a context of the appropriate type cannot be found.

```
Context.DestroyProcType: TYPE = PROCEDURE [Context.Data, Window.Handle];
```

A **DestroyProcType** is passed to **Create** so that the client can be notified when the context should be destroyed. It may be the result of the window being destroyed.

22.2 Constants and data objects

None.

22.3 Signals and errors

```
Context.Error: ERROR [code: Context.ErrorCode];
```

Error is the only error raised by any of the **Context** procedures.

```
Context.ErrorCode: TYPE = {duplicateType, windowIsNIL, tooManyTypes, other};
```

duplicateType is raised by **Create** if there is already a context of the given type on the window passed as an argument.

windowIsNIL is raised if the client has passed in a **NIL** window.

tooManyTypes is raised if **UniqueType** has been called too many times.

22.4 Procedures

```
Context.Acquire: PROCEDURE [type: Context.Type, window: Window.Handle]
  RETURNS [Context.Data];
```

The procedure **Acquire** retrieves the **data** field from the specified window. **NIL** is returned if no such context exists on the window. It also locks the context object so that no other calls on **Acquire** or **Destroy** with the same **type** and **window** will complete until the context is freed by a call on **Release**.

```
Context.Create: PROCEDURE [
  type: Context.Type, data: Context.Data, proc: Context.DestroyProcType,
  window: Window.Handle];
```

The procedure **Create** creates a new context of type **type** that contains **data**. The context is associated with the indicated window; it is said to "hang" on the window. If **window** already possesses a context of the specified type, the **ERROR Error[duplicateType]** is raised. If the **window** is **NIL**, the **ERROR Error[windowIsNIL]** is raised. The **proc** is supplied so that when the window is destroyed, all the context data can be destroyed (deallocated) in an orderly way.

```
Context.Destroy: PROCEDURE [type: Context.Type, window: Window.Handle];
```

The procedure **Destroy** destroys a context of a specific **type** on **window**. If the context exists on the window, it calls the **DestroyProcType** for the context being destroyed.

```
Context.DestroyAll: PROCEDURE [window: Window.Handle];
```

The procedure **DestroyAll** destroys all the contexts on **window**. **DestroyAll** can be very dangerous because Tajo keeps its window-specific data in contexts on the window. **DestroyAll** should not be used except in special circumstances. It is called by the routines that destroy windows.

```
Context.Find: PROCEDURE [type: Context.Type, window: Window.Handle]
  RETURNS [Context.Data];
```

The procedure **Find** retrieves the **data** field from the specified context for **window**. **NIL** is returned if no such context exists on the window.

```
Context.FindOrCreate: PROCEDURE [
  type: Context.Type, window: Window.Handle, createProc: Context.CreateProcType]
  RETURNS [Context.Data];
```

The procedure **FindOrCreate** resolves the outcome of the race that occurs when creating new contexts in a multi-process environment. If a context of type **type** exists on **window**, it returns the context's **data**; otherwise, it creates a context of **type** by calling **createProc** and then return **data**. If the **window** is **NIL**, the **ERROR Error[windowIsNIL]** is raised.

```
Context.NopDestroyProc: Context.DestroyProcType;
```

The procedure **NopDestroyProc** does nothing. It is provided as a convenience to clients that do not want to create their own "do-nothing" **DestroyProcType** to pass to **Create**.

```
Context.Release: PROCEDURE [type: Context.Type, window: Window.Handle];
```

The procedure **Release** releases the lock on the specified context object for **window** that was locked by the call on **Acquire**. If the specified context cannot be found or if it is not locked, **Release** is a no-op.

```
Context.Set: PROCEDURE [
  type: Context.Type, data: Context.Data, window: Window.Handle];
```

The procedure **Set** changes the actual **data** pointer of a context. Subsequent **Finds** return the new data. The client can change the data pointed to by the **data** field of a context at any time. Race conditions could occur if multiple processes are doing **Finds** for the same context and modifying the data. It is the client's responsibility to **MONITOR** the data in such cases. If the **window** is **NIL**, the **ERROR Error[windowIsNIL]** is raised.

```
Context.SimpleDestroyProc: Context.DestroyProcType;
```

The procedure **SimpleDestroyProc** merely calls the system heap deallocator on the **data** field. It is provided for clients whose context data is a simple heap node in the system zone.

```
Context.UniqueType: PROCEDURE RETURNS [type: Context.Type];
```

The procedure **UniqueType** is called if a client needs a unique **Type** not already in use either by Tajo or by another client. If no more unique **types** are available, the **ERROR Error[tooManyTypes]** is raised.

22.5 Discussion

Acquire and **Release** can be used in much the same way as a Mesa **MONITOR**. It is important that the client call **Release** for every context that has been obtained by **Acquire**; this is not done automatically. The cost of doing an **Acquire** is barely more than entering

MONITOR and doing a **Find**. Using this technique allows the client to monitor its data rather than its code.

If it is necessary for several tools to share global data, it is possible to place a context on **Window.rootWindow** that is never destroyed, even when the bitmap is turned off. To share a **Context.Type** without having to **EXPORT** a variable, you can use one in the range **(lastAllocated..last]**. Contact the support organization to have one allocated to you.

23

Display

The Display interface provides routines for painting into windows on the user's screen. (See **Window** for details of the Tajo window package.) Unless stated otherwise, all procedures that paint to the screen clip to the window's bad phosphor list. (This list is explained in the **Window** chapter.)

Some procedures in this interface are not available in the released boot file. **DisplayImpl** must be loaded before these procedures can be called.

23.1 Types

```
Display.BreakReason: TYPE = {normal, margin, stop};
```

BreakReason is returned by **Block**, **MeasureBlock**, and **ResolveBlock** to indicate why these procedures terminated.

normal all data is displayed.

margin the next character overlaps the margin.

stop the next character has no representation in the font.

```
Display.Brick: TYPE = LONG DESCRIPTOR FOR ARRAY OF CARDINAL;
```

Bricks are used by **Gray** and **Trajectory** to describe a gray pattern with which to tile a window. The maximum size of a **Brick** is 16 words; each word is one row of the pattern.

```
Display.TrajectoryProc: TYPE = PROC [Window.Handle] RETURNS [Window.Box, INTEGER];
```

TrajectoryProc is the type of the procedure that is passed to **Trajectory**. When called, the procedure should return a small area within the window where painting should occur. Think of it as a "brush stroke."

23.2 Constants and data objects

```
Display.fiftyPercent: Brick
```

This is a 50% gray pattern.

```
Display.infinity: INTEGER = INTEGER.LAST;
```

infinity is used as an argument to the **Block** and **Text** routines. It indicates that the operation should terminate at the right edge of the window.

```
Display.paintGrayFlags, bitFlags: BitBlt.BitBltFlags = [
  direction: forward, disjoint: TRUE, disjointItems: TRUE, gray: TRUE,
  srcFunc: null, dstFunc: or, reserved: 0];

Display.replaceGrayFlags, boxFlags: BitBlt.BitBltFlags = [
  direction: forward, disjoint: TRUE, disjointItems: TRUE, gray: TRUE,
  srcFunc: null, dstFunc: null, reserved: 0];

Display.xorGrayFlags, xorBoxFlags: BitBlt.BitBltFlags = [
  direction: forward, disjoint: TRUE, disjointItems: TRUE, gray: TRUE,
  srcFunc: null, dstFunc: xor, reserved: 0];

Display.replaceFlags: BitBlt.BitBltFlags = [
  direction: forward, disjoint: TRUE, disjointItems: TRUE, gray: FALSE,
  srcFunc: null, dstFunc: null, reserved: 0];

Display.textFlags, paintFlags: BitBlt.BitBltFlags = [
  direction: forward, disjoint: TRUE, disjointItems: FALSE, gray: FALSE,
  srcFunc: null, dstFunc: or, reserved: 0];

Display.xorFlags: BitBlt.BitBltFlags = [
  direction: forward, disjoint: TRUE, disjointItems: FALSE, gray: FALSE,
  srcFunc: null, dstFunc: xor, reserved: 0];
```

BitBlt.BitBltFlags are passed into several display procedures; they control what actually happens on the display. These flags are provided for some of the most common cases; they include painting, replacing, and **XOR**ing of text, bits, and gray patterns. Use **Display.paintGrayFlags** to paint black, **Display.replaceGrayFlags** to paint white, and **Display.xorGrayFlags** to invert. These flags are documented further in the *Mesa Processor Principles of Operation*.

23.3 Signals and errors

None.

23.4 Procedures

```
Display.Arc: PROC [
  window: Window.Handle, place: Window.Place, radius: INTEGER,
  startSector, stopSector: CARDINAL, start, stop: Window.Place,
  bounds: Window.BoxHandle ← NIL];
```

Arc displays a portion of a circular arc centered at **place** of **radius** in **window**. The arc goes from **start** in the **startSector** to **stop** in the **stopSector**. Sectors are simply octants numbered from 1 to 8, starting with 1 at NNE going clockwise. The arc is clipped to the **Window.Box** described by **bounds**; a **bounds** of **NIL** clips the arc at **window**'s bounding box. This procedure is not available in the released boot file. **DisplayImpl** must be loaded before it can be called.

```
Display.BitAddressFromPlace: PROC [
  base: BitBlt.BitAddress, x, y: NATURAL, raster: CARDINAL]
  RETURNS [BitBlt.BitAddress];
```

BitAddressFromPlace returns the **BitBlt.BitAddress** of the (**x**, **y**) coordinates in the bitmap described by **base**. **raster** is the number of bits per line in the bitmap. This procedure is provided as a utility to calculate the **address** parameter to **Display.Bitmap**.

```
Display.Bitmap: PROC [
  window: Window.Handle, box: Window.Box, address: BitBlt.BitAddress,
  bitmapBitWidth: CARDINAL, flags: BitBlt.BitBltFlags ← Display.paintFlags];
```

Bitmap paints the bitmap described by **address** and **bitmapBitWidth** into **box** in **window**, using **flags** to control the interaction with bits already displayed in the window. **box.dims.w** should be less than or equal to **bitmapBitWidth**. This procedure may also be used instead of **Display.Gray** to display a gray pattern that is not aligned relative to the window origin.

```
Display.Black: PROC [window: Window.Handle, box: Window.Box];
```

Black makes the region of **window** described by **box** black.

```
Display.Block: PROC [
  window: Window.Handle, block: Environment.Block,
  lineLength: INTEGER ← Display.infinity, place: Window.Place,
  font: WindowFont.Handle ← NIL, flags: BitBlt.BitBltFlags ← Display.textFlags,
  bounds: Window.BoxHandle ← NIL]
  RETURNS [newPlace: Window.Place, positions: CARDINAL, why: Display.BreakReason];
```

Block is used to display a block of characters in a window. **block** describes the block of characters to be displayed. The characters are painted into **window** starting at **place**. The total width of the characters painted will not exceed **lineLength**. If **lineLength** is **Display.infinity**, characters will be painted up to (but not past) the right edge of the window. Painting will also stop if **block** is consumed or a character is encountered that is not represented in **font**. **font** is the character font to be used; if **font** is **NIL**, the default font will be used. (See **WindowFont.SetDefault**.) **flags** is used to affect how the bits are painted into the window. **bounds** is an optional box to which the text should be clipped. **newPlace** is

where the next character would have been painted. **positions** is the number of characters painted. **why** is the reason painting was stopped.

```
Display.Character: PROC [
  window: Window.Handle, char: CHARACTER, place: Window.Place,
  font: WindowFont.Handle ← NIL, flags: BitBlt.BitBltFlags ← Display.textFlags,
  bounds: Window.BoxHandle ← NIL]
  RETURNS [Window.Place];
```

Character displays a single character. If the character has no representation in **font**, the special *undefined* character in the font will be displayed. **bounds** is an optional box to which the character should be clipped. The returned **Window.Place** is where the next character should be displayed.

```
Display.Circle: PROC [
  window: Window.Handle, place: Window.Place, radius: INTEGER,
  bounds: Window.BoxHandle ← NIL];
```

Circle displays a circle centered at **place** of **radius** in **window**. The circle is clipped to the **Window.Box** described by **bounds**; a **bounds** of **NIL** clips the circle to **window**'s bounding box. This procedure is not available in the released boot file. **DisplayImpl** must be loaded before it can be called.

```
Display.Conic: PROC [
  window: Window.Handle, a, b, c, d, e, errorTerm: LONG INTEGER,
  start, stop, errorRef: Window.Place,
  sharpCornered, unboundedStart, unboundedStop: BOOLEAN,
  bounds: Window.BoxHandle ← NIL];
```

Conic displays the portion of the curve of the equation $\mathbf{a}x^2 + \mathbf{b}y^2 + \mathbf{c}xy + \mathbf{d}x + \mathbf{e}y + f = 0$ in **window** from **start** to **stop**. Instead of passing in the last coefficient, *f*, this procedure takes the **errorTerm** resulting from substituting **start** into the equation. If the conic contains points whose radius of curvature is less than or equal two pixels, it must be displayed using multiple calls with **sharpCornered** boolean **TRUE**; otherwise **sharpCornered** should be **FALSE**. These "sharp-cornered" conics must be broken up into segments where the corners become a new segment's start and stop points. For example, a very long skinny ellipse must be displayed in two pieces. **errorRef** and the booleans **unboundedStart** and **unboundedStop** are ignored. The curve is clipped to the **Window.Box** described by **bounds**; a **bounds** of **NIL** clips to the **window**'s bounding box. This procedure is not available in the released boot file. **DisplayImpl** must be loaded before it can be called.

```
Display.Ellipse: PROC [
  window: Window.Handle, center: Window.Place, xRadius, yRadius: INTEGER,
  bounds: Window.BoxHandle ← NIL];
```

Ellipse only displays ellipses with axes parallel to the x-y coordinate system centered at **center** with an x radius of **xRadius** and a y radius of **yRadius** in **window**. The ellipse is clipped to the **Window.Box** described by **bounds**; a **bounds** of **NIL** clips the ellipse to **window**'s bounding box. Other types of ellipses must be displayed with the **Display.Conic** procedure. This procedure is not available in the released boot file. **DisplayImpl** must be loaded before it can be called.

```
Display.Gray: PROC [
  window: Window.Handle, box: Window.Box, gray: Brick ← Display.fiftyPercent,
  dstFunc: BitBlt.DstFunc ← null];
```

Gray paints the the gray pattern described by **gray** into the **box** region of **window**. **dstFunc** affects how the bits are painted into the window. The gray pattern is aligned relative to the window origin.

```
Display.Invert: PROC [window: Window.Handle, box: Window.Box];
```

Invert inverts the **box** region of **window**.

```
Display.Line: PROC [
  window: Window.Handle, start, stop: Window.Place,
  bounds: Window. BoxHandle ← NIL];
```

Line displays a single pixel-wide line from **start** to **stop** in **window**. The line is clipped to the **Window.Box** described by **bounds**; a **bounds** of **NIL** clips the line to **window**'s bounding box. This procedure is not available in the released boot file. **DisplayImpl** must be loaded before it can be called.

```
Display.MeasureBlock: PROC [
  window: Window.Handle, block: Environment.Block,
  lineLength: INTEGER ← Display.infinity, place: Window.Place,
  font: WindowFont.Handle ← NIL]
  RETURNS [newPlace: Window.Place, positions: CARDINAL, why: Display.BreakReason];
```

MeasureBlock is used to measure the length of a block of text if it were painted to the screen. The arguments and return values are the same as described by **Display.Block**.

```
Display.Point: PROC [window: Window.Handle, point: Window.Place];
```

Point turns a single pixel black at **point** in **window**, if it is visible.

```
Display.ResolveBlock: PROC [
  window: Window.Handle, block: Environment.Block,
  offsets: LONG POINTER TO ARRAY CARDINAL [0. . 0) OF CARDINAL,
  font: WindowFont.Handle ← NIL]
  RETURNS [positions: CARDINAL, why: Display.BreakReason];
```

ResolveBlock is used to determine the locations of characters in a block of text. The offset of the left edge of each character in **block** is stored into **offsets**. It is the client's responsibility to ensure that this array is long enough to hold the offsets of all the characters in block. This procedure terminates either because it has reached the end of block (**why** = **normal**) or it has reached a character that has no representation in font (**why** = **stop**). In either case, **positions** is the number of characters processed.

```
Display.Shift: PROC [
  window: Window.Handle, box: Window.Box, newPlace: Window.Place];
```

Shift does a bitblt-style move of part of the window contents. **box** describes the region of **window** to be moved to **newPlace**. This call may produce invalid areas within the window (bits that should be moved into visible areas of the window but are not available because they have either been clipped or obscured). To avoid difficulties with the client's display

procedure, it is not called; this call simply leaves the **window** marked invalid. It is the client's responsibility to call Window.**Validate** or Window.**ValidateTree** as soon as it has corrected its data structures to reflect the call. **Shift** does not invalidate the areas where the box has been moved "from." If they should be repainted, invalidating them is the client's responsibility. **Shift** does not clip the actual region painted to **window**'s bad phosphor list (see the **Window** chapter for an explanation of the bad phosphor list.)

```
Display.Text: PROC [
  window: Window.Handle, string: LONG STRING, place: Window.Place,
  font: WindowFont.Handle ← NIL, lineLength: INTEGER ← Display.infinity,
  flags: BitBlt.BitBltFlags ← Display.textFlags, bounds: Window.BoxHandle ← NIL]
  RETURNS [newPlace: Window.Place];
```

Text uses a single call on Display.**Block** to paint characters from **string** at **place** in **window**. The value returned is the window-relative place where the next character should go. Note that the string is painted only up to the first character that is not represented in **font**.

```
Display.TextInline: PROC [
  window: Window.Handle, string: LONG STRING, place: Window.Place,
  font: WindowFont.Handle ← NIL, lineLength: INTEGER ← infinity,
  flags: BitBlt.BitBltFlags ← Display.textFlags, bounds: Window.BoxHandle ← NIL]
  RETURNS [Window.Place] = INLINE {
  RETURN[Display.Block[
    window, [LOOPHOLE[@string. text], 0, string.length], lineLength, place,
    font, flags, bounds].newPlace]};
```

TextInline is an INLINE version of Display.**Text** provided for clients who are willing to trade some code space in their own module to avoid an extra procedure call at run time.

```
Display.Trajectory: PROC [
  window: Window.Handle, box: Window.Box ← Window.nullBox,
  proc: Display.TrajectoryProc, source: LONG POINTER ← NIL, bpl: CARDINAL ← 16,
  height: CARDINAL ← 16, flags: BitBlt.BitBltFlags ← Display.bitFlags,
  missesChildren: BOOLEAN ← FALSE, brick: Display.Brick ← NIL];
```

Trajectory is designed to avoid much of the overhead of successive calls to the normal display routines. **window** is the window of interest. **box** is the window region where painting might occur; the client promises it will not try to paint outside this area. **proc** is the client procedure that, when called, repeatedly returns a window-relative box in which painting should occur (think of it as a brush stroke) and the x-offset into the client's **source** data. To end the trajectory, **proc** should return Window.**nullBox**. The client may wish to alter the brush shape along the trajectory by defining the **source** bitmap as a wide one with several different brush shapes in it and then returning the x-offset into the source bitmap with the brush-box. **flags** is used to describe the type of painting that should be performed on each small area. The use of this argument is similar to Display.**Bitmap**. **brick** is a gray brick to be used if **flags.gray** is TRUE. (This is described in more detail for Display.**Gray**.) **missesChildren** is unused.

```
Display.White: PROCEDURE [window: Window.Handle, box: Window.Box];
```

White makes the region of **window** described by **box** white.

24

Window

The **Window** interface defines the window management package that Tajo uses. These procedures are mostly of interest to clients who are implementing their own subwindow types. (See **Display** for routines that paint into windows.)

24.1 Types

```
Window.Box: TYPE = RECORD [place: Window.Place, dims: Window.Dims];
```

Box describes a window-relative region. **place** describes the top left corner of the region. **[place.x + dims.w, place.y + dims.h]** describes the bottom right corner of the region. This point is actually outside the region described by the **Box**.

```
Window.BoxHandle: TYPE = LONG POINTER TO Box;
```

```
Window.Clarity: TYPE = {isClear, isDirty};
```

Clarity is used by a client of **InvalidateBox** to indicate whether an invalid region is known to be white.

isClear window package believes that the region is all white and performs no clearing.

isDirty window package believes that the region is not all white and clears it.

```
Window.Dims: TYPE = RECORD [w, h: INTEGER];
```

Dims is the size of a window. **w** is the the number of pixels in the window's width. **h** is the the number of pixels in the window's height.

```
Window.Gravity: TYPE = {nil, nw, n, ne, e, se, s, sw, w, c, xxx};
```

Gravity indicates what to do with the current contents of a window when it changes size.

nil the contents stay in the same place on the bitmap.

nw, n, ne, e, se, s, sw, w	the contents stay attached to the indicated compass point, which is either a corner or the middle of a side (e.g., for **nw** the contents stay in the upper-left corner).
c	the contents stay in the middle (i.e., trimming occurs equally at all edges).
xxx	no attempt is made to save the contents: the window is repainted.

```
Window.Handle: TYPE = LONG POINTER TO Window.Object;
```

Handle represents a window.

```
Window.MinusLandBitmapUnder: TYPE = [4];
```

MinusLandBitmapUnder is provided for clients who need to allocate their own window objects.

```
Window.MouseTransformerProc: TYPE = PROC [Window.Handle, Window.Place]
  RETURNS [Window.Handle, Window.Place];
```

MouseTransformerProc is not supported in this release.

```
Window.Object: TYPE = [18]
```

```
Window.Place: TYPE = UserTerminal.Coordinate;
```

Place is a window-relative coordinate.

```
Window.UnderChangedProc: TYPE = PROCEDURE [Window.Handle, Window.Box];
```

UnderChangedProc is not supported in this release.

24.2 Constants and data objects

```
Window.nullBox: Window.Box = [[0,0], [0,0]];
```

nullBox is a zero-sized **Box** at the upper-left corner of a window.

```
Window.rootWindow: READONLY Window.Handle;
```

rootWindow is the exported root of the window tree. It represents the entire display.

24.3 Signals and errors

```
Window.Error: ERROR [code: Window.ErrorCode];
```

Error is the only error raised by any of the **Window** procedures.

```
Window.ErrorCode: TYPE = {illegalBitmap, illegalFloat, windowNotChildOfParent,
  whosSlidingRoot, noSuchSibling, noUnderVariant,windowInTree,
  sizingWithBitmapUnder, illegalStack};
```

illegalBitmap — This error is never raised.

illegalFloat — The client passed illegal parameters to **Float**.

windowNotChildOfParent — The window passed as a parameter is not in the list of its parent's children. This error can be raised by any procedure that deals with a window; that is, by most of the procedures in the **Window** interface.

whosSlidingRoot — The client has attempted to move the root window.

noSuchSibling — The client has requested a change to the window tree, asking that a window's new sibling be a window that is not a child of its new parent.

noUnderVariant — A client has attempted to manipulate the **bitmapUnder** data of a window for which **underVariant** is FALSE.

windowInTree — An attempt was made to use one of the "Set" procedures on a window that is currently a descendant of **rootWindow**. In most cases, you should use one of the **SlideAndSizeAndStack** procedures instead.

sizingWithBitmapUnder — A client has tried to change the size of a window that currently has a bitmap under.

illegalStack — The client is attempting to move a window between parents, one of which is in the window tree and the other is not.

24.4 Procedures

```
Window.BitmapPlace: PROC [
  window: Window.Handle, place: Window.Place ←[0,0] ] RETURNS [Window.Place];
```

BitmapPlace returns the bitmap-relative coordinates of **place** in **window**.

```
Window.BitmapPlaceToWindowAndPlace: PROC [bitmapPlace: Window.Place]
  RETURNS [window: Window.Handle, place: Window.Place];
```

Given a bitmap-relative place, **bitmapPlace**, **BitmapPlaceToWindowAndPlace** returns the most deeply nested window containing **bitmapPlace** and the window-relative coordinates of **bitmapPlace**.

```
Window.BoxesAreDisjoint: PROC [a, b: Window.Box] RETURNS [BOOLEAN];
```

BoxesAreDisjoint returns TRUE if and only if **a** and **b** do not intersect.

```
Window.EnumerateInvalidBoxes: PROC [
  window: Window.Handle, proc: PROC [Window.Handle, Window.Box]];
```

EnumerateInvalidBoxes procedure calls **proc** for each of the invalid boxes of **window**; it should only be called from within **window**'s display procedure. **window** is passed through to **proc** as its first parameter. The second parameter to **proc** describes the region that is invalid. The invalid areas are clean unless the client has set **clearingNotRequired** for window; that is, there are no pixels in them that are currently black but should be white.

```
Window.EnumerateTree: PROC [
  root: Window.Handle, proc: PROC [window: Window.Handle]];
```

EnumerateTree calls **proc** for each window that is a descendant of **root**. **root** need not itself be a descendant of **Window.rootWindow**. The order of enumeration is not specified.

```
Window.Float: PROC [window, temp: Window.Handle,
  proc: PROC [window: Window.Handle]
    RETURNS [place: Window.Place, done: BOOLEAN]];
```

Float changes **window**'s position and adjusts the display. It requires that **window** be a bitmap-under window. It also requires that the user supply for scratch storage a **temp** window with a bitmap under, exactly the same size as **window** but not in the window tree. **Float** repeatedly calls **proc** and does a continuous move to the new place as long as **done** is **FALSE**. The window is forced to the top of the sibling stack before the move begins. A new place that would require moving the window so it is not completely visible is a client error. **ValidateTree** is called to pick up the bits that must be on the bitmap when the **window** is moved away. This procedure can raise the error **Error[illegalFloat]**.

```
Window.FreeBadPhosphorList : PROC [window: Window.Handle];
```

FreeBadPhosphorList forces the window package to ignore **window**'s bad phosphor list when painting to it.

```
Window.GetBitmapUnder: PROC [window: Window.Handle] RETURNS [LONG POINTER];
```

GetBitmapUnder returns a long pointer to the bitmap-under data for **window**. **window** must be a bitmap-under variant or **Error[noUnderVariant]** will be raised. If there is no current bitmap-under pointer, this procedure returns **NIL**.

```
Window.GetBox: PROC [Window.Handle] RETURNS [Window.Box];
```

GetBox returns the current **Box** for a window.

```
Window.GetChild: PROC [Window.Handle] RETURNS [Window.Handle];
```

GetChild returns the window's topmost (eldest) child.

```
Window.GetClearingRequired: PROC [Window.Handle] RETURNS [BOOLEAN];
```

GetClearingRequired returns the current value of the clearing-required flag for a window.

```
Window.GetDisplayProc: PROC [Window.Handle] RETURNS [PROC [Window.Handle]];
```

GetDisplayProc returns the window's display procedure.

```
Window.GetParent: PROC [Window.Handle] RETURNS [Window.Handle];
```

GetParent returns the the window's current parent.

```
Window.GetSibling: PROC [Window.Handle] RETURNS [Window.Handle];
```

GetSibling returns the window's topmost (eldest) sibling.

```
Window.InitializeWindow: PROC [
  window: Window.Handle, display: PROC [Window.Handle], box: Window.Box,
  parent: Window.Handle ← Window.rootWindow, sibling, child: Window.Handle ← NIL,
  clearingRequired: BOOLEAN ← TRUE, under: BOOLEAN ← FALSE];
```

InitializeWindow sets the values of the listed fields in the window object. This procedure should be called before **InsertIntoTree**. (Most Tajo clients should not need this procedure.)

```
Window.InsertIntoTree: PROC [window: Window.Handle];
```

InsertIntoTree adds the client-supplied **Window.Object** to the window tree. The caller must have set the following fields of the window object by calling **InitializeWindow** or one of the "Set" procedures: **parent**, **sibling**, **child**, **display**, **under**. **sibling** should be **NIL** if this window is to be the last child of its parent. The root window must have been defined before this procedure is called. The client can force all the just-inserted windows to be painted by calling **ValidateTree** and passing a window that contains all of the inserted windows. If an inserted window has a bitmap under and the new window is partially obscured (if all the bits needed for the bitmap under are not available), then **ValidateTree** is called on the parent of the inserted window to obtain those bits. This procedure can raise **Error[noSuchSibling]**. (Most Tajo clients should not need this procedure.)

```
Window.IntersectBoxes; PROC [b1,b2: Window.Box] RETURNS [box: Window.Box];
```

IntersectBoxes returns a **Box** that is the intersection of **b1** and **b2**. If their intersection is empty, **Window.nullBox** is returned.

```
Window.InvalidateBox: PROC [
  window: Window.Handle, box: Window.Box, clarity: Window.Clarity ← isDirty];
```

InvalidateBox adds the region described by **box** to the list of invalid regions of **window**. **clarity** controls whether the window package should clear the region; if **clarity** is **isClean**, the region is not cleared. **InvalidateBox** does not update the display; the client should call **Validate** on **window** to cause the window package to update the display. The client should not call its display procedure directly when its window needs repainting. Instead, it should update its data to reflect the newly desired content and call **InvalidateBox**. A call on **InvalidateBox** followed by a call on **Validate** may result in no call to the display procedure if, for instance, the invalidated areas stick out of the parent.

```
Window.IsBitmapUnderVariant:PROC[Window.Handle]RETURNS [BOOLEAN];
```

IsBitmapUnderVariant returns the value of the **under** parameter as of the last call on **InitializeWindow** for the window. If **InitializeWindow** has not been called, this procedure returns **FALSE**.

```
Window.IsDescendantOfRoot: PROC [Window.Handle] RETURNS [BOOLEAN];
```

IsDescendantOfRoot determines if the window is currently a part of the tree rooted at **Window.rootWindow**.

```
Window.IsPlaceInBox: PROC [place: Window.Place, box: Window.Box] RETURNS [BOOLEAN];
```

IsPlaceInBox is a utility that determines whether **place** is inside **box**. Points on **box**'s border are considered to be inside.

```
Window.ObscuredBySibling: PROC [Window.Handle] RETURNS [BOOLEAN];
```

ObscuredBySibling returns **TRUE** if and only if the **box** of an older sibling (one closer to the top of the sibling stack) intersects **window**'s box.

```
Window.RemoveFromTree: PROC [Window.Handle];
```

RemoveFromTree removes the argument window and its children from the visible window tree. (Most Tajo clients should never have to call this procedure.)

```
Window.Root: PROC RETURNS [Window.Handle];
```

Root returns **Window.rootWindow**.

```
Window.SetBitmapUnder: PROC [
  window: Window.Handle, pointer: LONG POINTER ← NIL,
  underChanged: Window.UnderChangedProc ← NIL,
  mouseTransformer: Window.MouseTransformerProc ← NIL]
  RETURNS [LONG POINTER];
```

SetBitmapUnder allows the client to specify a bitmap under for the window, allowing the window package to maintain the pixels that would appear on the display if the window did not exist. The window package can thus quickly adjust the display when the window is removed from the tree without having to call the display procedure of all the (partially) hidden windows. A client clears the data by passing in **NIL** for **pointer**. The old value of the data pointer is returned, and the client can free it at that time. The allocation of an appropriate amount of space is the caller's responsibility (see **Window.WordsForBitmapUnder**.) The **underChanged** and **mouseTransformer** parameters are ignored in the current release. While the bitmap under is in effect, the window's size cannot be changed. This procedure can raise **Error[noUnderVariant]**.

```
Window.SetChild: PROC [window, newChild: Window.Handle]
  RETURNS [oldChild: Window.Handle];
```

SetChild allows you to change the value of **window**'s eldest child. This procedure should not be called for a window that is part of the visible window tree: **Window.Error[inTree]** will be raised in this case. Use **Window.Stack** instead.

```
Window.SetClearingRequired: PROC [window: Window.Handle, required: BOOLEAN]
  RETURNS [old: BOOLEAN];
```

SetClearingRequired changes the value of the clearing required field in **window**. It returns the old value of this field.

```
Window.SetDisplayProc: PROC [Window.Handle, PROC [Window.Handle]]
  RETURNS [PROC [Window.Handle]];
```

SetDisplayProc sets the window display procedure. It returns the old display procedure.

```
Window.SetParent: PROC [window, newParent: Window.Handle]
  RETURNS [oldParent: Window.Handle];
```

SetParent allows you to change the value of **window**'s parent. This procedure should not be called for a window that is part of the visible window tree: **Window.Error[inTree]** will be raised in this case. Use **Window.Stack** instead.

```
Window.SetSibling: PROC [window, newSibling: Window.Handle]
  RETURNS [oldSibling: Window.Handle];
```

SetSibling allows you to change the value of **window**'s eldest sibling. This procedure should not be called for a window that is part of the visible window tree: **Window.Error[inTree]** will be raised in this case. Use **Window.Stack** instead.

```
Window.Slide: PROC [window: Window.Handle, newPlace: Window.Place];
```

Slide changes **window**'s **place** within its parent. This procedure can be used for any child movement. It can raise **Error[whosSlidingRoot]**. Tajo clients do not usually call this procedure directly.

```
Window.SlideAndSize: PROC [
  window: Window.Handle, newBox: Window.Box, gravity: Window.Gravity ← nw];
```

SlideAndSize changes both the **place** and the **dims** of **window**'s box relative to **window**'s parent. (See **Window.Gravity** for the use of **gravity** in changing the size of a window.) The window package tries to minimize the amount of repainting necessary. This procedure can raise **Error[sizingWithBitmapUnder]** and **Error[whosSlidingRoot]**. Tajo clients do not usually call this procedure directly.

```
Window.SlideAndSizeAndStack: PROC [
  window: Window.Handle, newBox: Window.Box, newSibling: Window.Handle,
  newParent: Window.Handle ← NIL, gravity: Window.Gravity ← nw];
```

SlideAndSizeAndStack performs the **SlideAndSize** and **Stack** functions; that is, it changes both **window**'s box and **window**'s location in the window tree. This procedure can raise **Error[sizingWithBitmapUnder]**, **Error[illegalStack]**, and **Error[whosSlidingRoot]**. Tajo clients do not usually call this procedure directly.

```
Window.SlideAndStack: PROC [
  window: Window.Handle, newPlace: Window.Place, newSibling: Window.Handle,
  newParent: Window.Handle ← NIL];
```

SlideAndStack performs the **Slide** and **Stack** functions; that is, it changes both **window**'s place and **window**'s location in the window tree. This procedure can raise **Error[illegalStack]**, and **Error[whosSlidingRoot]**. Tajo clients do not usually call this procedure directly.

```
Window.SlideIconically: PROC [window: Window.Handle, newPlace: Window.Place];
```

SlideIconically is not currently implemented.

```
Window.Stack: PROC[
  window, newSibling: Window.Handle, newParent: Window.Handle ← NIL];
```

Stack changes **window**'s location in the window tree. If **newParent** is not **NIL**, then **window** is moved to be a child of **newParent**. The sibling list containing **window** is modified so that **window** is now immediately above **newSibling** in the stack. Supplying **newSibling** = **NIL** puts **window** on the bottom of the sibling stack. Unless **window** is already on top, supplying **newSibling = window.GetParent.GetChild** puts **window** on the top of the stack. If **window** is on top, the previous expression is a client error that is not guarded against. This procedure can raise **Error[illegalStack]**. Tajo clients do not usually call this procedure directly.

```
Window.TrimBoxStickouts: PROC [
  window: Window.Handle, box: Window.Box] RETURNS [Window.Box];
```

TrimBoxStickouts returns a box that is the result of excluding any portion of **box** that sticks out of **window** or its ancestors.

```
Window.Validate: PROC [window: Window.Handle];
```

Validate calls **window**'s display procedure if **window** has any visible invalid regions.

```
Window.ValidateTree: PROC [window: Window.Handle ← Window.rootWindow];
```

ValidateTree calls the display procedure for each window in the tree rooted at **window** that has any visible invalid regions.

```
Window.WordsForBitmapUnder: PROC [window: Window.Handle] RETURNS [CARDINAL];
```

The **WordsForBitmapUnder** procedure returns the number of words of storage needed for a **bitmapUnder** for a window the size of **window.GetBox.dims**.

25

Caret

The **Caret** interface provides a way for clients to manage a blinking caret that marks the insertion point. It is intended for clients implementing their own subwindow types. The procedures in this interface create a caret, clear it, cause it to blink, and start or stop it from blinking, regardless of which client is the current manager. A client can also implement a set of actions to perform when another client forces it to relinquish control of the caret.

This interface does *not* determine where a caret should be displayed, nor can it paint the caret on the screen. The client must maintain the information necessary for positioning and displaying the caret. Whenever an action is to be performed on the caret, client procedures should not only implement the definition of the various caret actions but also position and display it.

25.1 Types

```
Caret.Action: TYPE = MACHINE DEPENDENT {
  clear(0), mark(1), invert(2), start(3), stop(4), reset(5), firstFree(6), last(255)};
```

action defines the operations that can be performed on a caret.

clear removes the caret.

mark creates the caret and sets it to the on (positive) polarity.

invert sets it to off (negative) polarity.

start starts the caret blinking between the on and off polarities.

stop stops it from blinking.

reset causes the current owner to relinquish control of the caret.

firstFree is used internally by **UniqueAction** and should not be used by Tajo clients.

```
Caret.ClientData: TYPE = LONG POINTER;

Caret.MarkProcType: TYPE = PROCEDURE [data: Caret.ClientData, action: Caret.Action];
```

A **MarkProcType** procedure is provided by the manager of a caret to execute actions on a caret.

25.2 Constants and data objects

None.

25.3 Signals and errors

None.

25.4 Procedures

```
Caret.ActOn: PROCEDURE [Caret.Action];
```

The **ActOn** procedure allows clients to act on the current caret without regard to the current owner.

```
Caret.NopMarkerProc: Caret.MarkProcType;
```

The **NopMarkerProc** procedure is used by a client that does not want to display anything on the screen when it is the manager of the caret. It is passed as the **marker** parameter to the **Set** procedure.

```
Caret.ResetOnMatch: PROCEDURE [data: Caret.ClientData];
```

The **ResetOnMatch** procedure allows a client to relinquish control of the blinking caret if it is currently the owner. If **data** is **NIL**, no actions are performed. Simply doing a **Caret.Set** with **data** set to **NIL** and a **marker** that is the **NopMarkerProc** does not accomplish the same effect because of race conditions in an arbitrary pre-emption environment.

```
Caret.Set: PROCEDURE [data: Caret.ClientData, marker: Caret.MarkProcType];
```

The **Set** procedure allows a client to become the manager of the caret. **data** is passed back to **marker** whenever it is called. If a client does not want to mark the display when it is the manager of the caret, it can use **NopMarkerProc** as its **marker**. If **data** is **NIL**, then the caret's current manager is forced to relinquish control. No client manages the caret until the next **Set** operation is performed with a non-nil **data** value.

```
Caret.UniqueAction: PROCEDURE RETURNS [Caret.Action];
```

The **UniqueAction** procedure allows clients to define private actions. Implementors of caret-marking procedures should thus ignore actions they do not implement.

26

Cursor

The **Cursor** interface provides a procedural interface to the hardware mechanism that implements the cursor on the screen. To prevent chaos, all tools must manipulate the cursor through this interface.

26.1 Types

```
Cursor.Defined: TYPE = Cursor.Type [activate..groundedText];
```

There is a distinction between user and system-manufactured cursors. To keep things straight, clients may access system cursors only by their type. The range **Defined** contains the system-manufactured cursors.

```
Cursor.Handle: TYPE = POINTER TO Cursor.Object;

Cursor.Info: TYPE = RECORD [type: Cursor.Type, hotX: [0..16), hotY: [0..16)];

Cursor.Object: TYPE = RECORD [info: Cursor.Info, array: UserTerminal.CursorArray];
```

The cursor facilities define an **Object** that contains a cursor type, a specification of which bit in the cursor is to be considered "hot", and a 16-by-16 array of bits that is the bitmap for the cursor (i.e., the array of bits that are or'ed into the display). When the cursor is on the screen, the "hot" bit is the place to which the cursor points.

```
Cursor.Type: TYPE = MACHINE DEPENDENT{
  activate(0), blank(1), bullseye(2), confirm(3), crossHairsCircle(4), ftp(5), ftpBoxes(6),
  hourGlass(7), lib(8), menu(9), mouseRed(10), mouseYellow(11), mouseBlue(12), mtp(13),
  pointDown(14), pointLeft(15), pointRight(16),pointUp(17), questionMark(18), retry(19),
  scrollDown(20), scrollLeft(21), scrollLeftRight(22), scrollRight(23), scrollUp(24),
  scrollUpDown(25), textPointer(26), typeKey(27), groundedText(28), last(377B)};
```

26.2 Constants and data objects

The cursors in the subrange **Type[activate..groundedText]** are built in (system supplied). Some special notes on what some of the built-in cursors look like follow:

activate	used by the Librarian interface to indicate that a libject is being activated. LIB is in the upper half, ACT in the lower.
ftp	used to indicate a file transfer in progress. FTP is along the diagonal from the upper left to the lower right; triangles are in in the lower-left and upper-right corners.
ftpBoxes	also used to indicate a file transfer in progress. Black quadrants are in the upper left and lower right, white quadrants elsewhere.
lib	used to indicate a Librarian transaction in progress. LIB is along the diagonal from the upper left to the lower right; triangles are in the lower-left and upper-right corners.
mouseRed	a three-button mouse with the left button highlighted.
mouseYellow	a three-button mouse with the center button highlighted.
mouseBlue	a three-button mouse with the right button highlighted.
textPointer	an arrow pointing up and to the left.
groundedText	a **textPointer** with a small bar though the tail.

26.3 Signals and errors

None.

26.4 Procedures

Cursor.Fetch: PROCEDURE [Cursor.Handle];

The **Fetch** procedure copies the current cursor object into the cursor object pointed to by **Handle**.

Cursor.FetchFromType: PROCEDURE [cursor: Cursor.Handle, type: Cursor.Defined];

The **FetchFromType** procedure copies the cursor object corresponding to **type** into the cursor object pointed to by **Handle**.

Cursor.GetInfo: PROCEDURE RETURNS [Cursor.Info];

The **GetInfo** procedure allows clients to find out about the current cursor.

Cursor.Invert: PROCEDURE RETURNS [BOOLEAN];

The **Invert** procedure makes each white bit in the current cursor black, and vice versa. It returns **TRUE** if the new state of the cursor is positive.

```
Cursor.MakeNegative: PROCEDURE;
```

The **MakeNegative** procedure is equivalent to **MakePositive** followed by **Invert**.

```
Cursor.MakePositive: PROCEDURE;
```

The **MakePositive** procedure restores the current cursor's polarity to be as if a **Set** or **Store** had just been done.

```
Cursor.MoveIntoWindow: PROCEDURE [
  window: Window.Handle, place: Window.Place];
```

The **MoveIntoWindow** procedure causes the cursor to appear at **place** in **window**.

```
Cursor.Set: PROCEDURE [Cursor.Defined];
```

The **Set** procedure sets the displayed cursor to be one of the system-defined cursors.

```
Cursor.Store: PROCEDURE [Cursor.Handle];
```

The **Store** procedure sets the displayed cursor to be a client-defined cursor.

```
Cursor.Swap: PROCEDURE [old, new: Cursor.Handle];
```

The **Swap** procedure places the old cursor object in to **old** ↑ and **Stores** the **new** cursor.

```
Cursor.UniqueType: PROCEDURE RETURNS [Cursor.Type];
```

The **UniqueType** procedure lets clients assign a unique type to their defined cursors. It returns a **Cursor.Type** that is different from all predefined types as well as different from any that has previously been returned by **UniqueType**.

27

Menu

The **Menu** interface gives a tool writer control over which menus the user sees and what actions an individual menu item performs. The General Tools section of the *XDE User's Guide* describes how menus appear to the user and how to interact with them.

27.1 Types

```
Menu.EnumerateFor: TYPE = {all, inSW, availableInSW};
```

EnumerateFor is used to control which menus will be passed back to you by **Enumerate**.

all	all menus instantiated with a window should be enumerated.
inSW	only menus instantiated with a subwindow are enumerated.
availableInSW	all menus that the user could display for a subwindow are enumerated (including the system menus and menus instantiated on the Tool window).

```
Menu.EnumerateProcType: TYPE =
  PROCEDURE [window: Window.Handle, menu: Menu.Handle]
  RETURNS [stop: BOOLEAN];
```

This procedure type is used with the **Enumerate** procedure. **window** is the window to which **menu** is attached, and **menu** is one of the menus that are being enumerated. If **stop** is **TRUE**, the enumeration is terminated.

```
Menu.Handle: TYPE = LONG POINTER TO Menu.Object;
```

Most procedures in the **Menu** interface take a **Handle** as an argument.

```
Menu.ItemHandle: TYPE = LONG POINTER TO Menu.ItemObject;
```

ItemHandle is not used by the **Menu** package but is provided as a convenience to the client.

```
Menu.ItemObject: TYPE = RECORD [
  keyword: LONG STRING, mcrProc: Menu.MCRType];
```

Each menu item has a **keyword** (a string of characters) and a *Menu Command Routine (MCR)* associated with it.

```
Menu.Items: TYPE = LONG DESCRIPTOR FOR ARRAY OF Menu.ItemObject;
```

A variable of type **Items** is a parameter to the **Create** operation. This variable is stored in **Object**; the data referenced by **Items** (the keywords and procedures) must not be deallocated until the menu is destroyed.

```
Menu.MCRType: TYPE = PROCEDURE [
  window: Window.Handle ← NIL, menu: Menu.Handle ← NIL,
  index: CARDINAL ← LAST[CARDINAL]];
```

A *Menu Command Routine (MCR)* is a procedure that is called when the user invokes the associated menu item. **index** allows the procedure to determine which menu item was selected. Clients have often found that using one MCR per menu is useful because only one large catch phrase need be written to handle common exception cases.

```
Menu.Object: TYPE = RECORD [
  permanent: BOOLEAN,
  nInstances: CARDINAL [0..77777B],
  name: LONG STRING,
  items: Menu.Items];
```

The **Object** contains the normally invariant data associated with a menu. An unlimited number of menus may be associated (instantiated) with the Tool window or any subwindow. The menu mechanism maintains a ring of menu instances (pointers to associated menus) for each subwindow (if there is at least one associated menu). One of these associated menus is taken to be the "current" menu for that subwindow. Some menus (at least the system global ones) want to be available from virtually every subwindow. This could be accomplished by creating an **Object** for each use, but the primary memory cost of multiple copies of an **Object** is large. In addition, you may want to dynamically alter the items contained in menus (such as lists of available fonts). As a result, a level of indirection is used. Thus, Tajo never copies a client's **Object**; instead it always keeps a pointer to that **Object**. It is the client's responsibility to guarantee that the **Object** is valid as long as Tajo has a pointer to it. The client should only **Make** or **Create** a menu once, but you may **Instantiate** that menu over as many windows as you like. **Objects** are created and destroyed by the menu implementation.

27.2 Constants and data objects

None.

27.3 Signals and errors

```
Menu.Error: ERROR [code: Menu.ErrorCode];

Menu.ErrorCode: TYPE = {
  isInstantiated, alreadyInstantiated, notInstantiated, contextNotAvailable,
  isPermanent, other};
```

isInstantiated	a client is attempting to destroy a menu that is currently instantiated by the user.
alreadyInstantiated	a client is attempting to instantiate a menu that is already instantiated.
notInstantiated	a client is attempting to un-instantiate a menu that is not instantiated.
contextNotAvailable	Tajo has detected an internal inconsistency in its data structures.
isPermanent	a client is attempting to destroy a permanent menu.

27.4 Procedures

```
Menu.Create: PROCEDURE [
  items: Menu.Items, name: LONG STRING, permanent: BOOLEAN ← FALSE]
  RETURNS [Menu.Handle];
```

The **Create** procedure allows a tool to create a menu. It returns a pointer to a menu **Object** named **name**, which is made up of **items**. The **permanent** flag indicates whether the created object can subsequently be destroyed. Ownership of **items** is passed to the menu mechanism. **name** is copied and you retain ownership of the original string, which may be a local **STRING**.

```
Menu.Destroy: PROCEDURE [Menu.Handle];
```

The **Destroy** procedure allows a tool to destroy a menu. It deallocates storage for the **Object** pointed to by **Handle**. It first verifies that the **Object** has an instantiation count = 0; if not, the **ERROR Error[isInstantiated]** is generated. See **Instantiate** and **Uninstantiate**. If the menu is permanent, the **ERROR Error[isPermanent]** is generated.

```
Menu.Enumerate: PROCEDURE [
  window: Window.Handle, which: Menu.EnumerateFor,
  proc: Menu.EnumerateProcType];
```

The **Enumerate** procedure enumerates the menus instantiated with a window. The **which** argument specifies which menus that **proc** will be called with during the enumeration. If **which** is **all**, **window** is expected to be a Tool window and all the menus instantiated with **window** are enumerated. If **which** is **inSW**, **window** is expected to be a subwindow and all the menus instantiated with the subwindow are enumerated. If **which** is **availableInSW**, **window** is expected to be a subwindow and all the menus that you could display are enumerated (this includes the system menus and menus instantiated on the Tool window). If **TRUE** is returned from **proc**, the enumeration is terminated.

```
Menu.Free: PROCEDURE [menu: Menu.Handle, freeStrings: BOOLEAN ← TRUE];
```

The **Free** procedure frees a menu, optionally freeing the copied strings. **Free** is the complement of **Make**. After freeing the items that were created in the call to **Make**, **Destroy** is called.

```
Menu.FreeItem: PROCEDURE [Menu.ItemObject];
```

The **FreeItem** procedure frees a menu item.

```
Menu.GetFont: PROCEDURE RETURNS [font: WindowFont.Handle];
```

The **GetFont** procedure allows a tool to get a handle for the font used for menus.

```
Menu.Instantiate: PROCEDURE [menu: Menu.Handle, window: Window.Handle];
```

The menus chosen for display depend on the window that the cursor is over. This allows the displayed menu stack to vary, depending on the window layout. The **Instantiate** procedure associates the menu with the passed window so it will be displayed when the cursor is over that window. It also increments a use count in **menu**. If this is the first menu to be instantiated in **window**, the window manager menu is also instantiated. If **menu** is NIL,only the system global window manager menu is instantiated. If **menu** is already instantiated, the ERROR **Error[alreadyInstantiated]** is generated. **Uninstantiate** is the complement of **Instantiate**.

```
Menu.Invoke: PROCEDURE [window: Window.Handle, place: Window.Place];
```

Invoke displays the menu stack that is available at that place in the window. This is normally called from a TIP.**NotifyProc** (see the TIP chapter).

```
Menu.Make: PROCEDURE [
  name: LONG STRING, strings: LONG DESCRIPTOR FOR ARRAY OF LONG STRING,
  mcrProc: Menu.MCRType, copyStrings: BOOLEAN ← TRUE,
  permanent: BOOLEAN ← FALSE]
  RETURNS [Menu.Handle];
```

The **Make** procedure makes a menu named **name** that has the elements contained in **strings**. When one of the strings is selected, the **mcrProc** is called, indicating the index of the string in the array. The **permanent** flag indicates whether the created object can subsequently be destroyed. The **copyStrings** flag indicates whether **strings** should be copied into the system heap. **Free** is the complement of **Make**. **Make** is usually followed by **Instantiate**.

```
Menu.MakeItem: PROCEDURE [keyword: LONG STRING, mcrProc: Menu.MCRType] RETURNS
[Menu.ItemObject];
```

The **MakeItem** procedure makes a menu item. **keyword** is copied and may be a local STRING.

```
Menu.MCRForKeyword: PROCEDURE [
  sw: Window.Handle, menuName, keyword: LONG STRING]
  RETURNS [mcr: Menu.MCRType, menu: Menu.Handle, index: CARDINAL];
```

The **MCRForKeyword** procedure allows the client to get the arguments necessary to invoke a menu item knowing only the subwindow, menu name, and item name. If the menu item is not found, the ERROR **Error[notInstantiated]** is generated.

```
Menu.SetFont: PROCEDURE [font: WindowFont.Handle];
```

The **SetFont** procedure allows a tool to set the font used for all menus.

```
Menu.SetPNR: PROCEDURE [window: Window.Handle];
```

If a window is not managed by Tajo (if it is a client-defined window type), the client may set the standard menu **PNR** by calling the **SetPNR** procedure. If a window is managed by Tajo, the standard menu **PNR** is already set up. (See also **PNR**.)

```
Menu.Uninstantiate: PROCEDURE [menu: Menu.Handle, window: Window.Handle];
```

The menus chosen for display depend on the window that the cursor is over. This allows the displayed menu stack to vary, depending on the window layout. The **Uninstantiate** procedure removes **menu** from the window so it will not be displayed when the cursor is over this window. It also decrements its use count. Eventual deallocation of the menu must be performed by the client. If this menu is not instantiated with this window, then the **ERROR Error[notInstantiated]** is generated. It is also possible that the **ERROR Error[contextNotAvailable]** will be generated, indicating that Tajo has detected an internal inconsistency in its data structures.

27.5 Examples

For an example of how to use menus, see ExampleTool in Appendix A.

28

Scrollbar

The **Scrollbar** interface provides a consistent user interface and mechanism for specifying and invoking scroll actions. It does not scroll (move bits on the screen).

28.1 Types

```
Scrollbar.Direction: TYPE = {forward, backward, relative};
```

A **Direction** is used to specify the type of scrolling requested.

forward scrolls the window so that data near the bottom (right) of the window is moved toward the top (left).

backward scrolls the window so that data near the top (left) of the window is moved toward the bottom (right).

relative indicates that the window should display the data at a relative location in the underlying source.

```
Scrollbar.Percent: TYPE = [0..100];
```

Percent controls the amount of information scrolled or the location in the file to be displayed. (See **ScrollProcType** for the interaction between the interpretation of **Direction** and **Percent**.) It is possible to overflow when multiplying a **Percent** with a **window.box.dims.w** while converting between percentage locations and coordinates.

```
Scrollbar.ScrollbarProcType: TYPE = PROCEDURE [window: Window.Handle]
  RETURNS [box: Window.Box, offset, portion: Scrollbar.Percent];
```

A **ScrollbarProcType** procedure gets the scrollbar data from the client to display it. **box** is the region of window occupied by the scrollbar; **offset** is the relative position in the file occupied by the first character in the window and **portion** is the percentage of the file displayed (the percentage of the file represented by the **offset** of the last character in the window minus the **offset** of the first character of the window).

```
Scrollbar.ScrollProcType: TYPE = PROCEDURE [
  window: Window.Handle, direction: Scrollbar.Direction,
  percent: Scrollbar.Percent];
```

A **ScrollProcType** procedure communicates to the client a user's scroll request. **window** is the window in which the scrollbar was created, and **direction** is the direction of scrolling desired. If **direction** is **relative**, **percent** specifies the location in the file to display; for example, 0 is the beginning, 100 is the end, and 50 is the middle. If **direction** is not **relative**, **percent** is the amount of the window to be scrolled; for example, 0 means "don't scroll at all," 100 means "scroll one window contents," 50 means "scroll so that half of the current window contents is still displayed."

```
Scrollbar.Type: TYPE = {horizontal, vertical};
```

Type indicates whether the scrollbar controls the up-down movement of data (**vertical**) or the left-right movement (**horizontal**).

28.2 Constants and data objects

None.

28.3 Signals and errors

```
Scrollbar.Error: ERROR [code: Scrollbar.ErrorCode];

Scrollbar.ErrorCode: TYPE = {alreadyExists, doesNotExist, other};
```

alreadyExists the client is attempting to add to a window a scrollbar of a type that already exists on that window.

doesNotExist is raised by **GetNotifier** and **SetNotifier** if no scrollbar exists on the window in question.

other is not used.

28.4 Procedures

```
Scrollbar.Adjust: PROCEDURE [window: Window.Handle, box: Window.Box] RETURNS [
  clientBox: Window.Box,
  verticalWindow: Window.Handle, verticalBox: Window.Box,
  horizontalWindow: Window.Handle, horizontalBox: Window.Box];
```

Adjust is used by the client whenever it changes the size or position of a subwindow that has scrollbars. The client calculates the **box** to contain both the subwindow and its scrollbar windows. **clientBox** describes the area that the subwindow (minus the scrollbars) should actually occupy. **verticalWindow** is the window used to display the vertical scrollbar, and **verticalBox** is the region that **verticalWindow** should occupy. **horizontalWindow** and **horizontalBox** are similar. **verticalWindow** or **horizontalWindow** is **NIL** if that type of scrollbar does not exist for the subwindow. (If the subwindow has no scrollbars, then both **verticalWindow** and **horizontalWindow** are **NIL** and **clientBox** equals

box.) The client must use this information for the actual Window.SlideAndSize for its subwindow and each of the scrollbar windows.

```
Scrollbar.Create: PROCEDURE [
  window: Window.Handle, type: Scrollbar.Type, scroll: Scrollbar.ScrollProcType, scrollbar:
  Scrollbar.ScrollbarProcType, notify: Scrollbar.ScrollProcType ← NIL];
```

Create creates a scrollbar in the subwindow **window** for vertical or horizontal scroll functions. **scroll** is called to request a scrolling action. **scrollbar** is called to obtain information about the scrollbar and its window. **notify** is called every time a scrolling action occurs; it permits the client to monitor scrolling actions. If **Create** is called for a subwindow that already has a scrollbar of that **type**, the error **Error[alreadyExists]** is generated.

```
Scrollbar.Destroy: PROCEDURE [window: Window.Handle, type: Scrollbar.Type];
```

Destroy deletes a scrollbar. If **Destroy** is called for a subwindow that has no scrollbar of that **type**, no operation is performed.

```
Scrollbar.GetNotifier: PROCEDURE [window: Window.Handle, type: Scrollbar.Type]
  RETURNS [Scrollbar.ScrollProcType];
```

GetNotifier is called to find out what **notify** procedure has been associated with **window** and **type**.

```
Scrollbar.HasScrollbar: PROCEDURE [
  window: Window.Handle, type: Scrollbar.Type] RETURNS [BOOLEAN];
```

HasScrollbar returns a TRUE if and only if **window** has a scrollbar of type **type**.

```
Scrollbar.SetNotifier: PROCEDURE [
  window: Window.Handle, type: Scrollbar.Type, notify: Scrollbar.ScrollProcType]
  RETURNS [Scrollbar.ScrollProcType];
```

SetNotifier is called to change the **notify** procedure associated with **window** and **type**. It returns the old **notify** procedure.

```
Scrollbar.WindowNowDelinked: PROCEDURE [window: Window.Handle];
```

WindowNowDelinked is used by the client when it removes a subwindow from a tool without destroying the scrollbar property associated with that window.

```
Scrollbar.WindowNowEnlinked: PROCEDURE [window: Window.Handle];
```

WindowNowEnlinked gets the scrollbar windows attached to the tool window when the client has inserted its window back as a son of the tool window.

28.5 Discussion

Clients of the **Tool** interface should not have to call **Adjust**, **WindowNowDelinked**, or **WindowNowEnlinked**.

29

Selection

The **Selection** interface is the mechanism that communicates the current selection among various tools. It is the responsibility of a client of this interface to provide for actual selection of text or graphics within its window(s). The client window containing the current selection is referred to as the *manager* of the current selection. The **Selection** interface also defines two abstractions known as the *trashbin* and the *insertion*. The trash bin saves the most recent text cuts for subsequent pastes. The insertion saves the most recent text inserted into a text subwindow. (Note that text inserted elsewhere, such as form subwindows, is not saved.)

Two classes of clients use the **Selection** interface. Most commonly, tools that wish to obtain the *value* of the current selectioncall **Convert** (or maybe **(Long)Number**, which in turn calls **Convert**). These tools need not be concerned with the details of how selection happens. There is one slightly tricky concept for such tools to understand--if they want the selection as a **STRING**, they should also be prepared to get the selection as a **Source** in case it is longer than **Selection.maxStringLength**.

The other class is those clients who wish to manage the current selection. In this case, the tool calls **Selection.Set** and provides procedures that may be called to convert the selection or perform various actions on it. The tool remains in control of the current selection until some other tool calls **Selection.Set**.

29.1 Types

```
Selection.Action: TYPE = MACHINE DEPENDENT {
  clear(0), mark(1), unmark(2), delete(3), clearIfHasInsert(4), firstFree(5), last(255)};
```

clear	"unselects" and dehighlights the current selection.
mark	highlights the current selection.
unmark	dehighlights the current selection.
delete	deletes the contents of the current selection. The manager of the current selection may decide against actually deleting it.

clearIfHasInsert same as **clear**, but only if the insertion point is in the selection.

firstFree is used internally by **UniqueAction** and should not be used by clients.

```
Selection.ActOnProcType: TYPE = PROCEDURE [
  data: Selection.ClientData, action: Selection.Action];
```

ActOnProcType procedures are provided by the manager of the selection to handle actions.

```
Selection.ClearTrashBinProcType: TYPE = PROCEDURE [data: Selection.ClientData];
```

ClearTrashBinProcType procedures are provided by the manager of the trashbin or the insertion.

```
Selection.ClientData: TYPE = LONG POINTER;
```

```
Selection.ConvertProcType: TYPE = PROCEDURE [
  data: Selection.ClientData, target: Selection.Target] RETURNS [LONG POINTER];
```

ConvertProcType procedures are provided by the manager of the selection, trashbin, or insertion to implement **Convert**.

```
Selection.DestroyProc: TYPE = PROCEDURE [source: Selection.Source];
```

DestroyProc procedures are provided for clean-up when a manager ceases to be the manager of the selection, trashbin, or insertion (when **Selection.Set** is called again).

```
Selection.Source: TYPE = LONG POINTER TO Selection.SourceObject;
```

```
Selection.SourceObject: TYPE = RECORD [
  data: LONG POINTER TO UNSPECIFIED, proc: Selection.SourceProc,
  destroy: Selection.DestroyProc];
```

The **Source** mechanism processes textual selections that are longer than a few hundred characters. It works as follows: The client asks for the current selection to be converted as a **Source** by calling **Convert** with a **Selection.Target** of **source.** The manager of the current selection creates an instance of the **Source** data structure and returns a pointer to it to the client. The client then makes repeated calls on **proc**, supplying a string of arbitrary size. The manager of the current selection fills the string with text and returns. The manager does not need to fill the string completely, but it must return some data with each call, as end-of-selection is indicated by returning an empty string. When the client receives a zero-length string, it must call the **destroy** procedure supplied in the **SourceObject**; otherwise, the space allocated for the source is lost.

```
Selection.SourceProc: TYPE = PROCEDURE [
  data: Selection.ClientData, string: LONG STRING];
```

SourceProc procedures are contained in **Selection.SourceObjects**, and are called by client procedures to have **string** filled with characters from the selection. The **data** that is passed to the **SourceProc** should be the **data** field of the **SourceObject** that contains the **SourceProc**. The selection source need not completely fill **string**, but must return at least one character unless the source is exhausted.

```
Selection.Target: TYPE = MACHINE DEPENDENT{
  window(0), subwindow(1), string(2), source(3), length(4), position(5), pieceList(6),
  longInteger(7), interpressMaster(8), potentialInterpressMaster(9), token(10),
  firstFree(11), last(255)};
```

Target describes the type of data to which a selection may be converted (see **Convert**). Tools that manage the current selection (by calling **Selection.Set**) may choose not to implement conversion to some (or all) of these types:

window	returns a **Window.Handle** to the window containing the selection.
subwindow	returns a **Window.Handle** to the subwindow containing the selection.
string	returns a **LONG STRING** allocated from the system heap that contains a copy of the selection. If the current selection is too large, the manager of the selection may return **NIL** when asked to convert to a **string**. The client program should then ask for the selection as a **source**.
source	returns a **Selection.Source** on the selection.
length	returns a **LONG POINTER TO LONG CARDINAL** containing the length of the selection in characters.
position	returns a **LONG POINTER TO LONG CARDINAL** containing the position in the source.
pieceList	returns a list of pieces, understood by the internals of **PieceSource**.
longInteger	returns **LONG POINTER TO LONG INTEGER** containing the result of converting the contents of the selection to a number.
interpressMaster	converts the contents of the selection into an Interpress master.
potentialInterpressMaster	returns **NIL** if the manager is not willing to produce an Interpress master, or a non-**NIL** pointer (to an otherwise uninteresting small quantity) if it is willing. Even though the quantity is uninteresting, the client must free it to the system heap, or storage will be lost (**Convert** uniformly returns a legitimate pointer to storage that the client should free.)
token	returns a **LONG STRING** allocated from the system heap that contains the first token of the current selection. What constitutes a token is not defined by the **Selection** interface; all that is necessary is that the manager and a client agreed to a definition .

firstFree	is used internally by **UniqueTarget** and should not be used by clients.

Only the following targets are supported by the standard Tajo selection manager: **length**, **source**, **string** (only if the length is less than **Selection.maxStringLength** characters), **subwindow**, **window**.

29.2 Constants and data objects

```
Selection.maxStringLength: CARDINAL = 200;
```

maxStringLength is the largest string that can be produced by **Convert**.

29.3 Signals and errors

None.

29.4 Procedures

```
Selection.ActOn: PROCEDURE [Selection.Action];
```

The **ActOn** procedure communicates a request for an action to the manager of the current selection. (See also **UniqueAction**.)

```
Selection.Clear: PROCEDURE;
```

The **Clear** procedure requests that the current selection be **cleared**. It is equivalent to calling **Selection.ActOn[clear]**.

```
Selection.ClearInsertionOnMatch: PROCEDURE [pointer: LONG POINTER];
```

It is sometimes difficult to determine if you are the manager of the current insertion. The **ClearInsertionOnMatch** procedure will clear the current selection if and only if the client is the current owner. A client is the current owner if **pointer** is equal to the latest **pointer** that was passed into **SetInsertion**.

```
Selection.ClearOnMatch: PROCEDURE [pointer: LONG POINTER];
```

It is sometimes difficult to determine if you are the manager of the current selection. The **ClearOnMatch** procedure will clear the current selection if and only if the client is the current owner. A client is the current owner if **pointer** is equal to the latest **pointer** that was passed into **Set**.

```
Selection.Convert: PROCEDURE [Selection.Target] RETURNS [LONG POINTER];
```

The **Convert** procedure will perform the requested conversion and return a **LONG POINTER** to the data. The data returned for many types of items is allocated out of the system heap. The storage ownership is passed to the recipient, which must deallocate it. (See **Target** for the effect of different conversion targets.) **NIL** is returned if the manager of the current selection does not implement the desired conversion. (See also **SourceObject**.)

```
Selection.ConvertInsertion: PROCEDURE [Selection.Target] RETURNS [LONG POINTER];
```

The **ConvertInsertion** procedure converts the contents of the insertion like **Convert**.

```
Selection.ConvertTrashBin: PROCEDURE [Selection.Target] RETURNS [LONG POINTER];
```

The **ConvertTrashBin** procedure converts the contents of the trash bin like **Convert**.

```
Selection.LongNumber: PROCEDURE [radix: CARDINAL ← 10] RETURNS [LONG CARDINAL];
```

The **LongNumber** procedure will perform the requested conversion to a number. If the current selection is not acceptable to the Mesa runtime, then **String.InvalidNumber** will be raised by the runtime and allowed to propagate through these procedures.

```
Selection.Number: PROCEDURE [radix: CARDINAL ← 10] RETURNS [CARDINAL];
```

The **Number** procedure will perform the requested conversion to a number. If the current selection is not acceptable to the Mesa runtime as a number, then **String.InvalidNumber** will be raised by the runtime and allowed to propagate through these procedures

```
Selection.Set: PROCEDURE [
  pointer: LONG POINTER, conversion: Selection.ConvertProcType,
  actOn: Selection.ActOnProcType];
```

The **Set** procedure allows a client to become the manager of the current selection by supplying the **Selection** interface with a pair of procedures. The **ActOnProcType** is called to modify the current selection. The **ConvertProcType** is called to get the value of the current selection. The value of **pointer** passed to **Set** will be used as the **data** argument in calls to **conversion** or **actOn**.

```
Selection.SetInsertion: PROCEDURE [
  pointer: LONG POINTER, conversion: Selection.ConvertProcType,
  clear: Selection.ClearTrashBinProcType];
```

The **SetInsertion** procedure allows a client to become the owner of the insertion.

```
Selection.SetTrashBin: PROCEDURE [
  pointer: LONG POINTER, conversion: Selection.ConvertProcType,
  clear: Selection.ClearTrashBinProcType];
```

The **SetTrashBin** procedure allows a client to become the owner of the trashbin.

```
Selection.UniqueAction: PROCEDURE RETURNS [Selection.Action];
```

The **UniqueAction** procedure allows a client to define its own private operations on the selection. It returns a new **Action** in **[firstFree..last]**.

```
Selection.UniqueTarget: PROCEDURE RETURNS [Selection.Target];
```

The **UniqueTarget** procedure allows a client to define its own private conversion type. It returns a new **Target** in **[firstFree..last]**.

30

ToolFont

The **ToolFont** interface provides Tajo's interface to the **WindowFont** facilities. These routines provide font storage management. (See also **WindowFont**.)

30.1 Types

None.

30.2 Constants and data objects

None.

30.3 Signals and errors

None.

30.4 Procedures

```
ToolFont.Create: PROCEDURE [MFile.Handle] RETURNS [WindowFont.Handle];
```

The **Create** procedure allocates a font object and initializes it. Do not call **Create** if **MFile.Handle** is **NIL**; it causes an error in **MSegment**.

```
ToolFont.Destroy: PROCEDURE [WindowFont.Handle];
```

The **Destroy** procedure destoys the data segment and font object. Do not call **Destroy** with a **NIL WindowFont.Handle**; it causes an address fault.

```
ToolFont.StringWidth: PROCEDURE [
    string: LONG STRING, font: WindowFont.Handle ← NIL] RETURNS [[0..LAST[INTEGER]]];
```

The **StringWidth** procedure computes the width of **string** in font **font**. If font is **NIL**, the default font is used (see **WindowFont.SetDefault**). If the width of **string** in the given font is wider than can be represented in an **INTEGER**, the return value will be meaningless. This routine maps non-printing characters (such as control characters.) into a font-specific default character. If **string** is **NIL**, an address fault results.

31

WindowFont

The **WindowFont** interface converts `.strike` fonts into a representation that makes it more convenient for Tajo's window package to display characters.

31.1 Types

```
WindowFont.Handle: TYPE = LONG POINTER TO WindowFont.Object;
```

The text-painting procedures of the **Display** interface take as an argument a **Handle** on an object from **WindowFont**. Most of the fields of a **Handle** are private to the implementation.

```
WindowFont.Object: TYPE = RECORD [
  height: [0..7777B] ← NULL,
  kerned: BOOLEAN ← FALSE,
  width: PACKED ARRAY CHARACTER [0C..377C] OF [0..255] ← ALL[0],
  raster: CARDINAL ← NULL,
  maxWidth: CARDINAL ← NULL,
  min, max: CHARACTER ← NULL,
  address: LONG POINTER,
  bitmap: LONG POINTER TO ARRAY [0..0) OF WORD ← NULL,
  xInSegment: LONG POINTER TO ARRAY CHARACTER [0C..0C) OF
   CARDINAL ← NULL];
```

The bits within the font object that define the character pictures are private to the implementation. The public interfaces only allow the client to determine the sizes of the characters in screen dots.

Each of the measurement values in **Object** is in units of bits.

height is the font height.

kerned must be **FALSE**, because fonts are not supported by the window package.

width contains the width of each character.

raster is the width of the bitmap.

maxwidth	is the width of the widest character in the font.
min, max	are the lowest and highest characters that exist in the font, respectively.
address	is the address in memory of the first word of the `.strike` font.
bitmap	is the address of the first word of the actual data for the character pictures.
xInSegment	contains the number of bits from the beginning of **bitmap** to the left edge of the character, for each character in the font.

31.2 Constants and data objects

```
WindowFont.defaultFont: READONLY WindowFont.Handle;
```

31.3 Signals and errors

```
WindowFont.Error: ERROR [code: WindowFont.ErrorCode];

WindowFont.ErrorCode: TYPE = {illegalFormat};
```

31.4 Procedures

```
WindowFont.CharIsDefined: PROC [
  char: CHARACTER, font: WindowFont.Handle ← WindowFont.defaultFont]
  RETURNS [BOOLEAN];
```

CharIsDefined returns **TRUE** if a picture exists for **char**, **FALSE** otherwise. If **font** is **NIL**, the **defaultFont** is used.

```
WindowFont.CharWidth: PROC [
  char: CHARACTER, font: WindowFont.Handle ← WindowFont.defaultFont]
  RETURNS [NATURAL];
```

CharWidth allows the client to determine the width of a character in screen dots. A **font** argument of **NIL** for these routines means use the **defaultFont**.

```
WindowFont.FontHeight: PROC [
  font: WindowFont.Handle ← WindowFont.defaultFont] RETURNS [NATURAL];
```

FontHeight allows the client to determine the height of the characters in a font in screen dots. A **font** argument of **NIL** for these routines means use the **defaultFont**.

```
WindowFont.Initialize: PROC [font: WindowFont.Handle];
```

The **Initialize** procedure creates an internal font of the client's choice. **font** points to a font record that is at least **Object.SIZE** words long. The client is responsible for setting **font.address** *before* calling **Initialize**. This address must point to the first word in memory of a `.strike` font. This implies, of course, that **font** cannot be **NIL**. Tajo clients do not usually call this procedure. (See **TajoFont.Create** for a more convenient way of initializing fonts.)

```
WindowFont.SetDefault: PROC [font: WindowFont.Handle];
```

The **SetDefault** procedure sets the **defaultFont** to be **font**. Using **defaultFont** before this procedure has been called is a client error. Tajo clients do not usually call this procedure.

32

AsciiSink

This interface implements a text sink that outputs Ascii text. (See **TextSink** for a description of text sinks.)

32.1 Types

```
AsciiSink.TabStops: TYPE = LONG DESCRIPTOR FOR ARRAY OF CARDINAL;
```

TabStops describes the tab settings for text output through an **AsciiSink**. Each element of the array specifies the number of pixels from the left margin for that tab stop.

32.2 Constants and data objects

None.

32.3 Signals and errors

None.

32.4 Procedures

```
AsciiSink.Create: PROC [font: WindowFont.Handle] RETURNS [TextSink.Handle];
```

Create takes a font to be used for output and returns a **TextSink.Handle**.

```
AsciiSink.GetTabs: PROC [sink: TextSink.Handle] RETURNS [AsciiSink.TabStops];
```

GetTabs returns the current tab stops for **sink**. A returned value of **NIL** means that the default tab stops (one every eight spaces) are in effect.

```
AsciiSink.Info: PROC [sink: TextSink.Handle] RETURNS [font: WindowFont.Handle];
```

Info returns the font with which the sink was created.

```
AsciiSink.IsIt: PROC [sink: TextSink.Handle] RETURNS [BOOLEAN];
```

IsIt returns TRUE if this sink is an **AsciiSink** (created by AsciiSink.**Create**) and FALSE otherwise.

```
AsciiSink.SetTabs: PROC [sink: TextSink.Handle, tabStops: AsciiSink.TabStops ← NIL];
```

SetTabs sets the tab stops for **sink**. If **tabStops** is defaulted, the default tab stops (one every eight spaces) are set.

33

BlockSource

This interface creates a text source (see **TextSource**) that is backed by an Environment.Block of Ascii characters. It is the same as a scratch source (see **ScratchSource**) with an access of read-only.

33.1 Types

```
BlockSource.Block: TYPE = Environment.Block;

BlockSource.Handle: TYPE = TextSource.Handle;
```

33.2 Constants and Data Objects

None.

33.3 Signals and Errors

None are defined by this interface; however, TextSource.Error can be raised by the procedure Info.

33.4 Procedures

```
BlockSource.Create: PROCEDURE [
  block: BlockSource.Block] RETURNS [source: BlockSource.Handle];
```

The **Create** procedure creates a block source. The characters in the block must not change as long as the source is using that block.

```
BlockSource.Info: PROCEDURE [
  source: BlockSource.Handle] RETURNS [block: BlockSource.Block];
```

Info returns the block backing the block source. **source** cannot be NIL. This procedure raises TextSource.Error[other] if **source** is not a pointer to a block source.

```
BlockSource.IsIt: PROCEDURE [source: BlockSource.Handle] RETURNS [yes: BOOLEAN];
```

IsIt returns **TRUE** if the **Handle** is a block source and **FALSE** otherwise. **source** cannot be **NIL**.

```
BlockSource.Set: PROCEDURE [source: BlockSource.Handle, block: BlockSource.Block];
```

Set changes the block backing the block sources; the old block is not deallocated. **source** cannot be **NIL**.

34

DiskSource

The **DiskSource** interface creates a text source (see the TextSource chapter) that is backed by a stream or a file in the local file system.

34.1 Types

None.

34.2 Constants and data objects

None.

34.3 Signals and errors

None.

34.4 Procedures

```
DiskSource.Create: PROCEDURE [
  name: LONG STRING, access: TextSource.Access, s: Stream.Handle ← NIL]
  RETURNS [source: TextSource.Handle];
```

The **Create** procedure creates a disk source. If **s** is not **NIL**, it is used as the stream backing the source. If **s** is **NIL**, a stream is opened on the file **name**. **access** may be either **read** or **append**. This procedure may raise **TextSource.Error[..., accessError, fileNameError, ...]**.

```
DiskSource.Info: PROCEDURE [source: TextSource.Handle]
  RETURNS [name: LONG STRING, s: Stream.Handle, access: TextSource.Access];
```

The **Info** procedure returns the name of the file backing the disk source, the stream backing the source, and the access on the source.

```
DiskSource.IsIt: PROCEDURE [source: TextSource.Handle] RETURNS [BOOLEAN];
```

The **IsIt** procedure returns **TRUE** if the **Handle** is a disk source and **FALSE** otherwise.

```
DiskSource.Rename: PROCEDURE [
  source: TextSource.Handle, newName: LONG STRING, access: TextSource.Access]
  RETURNS [TextSource.Handle];
```

The **Rename** procedure renames a currently existing disk source. The current disk source is destroyed and a disk source for the new file, with the specified **access,** is created. This procedure may raise **TextSource.Error[..., accessError, fileNameError, ...]**. **source** cannot be **NIL**.

```
DiskSource.SetMaxDiskLength: PROCEDURE [
  source: TextSource.Handle, maxLength: LONG CARDINAL];
```

The **SetMaxDiskLength** procedure provides a way to implement circular files, which are particularly useful for logs. When the source reaches **maxLength** characters in length, it starts over at the beginning of the stream, rather than extending the file. **source** cannot be **NIL**.

35

PieceSource

The **PieceSource** interface creates a text source (see **TextSource**) that is backed by a piece table maintained on a text source.

35.1 Types

None.

35.2 Constants and data objects

None.

35.3 Signals and errors

None.

35.4 Procedures

```
PieceSource.Create: PROCEDURE [original, scratch: TextSource.Handle]
  RETURNS [source: TextSource.Handle];
```

The **Create** procedure creates a piece source. **original** is the text source on which the piece table is made. The piece source takes over ownership of this text source. **scratch** is a text source with append access that the piece table code uses for maintaining the interim state.

```
PieceSource.Info: PROCEDURE [source: TextSource.Handle]
  RETURNS [original, scratch: TextSource.Handle];
```

The **Info** procedure returns the original and scratch text sources with which the piece source was created. **source** cannot be NIL.

```
PieceSource.IsIt: PROCEDURE [source: TextSource.Handle] RETURNS [yes: BOOLEAN];
```

The **IsIt** procedure returns TRUE if the **Handle** is a piece source and FALSE otherwise.

```
PieceSource.Put: PROCEDURE [source: TextSource.Handle, name: LONG STRING] RETURNS [new:
TextSource.Handle];
```

The **Put** procedure converts the piece table into a stream and stores it into the file named **name**. It returns a disk source with read access on the file after storing the contents of the piece table. **source** cannot be NIL. Any of the errors from MFile.**WriteOnly** may be raised.

```
PieceSource.Reset: PROCEDURE [source: TextSource.Handle]
  RETURNS [original: TextSource.Handle];
```

The **Reset** procedure causes the piece source to discard all modifications made to the piece table and return the original source passed to the PieceSource.**Create** procedure. The text source **scratch** is destroyed in the process. **source** cannot be NIL.

36

ScratchSource

The **ScratchSource** interface creates a text source (see **TextSource** for more information) that is backed by a block of virtual memory containing Ascii characters.

36.1 Types

None.

36.2 Constants and data objects

None.

36.3 Signals and errors

None.

36.4 Procedures

```
ScratchSource.Create: PROCEDURE [
  block: Environment.Block ← Environment.nullBlock, extraRoom: CARDINAL ← 0, access:
  TextSource.Access ← edit, expandable: BOOLEAN ← TRUE]
  RETURNS [source: TextSource.Handle];
```

The **Create** procedure creates a scratch source. **block** is storage that is used to back the source. A **block** of **nullBlock** means the source allocates the block using **MSegment.GetPages**. For any other **block** passed in, the source has as initial data any characters contained in the block. **extraRoom** is the amount of storage beyond the end of block that may be used by the source. If **expandable** is **FALSE** and the source runs out of room in the block while performing a replace operation, that operation returns a *no-change* value (see **TextSource**). If **expandable** is **TRUE**, the source takes over ownership of the block passed in; the block must be allocated using **MSegment.GetPages** so that the source may replace it with another larger block if necessary. In this case, the block is deallocated when the source is destroyed. **access** is the access desired on this source. A scratch source whose access is **read** and whose block is not null is the same as a block source (see **BlockSource**).

```
ScratchSource.Info: PROCEDURE [source: TextSource.Handle]
  RETURNS [block: Environment.Block, extraRoom: CARDINAL,
  access: TextSource.Access, expandable: BOOLEAN];
```

The **Info** procedure returns the block backing the scratch source, the amount of extra room left after the block, whether the block is expandable, and the access on the source. **source** cannot be **NIL**. **TextSource.Error[other]** is raised if **source** does not point to a scratch source.

```
ScratchSource.IsIt: PROCEDURE [source: TextSource.Handle] RETURNS [yes: BOOLEAN];
```

The **IsIt** procedure returns **TRUE** if the **Handle** is a scratch source and **FALSE** otherwise.

37

StringSource

The **StringSource** interface creates a text source (see **TextSource**) that is backed by a string containing Ascii text.

37.1 Types

None.

37.2 Constants and data objects

```
StringSource.cannotExpand: CARDINAL = ...;
```

cannotExpand is used by the procedure **InsertString** to indicate that a string is non-expandable. It will be deleted from the interface when it is next changed, because **InsertString** will also be deleted.

37.3 Signals and errors

None.

37.4 Procedures

```
StringSource.Create: PROCEDURE [
  ps: LONG POINTER TO LONG STRING, expandable: BOOLEAN]
  RETURNS [source: TextSource.Handle];
```

The **Create** procedure creates a string source with edit access. **ps** is a pointer to the string backing the source. If **ps** is **NIL**, **TextSource.Error[invalidParameters]** is raised. If **expandable** is **FALSE** and the string source runs out of room in the string (such as during a call to **source.replaceText**), **String.StringBoundsFault[ps]** is raised. If **expandable** is **TRUE** and the string source runs out of room in the string, it allocates a new string, copies the old string, and deallocates it. If **expandable** is **TRUE**, the string must have been allocated from the system heap; the string is deallocted when the source is destroyed using its **ActOn** procedure.

Note: The current implementation of string sources requires a contiguous block of memory large enough to completely contain the backing string. More important, when the string is expanded, a new larger string is allocated and copied, which requires $2*n + delta$ characters of memory in the system heap.

```
StringSource.DeleteSubString: PROCEDURE [
  ss: String.SubString, keepTrash: BOOLEAN] RETURNS [trash: LONG STRING];
```

The **DeleteSubString** procedure is no longer implemented. It will be deleted from the interface when the interfaces are next changed.

```
StringSource.Info: PROCEDURE [source: TextSource.Handle]
  RETURNS [ps: LONG POINTER TO LONG STRING, expandable: BOOLEAN];
```

The **Info** procedure returns the string backing the string source and whether the string is expandable. **source** cannot be **NIL**. If **source** is not a string source, it returns **NIL, FALSE**.

```
StringSource.InsertString: PROCEDURE [
  string: LONG POINTER TO LONG STRING, position: CARDINAL, toAdd: String.SubString, extra:
  CARDINAL];
```

The **InsertString** procedure is no longer implemented. It will be deleted from the interface when the interfaces are next changed.

```
StringSource.IsIt: PROCEDURE [source: TextSource.Handle] RETURNS [yes: BOOLEAN];
```

IsIt returns **TRUE** if the **Handle** is a string source and **FALSE** otherwise.

38

TextData

The **TextData** interface is not of interest to most clients. It defines data types that a few procedures in **TextSW** and **FormSW** need. The **TextDisplay** interface depends heavily on the following definitions.

38.1 Types

```
TextData.Insertion: TYPE = LONG POINTER TO TextData.InsertionObject;

TextData.InsertionMode: TYPE = {triangle, box};

TextData.InsertionObject: TYPE = RECORD [
  position: TextData.Position,
  place: Window.Place,
  mode: TextData.InsertionMode,
  marked: BOOLEAN];
```

Insertion points for editable text are typically marked by a blinking caret. The TextDisplay routines take a pointer to the insertion object so that they can maintain the necessary values. When an insertion point is displayed, it is usually a blinking triangle; however, the convention is that append-only editing is indicated by a blinking rectangular box.

```
TextData.MarkingAction: TYPE = MACHINE DEPENDENT{clear(0), mark(1), invert(2), (3)};
```

The client may ask the display routines to change the marking of a displayed insertion point.

clear causes the insertion point to no longer be visible.

mark forces the insertion point to be visible.

invert toggles the visibility of the insertion point.

```
TextData.Position: TYPE = TextSource.Position;
```

Text is addressed by **Position**, which is a **LONG CARDINAL**.

```
TextData.Selection: TYPE = LONG POINTER TO TextData.SelectionObject;

TextData.SelectionEntity: TYPE = MACHINE DEPENDENT {
  text(0), word(1), element(2), line(3), paragraph(5), document(7)};

TextData.SelectionMode: TYPE = MACHINE DEPENDENT {
  video(0), grayBox(1), underline(2), clearText(3), strikeOut(4), splat(6), (15)};
```

The **SelectionMode** is how the selection will be displayed to the user.

video video-inverts the selection.

grayBox displays the selection on a light gray background.

underline underlines the selection.

clearText selections are not indicated to the user.

strikeOut draws a one-bit-wide line through all characters of the selection.

splat raises ERROR.

```
TextData.SelectionObject: TYPE = RECORD [
  left, right: TextData.Position, entity: TextData.SelectionEntity,
  mode: TextData.SelectionMode, marked: BOOLEAN];
```

Text selections are also maintained by the **TextDisplay** routines and may be set by client code. A selection consists of the marking mode and the current entity. The entity is maintained with the selection so that multiple clicks can grow the selection to the next higher value.

```
TextData.SelectionType: TYPE = {select, extend};
```

SelectionType is used by the display routines to either make a new selection or adjust the old.

38.2 Constants and data objects

None.

38.3 Signals and errors

None.

38.4 Procedures

None.

39

TextSink

TextSource and **TextSink** isolate Tajo's uniform text display, selection, and editing facilities from the representation of text. The **TextSink** interface defines a *sink* for text that is displayed in a window. It defines the standard set of operations that display text, measure displayed text, and resolve display positions to character positions. For each representation of text, there should be at least one sink and one source. The default sources and sinks display Ascii characters. Specific implementations may use additional operations for setting or altering the state of a text sink. (See also the interface **AsciiSink**.)

A client who wishes to implement its own sink must implement the sink's operations with the semantics defined below. The text display code in Tajo invokes these operations, behind which hide the representation of the text. Although text is addressed by **Environment.Block**, only the sink and its corresponding source look inside the block.

39.1 Types

TextSink.Action: TYPE = {destroy, sleep, wakeup};

An **Action** is the parameter to the **ActOnProc** that tells the sink to change state.

destroy the sink should destroy itself, freeing all storage and releasing all resources associated with the text sink instance.

sleep the source should release whatever resources it can without losing information; it is a hint that the text sink will not be used for a while.

wakeup the sink is going to be used and should resume its normal state, undoing whatever was done for **sleep**.

Note: **sleep** and **wakeup** are only hints for storage and resource management; implementors must be able to handle all operations on sleeping text sources.

TextSink.ActionResult: TYPE = {ok, bad};

An **ActionResult** is the result of **ActOnProc**. [**Note**: only a result of **ok** is expected.]

```
TextSink.ActOnProc: TYPE = PROCEDURE [
  sink: TextSink.Handle, action: TextSink.Action] RETURNS [TextSink.ActionResult];
```

The sink's **ActOnProc** is invoked to change a sink's state.

```
TextSink.BreakReason: TYPE = {eol, consumed, margin};
```

A **DisplayBlockProc**, **MeasureBlockProc**, or **ResolveBlockProc** can stop displaying, measuring, or resolving for one of several reasons, any of which may mean that the procedure has not finished the task.

eol it encountered the end of a line in the text it is operating on.

margin it encountered the edge of the area in which it can operate on.

consumed it finished operating on the requested text.

```
TextSink.DisplayBlockProc: TYPE = PROCEDURE [
  sink: TextSink.Handle, block: TextSink.TextBlock, lineLength, offset: INTEGER, window:
  Window.Handle, place: Window.Place, bbop: Window.BBoperation,
  bbso: Window.BBsourcetype]
  RETURNS [
    newPlace: Window.Place, positions: CARDINAL, why: TextSink.BreakReason];
```

The sink's **DisplayBlockProc** displays text in a window. **block** is the text to be displayed. **lineLength** is the farthest that the displayed text can extend. **offset** is the offset from the edge of the window to the beginning of the region where the text is displayed; it is used in calculating the position of tabs. **window** is the **Window** in which the text is to be displayed, and **place** is the location where the displayed text should start. **bbop** and **bbso**, used in painting the text, are described as part of **Window**. The **DisplayBlockProc** returns **why**, a **BreakReason**. In addition to the reason for stopping, the routine returns the number of positions it displayed and the position in the window where the next text will be displayed. The **Environment.Block** referenced by **block** should be updated.

```
TextSink.FontInfoProc: TYPE = PROCEDURE [
  sink: TextSink.Handle] RETURNS [lineHeight, minWidth, maxWidth: CARDINAL];
```

The sink's **FontInfoProc** returns information about the font being used by the sink. **lineHeight** is the height of a line of text, and **minWidth** and **maxWidth** bound the width of characters.

```
TextSink.Handle: TYPE = LONG POINTER TO TextSink.Procedures;
```

A **Handle** is an object-oriented pointer to a pointer to a record of procedures that defines the operations on a text sink.

```
TextSink.MeasureBlockProc: TYPE = PROCEDURE [
  sink: TextSink.Handle, block: TextSink.TextBlock, lineLength, offset: INTEGER, place:
  Window.Place, placeIsLeft: BOOLEAN ← TRUE]
  RETURNS [
    newPlace: Window.Place, positions: CARDINAL, why: TextSink.BreakReason];
```

The sink's **MeasureBlockProc** measures text in a window. It behaves very much like the **DisplayBlockProc**, except that the characters are not actually painted in the window. Because no painting is done, neither the window nor the painting parameters are passed. The parameter **placeIsLeft** indicates the direction of the measuring. If **placeIsLeft** is TRUE, the window position is the leftmost edge of the text, and measuring should be done from left to right. If it is FALSE, the position is the rightmost edge of the text, and measuring should be done from right to left. The results returned from the **MeasureBlockProc** should be the same as those from the **DisplayBlockProc**, if **placeIsLeft** is TRUE and the other parameters are the same.

```
TextSink.PositionsInBlockProc: TYPE = PROCEDURE [
  sink: TextSink.Handle, block: TextSink.TextBlock] RETURNS [CARDINAL];
```

The sink's **PositionsInBlockProc** determines the number of positions the block represents, which is not necessarily the number of bytes in the block. The **sink** parameter is included to pass the instance data.

```
TextSink.Procedures: TYPE = LONG POINTER TO TextSink.ProceduresObject;

TextSink.ProceduresObject: TYPE = RECORD [
  actOn: TextSink.ActOnProc,
  displayBlock: TextSink.DisplayBlockProc,
  fontInfo: TextSink.FontInfoProc,
  measureBlock: TextSink.MeasureBlockProc,
  positionsInBlock: TextSink.PositionsInBlockProc,
  resolveBlock: TextSink.ResolveBlockProc];

TextSink.ResolveBlockProc: TYPE = PROCEDURE [
  sink: TextSink.Handle, block: TextSink.TextBlock, startX, xToFind, offset: INTEGER,
  halfCharResolve: BOOLEAN]
  RETURNS [newX: INTEGER, positions: CARDINAL, why: TextSink.BreakReason]
```

The sink's **ResolveBlockProc** locates the position corresponding to a place in the window. **block** is the **TextBlock** in which to search. **startX** is the place on the line that corresponds to the first character of **block**. **xToFind** is the place on the line where the corresponding character position is desired. These parameters are integers instead of **Window.Places** because the **MeasureBlockProc** assumes that the places are on the same line. **offset** is the offset from the edge of the window to the edge of the text display area, as in the **DisplayBlockProc** and **MeasureBlockProc**. **halfCharResolve** indicates what to do if **xToFind** corresponds to the rightmost part of a position. If **halfCharResolve** is TRUE, the position returned is the next position (round up); if it is FALSE, the position is the one containing the place (truncate). The **ResolveBlockProc** should return a place (**newX**), the distance that place is from **startX**, the number of character positions scanned, and the reason why it stopped resolving (**why**). If **newX = xToFind**, the procedure was successful. If **newX** is different from **xToFind**, **ResolveBlockPlace** is called again to find the desired place.

```
TextSink.TextBlock: TYPE = POINTER TO Environment.Block;
```

A text sink represents its information as a **TextBlock**.

39.2 Constants and data objects

None.

39.3 Signals and errors

```
TextSink.Error: ERROR [code: ErrorCode];
```

```
TextSink.ErrorCode: TYPE = {invalidSink, isBad, invalidParameters, other};
```

invalidSink	the sink is invalid.
isBad	the sink no longer works.
invalidParameters	the parameters were not sensible.

39.4 Procedures

None.

40

TextSource

TextSource and **TextSink** isolate Tajo's uniform text display, selection, and editing facilities from the representation of text. The **TextSource** interface defines a *source* of text that may be displayed in a window. It defines the standard set of operations that access a text source. A text source implementation is responsible for implementing text source operations on its underlying representation of the text. For each representation of text, there should be at least one sink and one source. Default sources and sinks display Ascii characters. Specific implementations may use additional operations for setting or altering the state of a text source. (See also **BlockSource**, **DiskSource**, **PieceSource**, **ScratchSource**, and **StringSource**.)

40.1 Types

TextSource.Access: TYPE = **{read, append, edit}**;

Access is provided for source implementations.

TextSource.Action: TYPE = **{destroy, mark, sleep, truncate, wakeup}**;

An **Action** is the parameter to the **ActOnProc** that tells the source to change state.

destroy the source should destroy itself, freeing all storage and releasing all resources associated with the text source instance.

mark it should mark the logical end of the data.

sleep it should release whatever resources it can without losing information. (This is a hint that the text source will not be used for a while.)

truncate it should truncate its data to its current length. (This has a noticable effect only for sources that have some representation in a file system.)

wakeup the source is going to be used and should resume its normal state, undoing whatever was done for **sleep**.

Note: **sleep** and **wakeup** are only hints for storage and resource management. Implementors must be able to handle all operations on sleeping text sources.

```
TextSource.ActOnProc: TYPE = PROC [source: TextSource.Handle, action: TextSource.Action];
```

The source's **ActOnProc** changes a source's state.

```
TextSource.Class: TYPE = {none, eol, alpha, space, other};
```

Class divides characters into classes; it is a parameter of the **ReadTextProc**.

```
TextSource.Direction: TYPE = {left, right};
```

Direction indicates the direction of a scan.

```
TextSource.DoEditActionProc: TYPE = PROC [
  source: TextSource.Handle, action: TextSource.EditAction, editPos: TextSource.Position]
  RETURNS [delta: LONG INTEGER];
```

The source's **DoEditActionProc** moves within the source. The result **delta** is the number of positions that the source backed up. Ascii sources may use **AsciiDoEditAction**.

```
TextSource.EditAction: TYPE = {none, backSpace, backWord, backLine};
```

EditAction enumerates the possible edit actions for **DoEditActionProc**. **none** means "no action should be taken." **backSpace** means "back up one position from **editPos**." **backWord** means "back up until the source is positioned at the beginning of the next alphanumeric character." **backLine** means "back up until the source is positioned just to the right of the last end-of-line."

```
TextSource.GetLengthProc: TYPE = PROCEDURE [
  source: TextSource.Handle] RETURNS [TextSource.Position];
```

The source's **GetLengthProc** obtains the number of **Positions** in a source. This operation is used extensively, and it should be implemented efficiently.

```
TextSource.Handle: TYPE = LONG POINTER TO TextSource.Procedures;
```

A **Handle** is an object-oriented pointer to a pointer to a record of procedures that defines the operations on a text source.

```
TextSource.Position: TYPE = LONG CARDINAL;
```

TextSource procedures operate in terms of **Positions**, which are displayable units.

```
TextSource.Procedures: TYPE = LONG POINTER TO TextSource.ProceduresObject;

TextSource.ProceduresObject: TYPE = RECORD [
  actOn: ActOnProc, doEditAction: DoEditActionProc, getLength: GetLengthProc,
  readText: ReadTextProc, replaceText: ReplaceTextProc,
  scanText: ScanTextProc, setLength: SetLengthProc];
```

```
TextSource.ReadTextProc: TYPE = PROCEDURE [
  source: TextSource.Handle, position: TextSource.Position, maxLength: CARDINAL,
  class: TextSource.Class]
  RETURNS [block: Environment.Block, next: TextSource.Position];
```

The source's **ReadTextProc** obtains a block of text. The block should contain text in position **position** and contain at most **maxLength** characters. **class** is used as a hint to limit the amount of characters read. If **class** is not **none**, the block may be terminated after a character of that class is read. (See the Discussion section for a discussion of limitations.)

```
TextSource.ReplaceTextProc: TYPE = PROCEDURE [
  source: TextSource.Handle, block: Environment.Block, from, to: TextSource.Position,
  deleteToTrashbin: BOOLEAN ← TRUE]
  RETURNS [new: TextSource.Position, delta: LONG INTEGER];
```

The source's **ReplaceTextProc** replaces part of the source with a block of text. The source positions to be replaced are those between positions **from** and **to**. The text to insert in that place is in **block**. If **deleteToTrashbin** is **TRUE**, the data removed from the source should be placed in the trash bin, where it can be recovered. The procedure should return **new**, the position at the start of the inserted text, and **delta**, the change in the source's size resulting from this operation.

```
TextSource.ScanType: TYPE = {
  alpha, invisible, line, nonAlpha, word, leftMark, rightMark, spare};
```

ScanType, a parameter to **ScanTextProc**, defines the type of character that will terminate the scan.

```
TextSource.ScanTextProc: TYPE = PROCEDURE [
  source: TextSource.Handle, start: TextSource.Position, type: TextSource.ScanType,
  direction: TextSource.Direction]
  RETURNS [position: TextSource.Position];
```

The source's **ScanTextProc** scans a source, starting at the specified position and going in the specified direction until a character of the requested type is found. The position of the matching character should be returned; if no character of the requested class can be found, **nullPosition** should be returned.

```
TextSource.SetLengthProc: TYPE = PROCEDURE [
  source: TextSource.Handle, position: TextSource.Position]
  RETURNS [TextSource.Position];
```

The source's **SetLengthProc** sets the number of positions in a source. **position** is the length to be set; the return value is the actual number of positions the source was set to. Attempting to lengthen most sources with this operation is undefined and will produce unexpected results.

```
TextSource.State: TYPE = {asleep, awake, bad};
```

State is provided for source implementations.

40.2 Constants and data objects

```
TextSource.cannotExpand: CARDINAL = LAST[CARDINAL];
```

cannotExpand may be used as a parameter to **AsciiInsertBlock** to indicate that the string may not be expanded.

```
TextSource.nullPosition: TextSource.Position = LAST[LONG CARDINAL];
```

nullPosition is returned by a **ScanTextProc** if no character of the requested class can be found.

40.3 Signals and errors

```
TextSource.Error: ERROR [code: TextSource.ErrorCode];

TextSource.ErrorCode: TYPE = {
  fileNameError, accessError, isBad, invalidParameters, other};
```

fileNameError	either the file doesn't exist or bad file name syntax was used.
accessError	in operation that violates the created access option was attempted.
isBad	the source no longer exists. This occurs on core swaps when the file is deleted.
invalidParameters	the parameters were not sensible.

```
TextSource.SearchFailed: ERROR;
```

SearchFailed is raised by **AsciiTextSearch** if there is no match.

40.4 Procedures

```
TextSource.AsciiAppend: PROCEDURE [
  string: LONG STRING, source: TextSource.Handle, start: TextSource.Position,
  n: CARDINAL];
```

The **AsciiAppend** procedure appends **n** characters onto **string** from **source** starting at position **start**. It may raise **String.StringBoundsFault** if **string** does not have room for **n** characters.

```
TextSource.AsciiDeleteSubString: PROCEDURE [
  ss: String.SubString, keepTrash: BOOLEAN] RETURNS [trash: LONG STRING];
```

The **AsciiDeleteSubString** procedure deletes a substring in the source and optionally returns the deleted substring.

```
TextSource.AsciiDoEditAction: TextSource.DoEditActionProc;
```

AsciiDoEditAction is a standard **DoEditActionProc** on an Ascii text source.

```
TextSource.AsciiInsertBlock: PROCEDURE [
  string: LONG POINTER TO LONG STRING, position: CARDINAL, toAdd: Environment.Block, extra:
  CARDINAL];
```

The **AsciiInsertBlock** procedure inserts the contents of a block into a string, starting at a specified position. If there is not enough room in the string and **extra** is **cannotExpand**, then **String.StringBoundsFault** is raised; otherwise, the string is expanded.

```
TextSource.AsciiScanText: TextSource.ScanTextProc;
```

The **AsciiScanText** procedure is a standard **ScanTextProc** on an Ascii text source.

```
TextSource.AsciiTestClass: PROCEDURE [
  char: CHARACTER, class: TextSource.Class] RETURNS [equal: BOOLEAN];
```

The **AsciiTestClass** procedure tests to see if a character is a member of a **Class**.

```
TextSource.AsciiTextSearch: PROCEDURE [
  source: TextSource.Handle, string: LONG STRING, start: TextSource.Position ← 0,
  stop: TextSource.Position ← LAST[LONG CARDINAL]]
  RETURNS [lineStart, left: TextSource.Position];
```

The **AsciiTextSearch** procedure searches a range of positions in a source for an instance of **string**. It returns both the leftmost position of the match and the position of the first character in the line that contains the match. If there is no match, it raises the error **SearchFailed**.

```
TextSource.ActOn: TextSource.ActOnProc = INLINE {...};

TextSource.DoEditAction: TextSource.DoEditActionProc = INLINE {...};

TextSource.GetLength: TextSource.GetLengthProc = INLINE {...};

TextSource.ReadText: TextSource.ReadTextProc = INLINE {...};

TextSource.ReplaceText: TextSource.ReplaceTextProc = INLINE {...};

TextSource.ScanText: TextSource.ScanTextProc = INLINE {...};

TextSource.SetLength: TextSource.SetLengthProc = INLINE {...};
```

These procedures are for clients who wish to use object notation when dealing with sources. (See the next section.)

40.5 Discussion

The following additional semantic rules for reading text sources ease the job of implementing text sources and discontinuous sources. *Discontinuous sources* are text sources that either have holes in them or contain embedded sequences of non-textual data (such as text files with formatting information).

A text source may not return more text than was requested.

A single call on read may not return text that is not contiguous in the text source's address space (that is, it cannot concatenate two discontiguous runs of text).

A text source may return less text than was requested.

A text source may only return no text (i.e., length = 0) if the position is equal to the value returned by **getLength** or **pos** is greater than **position**.

The following code fragment shows an example of the **INLINE** procedures described above with object notation:

```
source: TextSource.Handle;
lastPosition: TextSource.Position;
lastPosition ← source.GetLength;
```

This is equivalent to:

```
source: TextSource.Handle;
lastPosition: TextSource.Position;
lastPosition ← source.getlength[source];
```

IV

User input and events

User input and other events in the Xerox Development Environment are handled by the **UserInput**, **Event**, **EventTypes**, and **TIP** interfaces. These interfaces are useful in tool-building because they allow programmers to concentrate on design rather than details of exactly how the system handles user type-in or other actions. The **UserInput** interface is the most commonly used for handling keyboard type-in, especially the important **ABORT** key.

IV.1 Events

Events are initiated by users and by tools or processes that want to notify a tool about a change in state, to request permission to boot the volume (which cannot be done during disk writes, for instance), or to start other major system-level activities. **Event** and **EventTypes** are used this way, in particular. They interact with an important Pilot-level interface called the *supervisor*, which keeps track of the tools and the events they want to be notified about. (For more information about the supervisor, refer to the *Pilot Programmer's Manual*.)

TIP (terminal interface package) is less frequently used by most programmers than the other interfaces in this section. TIP tables map between keyboard keys (or mouse clicks) and their meanings. Restructuring this mapping is a task for more advanced XDE programmers. The next section gives a brief overview of TIP tables and gives examples. The **TIP** chapter gives more detail.

IV.2 TIP tables

The system uses TIP tables to look up and execute commands based on user-initiated actions. It employs both a process to watch for user actions and a queue to store them until it can process them.

The **StimLev** process watches the hardware for user actions and queues them along with their time of occurrence in the *user action queue*, which is the queue of key transitions and mouse movements.

The **Matcher**, also called the **Notifier**, figures out which window and TIP table a given event is intended for. If a left side of a TIP statement has been matched, the **Notifier** calls

the associated **NotifyProc** with a list of results. If no match is found, the action is discarded.

A **NotifyProc** is a process called (by the **Notifier**) to let a TIP table know when a desired condition is true, by setting appropriate values.

IV.2.1.1 Example of a NotifyProc

The right sides of TIP statements are usually atoms, but they can also be window-relative coordinates, characters, and numbers. A **NotifyProc** is given a list of such results when it is called. There are two handy routines for stepping through this list: **TIP.First[] and TIP.Rest[]**.

```
TipMe: TIP.NotifyProc = {
  FOR input: TIP.Results results, input.Rest UNTIL input = NIL DO
  WITH z: input.First SELECT FROM
    char = > {
      IF ~UserInput.StuffCharacter[window, z.c] THEN
      UserTerminal.BlinkDisplay[]};
    coords = > tipPlace z.place;
    atom = >
      SELECT z.a FROM
      Exit = > {trackOnGrid FALSE; SetMouseTracking[FALSE]};
      Enter = > EnterWindow[clear];
      Copy = > CopyFunction[];
      SuperCopy = > SuperCopyFunction[];
      Delete = > DeleteFunction[];
      DrawLine = > DrawFunction[];
      TakeInputFocus = > UserInput.SetInputFocus[pictureWindow, DontCare, TRUE];
      Stuff = >
        IF ~UserInput.StuffCurrentSelection[window] THEN
        UserTerminal.BlinkDisplay[];
        ENDCASE;
    string = >
      IF ~UserInput.StuffString[window, z.s] THEN UserTerminal.BlinkDisplay[];
      ENDCASE;
  ENDLOOP};
```

The notify procedure **TIPMe** looks at the results and understands atoms and string input.

IV.2.2 TIP table keyword semantics

The keywords **TRIGGER** and **AND** refer to events that have just happened; that is, the event in question has just been dequeued from the User Action Queue.

The keywords **ENABLE** and **WHILE** refer to events that have already happened and are still true. These events are sometimes called *enabling conditions*.

Essentially, the whole TIP table can be viewed as a **SELECT** statement. The match process is continuously reading key transitions, mouse movements, or key states from the input queue. A **TRIGGER** statement has the effect of looking at the next action recorded in the

input queue and branching to the appropriate choice. An **ENABLE** statement implies selection between the choices according to the current state of the keyboard or the mouse keys. **AND** terms connect sequences of **TRIGGER** terms. They might be mixed with **ENABLE** terms, which are characterized by **WHILE**.

A timeout following a trigger indicates a timing condition that must hold between this trigger and its predecessor. The number associated with the timeout expresses a time interval in milliseconds. Events starting with the same sequence of trigger or enable terms are expressed as nested statements. Result items may be identifiers, numbers, strings, or the keywords COORDS, BUFFEREDCHAR, CHAR, KEYS, or TIME. The results of the successfully parsed event are passed to the client.

IV.2.3 TIP table syntax example

This example of a TIP table uses **TRIGGER, AND, ENABLE,** and **WHILE**:

```
SELECT TRIGGER FROM                          -- something has just happened
A Down => Foo;
B Down => SELECT ENABLE FROM                 -- what else is true right now?
  C Up => {Atom1 Atom2 Atom3 Atom4};
  E Down => Atom1, Atom2, Atom3, Atom4;      --if more than one is true,the
                                             first is matched
  ENDCASE;
H Up AND K Down => HAndK;                    -- H has just gone Up; we'll
                                             wait to see if K Down is the
                                             next action
M Up WHILE L Up => MAndL;                    -- M has just gone Up and L is
                                             already Up
ENDCASE..                                    -- TIP bug!  You need at least
                                             two .'s to end a TIP table
```

IV.2.4 How to create a TIP table

The following procedure sets up a typical TIP table:

```
MakeMySWs: Tool.MakeSWsProc =
  BEGIN
    msgSW Tool.MakeMsgSW[window: window, lines: 2];
    formSW Tool.MakeFormSW[window: window, formProc: AnchorsAway];
    frameWindow ToolWindow.CreateSubwindow[parent: window];
    tool.AddThisSW[window: window, sw: frameWindow, swType: vanilla];
    pictureWindow TajoOps.AllocateWindow[];
    Window.InitializeWindow[
      window: pictureWindow, display: DisplayPictureWindow,
      box: [[0, 0], [30000, 30000]], parent: frameWindow];
    Window.InsertIntoTree[pictureWindow];
    UserInput.CreateStringInOut[
      window: pictureWindow, in: AddStringToLabel, out: AddStringToLabel];
    scrollbar.Create[
      window: frameWindow, type: horizontal, scroll: HScroll,
      scrollbar: HScrollBar];
    Scrollbar.Create[
```

```
        window: frameWindow, type: vertical, scroll: VScroll,
        scrollbar: VScrollBar];
      TIP.CreateClient[window: pictureWindow, table: tipTable, notify: TipMe];
    END;

tipTable: TIP.Table NIL;

  Init: PROC = {
    tipContents: STRING = "   -- the default TIP table

OPTIONS DefaultKeys;
SELECT TRIGGER FROM
COPY Down => SELECT ENABLE FROM
  CONTROL Down => SuperCopy;
  ENDCASE => Copy;
DELETE Down => Delete;
Three Down AND DOIT Down BEFORE 100 => DrawLine;
Six Down => CHAR, CHAR;
Seven Down => ""8"";
STUFF Down => Stuff;
ENTER => Enter;
EXIT => Exit;
Point Down => COORDS, TakeInputFocus;
ENDCASE...

"L;
    MakeAtoms[];
    tipTable TIP.CreateTable[contents: tipContents,
      file: "TugBoat.TIP"L ! TIP.InvalidTable => RESUME];
    toolWindow Tool.Create[
      makeSWsProc: MakeMySWs, clientTransition: MyTransitionProc, name:
"TugBoat",
      tinyName1: "toot toot"L, cmSection: "TugBoat"L];
END;

MakeAtoms: PROC = {
    Enter Atom.MakeAtom["Enter"L];
    Exit Atom.MakeAtom["Exit"L];
    Copy Atom.MakeAtom["Copy"L];
    SuperCopy Atom.MakeAtom["SuperCopy"L];
    Delete Atom.MakeAtom["Delete"L];
    DrawLine Atom.MakeAtom["DrawLine"L];
    Stuff Atom.MakeAtom["Stuff"L];
    Paste Atom.MakeAtom["Paste"L];
    TakeInputFocus Atom.MakeAtom["TakeInputFocus"L];
    };
```

IV.3 More advanced topics

See the TIP chapter for a list of the basic commands. These are given here as hints for more experienced programmers.

NewManager — A **NewManager** command is used when you want to lock up the notifier, as is done in scrollbars, confirm cursors, adjusting and growing windows, the hourglass, adjusting and selecting text, FontMonster, and so forth.

TIP Tree — A TIP tree is a hierarchical series of TIP tables that can be searched until you find a match, reach the root table, or encounter an opaque table.

PushLocal — The **PushLocal** command is used when you want another TIP table, but you still want the window's **NotifyProc** to get the atoms. Example: a keyhack TIP table that maps **G Down = > ""BEGIN ! END""**

PushGlobal — The **PushGlobal** command is used when you have created a TIP table of global interest and want it to be searched by all processes. Perhaps **DOIT Up WHILE USERABORT Down = > ReBoot?**

"I" switch **orRESUME** — **TIP.CreateTable[]** uses the **contents:** field to build the TIP table instead of looking for the TIPC and then the TIP file.

Opaque Table — An opaque table is used if you want no further TIP table searching to be done. If this TIP table doesn't handle the current sequence of user actions, they are discarded.

ActionToWindow — The **ActionToWindow** command sends all user input to the window with the input focus except: **Adjust, Menu, Point, FIND, JFIRST, MENU,** and **USERABORT.**

CreateClient — **CreateClient** is used when you want your own TIP table to be the only one for a window, disjoint from the TIP tree.

IV.3.1 The GPM macro package

The GPM Macro Package translates mouse and keyboard interface language into encrypted code that is very compact and difficult to read. It is briefly documented here but is not recommended for extensive use except by experienced programmers.

A macro call consists of a macro name and a list of actual parameters, each separated by a comma. The name is preceded by a left square bracket ([) and the last parameter is followed by a right square bracket. A macro is defined by the special macro DEF, which takes two arguments: the name of the macro to be defined and the defining string. The defining string may contain special symbols that stand for the formal parameters. Enclosing any string in parentheses prevents evaluation of any macro calls inside; in place of evaluation, one "layer" of quotes is removed. It is usual to enclose the defining string of a macro definition in string quotes in order to prevent any macro calls or uses of formal parameters from being effective during the process of definition.

Here is a macro:

```
[DEF,IfShift,(SELECT ENABLE FROM
     LeftShift Down | RightShift Down = > ~1;
ENDCASE = > ~2)]

BS Down = > [IfShift,BackWord,BackSpace]
```

Here is a macro and its expansion from Mouse.TIP:

```
[DEF,ButtonEvents,(
     [DEF,ButtonEvent,(
        [DEF,SHIFT,(LeftShift Down | RightShift Down)]
        [DEF,CTRL,(CONTROL Down)]
        [DEF,COM,(COMMAND Down)]
        [DEF,TC,(TIME COORDS)]
        ~1 ~2 = > SELECT ENABLE FROM
   [SHIFT] = > SELECT ENABLE FROM
     [CTRL] = > SELECT ENABLE FROM
        [COM] = > { [TC] Command Control Shift ~1~2 };
        ENDCASE = > { [TC] Control Shift ~1~2 };
     ENDCASE = > SELECT ENABLE FROM
        [COM] = > { [TC] Command Shift ~1~2 };
        ENDCASE = > { [TC] Shift ~1~2 };
   [CTRL] = > SELECT ENABLE FROM
     [COM] = > { [TC] Command Control ~1~2 };
     ENDCASE = > { [TC] Control ~1~2 };
   [COM] = > { [TC] Command ~1~2 };
   ENDCASE = > { [TC] ~1~2 })]
     [ButtonEvent,~1,Down];[ButtonEvent,~1,Up])]

   [ButtonEvents, Point]

-- Expansion of Mouse.TIP

OPTIONS
   Fast;                          -- Top-level trigger select

SELECT TRIGGER FROM               -- Mouse and button actions

MOUSE = > SELECT ENABLE FROM
   Point Down = > COORDS, PointMotion;
   Menu Down = > COORDS, MenuMotion;
   Adjust Down = > COORDS, AdjustMotion;
   ENDCASE;

   Point Down = > SELECT ENABLE FROM
     LeftShift Down | RightShift Down = > SELECT ENABLE FROM
        CONTROL Down = > SELECT ENABLE FROM
   COMMAND Down = > { TIME COORDS Command Control Shift PointDown };
   ENDCASE = > { TIME COORDS Control Shift PointDown };
     ENDCASE = > SELECT ENABLE FROM
```

```
    COMMAND Down => { TIME COORDS Command Shift PointDown };
      ENDCASE => { TIME COORDS Shift PointDown };
    CONTROL Down => SELECT ENABLE FROM
      COMMAND Down => { TIME COORDS Command Control PointDown };
      ENDCASE => { TIME COORDS Control PointDown };
    COMMAND Down => { TIME COORDS Command PointDown };
    ENDCASE => { TIME COORDS PointDown };

  Point Up => SELECT ENABLE FROM
    LeftShift Down | RightShift Down => SELECT ENABLE FROM
      CONTROL Down => SELECT ENABLE FROM
    COMMAND Down => { TIME COORDS Command Control Shift PointUp };
    ENDCASE => { TIME COORDS Control Shift PointUp };
      ENDCASE => SELECT ENABLE FROM
    COMMAND Down => { TIME COORDS Command Shift PointUp };
    ENDCASE => { TIME COORDS Shift PointUp };
      CONTROL Down => SELECT ENABLE FROM
      COMMAND Down => { TIME COORDS Command Control PointUp };
      ENDCASE => { TIME COORDS Control PointUp };
    COMMAND Down => { TIME COORDS Command PointUp };
    ENDCASE => { TIME COORDS PointUp };

  Menu Down => SELECT ENABLE FROM
    LeftShift Down | RightShift Down => SELECT ENABLE FROM
      CONTROL Down => SELECT ENABLE FROM
    COMMAND Down => { TIME COORDS Command Control Shift MenuDown };
    ENDCASE => { TIME COORDS Control Shift MenuDown };
      ENDCASE => SELECT ENABLE FROM
    COMMAND Down => { TIME COORDS Command Shift MenuDown };
    ENDCASE => { TIME COORDS Shift MenuDown };
    CONTROL Down => SELECT ENABLE FROM
      COMMAND Down => { TIME COORDS Command Control MenuDown };
      ENDCASE => { TIME COORDS Control MenuDown };
    COMMAND Down => { TIME COORDS Command MenuDown };
    ENDCASE => { TIME COORDS MenuDown };

  Menu Up => SELECT ENABLE FROM
    LeftShift Down | RightShift Down => SELECT ENABLE FROM
      CONTROL Down => SELECT ENABLE FROM
    COMMAND Down => { TIME COORDS Command Control Shift MenuUp };
    ENDCASE => { TIME COORDS Control Shift MenuUp };
      ENDCASE => SELECT ENABLE FROM
    COMMAND Down => { TIME COORDS Command Shift MenuUp };
      ENDCASE => { TIME COORDS Shift MenuUp };
    CONTROL Down => SELECT ENABLE FROM
      COMMAND Down => { TIME COORDS Command Control MenuUp };
      ENDCASE => { TIME COORDS Control MenuUp };
    COMMAND Down => { TIME COORDS Command MenuUp };
    ENDCASE => { TIME COORDS MenuUp };
```

```
Adjust Down = > SELECT ENABLE FROM
   LeftShift Down | RightShift Down = > SELECT ENABLE FROM
      CONTROL Down = > SELECT ENABLE FROM
   COMMAND Down = > { TIME COORDS Command Control Shift AdjustDown };
   ENDCASE = > { TIME COORDS Control Shift AdjustDown };
      ENDCASE = > SELECT ENABLE FROM
   COMMAND Down = > { TIME COORDS Command Shift AdjustDown };
      ENDCASE = > { TIME COORDS Shift  AdjustDown };
   CONTROL Down = > SELECT ENABLE FROM
      COMMAND Down = > { TIME COORDS Command Control AdjustDown };
      ENDCASE = > { TIME COORDS Control AdjustDown };
   COMMAND Down = > { TIME COORDS Command AdjustDown };
   ENDCASE = > { TIME COORDS AdjustDown };

Adjust Up = > SELECT ENABLE FROM
   LeftShift Down | RightShift Down = > SELECT ENABLE FROM
      CONTROL Down = > SELECT ENABLE FROM
   COMMAND Down = > { TIME COORDS Command Control Shift AdjustUp };
   ENDCASE = > { TIME COORDS Control Shift AdjustUp };
      ENDCASE = > SELECT ENABLE FROM
   COMMAND Down = > { TIME COORDS Command Shift AdjustUp };
      ENDCASE = > { TIME COORDS Shift  AdjustUp };
   CONTROL Down = > SELECT ENABLE FROM
      COMMAND Down = > { TIME COORDS Command Control AdjustUp };
      ENDCASE = > { TIME COORDS Control AdjustUp };
   COMMAND Down = > { TIME COORDS Command AdjustUp };
   ENDCASE = > { TIME COORDS AdjustUp };

ENTER = > Enter;
EXIT = > Exit;

USERABORT Down = > Abort;

ENDCASE...
```

IV.3.2 Another TIP example

The following TIP table simulates a TeleVideo 920c terminal:

```
SELECT TRIGGER FROM
   A Down WHILE B Up WHILE C Up WHILE D Up ... WHILE Z Up = > CHAR
   B Down WHILE A Up WHILE C Up WHILE D Up ... WHILE Z Up = > CHAR
   .
   .
ENDCASE...
```

IV.4 Interface abstracts

Event is used with the **EventTypes** interface to allow clients to be notified of events that take place asynchronously on a system-wide basis.

EventTypes is used with the **Event** interface to allow clients to be notified of events that take place asynchronously on a system-wide basis.

TIP provides facilities for handling user input, including all key and mouse actions.

UserInput provides the client with routines for interpreting user actions and notifying tools of a change in the user state.

41

Event

The **Event** interface allows clients to be notified of actions (or events) that take place asynchronously on a system-wide basis. The actual notification mechanism is supplied by the **Supervisor** (see the **Supervisor** chapter of the *Pilot Programmer's Manual* for details). Tajo and CoPilot both invoke **Supervisor.NotifyDirectSubsystems** to notify clients of events that may interest them.

Event is used with the interface **EventTypes** to define events of interest. The **Event** interface contains **Supervisor.SubsystemHandle**s, on which a client may add dependencies. A **Supervisor.SubsystemHandle** may be thought of as a class of related events. A client specifies interest in a particular class of events by registering a dependency on the **Supervisor.SubsystemHandle** obtained from **Event**, specifying it as the implementor. The interface **EventTypes** provides some of the specific **Supervisor.Event**s that are raised. A client that has registered to be notified about a class of events uses the **Supervisor.Event** to determine which element of that class has occurred.

To write a program that will be notified about an event, first find the event definition in **EventType** and then add a dependency on the corresponding **Supervisor.SubsystemHandle** defined in **Event**. Unfortunately, there is not always a one-to-one correspondence between the events defined in **EventType** and **Supervisor.SubsystemHandle**s in **Event**. You must consider an event to be defined by the pair of items, one from **Event** and one from **EventTypes**.

41.1 Types

```
Event.Handle: Type = LONG POINTER TO Object;

Object: TYPE;
```

This type is for use with **Event.StartingProcess** and **Event.DoneWithProcess**.

41.2 Constants and data objects

```
Event.aboutToSwap: READONLY Supervisor.SubsystemHandle;
```

The **aboutToSwap** event class is used by the debugger and the Herald Window in Tajo to request permission to swap back to its client. The associated **EventType**s are

aboutToAbortSession, **aboutToBoot** and **aboutToBootPhysicalVolume**. Clients may optionally abort this event by raising the error **Supervisor.EnumerationAborted** from their agent procedures. If no client vetoes the request to swap, the debugger broadcasts the appropriate event in **Event.swapping**. (See the discussion at the end of this chapter.)

Event.displayState: READONLY Supervisor.SubsystemHandle;

Event.displayState is used to tell whether the display is on or off. Its **EventTypes** are **displayOff** and **displayOn**. (This event is not used by Tajo or CoPilot; it is included for future use.)

Event.fileSystem: READONLY Supervisor.SubsystemHandle;

File-system events that may interest clients include changing the search path, creating or deleting directories, and opening or closing volumes. The related **EventTypes** are **aboutToChangeSearchPath**, **newSearchPath**, **abortedSearchPathChange**, **directoryCreated**, **directoryDeleted**, **volumeOpened**, and **volumeClosed**. Events from **EventTypesExtra** are **aboutToOpenVolume** and **aboutToCloseVolume**.

Event.fileWindow: READONLY Supervisor.SubsystemHandle;

The **fileWindow** event class is concerned with events that affect windows maintained by the **FileWindow** interface. The **eventData** passed to the agent procedure is a **Window.Handle** for the affected window. The following events are defined in **EventTypes** for events on windows: **createWindow**, **destroy**, **edit**, **load**, **reset**, and **store**.

Event.powerOff: READONLY Supervisor.SubsystemHandle;

The **powerOff** event class is available for clients interested in performing some action before the machine powers down. The associated event defined in **EventTypes** is also called **powerOff**.

Event.primaryCredentials: READONLY Supervisor.SubsystemHandle;

The **primaryCredentials** event class is available for clients interested in monitoring changes to the user name and password. The associated event defined in **EventTypes** is also called **primaryCredentials**.

Event.swapping: READONLY Supervisor.SubsystemHandle;

The **swapping** event class is concerned with swapping; that is, with returning from the debugger to the client volume or entering the debugger from the client volume. These events cannot be vetoed; clients wishing to veto swaps should register for **Event.aboutToSwap** (see the end of this chapter for examples). Because clients can veto a swap, they must be notified whether the swap took place. Therefore, associated events defined in **EventTypes** fall into three categories: swap-out reasons, swap-in reasons, and swap-cancellations. The swap-in reasons are **newSession** and **resumeSession**. Swap-out reasons are **abortSession** **bootPhysicalVolume** and **resumeDebuggee**. **swapCancelled** and **bootPhysicalVolumeCancelled** are cancellation reasons. (Some private defaults that are unavailable to clients are used internally.)

Event.tajoDefaults: READONLY Supervisor.SubsystemHandle;

The **tajoDefaults** event class is concerned with system-wide defaults; the ones currently defined in **EventTypes** are **debugging**, **librarian**, **domain**, **organization**, **registry**, **fileServerProtocol**, and **systemFont**.

```
Event.toolWindow: READONLY Supervisor.SubsystemHandle; )
```

The **toolWindow** event class notifies clients when a tool is activated, deactivated, or created; the corresponding **EventTypes** are **createTool**, **activate**, and **deactivate**.

41.3 Signals and errors

None.

41.4 Procedures

The two procedures **Event.StartingProcess** and **Event.DoneWithProcess** keep track of non-notifier processes that are not otherwise protected against swapping. Conceptually, these procedures are actually counters: **Event.StartingProcess** adds 1 to the current count of running processes, and **Event.DoneWithProcess** decrements the count. When a swapping event occurs, it is aborted if the count of running processes is non-zero.

```
Event.DoneWithProcess: PROCEDURE [Event.Handle];
```

The parameter passed to this procedure is obtained by calling **Event.StartingProcess.**

```
Event.StartingProcess: PROCEDURE[id: LONG STRING] RETURNS[Handle];
```

Event.StartingProcess adds 1 to the total count of running processes. **id** is a message posted in the Herald Window if the swapping event is aborted.

41.5 Examples

The interface **EventTypes** provides some of the specific **Supervisor.Events** that are raised while the **Event** interface contains **Supervisor.SubsystemHandles** to which the client may wish to add dependencies. A typical fragment of client code might appear as follows:

```
NoteCredentialsChange: Supervisor.AgentProcedure =
    BEGIN
    SELECT event FROM
        EventTypes.primaryCredentials = > ...
        EventTypes.registry = > ...
        ENDCASE;
    END;
-- mainline
me: Supervisor.SubsystemHandle =
    Supervisor.CreateSubsystem[agent: NoteCredentialsChange,
instanceData: myInstanceData];
Supervisor.AddDependency[client: me, implementor: Event.tajoDefaults];
Supervisor.AddDependency[
```

```
    client: me, implementor: Event.primaryCredentials];
...
```

Tool writers should pay particular attention to the events involved in a world swap. When the user asks to leave CoPilot and return to the client, CoPilot notifies on the event **Event.aboutToSwap**. If any tool is unwilling or unable to stop for a world swap, it should abort this event by raising the error **Supervisor.EnumerationAborted**. If no clients abort the swap, CoPilot notifies on the event **Event.swapping** with a swap-out reason (**EventType.abortSession**, **EventType.resumeDebugee**, or **EventType.bootPhysicalVolume**). All tools are expected to stop when this event is notified. When CoPilot is re-entered for any reason, it raises the event **Event.swapping** with a swap-in reason (**EventType.newSession** or **EventType.resumeSession**) to let tools know that they can resume processing. The following example is typical of the swapping behavior expected of tools:

```
swapDone: CONDITION;
subsystemRunning, swapping: BOOLEAN ← FALSE;

aboutToSwapAgent: Supervisor.SubsystemHandle =
    Supervisor.CreateSubsystem[agent: AboutToSwap];
swappingAgent: Supervisor.SubsystemHandle =
    Supervisor.CreateSubsystem[agent: Swapping];

StartSubsystem: ENTRY PROCEDURE = {
    IF swapping THEN WAIT swapDone;
    subsystemRunning ← TRUE};
SubsystemStopped: ENTRY PROCEDURE = {subsystemRunning ← FALSE};

AboutToSwap: ENTRY Supervisor.AgentProcedure =
  BEGIN
  ENABLE UNWIND => NULL;
  IF subsystemRunning THEN {
    HeraldWindow.AppendMessage["MyTool busy: aborting swap."L];
    ERROR Supervisor.EnumerationAborted};
  END;

Swapping: ENTRY Supervisor.AgentProcedure =
  BEGIN
  ENABLE UNWIND => NULL;
  SELECT event FROM
   EventTypes.newSession, EventTypes.resumeSession, EventTypes.swapCancelled,
    EventTypes.bootPhysicalVolumeCancelled => {
    swapping ← FALSE; BROADCAST swapDone};
   EventTypes.abortSession, EventTypes.resumeDebuggee,
    EventTypes.bootPhysicalVolume =>
        swapping ← TRUE;
   ENDCASE;
  END;


-- mainline
Supervisor.AddDependency[client: aboutToSwapAgent,
implementor: Event.aboutToSwap];
```

```
Supervisor.AddDependency[
    client: swappingAgent, implementor: Event.swapping];
DO
 SubsystemStopped[];
    -- wait for user input from the Notifier
    StartSubsystem[];
    -- perform computation
    ENDLOOP;
...
```

42

EventTypes

The **EventTypes** interface allows clients to be notified of actions (or events) that take place asynchronously on a system-wide basis. The actual notification mechanism is supplied by the **Supervisor** (see the **Supervisor** chapter of the *Pilot Programmer's Manual* for details). Each of the **EventTypes** defined here is passed as the result of a Supervisor.**NotifyDirectSubsystems** for one of the events defined in the **Event** interface.

The interface **Event** is used with **EventTypes** to define events of interest. The **Event** interface contains Supervisor.**SubsystemHandle**s on which a client may add dependencies. A Supervisor.**SubsystemHandle** may be thought of as a class of related events; a client specifies interest in a particular class of events by adding a dependency on the corresponding Supervisor.**SubsystemHandle**. The interface **EventTypes** provides some of the specific Supervisor.**Event**s that are raised. A client that has registered to be notified about a class of events uses the Supervisor.**Event** to determine which element of that class has actually occurred.

Two of the **EventTypes** documented in this chapter are actually in the **EventTypesExtra** interface. They are EventTypesExtra.**aboutToOpenVolume** and EventTypesExtra.**aboutToCloseVolume**.

42.1 Types

The following **EventTypes** are used by Tajo for internal bookkeeping:

EventTypes.**CredentialEvents**: TYPE = ...

EventTypes.**DebugEvents**: TYPE = ...

EventTypes.**DisplayEvents**: TYPE = ...

EventTypes.**FileSystemEvents**: TYPE = ...

EventTypes.**FileWindowEvents**: TYPE = ...

EventTypes.**OtherEvents**: TYPE = ...

```
EventTypes.SpareEvents: TYPE = ...

EventTypes.TajoDefaultEvents: TYPE = ...

EventTypes.ToolWindowEvents: TYPE = ...

EventTypes.VetoEvents: TYPE = ...;
```

42.2 Constants and data objects

```
EventTypes.abortedSearchPathChange: Supervisor.Event = [EventTypes.firstFileSystem + 2];
```

abortedSearchPathChange is an event in the event class **Event.fileSystem**. It means that a previous notification that the search path would change has been aborted.

```
EventTypes.abortSession: Supervisor.Event = [EventTypes.firstDebugEvent + 5];
```

abortSession is an event in the event class **Event.swapping**. It means that the user has quit a debugging session and is returning to the client.

```
EventTypes.aboutToAbortSession: Supervisor.Event = [EventTypes.firstVetoEvent + 1];
```

aboutToAbortSession is an event in the event class **Event.aboutToSwap**. It informs interested clients that a world swap is about to occur. Tools should behave as though they will be interrupted but expect to be resumed later. This event should be vetoed by any process that is unable or unwilling to stop for the duration of the world swap.

```
EventTypes.aboutToBoot: Supervisor.Event = [EventTypes.firstVetoEvent];
```

aboutToBoot is an event in the event class **Event.aboutToSwap**. It means that a HeraldWindow boot is about to occur. In this case, the state of the current volume is going to disappear, never to return. Processes doing something physically destructive across reboots, like writing on the disk, should veto this event.

```
EventTypes.aboutToBootPhysicalVolume: Supervisor.Event = [EventTypes.firstVetoEvent + 3];
```

aboutToBootPhysicalVolume is an event in the event class **Event.aboutToSwap**. It means that the physical volume is about to be booted. It is similar to **aboutToBoot** because the current state will disappear, never to return. Processes doing something physically destructive across reboots, like writing on the disk, should veto this event.

```
EventTypes.aboutToChangeSearchPath: Supervisor.Event = [EventTypes.firstFileSystem];
```

aboutToChangeSearchPath is an event in the event class **Event.fileSystem**. It means that the current search path is about to be changed. Clients may veto this event.

```
EventTypesExtra.aboutToCloseVolume: Supervisor.Event = [EventTypes.firstFileSystem + 8];
```

aboutToCloseVolume is an event in the event class **Event.fileSystem**. It means that a logical volume is about to be closed. The parameter **eventData** passed to the agent

procedure contains the volume id of the volume that will be closed. Clients may veto this event.

```
EventTypesExtra.aboutToOpenVolume: Supervisor.Event = [EventTypes.firstFileSystem + 7];
```

aboutToOpenVolume is an event in the event class **Event.fileSystem**. It means that a logical volume is about to be opened, The parameter **eventData** passed to the agent procedure contains the volume id of the volume that will be opened. Clients may veto this event.

```
EventTypes.aboutToResume: Supervisor.Event = [EventTypes.firstVetoEvent + 2];
```

aboutToResume is an event in the event class **Event.aboutToSwap**. It means that the debugging session is about to be resumed. Clients should behave as though they will be temporarily interrupted, to be resumed later. Processes unable to stop for the duration of the world swap should veto this event.

```
EventTypes.activate: Supervisor.Event = [EventTypes.firstToolWndowEvent + 1];
```

activate is an event in the event class **Event.toolWindow**. It means that a particular tool has been activated. The window handle for the tool is passed as the **eventData**.

```
EventTypes.bootPhysicalVolume: Supervisor.Event = [EventTypes.firstDebugEvent + 7];
```

bootPhysicalVolume is an event in the event class **Event.swapping**. It means that no client has vetoed the previous **aboutToBootPhysicalVolume**, and the physical volume will be booted.

```
EventTypes.bootPhysicalVolumeCancelled: Supervisor.Event = [EventTypes.firstDebugEvent +
4];
```

bootPhysicalVolumeCancelled is an event in the event class **Event.swapping**. It means that at least one client has vetoed the previous **aboutToBootPhysicalVolume**, and the physical volume will not be booted.

```
EventTypes.createTool: Supervisor.Event = [EventTypes.firstToolWindowEvent];
```

createTool is an event in the class **Event.toolWindow**. It means that a tool has just been created. The window handle for the tool is passed as the **eventData**.

```
EventTypes.createWindow: Supervisor.Event = [EventTypes.firstFileWindowEvent];
```

createWindow is an event in the event class **Event.fileWindow**. It means that a new file window has been created. The window handle for the new window is passed as the **eventData**.

```
EventTypes.deactivate: Supervisor.Event = [EventTypes.firstToolWindowEvent + 2];
```

deactivate is an event in the class **Event.toolWindow**. It means that a tool has just been deactivated. The window handle for the tool is passed as the **eventData**. This event can be vetoed.

EventTypes.**debugging**: Supervisor.**Event** = [EventTypes.**firstDefaultEvent** + 1];

debugging is an event in the event class Event.**tajoDefaults**. It means that the value of the variable **debugging**, maintained in the **Profile** module, has changed.

EventTypes.**destroy**: Supervisor.**Event** = [EventTypes.**firstFileWindowEvent** + 1];

destroy is an event in the event class Event.**fileWindow**. It means that a file window has been destroyed. The window handle for the window is passed as the **eventData**.

EventTypes.**directoryCreated**: Supervisor.**Event** = [EventTypes.**firstFileSystem** + 3];

directoryCreated is an event in the class Event.**fileSystem**. It means that a new directory has just been created.

EventTypes.**directoryDeleted**: Supervisor.**Event** = [EventTypes.**firstFileSystem** + 4];

directoryDeleted is an event in the class Event.**fileSystem**. It means that an old directory has just been deleted.

EventTypes.**displayOff**: Supervisor.**Event** = [EventTypes.**firstDisplayEvent**];

displayOff is not currently used; it is included for future use.

EventTypes.**displayOn**: Supervisor.**Event** = [EventTypes.**firstDisplayEvent** + 1];

displayOn is not currently used; it is included for future use.

EventTypes.**domain**: Supervisor.**Event** = [EventTypes.**firstDefaultEvent** + 3];

domain is an event in the class Event.**tajoDefaults**. It means that the value of the variable **domain**, maintained in the **Profile** module, has changed.

EventTypes.**edit**: Supervisor.**Event** = [EventTypes.**firstFileWindowEvent** + 2];

edit is an event in the event class Event.**fileWindow**. It means that a file window has been opened for editing. The window handle for the window is passed as the **eventData**.

EventTypes.**fileServerProtocol**: Supervisor.**Event** = [EventTypes.**firstDefaultEvent** + 6];

fileServerProtocol is an event in the class Event.**tajoDefaults**. It means that the file server protocol, which is maintained by the **Profile** module, has changed from NS to PUP or vice versa. This event will never occur in a product configuration because file server protocols are always NS. This event type will be removed in a future release.

The following **EventTypes** are used by Tajo for internal bookeeping:

firstCredentialEvent, **firstDebugEvent**, **firstDefaultEvent**, **firstDisplayEvent**, **firstFileSystemEvent**, **firstFileWindowEvent**, **firstOtherEvent**, **firstSpare** and **firstVetoEvent**.

EventTypes.flushSymbols: Supervisor.Event = [EventTypes.firstDebugEvent];

flushSymbols is in the event class Event.swapping and is for private use by the debugger.

EventTypes.librarian: Supervisor.Event = [EventTypes.firstDefaultEvent + 2];

librarian is an event in the event class Event.tajoDefaults. It means that the value of the default librarian server, maintained in the **Profile** module, has changed.

EventTypes.load: Supervisor.Event = [EventTypes.firstFileWindowEvent + 3];

load is an event in the event class Event.fileWindow. It means that a file window has been loaded with a new file. The window handle for the window is passed as the **eventData**.

EventTypes.newSearchPath: Supervisor.Event = [EventTypes.firstFileSystem + 1];

newSearchPath is an event in the event class Event.fileSystem. It means that the previous notification of **aboutToChangeSearchPath** was not vetoed, and the search path will be changed.

EventTypes.newSession: Supervisor.Event = [EventTypes.firstDebugEvent + 1];

newSession is an event in the event class Event.swapping. It represents a swapping-in reason and means that CoPilot has been entered for debugging for the first time in a session.

EventTypes.organization: Supervisor.Event = [EventTypes.firstDefaultEvent + 4];

organization is an event in the event class Event.tajoDefaults. When the organization field of the Profile Tool changes, interested clients arenotified with the reason EventTypes.organization.

EventTypes.powerOff: Supervisor.Event = [EventTypes.firstOtherEvent];

powerOff is an event in the event class Event.powerOff. It means that the machine is about to be turned off.

EventTypes.primaryCredentials: Supervisor.Event = [EventTypes.firstCredentialEvent];

primaryCredentials is an event in the event class Event.primaryCredentials. It means that the user name, password, or both have changed.

EventTypes.registry: Supervisor.Event = [EventTypes.firstDefaultEvent + 5];

registry is an event in the event class Event.tajoDefaults. It means that the value of the default registry, maintained in the Profile module, has changed.

EventTypes.reset: Supervisor.Event = [EventTypes.firstFileWindowEvent + 4];

reset is an event in the event class Event.fileWindow. It means that a file window has been reset. The window handle for the window is passed as the **eventData**.

```
EventTypes.resumeDebuggee: Supervisor.Event = [EventTypes.firstDebugEvent + 6];
```

resumeDebuggee is an event in the event class **Event.swapping**. It means that the user is proceeding from a debugging session and returning to the client.

```
EventTypes.resumeSession: Supervisor.Event = [EventTypes.firstDebugEvent + 2];
```

resumeSession is an event in the event class **Event.swapping**. It means that Copilot has been re-entered for debugging.

```
EventTypes.store: Supervisor.Event = [EventTypes.firstFileWindowEvent + 5];
```

store is an event in the event class **Event.fileWindow**. It means that the file in a file window open for editing has been saved or stored and the window is no longer open for editing. The window handle for the window is passed as the **eventData**.

```
EventTypes.swapCancelled: Supervisor.Event = [EventTypes.firstDebugEvent + 3];
```

swapCancelled is an event in the class **Event.swapping**. It means that a previous notification of a swap was vetoed by at least one client.

```
EventTypes.systemFont: Supervisor.Event = [EventTypes.firstDefaultEvent + 7];
```

systemFont is an event in the class **Event.tajoDefaults**. It is used to notify clients when the default font used to display text is changed.

```
EventTypes.tellFileSystemSwappingIn:Supervisor.Event = [EventTypes.firstDebugEvent + 9];
EventTypes.tellFileSystemSwappingOut:Supervisor.Event = [EventTypes.firstDebugEvent + 8];
```

These two events, in the class **Event.swapping**, are for private use by CoPilot.

```
EventTypes.volumeClosed:Supervisor.Event = [EventTypes.firstFileSystem + 6];
EventTypes.volumeOpened:Supervisor.Event = [EventTypes.firstFileSystem + 5];
```

These two events, in the class **Event.fileSystem**, are used to notify when a logical volume has been opened or closed. The parameter **eventData**, passed to the agent procedure, contains the volume id of the volume that is opening or closing.

42.3 Signals and errors

None.

42.4 Procedures

None.

42.5 Examples

See **Event**.

43

TIP

TIP allows you to customize the keyboard for programming the user interface. It translates hardware-level actions from the keyboard, mouse, and keyset into higher-level client action requests (result lists). The acronym TIP stands for *t*erminal *i*nterface *p*ackage. (See also the **UserInput** chapter.)

43.1 Types

```
TIP.DownUp: TYPE = Keys.DownUp; -- {down, up}
```

DownUp is an enumerated type that describes the two possible key and button states.

```
TIP.GlobalTable: TYPE = {
  root, formSW, textSW, fileWindow, ttySW, executive, spare1, spare2};
```

These are the indices of the predefined global TIP table array **TIP.globalTable**.

```
TIP.KeyName: TYPE = Keys.KeyName;
```

KeyName is an enumerated type that describes the keyboard and mouse buttons. It is used to index the table **TIP.actionToWindow** and is provided here for convienience. (See the *Pilot Programmer's Manual* for a complete list of its elements.)

```
TIP.NotifyProc: TYPE = PROCEDURE [window: Window.Handle, results: TIP.Results];
```

When a sequence of user actions matching the left side of a statement in a TIP table occurs, a **NotifyProc** is called with the results list of that statement.

```
TIP.ResultElement: TYPE = RECORD [
  SELECT type: * FROM
    char = > [c: CHARACTER],
    coords = > [place: Window.Place],
    keys = > [keys: LONG POINTER TO Keys.KeyBits],
    atom = > [a: Atom.ATOM],
    int = > [i: LONG INTEGER],
    string = > [s: LONG STRING],
```

```
time => [time: System.Pulses],
ENDCASE];
```

The right side of a statement in a TIP table is a list of results to be passed to the client when the specified action(s) occurs. Each element in this list is described by a **ResultElement**. (See also the descriptions of **TIP.Results**, **TIP.First**, and **T,IP.Rest**.) **Note:** The place in a **coords ResultElement** is relative to the **window** argument of the **NotifyProc**.

```
TIP.Results: TYPE = LONG POINTER TO TIP.ResultsList;
TIP.ResultsList: TYPE;
```

A **NotifyProc** is passed a list of results. The client enumerates the list with the procedure **TIP.Rest** and extracts the elements of the list with the procedure **TIP.First** (*q.v.*).

```
TIP.Table: TYPE = LONG POINTER TO TIP.TableObject;
TIP.TableObject: TYPE;
```

A **Table** is a pointer to the internal representation of a TIP table.

43.2 Constants and data objects

```
TIPExtras.clickTimeout: System.Pulses;
```

clickTimeout determines the maximum time allowed between two clicks of a multi-click. If a mouse button goes down more than **clickTimeout Pulses** after the previous button transition, it is treated as a separate selection action. The current selection does not go to the next level of the selection hierarchy (Character -> Word -> Line -> Window). This value is global for the entire environment. This item is currently in the **TIPExtras** interface.

```
TIP.actionToWindow: PACKED ARRAY Keys.KeyName OF BOOLEAN;
```

actionToWindow determines if a user action should be sent to the window containing the cursor (**TRUE**) or to the window containing the current input focus (**FALSE**). This array is global for the entire environment. It is initialized for all actions to go to the input focus, except those associated with the **Adjust**, **Menu**, and **Point** mouse buttons and the **FIND**, **JFIRST**, **MENU**, and **USERABORT** keys.

```
globalTable: READONLY ARRAY GlobalTable OF TIP.Table;
```

Elements of **globalTable** are predefined, globally available TIP tables. These tables are definied at boot time and do not change after they are initialized. Figure 43.1 shows how they form a chain. (See **TIP.PushGlobal** and **TIP.PushLocal** for examples of how they are used.)

```
TIPExtras.mouseTIP: TIP.Table;
```

mouseTIP is a convenient TIP table that is made available to clients who need to watch mouse events, both buttons and tracking. It has no successor tables. This item is currently in the **TIPExtras** interface.

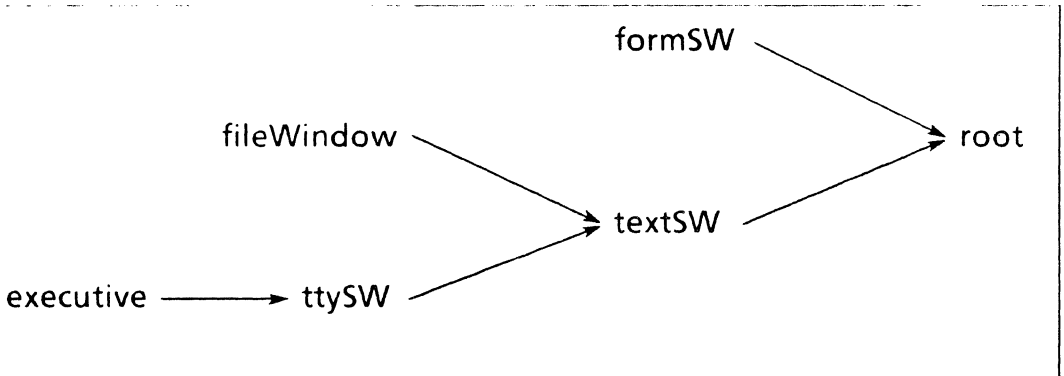


Figure 43.1: Dependency Structure of Global Tables

43.3 Signals and errors

```
TIP.InvalidTable: SIGNAL [type: TIP.TableError, message: LONG STRING];
TIP.TableError: TYPE = {fileNotFound, badSyntax};.
```

InvalidTable is raised only by **TIP.CreateTable**. The type is **fileNotFound** if the file could not be found and the **contents** string was empty. **fileNotFound** is raised as an **ERROR**. The type is **badSyntax** if the current file is syntactically incorrect. If **badSyntax** is **RESUME**d, and **contents** is not empty, the **contents** are written into **file** and it is reparsed. If the file has been overwritten or **contents** is empty and a syntax error occurs, the error will be **badSyntax**. In this case, if the signal is resumed, **CreateTable** simply returns **NIL**.

43.4 Procedures

```
TIP.CreateClient: PROCEDURE [
   window: Window.Handle, table: TIP.Table ← NIL, notify: TIP.NotifyProc ← NIL];
```

CreateClient makes window a potential TIP client. If **window** is already a TIP client and **table** or **notify** is **NIL**, then the old value is retained. **Note:** This procedure is not called by most clients of this interface since all windows created by **Tool**, **ToolWindow**, or any of the Subwindow interfaces are already TIP clients. Call this routine only if you are creating your own subwindow type.

```
TIP.CreateTable: PROCEDURE [
   file: LONG STRING ← NIL, opaque: BOOLEAN ← FALSE, z: UNCOUNTED ZONE ← NIL,
   contents: LONG STRING ← NIL ]
   RETURNS [table: TIP.Table];
```

CreateTable generates a **TIP.Table** from the text file named by **file** (which may not be **NIL**). If opaque is **TRUE**, then unrecognized actions are discarded without searching the table chain past this entry. **table** will be allocated in **z**. If **z** is **NIL**, it is allocated from a zone owned by the TIP table manager. There is no procedure provided for destroying TIP tables, so if you will want to free **table** later, provide a zone that you may destroy. Further, note that you should not destroy tables on which you haved done a push global. **contents** is the default contents of **file** and will be used if (1) you boot with the "I" switch, (2) **file** cannot be read, or (3) you **RESUME TIP.InvalidTable[badSyntax, ...]**. (See **TIP.InvalidTable** for further details on how to treat that **SIGNAL**.)

When **file** is parsed, a compiled form of the table is written into a file with a name constructed by appending a "C" on the end of **file**. **file** should typically have the extentsion ".TIP".

When **CreateTable** is called, if a ".TIPC" file exits that was created from **file**, the ".TIPC" file is used to generate **table**.

This procedure may raise the SIGNAL **TIP.InvalidTable**.

TIP.DestroyClient: PROCEDURE [**window**: **Window.Handle**];

DestroyClient frees the resources allocated by **CreateClient**.

TIP.First: PROCEDURE [**results**: **TIP.Results**] RETURNS [**TIP.ResultElement**];

First returns the first **TIP.ResultElement** associated with the list **results**.

TIP.FlushUserInput: PROCEDURE;

FlushUserInput empties the queue of pending user actions (type-ahead and button-ahead).

TIP. GetNotifyProc: PROCEDURE [**window**: **Window.Handle**] RETURNS [**TIP.NotifyProc**];

GetNotifyProc returns the **TIP.NotifyProc** associated with **window**.

TIP. GetNotifyProcFromTable: PROCEDURE [**table**: **TIP.Table**] RETURNS [**TIP.NotifyProc**];

GetNotifyProc FromTable returns the **TIP.NotifyProc** associated with **table**.

TIP. GetPlace: PROCEDURE [**window**: **Window.Handle**] RETURNS [**Window.Place**];

GetPlace returns the **window**-relative coordinate of the last user action that was matched. **GetPlace** should be invoked only while in the call stack of a **TIP.NotifyProc**.

TIP. GetTable: PROCEDURE [**window**: **Window.Handle**] RETURNS [**TIP.Table**];

GetTable returns the head of the **TIP.Table** chain associated with **window**.

TIP.NewManager: PROCEDURE [
  window: **Window.Handle**, **table**: **TIP.Table**, **notify**: **TIP.NotifyProc** ← NIL];

NewManager sends all user actions through **table** and **notify** using **window**, instead of through the **window**, **table**, and **notify** procedure determined by **TIP.actionToWindow** and the **Match** process. If **table** is NIL, the standard mechanisms determine where actions are sent.

TIP.NextTable: PROCEDURE [**table**: **TIP.Table**] RETURNS [**next**: **TIP.Table**];

NextTable returns the TIP table following **table** in the chain. **next** will be NIL if there is no successor table.

TIP.PushGlobal: PROCEDURE [
  push: **TIP.Table**, **onto**: **TIP.GlobalTable**, **opaque**: BOOLEAN ← FALSE];

TIP.PushLocal: PROCEDURE [**push**, **onto**: **TIP.Table**, **opaque**: BOOLEAN ← FALSE];

PushGlobal and **PushLocal** manipulate the relationships among TIP.Tables. If **opaque** is TRUE, unrecognized user actions will be discarded without searching the table chain past the opaque entry.

PushGlobal inserts **push** after the global table indexed by **onto**.

PushLocal appends the chain of TIP.Tables headed by **onto** the successor of the chain headed by **push**. Other clients sharing **onto** will not be affected.

Note: Never supply the same actual parameter to the formal paramenter **push** more than once.

```
TIP.Rest: PROCEDURE [results: TIP.Results] RETURNS [TIP.Results];
```

Rest advances **results** one element. NIL is returned when **results** is exhausted.

```
TIP. SetNotifyProc: PROCEDURE [window: Window.Handle, notify: TIP.NotifyProc]
  RETURNS [oldNotify:TIP.NotifyProc];
```

SetNotifyProc sets the TIP.NotifyProc associated with **window** to be **notify** and returns the old TIP.NotifyProc.

```
TIP. SetNotifyProcForTable: PROCEDURE [table: TIP.Table, notify: TIP.NotifyProc]
  RETURNS [oldNotify: TIP.NotifyProc];
```

SetNotifyProcForTable sets the TIP.NotifyProc associated with **table** to be **notify** and returns the old TIP.NotifyProc. Note: results from statements in **table** go to **notify** instead of to the **NotifyProc** for whatever window this chain is associated with.

```
TIP. SetTable: PROCEDURE [window: Window.Handle, table: TIP.Table]
  RETURNS [oldTable: TIP.Table];
```

SetTable sets the TIP.Table associated with **window** to be **table** and returns the old TIP.Table.

43.5 Discussion

TIP tables describe the translation from keyboard and mouse actions into client actions. Every time a user action (key transition, button transition, or mouse movement) occurs, the TIP software determines which window that event is for and looks the event up in the first table of the chain of TIP tables associated with that window. If the event matches the left side of a statement in that TIP table, the right side (result list) of the statement is passed to the **NotifyProc** for that window. If no match is found, the next table in the chain is checked, and so on. If no match is found in any table, the event is discarded.

43.5.1 Overview

A TIP table specifies a translation between a sequence of user actions and a sequence of client actions. These tables are created and linked by the client and made available to the translation process (the TIP matcher).

The Stimulus Level (or StimLev) watches the hardware for user actions and queues them, along with their time of occurrence, in the User Action Queue.

The match process (also called the Matcher or Notifier) dequeues each user action and then determines which window this event is associated with. If the entry in TIP.**actionToWindow** is TRUE, the window is the one containing the cursor; otherwise, it is the window with the input focus. After determining the appropriate window, the match process gets the first TIP table in the chain associated with that window. It then attempts to match the user action against statements in that TIP table and succeeding tables until (1) a match is found, (2) an opaque table is encountered, or (3) the end of the table chain is reached. If no match is found, that user action is discarded and the match process dequeues the next user action. If a match is found, the appropriate notify procedure is called. (Normally this is the notify procedure for that window.) Thus you can add a table with results that the client window's notification procedure is expected to understand without having to write an interpreter for those results yourself. In special circumstances, a notify procedure can be associated with the TIP table itself. In that case, the table's notify procedure is called instead of the window's.

Predefined system-supplied global tables provide basic mouse, chord, and character facilities, or change or add functions to these basic facilities for specific window types. The structure of tables is an inverted tree or a number of linked lists with separate heads and common tails. For example, TTY subwindows need to transmit an ASCII backspace character when the BS key goes down. TTY subwindows are also text subwindows, and text subwindows already define BS as an editing action. Therefore TTY subwindows have their own table that overrides the text subwindow definition of the action to take when this key goes down.

Fine point: The StimLev will not enqueue more than a certain fixed number of continuous mouse motions. After n continuous mouse motions are enqueued, with none dequeued and no intervening user actions, instead of enqueuing the $n+1$st, it replaces the nth with the $n+1$st.

43.5.2 Using TIP tables

If you need to become a client of this interface, use TIP.**CreateTable** to make a TIP.**Table** from its user-editable disk representation. (See the following sections for details on the internals of a .TIP file.) Link this table with whatever system-supplied TIP table you find useful, using TIP.**PushGlobal**, TIP.**PushLocal**, or TIP.**CreateClient**, and passing it your table and your **NotifyProc**.

When the match process recognises an event, the **NotifyProc** is called with the parameter **results**, which is a list of values collected from the table while parsing an event. This list structure is opaque; clients should use the procedures TIP.**First** and TIP.**Rest** to access its elements.

43.5.3 Syntax of TIP tables

Here is the BNF description for syntactically correct TIP tables. Non-terminals are boldface, terminals are non-bold Titan (such as `FastMouse`). The characters "`"`", "`.`", "`;`", "`,`", "`=>`", "`{`", and "`}`" in the BNF below are terminal symbols.

```
TIPTable            :: = Options TriggerStmt .

Options             :: = empty | OPTIONS OptionList ;
OptionList          :: = Option | Option , OptionList
Option              :: = SmallOrFast | PrintOrDefaultKeys | FastOrSlowMouse
SmallOrFast         :: = Small | Fast
PrintOrDefaultKeys  :: = PrintKeys | DefaultKeys
FastOrSlowMouse     :: = FastMouse | SlowMouse
Expression          :: = AND TriggerChoice | WHILE EnableChoice | => Statement
Statement           :: = TriggerStmt | EnableStmt | Results

TriggerStmt         :: = SELECT TRIGGER FROM TriggerChoiceSeries
EnableStmt          :: = SELECT ENABLE FROM EnableChoiceSeries

TriggerChoiceSeries :: = TriggerChoice ; TriggerChoiceSeries
                       | TriggerChoice ENDCASE FinalChoice
                       | ENDCASE FinalChoice
EnableChoiceSeries  :: = EnableChoice ; EnableChoiceSeries
                       | EnableChoice ENDCASE FinalChoice
                       | ENDCASE FinalChoice

TriggerChoice       :: = TriggerTerm Expression
EnableChoice        :: = EnableTerm Expression
FinalChoice         :: = empty | => Statement

TriggerTerm         :: = ( Key | MOUSE | ENTER | EXIT ) TimeOut
EnableTerm          :: = KeyEnableList | PredicateIdent

TimeOut             :: = empty | BEFORE Number | AFTER Number

KeyEnableList       :: = Key | Key | KeyEnableList
                       Note: the | between Key and KeyEnableList is a
                       terminal and must be entered.
Key                 :: = KeyIdent UP | KeyIdent DOWN

Results             :: = ResultItem | ResultItem , Results | ResultItem Expression
                       | { ResultItems }
ResultItems         :: = ResultItem | ResultItem ResultItems
ResultItem          :: = COORDS | CHAR | KEYS | TIME | String | Number
                       | ResultIdent

String              :: = "any sequence of characters not containing a ""

ResultIdent         :: = Ident
KeyIdent            :: = Ident
PredicateIdent      :: = Ident
```

43.5.4. Semantics of TIP tables

```
TIPTable            :: = Options TriggerStmt .
```

Note that TIP tables terminate with a period.

```
Options             :: = empty | OPTIONS OptionList ;
OptionList          :: = Option | Option , OptionList
Option              :: = SmallOrFast | PrintOrDefaultKeys | FastOrSlowMouse
SmallOrFast         :: = Small | Fast
PrintOrDefaultKeys  :: = PrintKeys | DefaultKeys
FastOrSlowMouse     :: = FastMouse | SlowMouse
```

`Small` Indicates to table builder that you favor storage over lookup speed (default).

`Fast` Indicates to table builder that you favor lookup speed over storage.

`PrintKeys` As above, but only printing keys (not 'Return', control combinations, or mouse actions).

`DefaultKeys` Adds all normal keyboard events, including control characters.

`FastMouse` Indicates to table matcher that you want to see ALL mouse movement when you use **TriggerTerm** MOUSE.

`SlowMouse` Indicates to table matcher that you want to see only the last mouse motion when you use **TriggerTerm** MOUSE (default).

```
Expression          :: = AND TriggerChoice | WHILE EnableChoice | => Statement
```

AND TriggerChoice matches if and only if **TriggerChoice** happens *immediately after* the preceding choice. For example, A Down AND B Down means "A goes down and then B goes down" (with no intervening actions like A Up or Mouse motion).

WHILE EnableChoice matches if **EnableChoice** is also true at this point. For example, A Down WHILE B Down matches if A goes down while B is down.

=> Statement continue processing at **Statement** (used for results and common prefixes)

```
Statement           :: = TriggerStmt | EnableStmt | Results
TriggerStmt         :: = SELECT TRIGGER FROM TriggerChoiceSeries
EnableStmt          :: = SELECT ENABLE FROM EnableChoiceSeries
```

EnableStmt matches if any of the **EnableChoiceSeries** *has already happened.*

TriggerStmt matches if any of the **TriggerChoiceSeries** *has just happened.*

```
TriggerChoiceSeries :: = TriggerChoice ; TriggerChoiceSeries
                       | TriggerChoice ENDCASE FinalChoice
                       | ENDCASE FinalChoice
EnableChoiceSeries  :: = EnableChoice ; EnableChoiceSeries
                       | EnableChoice ENDCASE FinalChoice
                       | ENDCASE FinalChoice
TriggerChoice       :: = TriggerTerm Expression
EnableChoice        :: = EnableTerm Expression
FinalChoice         :: = empty | => Statement
```

TriggerTerm	:: = (**Key** | MOUSE | ENTER | EXIT) **TimeOut**
TimeOut	:: = **empty** | BEFORE **Number** | AFTER **Number**
Key	matches if the appropriate key transition occurs.
MOUSE	matches if there is mouse motion (useful for tracking the mouse).
ENTER	matches if the mouse enters the window.
EXIT	matches if the mouse leaves the window.
BEFORE **Number**	matches if the associated **TriggerTerm** happens within a given number of milliseconds of the preceding (matched) user action. For example, A Down AND B Down BEFORE 200 matches if A went down and then B went down within 1/5 second (and if there were no intervening actions).
AFTER **Number**	matches if the associated **TriggerTerm** happens a given number of milliseconds or more after the preceding user action. For example, A Down AND B Down AFTER 200 matches if A went down and then B went down more than 1/5 second later (and if there were no intervening actions).
EnableTerm	:: = **KeyEnableList** | **PredicateIdent**
KeyEnableList	:: = **Key** | **Key** | **KeyEnableList** *Note: the | between* **Key** *and* **KeyEnableList** *is a terminal and must be entered.*
KeyEnableList	is true if any of the **Key**s are true.
Key	:: = **KeyIdent** UP | **KeyIdent** DOWN
Key	is true if the appropriate transition has happened (either is the current user action if part of a trigger term, or has already happened if an enable term).
Results	:: = **ResultItem** | **ResultItem** , **Results** | **ResultItem Expression** | { **ResultItems** }
ResultItems	:: = **ResultItem** | **ResultItem ResultItems**
ResultItem	:: = COORDS | CHAR | KEYS | TIME | **String** | **Number** | **ResultIdent**
String	:: = **"any sequence of characters not containing a ""**
ResultIdent	:: = **Ident**
COORDS	returns a coord **ResultElement** with the coords of the last user action
CHAR	returns a char **ResultElement** with the character representation of the last user action
KEYS	returns a keys **ResultElement** with the current state of all the keys. (This is not recommended in normal usage. Usually a more complex TIP table is indicated if you are using this result.)
TIME	returns a time **ResultElement** with the time of the last (matched) user action.
String	returns a string **ResultElement**.

Number	returns an int **ResultElement**.
ResultIdent	returns an atom **ResultElement**.
KeyIdent	:: = Ident
One of:	A ... Z, One, Two, Three, ... Zero, Adjust, AGAIN, Arrow, ATTENTION, BackSlash, BS, CLIENT1, CLIENT2, Comma, COMMAND, COMPLETE, CONTROL, COPY, Dash, DELETE, DOIT, Equal, EXPAND, FIND, HELP, JFIRST, JSELECT, Keyset1, Keyset2, Keyset3, Keyset4, Keyset5, LeftBracket, LeftShift, LOCK, Menu, MENU, MOVE, NEXT, PASTE, Period, Point, Quote, RESERVED, RETURN, RightBracket, RightShift, SCROLLBAR, SemiColon, Slash, Space, STUFF, TAB, UNDO, USERABORT, A8, A9, A10, A11, A12, L1, L4, L7, L10, Key47, R3, R4, R9, R10
PredicateIdent	:: = Ident
PredicateIdent	is not currently implemented.

The whole match process can be viewed as a **SELECT** statement that is continuously reading key transitions, mouse movements, or key states from the input queue. A trigger statement looks at the next action recorded in the input queue and branches to the appropriate choice. An enable statement selects between the different choices according to the current state of the keyboard or the mouse keys. Trigger terms may appear in sequence, separated by **AND**. They may be mixed with enable terms, which in turn are characterized by the keyword **WHILE**. A timeout following a trigger indicates a timing condition that has to hold between this trigger and its predecessor. The number associated with the timeout expresses a time interval in milliseconds. Events starting with the same sequence of trigger and/or enable terms are expressed as nested statements. Result items may be names, numbers, strings, or the keywords **COORDS**, **CHAR**, **KEYS**, or **TIME**. The results of a successfully parsed event are passed to the user as an opaque list whose elements are extracted with the procedures **TIP.First** and **TIP.Rest**. Names appear as identifiers, numbers as **LONG INTEGERS**, and strings as **LONG STRINGS**. **Char** comes as **CHARACTER** containing the ASCII interpretation of the key involved with the event. **Coords** results in a **Window.Place** containing the mouse coordinates of the event.

For example, the **PrintKeys** entry for the letter "a" can be represented as:

```
SELECT TRIGGER FROM
  A Down WHILE CONTROL Up => CHAR;
...
```

This event is triggered when the A key goes down only if the **CONTROL** key is up. It puts a result on the list that will be the character **a**.

A more elaborate example may look like this:

```
SELECT TRIGGER FROM
  Point Down =>
    SELECT TRIGGER FROM
      Point Up BEFORE 200 AND Point Down BEFORE 200 =>
        SELECT ENABLE FROM
```

```
        LeftShift Down => COORDS, ShiftedDoubleClick
        ENDCASE        => COORDS, NormalDoubleClick;
Adjust Down BEFORE 300 => PointAndAdjust;
ENDCASE => COORDS, SimpleClick;
...
```

This table produces the result element (atom) **NormalDoubleClick** along with the mouse coordinates if the left mouse button goes down, remains there not longer than 200 ms, and goes down again before another 200-ms lapse. The result is **ShiftedDoubleClick** if the same actions occur and the left shift key is down. If the right mouse button also goes down less than 300 ms after the initial **Point Down** and the right mouse button also goes down, **PointAndAdjust** results. Finally, the table specifies the result **SimpleClick** (with coordinates) if **Point** goes down but none of the succeeding actions occurs.

Following is a list of namesyou might want to use for the keys :

Letters: A ... Z.

Numbers: One, Two, Three, ... Zero.

Functions: Adjust, AGAIN, Arrow, ATTENTION, BackSlash, BS, CLIENT1, CLIENT2, Comma, COMMAND, COMPLETE, CONTROL, COPY, Dash, DELETE, DOIT, Equal, EXPAND, FIND, HELP, JFIRST, JSELECT, Keyset1, Keyset2, Keyset3, Keyset4, Keyset5, LeftBracket, LeftShift, LOCK, Menu, MENU, MOVE, NEXT, PASTE, Period, Point, Quote, RESERVED, RETURN, RightBracket, RightShift, SCROLLBAR, SemiColon, Slash, Space, STUFF, TAB, UNDO, USERABORT.

Others: A8, A9, A10, A11, A12, L1, L4, L7, L10, Key47, R3, R4, R9, R10

There are no names for shifted characters like left or right paren. Instead, you must specify one or both shift keys plus the unshifted key name. For example, **Nine Down WHILE LeftShift Down** instead of **LeftParen Down**

43.5.5. GPM: macro package

The macro package used in TIP is based on the "General-Purpose Macrogenerator" described by Strachey in *Computer Journal* (October 1965). The following summary is based on that article; refer to the article itself for more details.

A macro call consists of a macro name and a list of actual parameters, each separated by a comma. The name is preceded by a left square bracket ([) and the last parameter is followed by a right square bracket. A macro is defined by the special macro DEF, which takes two arguments: the name of the macro to be defined and the defining string. The defining string may contain the special symbols ~1, ~2, etc., which stand for the first, second, etc., formal parameters. Enclosing any string in parentheses prevents evaluation of any macro calls inside; in place of evaluation, one "layer" of string quotes is removed. It is usual to enclose the defining string of a macro definition in string quotes to prevent any macro calls or uses of formal parameters from being effective during definition.

Here are some sample macros and an example:

```
-- macro definitions
[DEF,LSHIFT,(LeftShift Down)]
[DEF,RSHIFT,(RightShift Down)]
[DEF,EitherShift,(
  [LSHIFT] => ~1;
  [RSHIFT] => ~1)]

-- trigger cases
SELECT TRIGGER FROM
BS Down => SELECT ENABLE FROM
  [EitherShift,{BackWord}];
  ENDCASE => {BackSpace};
-- more cases ...
ENDCASE...
```

The above example expands to:

```
BS Down => SELECT ENABLE FROM
  LeftShift Down => BackWord;
  RightShift Down => BackWord;
  ENDCASE => {BackSpace};
```

44

UserInput

The **UserInput** interface provides the client with routines that manage the input focus, user type-in, the periodic notifier, and **UserAbort**. (See TIP for keyboard-handling facilities.)

The **User TypeIn** facility, which is built with the TIP facilities, lets the client supply a procedure to be called whenever actions have taken place that correspond to a character being typed (such as key down, key up, shift). Type-in frees the client from being concerned about how it is done.

The procedure **UserAbort** may be called to see if the user has pressed **ABORT** while the cursor is in its window. Periodic notifiers are useful for procedures that are to be performed at regular intervals and for procedures that must be executed from within the Notifier. (See the examples at the end of this chapter.)

44.1 Types

```
UserInput.AttentionProcType: TYPE = PROC [window: Window.Handle];
```

An **AttentionProcType** is called whenever the **ABORT** key is pressed. It is called "outside" the Notifier as soon as the stimulus level sees the key go down.

```
UserInput.CaretProcType: TYPE = PROC window:
  [Window.Handle, startStop: UserInput.StartStop];
```

A **CaretProcType** is called when the input focus changes. **startStop** is a flag designating whether to start or stop blinking the caret marking the type-in point (see **CreateStringInOut**).

```
UserInput.PeriodicNotifyHandle: TYPE [1];
```

Clients sometimes want to wake up at regular time intervals to do some operation. However, a client may need operations that, if done while the Notifier was invoking some other operation, would either preempt the user or cause serious problems in Tajo (such as blinking the type-in caret). Thus, the *periodic notification* mechanism is provided.

```
UserInput.PeriodicProcType: TYPE = PROC [window: Window.Handle, place: Window.Place];
```

A **PeriodicProcType** is one that is called by the periodic notifier at regular intervals, as described in **CreatePeriodicNotify**. **window** is the window passed to **CreatePeriodicNotify**; **place** is the window-relative place of the cursor when the procedure is called.

```
UserInput.StartStop: TYPE = {start, stop};
```

This is used as an argument to a **CaretProcType**.

```
UserInput.StringProcType: TYPE = PROC [window: Window.Handle, string: LONG STRING];
```

By attaching a **StringProcType** to **window**, the system converts keystrokes into strings, so the client can ignore the details of keys going up and down.

44.2 Constants and data objects

```
UserInput.caretRate: Process.Ticks;
```

```
UserInput.noSuchCharacter: CHARACTER = 377C;
```

noSuchCharacter is used by the TIP match process when a request is made to translate a key into a character and there is no translation..

```
UserInput.nullPeriodicNotify: UserInput.PeriodicNotifyHandle = ...;
```

Allows a client to initialize its handle to a well-known null value.

44.3 Signals and errors

```
UserInput.Error: ERROR [code: UserInput.ErrorCode];
```

```
UserInput.ErrorCode: TYPE = {
    windowAlreadyHasStringInOut, noStringInOutForWindow, noSuchPeriodicNotifier,
    other};
```

```
UserInput.ReturnToNotifier: ERROR [string: LONG STRING];
```

A client may be deep in the call stack of some Notifier-invoked operation from which it wishes to unwind. The **ERROR ReturnToNotifier** can be raised and will be caught at the top level of the TIP match process. Clients can catch this error, post a message with **string** in it, and let the error propagate up.

44.4 Procedures

```
UserInput.CancelPeriodicNotify: PROC [
  UserInput.PeriodicNotifyHandle] RETURNS [nil: UserInput.PeriodicNotifyHandle];
```

CancelPeriodicNotify stops the periodic notification passed in by removing the notification from Tajo's list of registered procedures and returns **nullPeriodicNotify**. This procedure raises **Error[noSuchPeriodicNotifier]** if the handle passed in is not valid. (Calling it with **nullPeriodicNotify** has no effect.)

```
UserInput. ClearInputFocusOnMatch: PROC [w: Window.Handle];
```

ClearInputFocusOnMatch clears the input focus in a window if that window has the input focus. This procedure is usually called by clients who are implementing their own window type when they are destroying a window.

```
UserInput.CreateIndirectStringIn: PROC [from, to: Window.Handle];
```

CreateIndirectStringIn redirects input from one subwindow to another. **Error[windowAlreadyHasStringIn]** will be raised if the **from** already has type-in.

```
UserInput.CreateIndirectStringInOut: PROC [from, to: Window.Handle];
```

CreateIndirectStringInOut redirects input and output from one subwindow to another. **Error[windowAlreadyHasStringInOut]** will be raised if the window already has type-in or type-out.

```
UserInput.CreateIndirectStringOut: PROC [from, to: Window.Handle];
```

CreateIndirectStringOut redirects output from one subwindow to another. **Error[windowAlreadyHasStringInOut]** will be raised if the window already has type-out

```
UserInput.CreatePeriodicNotify: PROC [
  proc: UserInput.PeriodicProcType, window: Window.Handle, rate: Process.Ticks]
  RETURNS [UserInput.PeriodicNotifyHandle];
```

CreatePeriodicNotify registers a periodic notification with Tajo. **proc** is called once every interval defined by **rate** as long as no TIP client notifications are taking place. If **rate** = 0, it runs once and then destroys itself. The **proc** has a parameter of type **Window.Handle**. When **proc** is called, it is passed the value of **window** used in the call of **CreatePeriodicNotify**, *not* of the window that currently contains the cursor or input focus.

```
UserInput.CreateStringInOut: PROC [
  window: Window.Handle, in, out: UserInput.StringProcType,
  caretProc: UserInput.CaretProcType ← UserInput.NopCaretProc];
```

CreateStringInOut attaches **StringProcType** procedures to the given window, allowing the client program to be unconcerned with the details of keyboard activity within the window. The **caretProc** is used to turn on and off a blinking caret at the type-in point when the input focus changes to and from **window**. **CreateStringInOut** must be called before **UserInput.SetStringInOut**. It is usually not called directly by clients, but is called as a side effect of creating a window of a type that accepts type-in(-out).

```
UserInput.DestroyIndirectStringIn: PROC [Window.Handle];
```

DestroyIndirectStringIn halts redirection of input from the subwindow.

```
UserInput.DestroyIndirectStringInOut: PROC [Window.Handle];
```

DestroyIndirectStringInOut halts redirection of input and output.

```
UserInput.DestroyIndirectStringOut: PROC [Window.Handle];
```

DestroyIndirectStringOut halts redirection of output from the subwindow.

```
UserInput.DestroyStringInOut: PROC [Window.Handle];
```

DestroyStringInOut removes the procedures supplied for dealing with input and output in the subwindow.

```
UserInput. FocusTakesInput: PROC RETURNS [BOOLEAN];
```

FocusTakesInput returns **TRUE** if the current input focus accepts input, **FALSE** otherwise.

```
UserInput.GetDefaultWindow: PROC RETURNS [Window.Handle];
```

GetDefaultWindow procedure returns the current default window.

```
UserInput. GetInputFocus: PROC RETURNS [Window.Handle];
```

GetInputFocus returns the window that currently has the input focus.

```
UserInput.NopCaretProc: UserInput.CaretProcType;
```

NopCaretProc does nothing when called.

```
UserInput.NopStringProc: UserInput.StringProcType;
```

NopStringProc does nothing when called.

```
UserInput.ResetUserAbort: PROC [Window.Handle];
```

ResetUserAbort sets the state of the window to appear that the user has not aborted its operation.

```
UserInput.SetAttention: PROC [ Window.Handle, attention: UserInput.AttentionProcType]
```

SetAttention sets the attention procedure for the window. The procedure **attention** is called asynchronously whenever the **USERABORT** key is pressed. If no attention proc is available for the window, **UserAbort** is set for that window.

```
SetInputFocus: PROC [ w: Window.Handle, notify: PROC [Window.Handle, LONG POINTER],
takesInput: BOOLEAN, data: LONG POINTER ← NIL];
```

SetInputFocus should be called by your **TIP.NotifyProc** when you want to set the input focus. It makes **w** the target of type-in. If **w** allows type-in, then **takesInput** should be set to **TRUE**; otherwise, **takesInput** should be set to **FALSE**. **notify** is called when **w** loses the input focus. It is passed **data** as the value of its **LONG POINTER** parameter.

```
UserInput.SetStringIn: PROC [
  window: Window.Handle, proc: UserInput.StringProcType]
  RETURNS [old: UserInput.StringProcType]
```

SetStringIn alters the procedure to be called for a window with existing type-in. **Error[noStringInOutForWindow]** can be raised if the window has no type-in.

```
UserInput.SetStringOut: PROC [
  window: Window.Handle, proc: UserInput.StringProcType]
  RETURNS [old: UserInput.StringProcType]
```

SetStringOut alters the procedure to be called for a window with existing type-out. **Error[noStringInOutForWindow]** can be raised if the window has no type-out.

```
UserInput.SetUserAbort: PROC [Window.Handle]
```

SetUserAbort sets the state of the window to appear that the user has aborted its operation. It does not call the window's attention procedure, even if there is one.

```
UserInput.StringOut: PROC [window: Window.Handle, string: LONG STRING]
```

StringOut allows a client to output directly to a window, bypassing any input filtering that might have been performed.

```
UserInput.StuffCharacter: PROC [
  window: Window.Handle, char: CHARACTER] RETURNS [BOOLEAN]
```

StuffCharacter allows a client to drive the type-in mechanism as though a character were coming from the user. The returned **BOOLEAN** is **TRUE** only if **window** was prepared to accept input.

```
UserInput.StuffCurrentSelection: PROC [Window.Handle] RETURNS [BOOLEAN]
```

StuffCurrentSelection allows a client to drive the type-in mechanism as though the contents of the current selection were coming from the user. (See the **Selection** interface for a description of the current selection.) The returned **BOOLEAN** is **TRUE** only if **window** was prepared to accept input.

```
UserInput.StuffString: PROC [window: Window.Handle, string: LONG STRING] RETURNS [BOOLEAN]
```

StuffString allows a client to drive the type-in mechanism as though **string** were coming from the user. The returned **BOOLEAN** is **TRUE** only if **window** was prepared to accept input.

```
UserInput.StuffTrashBin: PROC [Window.Handle] RETURNS [BOOLEAN]
```

StuffTrashBin allows a client to drive the type-in mechanism as though the user had typed in the exact contents of the last deletion. (See the **Selection** interface for a description of the trash bin.) The returned **BOOLEAN** is **TRUE** only if **window** was prepared to accept input.

```
UserInput.UserAbort: PROC [Window.Handle] RETURNS [BOOLEAN];
```

A client operation that runs for more than a few seconds can poll **UserAbort** on its window to see if you have indicated that you want to abort the operation in that window. If **window** is **NIL**, the **UserInput** package checks to see whether you have done a global abort. When the TIP match process calls a client, this flag is cleared. (See the *XDE User's Guide* for the abort procedure.) If there is an attention procedure for the window, **UserAbort** is not set automatically.

```
UserInput.WaitForConfirmation: PROC RETURNS [place: Window.Place, okay: BOOLEAN];
```

Before calling this procedure, the client should call **Cursor.Set[mouseRed]**. **WaitForConfirmation** then gets the confirmation from the user. If **okay** = **TRUE**, then the user pushed the point button; otherwise the user pushed either the menu or the adjust

buttons. **place** is the bitmap-relative position of the cursor when the button went down. The cursor should be set back to its previous type upon return from this procedure. This procedure does the equivalent of **TIP.NewManager[NIL, NIL, NIL]** as a side effect.

```
UserInput.WaitNoButtons: PROC;
```

WaitNoButtons returns when all the mouse buttons are released. This procedure does the equivalent of **TIP.NewManager[NIL, NIL, NIL]** as a side effect.

44.5 Examples

The following example shows a periodic notifier updating a display of the volume page count in a tool. The page count is updated every 20 seconds if the Notifier is not otherwise occupied.

```
window: Window.Handle ← ...; -- this tool's window

pageNotifier: UserInput.PeriodicNotifyHandle ← UserInput.nullPeriodicNotify;

Cleanup: ToolWindow.TransitionProcType = { -- the Transition Proc for window
    IF old = active THEN
        pageNotifier ← UserInput.CancelPeriodicNotify[pageNotifier];
    IF new = active THEN
        pageNotifier ← UserInput.CreatePeriodicNotify[
            proc: UpdatePageCount, window: window,
            rate: Process.MsecToTicks[20000]]};

UpdatePageCount: UserInput.PeriodicProcType = {
    -- code to update page count on screen;
    -- this procedure probably ignores both input parameters
    };
```

The following example shows a *kamikaze* periodic notifier, one whose only purpose is to let a procedure be executed from the Notifier process, such as booting another volume. Rather than executing the procedure at regular intervals, it is executed once and then the periodic notifier is destroyed.

```
kamikaze: UserInput.PeriodicNotifyHandle ← UserInput.nullPeriodicNotify;

RunProc: UserInput.PeriodicProcType = {
    kamikaze ← UserInput.UserInput.nullPeriodicNotify;};

    -- main body
    -- call procedure that must be run from Notifier;
    IF kamikaze = UserInput.nullPeriodicNotify THEN
        kamikaze ← UserInput.CreatePeriodicNotify[
            proc: RunProc, window: NIL,
            rate: 0]; -- rate of 0 means only execute once
```

V

File management

File management interfaces support loading, storing, and transfering files among local and remote disks or other storage media. These interfaces hide the details of the various types of storage hardware from the software, thus presenting a uniform surface to the tools that must interact with these media.

If you are less experienced with the XDE, it would be helpful to study the **MStream**, **MFile**, and **FileTransfer** interfaces before the others. Also, the **FileName** interface, though not as important by itself, is used by other File Management interfaces, so you should familiarize yourself with it.

The other file management interfaces allow the more advanced XDE programmers to exercise more control over the specifics of data access.

V.1 Overview

The XDE file system views processes as cooperative, allowing sophisticated file sharing among independent processes. If one process wishes to use a file in a way that conflicts with the way a second process is using it, the process that is using the file may be asked to relinquish it. For example, if a process wants to write a file being read by another process, the process reading the file is asked to stop. In addition, a process may ask to be notified when a file becomes available for a particular use. The processes that share files need neither communicate explicitly nor know one another's identities.

The XDE file system facilitates cooperation among processes by asking clients to provide procedures that the file system can call to ask a client to give up a file (**PleaseReleaseProc**) or tell a client that a file is available (**NotifyProc**). Such procedures are called *call-back procedures* because the file system uses them to call back the client at its discretion.

The next three sections of this chapter describe the file system procedures clients use for accessing and sharing files. The File access section discusses how a client gains access to a file and how it can respond if the file system asks it to give up a file. The Notification section discusses the mechanism by which a client might ask to be notified that a file is available for access. The Append files section discusses a controlled type of file access that lets clients read and write the same file at the same time.

The Examples section contains three examples of the file system's cooperative features. The section on Concurrency problems discusses the subtleties of writing the call-back procedures that clients provide, and the Implementation section discusses the implementation of this file system.

V.2 File access

To use a file, a client must have a handle on the file that identifies it in other calls to the file system. To obtain a handle, the client calls the procedure **Acquire** (see Example V.1).When a client finishes with a file, it must release its handle and relinquish its use by calling **Release**. The access parameter indicates how the file is to be used. **Anchor access** is used both to keep a file from being deleted or renamed and to change some file properties. **readOnly**, **writeOnly**, **readWrite**, **delete**, and **rename** are self-evident; **append** will be discussed below. **null** is used only for client initialization, not to acquire a file. The **release** parameter, used for asking the client to relinquish its use of the file, is discussed in the following paragraphs.

ExampleV.1: Procedures for acquiring and releasing files

```
Acquire: PROCEDURE [
   name: LONG STRING,
   access: Access,
   release: ReleaseData,
   mightWrite: BOOLEAN,
   initialLength: LONG CARDINAL,
   type:Type]

RETURNS [Handle];

Release: PROCEDURE [file: Handle];

Access: TYPE = {anchor, readOnly, readWrite, writeOnly, append, delete, rename,
null};
Type: TYPE = {unknown, text, binary};
```

The file system checks that the requested use of the file does not conflict with its other current uses. If there is no conflict, the file system asks each of the current owners of a conflicting handle on that file to release its handle, by calling the **PleaseReleaseProc** associated with the conflicting handle (see Example V.2). If all the clients with conflicting declarationshandles release them, the request is honored and the new use is granted. Otherwise, the request is denied.

Example V.2: **PleaseReleaseProc** declarations

```
PleaseReleaseProc: TYPE = PROCEDURE [file: Handle, instanceData: LONG POINTER]
   RETURNS [ReleaseChoice];

ReleaseData: TYPE = RECORD [
   proc: PleaseReleaseProc,
   clientInstanceData: LONG POINTER];

ReleaseChoice: TYPE = {later, no, goAhead, allowRename};
```

When a client's **PleaseReleaseProc** is called, the client can do one of four things. It can refuse to relinquish its use of the file, returning the value **no**, in which case the conflicting request cannot be honored. If it returns **allowRename**, it refuses to relinquish its use of the file, but allows the file to be renamed. If it returns **goAhead**, it relinquishes the file, and the file system releases its handle. (It is a client error to use this handle thereafter.) If it returns **later**, it promises to release the file soon, so the file system should delay processing the new request until that handle has been released.

Let us consider how the different return values might be used. If a client's **PleaseReleaseProc** were called in the middle of writing a file, it returns **no**. A file-cache client returns the value **goAhead** after it removes the requested file from the cache. If a client notes that it is already in the process of releasing a file when its **PleaseReleaseProc** is called, it returns **later**.

If a client is concerned only with a file's contents, not with its name, it returns **allowRename** from its **PleaseReleaseProc**. A loader is such a client; it does not want the contents of a loaded program to change, but does not care if the program is renamed.

A client can acquire a new handle on a file for each use of it. Alternatively, a client can change the use associated with a given handle by calling the procedure **SetAccess** (see Example V.3). For instance, a client can acquire a file with **readOnly** access and change the access to **readWrite** only when it determines that it must write into the file.

Example V.3: **SetAccess** declarations

```
SetAccess: PROCEDURE [file: Handle, access: Access];
```

SetAccess provides a quicker way to release and reacquire a file with a new access. In particular, **PleaseReleaseProcs** are called if required to obtain the new access, and file notification takes place if appropriate (see the section on Notification)

V.3 Notification

A client can ask the file system to notify it whenever a file (or class of files) becomes available for some particular access. For example, when a client is denied access to a file, it might want to be awakened when that file is available so it can try again.

AddNotifyProc is called to register such a request with the file system, and the procedure **RemoveNotifyProc** is called to remove it (see Example V.4). The filter parameter determines the class of files of interest. The name field of the filter is a pattern to be matched against file names. Patterns can include wildcard characters that match zero or more characters in a file name. The type field of the filter is the type of the files that the client is interested in; if type is unknown, all types match. The access field of the filter ensures that the client is notified only when a file with the needed access becomes available, such as when a file that was being written becomes available for reading.

Example V.4: **NotifyProc** declarations

```
AddNotifyProc: PROCEDURE [
  proc: NotifyProc, filter: Filter, clientInstanceData: LONG POINTER];

  RemoveNotifyProc: PROCEDURE [
    proc: NotifyProc, filter: Filter, clientInstanceData: LONG POINTER];

  Filter: TYPE = RECORD[name: LONG STRING, type: Type, access: Access];

  NotifyProc: TYPE = PROCEDURE [name: LONG STRING, file: Handle,
  clientInstanceData: LONG POINTER]
    RETURNS [removeNotifyProc: BOOLEAN];
```

When the file system determines that the conditions of a filter have been satisfied, it calls the **NotifyProc** passed in with the filter. The **name** parameter is the name of the file; **file** is a handle on the file; and **clientInstanceData** is the value passed to the file system when **AddNotifyProc** was called. The **NotifyProc** returns **TRUE** when it wishes to be removed from the file system's notification list.

Because a client can acquire a file for a conflicting access before other interested clients have been notified that the file is available for some weaker access, there is no guarantee that a client will be called for every state change of a file. For instance, clients to be notified that a file is available for **readOnly** access will not be notified if another client acquires the file for **readWrite** access in the interim. When a client is notified, however, it is guaranteed that it can acquire the file for its desired access.

V.4 Append files

A client may request append access to a file for typescript applications in which a file can be concurrently read and written. In such an application, the file can be divided into two parts: an unchangeable initial portion and a final portion that may still be changed. The read length of the file divides these two sections.

A client with append access to a file may change either the contents of the final portion of the file or its size. This client is also responsible for setting the read length of the file when it has finished writing a new section of the file. The read length may never decrease.

The file system always honors requests to read a file for which another client has append access. It will appear to the reader, however, that the file is only as long as its read length at the time it was acquired. To encourage a client with append access to let the reader read as much as possible, the file system will call the **PleaseReleaseProc** of the append client, ignoring the result returned. This allows the append client to set the read length from the **PleaseReleaseProc**, which may have been called because some client is trying to read the append file.

Append files are particularly useful for applications in which a client is continually adding information to the end of a file, but another client needs to read the current contents. For instance, a command executive program may write a typescript of commands typed by the user together with their output. One of the executive's commands may store a file. It is useful to store a copy of the typescript file itself. If the typescript file is an append file and the command executive sets the read length to the

end of the output from the previous command, the executive can store the contents of its own typescript file up to the point where the command was issued.

The next section gives another example of append files.

V.5 Examples

The Xerox Development Environment uses the file system's cooperative features to solve several problems that can be quite awkward otherwise, such as those involved in dealing with windows that display files, file managers, and append files for processing data.

V.5.1 File windows

While a file is loaded in a window so that you can read or edit it, the file window program has a handle on it. Some other client may need to write into that file. For instance, you may load the compiler error log into a window to look at your compilation errors while you edit your source file. If, after finishing the edits, you recompile the source file without unloading the compiler log, the compiler will need to write into the log if it encounters additional errors.

If you are not editing the file in a file window, the file window program will unload the window and relinquish ownership of the file. When the **PleaseReleaseProc** is called for the file window's file handle, the file window program checks the state of the window. If the file is being edited, it refuses to release the handle. Otherwise, it unloads the window, registers a **NotifyProc** for read access on the file, and relinquishes ownership of the file.

When the client that was writing the file completes and releases its handle, the file system notices that read access has become available on the file. Since it can satisfy the file window's notification request, it calls the file window's **NotifyProc**. The file window program acquires the file for read once more and reloads it into the window. Hence, a client will not be blocked if a file that it needs has been left loaded in a window; file windows automatically update themselves to the most recent version of whichever files they contain.

V.5.2 File managers

Some clients cache file handles that are expected to be needed again, some of which may be in use. This saves looking up the file in the file system each time it is needed and remapping the file contents into memory.

When another client needs a file maintained by the file manager, the **PleaseReleaseProc** for the file manager checks its reference counts to see if the file is in use. If not, the file manager clears the file from its set of file handles and relinquishes ownership.

Hence, the **PleaseReleaseProc** facility allows a client to gain the performance advantages of a file cache without interfering with other clients that need to use the files in the cache.

V.5.3 Append file processing

Consider a data analysis system in which one process continuously gathers data that several other processes analyze as the data is gathered. These processes can be coordinated straightforwardly using an append file.

As data comes in, the data-gathering process appends it to an append file and sets the new read length for the file. Setting the read length causes file notification to take place for read access.

The analysis processes have a **NotifyProc** on the append file. When the file is extended with new information, the **NotifyProc** is called. The **NotifyProc** performs a broadcast on a condition variable to awaken the analysis processes blocked waiting for data. Each analysis process loops, checking to see whether there is more data by comparing the current read length of the file with the last length processed. If there is no new data, the process again waits on the condition variable. If there is more data, the process acquires the append file for read access and processes the data starting from where it last left off, continuing to the end of the file.

Note that the data-gathering process does not need to know the identity of or even the number of analysis processes. It simply provides a service to whatever clients may be interested. Analysis processes can be added or removed dynamically without affecting other processes.

Only one copy of the data need be produced, since it can be freely shared among the analysis processes. The analysis processes can read any available data at any time, not just the previously unseen data.

V.6 Concurrency problems in writing call-back procedures

Writing call-back procedures correctly is often difficult because the client must be prepared to have its call-back procedures invoked at any time. Although clients that use **PleaseReleaseProcs** and **NotifyProcs** may appear to be simple sequential programs, subtle synchronization issues are involved in the interprocess communication between the client, the file system, and (indirectly) other clients.

The difficulties are inherent in writing multi-process programs. As the means of communication, the call-back procedures expose these difficulties. Note that clients need not master the subtleties of call-back procedures to use the file system. They can choose instead not to cooperate in their use of files, using a system-provided **PleaseReleaseProc** that always returns **no**. Often, tools are first written with little or no cooperation and gradually evolve to allow more. The rest of this section discusses the difficulties in writing call-back procedures.

As an example of the type of locking that the client must do, the client monitors data accessed by its **PleaseReleaseProc** and carefully synchronizes which process has released the file. To see how this might be done, consider the code fragments in Example V.5. In this example, the **PleaseReleaseProc** returns **later** if the client is done with the file and is in the process of releasing it. Otherwise, it will return **no**. The state of the file, **state**, is

always changed by the client and examined by the **PleaseReleaseProc** from within the client monitor.

Example V.5: Example **PleaseReleaseProc**

```
FileState: TYPE = {busy, beingReleased, released};
file: Handle;
state: FileState ← released;

ChangeState: ENTRY PROCEDURE [newState: FileState] =
  BEGIN
  state ← newState;
  END;

-- PleaseReleaseProc for file

MyReleaseProc: ENTRY ReleaseProc =
  BEGIN
  SELECT state FROM
    busy = > RETURN[no];
    beingRelease, released = > RETURN[later];
  ENDCASE;
  END;

-- code to acquire file

ChangeState[busy];
file ← MFile.Acquire["FileName", readWrite, []];

-- code to release file when done

ChangeState [ beingReleased];
MFile.Release[file];
ChangeState[ released];
```

This is an extremely simple **PleaseReleaseProc**. The only difference between providing it and none at all is that later will be returned during the small interval after the client has decided to release the file but before that operation is complete. If some other client requests the file in that interval, that second client will succeed when otherwise it would not.

Because many clients may be calling it simultaneously, the file system must lock some of its internal data structures while it calls the client-provided **PleaseReleaseProc** or **NotifyProc**. Although this lock is essential for preserving the consistency of data structures and behavior, it means that some file system operations cannot be invoked from a **PleaseReleaseProc** or **NotifyProc** without causing deadlock.

As an example of the type of locking that the file system must do, the file system must guarantee that once a client has released a file, the file system will not call the associated **PleaseReleaseProc**. Thus, while the file system is calling the **PleaseReleaseProc** for a file, it blocks all attempts to call **Release** on that file. This blocking guards against the case in which the call on the **PleaseReleaseProc** is blocked on a client monitor while the client has called **Release** on that file. If the file system executes the **Release** before the **PleaseReleaseProc** completes, it will appear to the client

that the **PleaseReleaseProc** was called after the **Release** completed, as seen in Example V.6.

Example V.6: Race condition if file system permitted **Release** to execute while calling a **PleaseReleaseProc.**

Client	File System
1. enter monitor to release file	
	2. call **PleaseReleaseProc** (blocks on client monitor)
3. call **Release**	
	4. process call of **Release** and return to client
5. leave monitor	
6. process call of **PleaseReleaseProc**	
	7. call from 2 completes

A **PleaseReleaseProc** should not wait for a monitor that may be held by a process waiting for the file system. In Example V.5, it is important that the actual release of the file was done outside the client monitor. Instead, only the state change of the file is protected by the monitor, and **Release** is called from outside the monitor. Otherwise, the deadlock sequence in Example V.7 might occur.

Example V.7: Client-caused deadlock in **PleaseReleaseProc**

Client 1	Client 2	File System
1. enter monitor to release file X		
	2. call Acquire on file X	
		3. lock data structure for file X
		4. call Client1's **PleaseReleaseProc** for file X (blocks on Client 1's monitor)
5. call **Release** (blocks on file system's lock on file X'sdata structure)		

Some of the file system procedures may not be called from within a **PleaseReleaseProc**; these include **Acquire**, **Release**, or **SetAccess**. If the **PleaseReleaseProc** calls one of these procedures, the process will deadlock on the file system's monitor for that file. If it must call one of these procedures, it must fork another process to perform the call and not wait for that process to complete, since the process will not complete until the **PleaseReleaseProc** returns. The return value **later** from a **PleaseReleaseProc** may indicate that a process has been forked that will release the file.

Writing **PleaseReleaseProcs** and **NotifyProcs** requires very careful thought and attention as well as a good understanding of the principles of multi-process programs.

V.7 Interface abstracts

FileName provides facilities for parsing local and remote file names.

FileTransfer provides a uniform interface for the manipulation of files. It makes invisible to the client whether the files are in the local file system or on a remote file server. It provides facilities for copying files, opening streams on files, and enumerating files.

MFileProperty defines a list of registered client-defined file properties for files in the development environment file system.

MLoader provides the facilities for loading and running programs stored in files in the development environment file system.

MSegment maps files in the development environment file system into memory.

MStream creates streams on local files. The facilities of Pilot's **Stream** interface as well as the operations in **MStream** are used to manipulate the streams provided by this interface.

45

FileName

The FileName interface provides a general data structure and procedures for dealing with file names, whether remote or local. This allows clients and interfaces to communicate through a standard representation of file names. The **FileTransfer** interface, for example, takes a **FileName.VFN** as a parameter to all of its procedures that operate on files.

45.1 Types

```
FileName.VirtualFilename, VFN: TYPE = LONG POINTER TO VirtualFilenameObject;

FileName.VirtualFilenameObject: TYPE = RECORD [
  host, directory, name, version: LONG STRING];
```

45.2 Constants and data objects

None.

45.3 Signals and errors

```
FileName.Error: SIGNAL
```

Error is raised by **AllocVFN** and **UnpackFilename,** indicating that the client provided an invalid file name. A file name has the following syntax, with all fields optional:

[*host*]$dir_1/dir_2/.../dir_n/filename!version$

It is also raised by **GetRemoteName** and **SetRemoteName** when certain string lengths are exceeded.

45.4 Procedures

```
FileName.AllocVFN: PROCEDURE [LONG STRING]
  RETURNS [FileName.VirtualFilename];
```

The **AllocVFN** procedure allocates a new **VirtualFilenameObject** and parses its parameter into a **VirtualFilename**. The strings in the object are allocated from the system heap; they

are part of the object and clients are free to replace them. The object itself is not allocated from the system heap and must be deallocated by **FreeVFN**. (See examples at the end of the chapter.) Note that a client is free to allocate its own **VirtualFilenameObject** from someplace other than the system heap (such as its private heap, its local frame, or its global frame). However, the strings in the **VirtualFilenameObject** must be allocated from the system heap so that **FileName** can change their sizes as necessary. This procedure can raise **Error** if the file name provided cannot be parsed.

```
FileName.FreeFilename: PROCEDURE [s: LONG STRING];
```

The **FreeFilename** procedure frees a string allocated with **PackFilename**.

```
FileName.FreeVFN: PROCEDURE [FileName.VirtualFilename];
```

The **FreeVFN** procedure frees a **VirtualFilenameObject**. It also frees its component strings to the system heap. The **VirtualFilenameObject** must have been allocated by **AllocVFN**.

```
FileName.GetRemoteName: PROCEDURE [file: MFile.Handle, remoteName: LONG STRING];
```

The **GetRemoteName** procedure copies the remote name associated with **file** into the parameter **remoteName**. If **remoteName** is not long enough to hold the complete name, **Error** is raised.

```
FileName.NormalizeVFN: PROCEDURE [vfn: FileName.VirtualFileName];
```

The **NormalizeVFN** procedure reparses the information in the **VirtualFilename** so that all host information is in the host field, all directory information is in the directory field, and so forth. All strings in **vfn** must be allocated from the system heap, since **NormalizeVFN** may return them to the system heap while reparsing the information.

```
FileName.PackFilename: PROCEDURE [
  vfn: FileName.VirtualFileName, h, d, n, v: BOOLEAN ← FALSE]
  RETURNS [s: LONG STRING];
```

The **PackFilename** procedure converts the information in selected fields of a **VirtualFilename** into a string, adding appropriate delimiters when necessary. **h**, **d**, **n**, and **v** indicate whether the host, directory, name, and version fields, respectively, are to be included in the string returned. Hosts are delimited by [], directories are terminated by > or /, and versions are preceded by !. If no version appears in **vfn**, enough room is left in **s** for a version at least six characters long. "<" receives no special treatment but is considered a normal character in a file name field. **s** is allocated from the system heap; it must be freed by the client with **FreeFilename** .

```
FileName.ResetVFN: PROCEDURE [
  vfn: FileName.VirtualFileName, h, d, n, v: BOOLEAN ← FALSE];
```

The **ResetVFN** procedure resets selected fields of a **VirtualFilename** to **NIL**, freeing the associated storage to the system heap. **h**, **d**, **n** and **v** indicate whether the host, directory, name, and version fields, respectively, are to be reset.

```
FileName.SetRemoteName: PROCEDURE [file: MFile.Handle, remoteName: LONG STRING];
```

The **SetRemoteName** procedure sets the remote name property of **file** to be **remoteName**. If the length of **remoteName** exceeds 150, **Error** is raised.

```
FileName.UnpackFilename: PROCEDURE [
  s: LONG STRING, vfn: FileName.VirtualFileName];
```

The **UnpackFilename** procedure parses a string into a **VirtualFilename**. If a directory is present in **vfn** and the directory within **s** does not begin with <, then the directory from **s** is appended. Otherwise, the directory is overwritten. **UnpackFilename** creates **VirtualFilenames** that no longer have a final > on the directory string. This procedure raises **Error** if the file name, **s**, cannot be parsed. See examples below.

45.5 Examples

This example describes how file names are parsed by **AllocVFN** and **UnpackFilename**. These procedures differ only in that **AllocVFN** first allocates a **VFN** before unpacking. The string

```
s = "[Server]AlphaMesa/Defs/FileName.mesa!2
```

is unpacked into

```
host:       server
directory:  AlphaMesa/Defs
name:       FileName.mesa
version:    2
```

If **s** = `doc/New.doc` and **vfn.directory** = `emerson` with the remaining fields NIL, **s** is unpacked into

```
host:       NIL
directory:  emerson/doc
name:       New.doc
version:    NIL
```

Note: FileName performs only minimal error checking, forcing the client to pass in properly formatted file names.

46

FileTransfer

The **FileTransfer** interface provides a uniform interface for manipulating files, whether they are local (and in the Xerox Development Environment file system) or remote. It provides facilities for copying files, opening streams on files, and enumerating files. It insulates the client from the network; in particular, **FileTransfer** does not give up if a connection to a file server cannot be opened on the first attempt. Examples of the use of **FileTransfer** are given at the end of the chapter.

46.1 Types

```
FileTransfer.CheckAbortProc: TYPE = PROCEDURE [clientData: LONG POINTER]
  RETURNS [abort: BOOLEAN];
```

The **CheckAbortProc** of a **Connection** is called at intervals to see whether the user has aborted an operation. The client may also attach some instance data to a **Connection** that is passed back to **CheckAbortProc**. When a **CheckAbortProc** returns **TRUE**, the error **ABORTED** is raised.

```
FileTransfer.ClientProc: TYPE = PROCEDURE [clientData: LONG POINTER];
```

Clients can specify **ClientProcs** for doing logins and giving an indication of progress. These procedures are passed the **clientData** associated with the corresponding **Connection**.

```
FileTransfer.Confirmation: TYPE = MACHINE DEPENDENT {
  do(0), skip, abort, firstPrivate(8), null(255)};
```

A **Confirmation** is returned from a **VetoProc** to give the client fine control over certain operations. **do** means that **FileTransfer** should perform the operation. **skip** means that the current file operation should not be performed, but that **FileTransfer** should proceed to the next file operation in this command. **abort** means that this and all succeeding operations should not be performed. **skip** and **abort** are identical for procedure calls involving single files. All other **Confirmation** values, including **firstPrivate** and **null**, act like **skip**.

```
FileTransfer.Connection: TYPE = LONG POINTER TO ConnectionObject;
```

FileTransfer requires a **Connection** for most operations. A **Connection** contains only state information used by **FileTransfer**. Large amounts of system resources are used during a

remote operation or until the **Close** procedure is called after a remote operation. A **ConnectionObject** contains private data and must be monitored by the client if it is to be accessed by multiple processes simultaneously.

```
FileTransfer.ConnectionObject: TYPE = ... ;

FileTransfer.DesiredProperties: TYPE = PACKED ARRAY ValidProperties OF BOOLEAN ← ALL[FALSE];
```

DesiredProperties is the type of property array passed to **SetDesiredProperties** and returned from **GetDesiredProperties**.

```
FileTransfer.FileInfo: TYPE = LONG POINTER TO FileInfoObject;

FileTransfer.FileInfoObject: TYPE = RECORD [
  host, directory, body, version, author: LONG STRING ← NIL,
  create, read, write: Time.Packed ← System.gmtEpoch,
  size: LONG CARDINAL ← 0,
  type: FileType ← unknown,
  oldFile: BOOLEAN ← TRUE,
  readProtect: BOOLEAN ← FALSE,
  ...];
```

A **FileInfoObject** contains information about a local or remote file. **host**, **directory**, **body**, and **version** are the pieces of the file name. **author** is the name of the user that created the file. **create**, **read**, and **write** are the times that the file was created, last read, and last written. **size** is the size of the file in bytes. **type** is the type of the file. **oldFile** is **TRUE** if and only if the file exists; if **oldFile** is **FALSE**, information other than the file name is undefined. **readprotect** is **TRUE** if and only if the file exists and is read-protected. Only those fields indicated by **FileTransfer.SetDesiredProperties** will be valid. Initially, all fields are valid.

```
FileTransfer.FileType: TYPE = {unknown, text, binary, directory, null};

FileTransfer.InfoProc: TYPE = PROCEDURE [FileTransfer.Connection]
  RETURNS [source, target: FileTransfer.FileInfo];
```

The **InfoProc** is used inside a **VetoProc** to obtain information about files if it is needed for deciding whether to veto the operation.

```
FileTransfer.ListProc: TYPE = PROCEDURE [
  conn: FileTransfer.Connection, clientData: LONG POINTER, file: LONG STRING,
  post: FileTransfer.MessageProc, info: FileTransfer.InfoProc]
  RETURNS [FileTransfer.Confirmation];
```

The **ListProc** is called for each file in an enumeration. The **ListProc** returns a **Confirmation**; A confirmation of other than **do** aborts the enumeration. The parameters **post** and **clientData** can be used by the **ListProc** for output. The **InfoProc** can be called by the **ListProc** to obtain information about the file if more than the file name is needed. The parameter **file** contains the fully qualified name of the file, including the directory, name, and version number.

```
FileTransfer.MessageProc: TYPE = PROCEDURE [
  clientData: LONG POINTER, level: Severity, s1, s2, s3, s4: LONG STRING ← NIL];
```

The **MessageProc** is used by **FileTransfer** for feedback. It is called to notify the user of errors and of details of the operations taking place. **level** indicates the importance of a message; it can be used by the **MessageProc** to filter out undesired feedback.

```
FileTransfer.ServerType: TYPE = MACHINE DEPENDENT{unknown(0), local, IFS, tenex, ns,
null(7)};
```

ServerType is the type of a host; it is defaulted to **local** when a connection is created. When a remote operation is to be performed, the **ServerType** defaults to **Profile.defaultFileServerProtocol**. The values **IFS** and **Tenex** correspond to protocols no longer supported; they should not be used. Support for **ServerType** will be dropped in a future release.

local	the machine on which the program is running.
IFS	an interim file server.
tenex	a machine running tenex.
ns	a product file server.

```
FileTransfer.Severity: TYPE = {verbose, terse, warning, fatal};
```

Severity indicates the urgency of a message sent to a **MessageProc**. **verbose** is the least important information, and **fatal** is the most important information.

```
FileTransfer.StreamType: TYPE = {remote, local, temporary};
```

When a client creates a stream on a remote file, it may also supply information about the way the stream is to be accessed; the **StreamType** is ignored for local files.

remote	the client intends to read the stream quickly (fast enough so that the file server does not time out) and will not position the stream.
local	works the same as if the client had done a **Copy**, then opened a stream; all local stream operations are valid on such streams.
temporary	is the same as **local**, except that a temporary local file is created; the file is deleted when the stream is destroyed.

```
FileTransfer.ValidProperties: TYPE = {host, directory, body, version, author, size, type,
oldFile, readProtect};
```

ValidProperties are the properties of a file that the client can know about through **FileTransfer**, that is, the fields of the **FileInfoObject**.

```
FileTransfer.VetoProc: TYPE = PROCEDURE [
  conn: FileTransfer.Connection, clientData: LONG POINTER,
  post: FileTransfer.MessageProc, info: FileTransfer.InfoProc,
  showingDates: BOOLEAN]
  RETURNS [confirm: FileTransfer.Confirmation, showDates: BOOLEAN];
```

A **VetoProc** is used by the operations **Copy**, **Delete**, **ReadStream**, and **StoreStream** to give the client fine control over these operations. The return value **confirm** tells **FileTransfer** how to proceed. The return value **showDates** indicates whether the date should be included in the message output to the **MessageProc** on succeeding files. **showDates** is ignored on a call to **Delete**. If the **VetoProc** wants to send an output, it can call **post** with **clientData** and the message. The client **VetoProc** can obtain information about the file(s) involved in the operation by calling **info**. If the **vetoProc** was called by **Copy**, **info** returns information about both the source and target file; if it was called by **Delete**, **ReadStream**, or **StoreStream**, target is **NIL** and information about the file is returned in **source**.

46.2 Constants and data objects

None.

46.3 Signals and errors

```
FileTransfer.Error: SIGNAL [conn: Connection, code: ErrorCode];

FileTransfer.ErrorCode: TYPE = MACHINE DEPENDENT {
  illegalParameters(0), invalidObject, notAStream, illegalLogin(4), IllegalConnect, skip,
  retry, cantModify, directoryFull, notFound, spare1, spare2, unknown(31)};
```

illegalParameters — the client provided illegal parameters to a call to **FileTransfer**.

invalidObject — the client provided a connection that was **NIL**, smashed, or has been freed.

notAStream — the client provided a stream that was **NIL**, smashed, freed, or not created by **FileTransfer**.

illegalLogin — the operation could not proceed because of illegal login credentials, and no login procedure was provided through **SetProcs**.

illegalConnect — the operation could not proceed because of illegal connect credentials.

skipOperation — the code **skipOperation** is raised whenever an operation fails and should not be retried; it probably means that user intervention is required to make the operation succeed. Details will have been reported by calls to the **MessageProc** supplied by **SetProcs** before this error is raised.

skipFile — the code **skipFile** is only raised when attempting a remote to remote **Copy**. For some reason, a particular file in the enumeration could not be accessed. In this case, the signal may be

resumed to coninue the enumeration. Details will have been reported by calls to the **MessageProc**.

retry — This code is raised from calls to **ReadNextStream** and any of the **Stream.Get** procedures on a **FileTransfer** stream when the connection to a remote file server has timed out; the **ReadStream** enumeration should be restarted.

cantModify — the operation could not get proper access to modify a file. If the operation involves wildcards, this error will not be raised until **FileTransfer** has attempted the operation on all files involved; it is only raised once.

directoryFull — the remote directory or local volume is full.

notFound — the file was not found.

accessDenied — either the current primary credentials are not sufficient for the operation to proceed, or secondary credentials are required.

unknown — the operation uncoverd an implementation error.

The codes **illegalParameters**, **invalidObject**, and **notAStream** are client errors. The **ErrorCodes** **illegalLogin**, **illegalConnect**, **skipOperation**, **skipFile**, **retry**, **cantModify**, **directoryFull**, **notFound**, and **accessDenied** are not normally client errors; however, they should be caught by client code. The client is also expected to catch the error **ABORTED**, which is raised if the **checkAbortProc** returns **true** when called by **FileTransfer**.

46.4 Procedures

FileTransfer insists that the client parse file names into a **FileName.VirtualFilename** (or **VFN**). The **Filename** interface provides procedures for converting between strings and **VirtualFilenames** (see the **FileName** chapter). All procedures in **FileTransfer** that manipulate files take **VirtualFilenames** as parameters. Any field (with the exception of **host**) contained in a **VirtualFilename** may contain wildcard characters. However, the interpretation of these characters is left entirely to the file system that contains the file.

```
FileTransfer.Close: PROCEDURE [FileTransfer.Connection];
```

The **Close** procedure frees any resources used to communicate with remote hosts; it does not destroy the **ConnectionObject**. It can raise **Error[invalidObject]**.

```
FileTransfer.CodeToString: PROCEDURE [FileTransfer.ErrorCode, LONG STRING];
```

The **CodeToString** procedure translates the code describing a **FileTransfer.Error** into a client-provided string. If the string is not long enough, **CodeToString** fills it in with as much information as will fit. If the client has provided a **MessageProc** by **SetProcs**, the error has already been reported by the **MessageProc**, and it may not be necessary to convert the **ErrorCode** to a message.

```
FileTransfer.Copy: PROCEDURE [
  sourceFile, destFile: FileName.VirtualFileName,
```

```
    sourceConn, destConn: FileTransfer.Connection ← NIL,
    veto: FileTransfer.VetoProc ← NIL, showDates: BOOLEAN ← FALSE];
```

The **Copy** procedure copies files. It will copy between any combination of remote and local **VirtualFilenames**. If the source **VFN** contains wildcard characters, a single invocation of **Copy** may copy several files. If the name field of **destFile** is **NIL**, the name portion of the source file will be used. Wildcards may not be used in **destFile**. If a client knows that a file is local (and hence no connection need be established with a file server), the connection parameter corresponding to that file may be **NIL**. The **showDates** parameter in **Copy** indicates whether **FileTransfer** should print the file creation date after the file name in its feedback messages that are sent to the connection's **MessageProc**. If the **VetoProc** is not **NIL**, it is called before each transfer operation to give the client closer control of which files are copied. This procedure can raise **Error[..., illegalParameters, invalidObject, illegalLogin, illegalConnect, notFound, directoryFull, spare1, skip, cantModify, retry, ...]**. The error **ABORTED** can also be raised.

```
FileTransfer.Create: PROCEDURE RETURNS [FileTransfer.Connection];
```

The **Create** procedure makes a new **ConnectionObject**.

```
FileTransfer.Delete: PROCEDURE [
    conn: FileTransfer.Connection, file: FileName.VirtualFileName,
    veto: FileTransfer.VetoProc ← NIL];
```

The **Delete** procedure can be used to delete files, either local or remote. If the file is remote and contains wildcard characters, several files may be deleted. If the **VetoProc** is not **NIL**, it is called before each delete operation to give the client closer control. This procedure can raise **Error[..., invalidObject, illegalParameters, illegalLogin, illegalConnect, cantModify, skip, notFound, spare1, ...]**. The error **ABORTED** can also be raised.

```
FileTransfer.Destroy: PROCEDURE [FileTransfer.Connection];
```

The **Destroy** procedure frees a **ConnectionObject**, closing the connection if it is open. This procedure can raise **Error[invalidObject]**. **Note**: Unpredictable results occur if **Destroy** is called from within a catch phrase on a call to any **FileTransfer** procedure for that same connection.

```
FileTransfer.Enumerate: PROCEDURE [
    conn: FileTransfer.Connection, files: FileName.VirtualFileName,
    proc: FileTransfer.ListProc];
```

The **Enumerate** procedure enumerates the files specified by **files**, calling **proc** for each file. **proc** returns a **Confirmation**; if the confirmation is something other than **do**, the enumeration stops. The file name information (the **host**, **directory**, **body**, and **version** fields) returned by the **InfoProc** passed to **proc** are filled in from the **files** parameter of **Enumerate**. Since this name information can contain wildcards, the **file** parameter passed to the **InfoProc** can be used to obtain the actual name of each enumerated file. This procedure can raise **Error[..., invalidObject, illegalLogin, illegalConnect, skip, notFound, spare1, ...]**. The error **ABORTED** can also be raised.

```
FileTransfer.GetDesiredProperties: PROCEDURE [FileTransfer.Connection]
  RETURNS [props: DesiredProperties];
```

The **GetDesiredProperties** gives the current list of valid properties returned in a **FileInfoObject**.

```
FileTransfer.GetProcs: PROCEDURE [conn: FileTransfer.Connection]
  RETURNS [clientData: LONG POINTER, messages: FileTransfer.MessageProc,
  login, noteProgress: FileTransfer.ClientProc];
```

The **GetProcs** procedure returns the values of the client-provided procedures associated with a **Connection**. This procedure can raise **Error[invalidObject]**.

```
FileTransfer.GetServerType: PROCEDURE [
  conn: FileTransfer.Connection, host: LONG STRING]
  RETURNS [FileTransfer.ServerType];
```

The **GetServerType** procedure returns the type of file server; if the host string is empty, the last host used with the **Connection** determines the type of server. This procedure can raise **Error[invalidObject]**.

```
FileTransfer.GetStreamInfo: PROCEDURE [remoteStream: Stream.Handle]
  RETURNS [FileTransfer.FileInfo];
```

The **GetStreamInfo** procedure returns information on the file behind a stream. This procedure can raise **Error[notAStream, skip]**.

```
FileTransfer.GetStreamName: PROCEDURE [remoteStream: Stream.Handle]
  RETURNS [file: LONG STRING];
```

The **GetStreamName** procedure returns the fully qualified name of the file behind a stream. For local streams, this is faster than **GetStreamInfo**. The string returned belongs to the implementation and should not be freed by the client. This procedure can raise **Error[notAStream]**.

```
FileTransfer.HighestVersion: PROCEDURE [
  conn: FileTransfer.Connection, remote: FileName.VirtualFileName]
  RETURNS [exists: BOOLEAN];
```

The **HighestVersion** procedure takes a **VirtualFileName** that refers to a remote file and updates the version field of the **VirtualFileName** to the highest version of that file existing on the remote file server. If there is no file by that name on the file server, it returns **FALSE**. This procedure can raise **Error[. . , illegalParameters, illegalLogin, illegalConnect, spare1, skip, ...]** and **ABORTED**.

```
FileTransfer.LocalVFN: PROCEDURE [
  conn: FileTransfer.Connection, vfn: FileName.VirtualFileName]
  RETURNS [BOOLEAN];
```

The **LocalVFN** procedure returns **TRUE** if the connection and **VirtualFilename** passed to it refer to a local file. If the host field of the **VirtualFilename** is empty, the last host used with

the **Connection** determines the location of a file. This procedure can raise **Error[invalidObject]**.

```
FileTransfer.LookUp: PROCEDURE [
  conn: FileTransfer.Connection, file: FileName.VirtualFileName]
  RETURNS [fileInfo: FileTransfer. FileInfo];
```

The **LookUp** procedure is not implemented.

```
FileTransfer.ReadNextStream: PROCEDURE [Stream.Handle] RETURNS [Stream.Handle];
```

Streams can be enumerated using the **ReadNextStream** procedure. The first stream is read using **ReadStream**. Successive streams can be obtained by calling **ReadNextStream** with the last stream from the enumeration. **ReadNextStream** returns **NIL** when there are no more streams. As a side effect, the stream passed in is deleted, so the client should not attempt the same. This procedure can raise **Error[... , notAStream, retry, ...]**. The error **ABORTED** can also be raised. (See the example of stream enumeration at the end of this chapter.)

```
FileTransfer.ReadStream: PROCEDURE [
  conn: FileTransfer.Connection, files: FileName.VirtualFileName,
  veto: FileTransfer.VetoProc ← NIL, showDates: BOOLEAN ← FALSE,
  type: FileTransfer.StreamType ← remote]
  RETURNS [Stream.Handle];
```

The **ReadStream** procedure opens a stream on a file, either local or remote. If the **VetoProc** is not **NIL**, it is called before the stream is obtained to permit the client closer control. The **VirtualFilename** passed **ReadStream** may contain wildcards. **ReadStream** returns a stream on the first file that matches **files**; **NIL** is returned if no matches are found. Successive streams can be obtained by calling **ReadNextStream** with the last stream from the enumeration. This procedure can raise **Error[... , notAStream, illegalLogin, illegalConnect, retry, skip, notFound, spare1, ...]**. The error **ABORTED** can also be raised. **Note**: In the case of an **ns** server, veto is called from a separate process, so the client must catch all signals raised from within **veto** with code inside the veto procedure, or they will not be caught.

```
FileTransfer.Rename: PROCEDURE [
  conn: FileTransfer.Connection, old, new: FileName.VirtualFileName];
```

The **Rename** procedure is used to rename a file on a single file system; the credentials associated with the connection must permit access to both **VirtualFilenames**. This procedure can raise **Error[... , invalidObject, illegalParameters, illegalLogin, illegalConnect, notFound, spare1, ...]** and **ABORTED**.

```
FileTransfer.SetDefaultServerType: PROCEDURE [
  conn: FileTransfer.Connection, type: FileTransfer.ServerType];
```

If **FileTransfer** is unable to determine the type of host, it uses a default type (as determined by **Profile.defaultFileServerProtocol**); the procedure **SetDefaultServerType** sets the default for a **Connection** to be **type**. This procedure can raise **Error[invalidObject]**.

```
FileTransfer.SetDesiredProperties: PROCEDURE [
  conn: FileTransfer.Connection, props: DesiredProperties];
```

On any succeeding calls to procedures that return **FileInfoObject**s, only properties with **TRUE** values indicated by **props** are valid .

```
FileTransfer.SetPrimaryCredentials: PROCEDURE [
  conn: FileTransfer.Connection, user, password: LONG STRING];
```

The **SetPrimaryCredentials** procedure sets the primary credentials to be used for a **Connection**. If no primary credentials have been supplied, **FileTransfer** uses the user name and password maintained by **Profile**. If these do not work, **FileTransfer** calls the **login** procedure associated with **conn**. Finally, if there is no **login** procedure, **FileTransfer** raises the error **Error** with a code of **illegalLogin**. This procedure can raise **Error[invalidObject]**.

```
FileTransfer.SetProcs: PROCEDURE [
  conn: FileTransfer.Connection, clientData: LONG POINTER,
  messages: FileTransfer.MessageProc ← NIL, login: FileTransfer.ClientProc ← NIL,
  noteProgress: FileTransfer.ClientProc ← NIL,
  checkAbort: FileTransfer.CheckAbortProc ← NIL];
```

The **SetProcs** procedure lets a client specify for a **Connection** the procedures to be used for certain functions. **NIL** parameters do not change the values in the **Connection**. The **MessageProc** is used by **FileTransfer** for user feedback; it is called to notify the user of errors and details of the operations taking place. The **login** procedure is called if **FileTransfer** needs a set of valid primary credentials. The **noteProgress** procedure is called by **FileTransfer** at intervals during the actual transfer of bytes between remote and local machines (during calls to **Copy** and **StoreStream**) so the client can provide feedback during a transfer. **checkAbort** is called at intervals to see whether the user has aborted the operation. If it returns **TRUE**, **FileTransfer** raises the error **ABORTED**. If no **CheckAbort** is specified, **FileTransfer** checks if the **ABORT** key has been pressed, and if so, raises **ABORTED**. The client may also attach some instance data, **clientData**, to a **Connection** that is passed back to each of these client-provided procedures. This procedure can raise **Error[invalidObject]**.

```
FileTransfer.SetSecondaryCredentials: PROCEDURE [
  conn: FileTransfer.Connection, connectName, connectPassword: LONG STRING];
```

The **SetSecondaryCredentials** procedure sets the secondary (connect) credentials to be used for a **Connection**. When FileTransfer needs secondary credentials and none have been set, **Error[spare1]** is raised. If secondary credentials have been set and they are invalid, **Error[illegalConnect]** is raised. This procedure can raise **Error[invalidObject]**.

```
FileTransfer.StoreStream: PROCEDURE [
  conn: FileTransfer.Connection, remote: FileName.VirtualFileName,
  veto: FileTransfer.VetoProc ← NIL, showDates: BOOLEAN ← FALSE,
  stream: Stream.Handle, creation: Time.Packed, bytes: LONG CARDINAL,
  fileType: FileTransfer.FileType];
```

The **StoreStream** procedure stores the contents of a stream into a remote file. **StoreStream** is passed a **VirtualFileName** that may not contain wildcards; the **version** field of the **VirtualFilename** is updated by **FileTransfer**. The **VetoProc** is called before the stream is

stored. **showDates** indicates whether **FileTransfer** should also output the creation date of a file when it posts its name with the connection's **MessageProc**. **stream** is the stream to be copied into the remote file. **creation** is the creation date to be given to the remote file. **bytes**, the length of the file, should be supplied if the client needs this information from **info** in its **veto** procedure. In the case of an **ns** server, **bytes** provides a hint of the file's size. **fileType** is the type of the file. This procedure can raise **Error**[... , **invalidObject, illegalParameters, illegalLogin, illegalConnect, retry, skip, spare1,** ...]. The error **ABORTED** can also be raised.

```
FileTransfer.WriteStream: PROCEDURE [
  conn: FileTransfer.Connection,file: FileTransfer.VirtualFileName,
  veto: FileTransfer.VetoProc ← NIL, showDates: BOOLEAN ← FALSE,
  creation: Time.Packed, fileType: FileTransfer.FileType]
  RETURNS [Stream.Handle];
```

The **WriteStream** procedure is not implemented.

46.5 Examples

A common use of **ReadStream/ReadNextStream** is to perform the same operation on a list of streams obtained from a **VFN** containing the "*" wildcard:

```
sh: Stream.Handle;
conn: Connection;
vfn: FileName.VirtualFilename;
fileName: LONG STRING;
...

vfn ← FileName.AllocVFN[fileName];
...
sh ← ReadStream[conn, vfn, remote];
WHILE sh # NIL DO
  -- Process the stream
  sh ← ReadNextStream[sh];
  ENDLOOP;
-- It is not necessary to delete the stream
FileName.FreeVFN[vfn];
```

47

MFile

The MFile interface provides operations on files, directories, and search paths. All files have a property list that contains the file name; the byte length; read, write, and creation dates; delete-protect, write-protect, and read-protect bits; and the file's type. Clients may add other properties to the property list.

The syntax of file names used by the development environment file system is defined in the *XDE User's Guide*; its conventions are summarized here for convenience.

The Xerox Development Environment file system provides a hierarchical directory structure. The top-level directory on a volume is named the same as the volume; that is, *<VolumeName>* specifies the top-level directory for volume *VolumeName*. A file name is fully specified (or fully qualified) if it starts with the root directory of a volume; that is, if it starts with <VolumeName>.

The search path is a sequence of directories used for looking up files that are not fully specified. The file system looks up a file name on the search path by searching for it in each successive directory until a match is found. There is only one search path in the file system. It is not possible to set up several concurrent search environments.

Directories on the search path may be write-protected, in which case it is not possible to change any of the files in the directory or add or delete files from it. If a file looked up on the search path is to be created or written into, two problems can occur: no match could be found, or the first match might occur in a search path directory that is write-protected. In this case, the file is created in the first directory in the search path that is not write-protected. This directory acts somewhat like a working directory. There must always be at least one directory in a search path that is not write-protected. If the search path contains directories that are all write-protected or on read-only volumes and a file must be created, **Error [illegalSearchPath]** is raised. If the first directory in the search path is write-protected, anomalies (to the client) may result, such as the file that is written may not necessarily be the file that is subsequently read.

47.1 Types

```
MFile.Access: TYPE = MACHINE DEPENDENT {
  anchor(0), readOnly, readWrite, writeOnly, log, delete, rename, null}
```

anchor access is requested to ascertain that a file exists or read its properties, for example. Anchor access is not enough to permit a client to read or write a file, but it does keep it from being deleted or renamed.

readOnly access permits the contents of the file to be read but not written.

readWrite access permits the contents of the file to be read and written and permits the length of the file to change.

writeOnly access permits the contents of the file to be written but not read and permits the length of the file to change.

log access truncates the file to zero length and permits new data to be appended to it. It is provided so that a client can let other clients read initial portions of a file that it is writing. (See **MFile.Log**.)

delete access permits a file to be deleted.

rename access permits the name/file binding of a file to be changed, either by renaming a file or swapping two files.

null access is provided only as an initialization value; it is not possible to acquire a handle with null access. Calling procedures with an access of **null** raises **Error[nullAccess]**.

```
MFile.ByteCount:TYPE = LONG CARDINAL;
```

The type **ByteCount** is used to specify length in bytes.

```
MFile.EnumerateProc: TYPE = PROCEDURE [
  name, fullName: LONG STRING, fileProc: MFile.FileAcquireProc, type: MFile.Type,
  spIndex: CARDINAL]
  RETURNS [done: BOOLEAN ← FALSE];
```

A client-provided **EnumerateProc** is called on every file matched by **EnumerateDirectory**. The **name** parameter is the name of the file, stripped of all directory information. **fullName** is the fully qualified name of the file starting at the character corresponding to the first character of the pattern. If no search path entry was used in the enumeration, it is the fully qualified name; if the search path was used, it is the part of the fully qualified name following the search path directory. (For example, if the pattern matched were **<Tajo>***, the **fullname** might be **<Tajo>Defs>Environment.bcd**; if the pattern matched were * and the search path contained **<Tajo>**, the **fullname** might be **Defs>Environment.bcd**). Enumerations do not lock out other operations, so you can call your **EnumerateProc** on a file that has since been deleted from the directory, and so forth. The error **noSuchFile** is raised if the current file has been deleted by some other process during the enumeration. The **fileProc** is provided so that the client can obtain a **Handle** on the file if desired. **type** is the type of the file (such as **text**, **binary**, or **directory**). If the search path was used to resolve the pattern, the **spIndex** parameter indicates which

directory was used for this file. If the search path was not used, **spIndex** has the value **MFile.searchPathNotUsed**. If the search path was used, the fully qualified name of the file is the concatenation of search path entry **spIndex** with **fullName**. The parameter **type** can be used to filter out files you aren't interested in. **done** indicates whether the client wishes to terminate the enumeration.

```
MFile.EnumRec: TYPE = ...;
```

The file system maintains its enumeration state for **GetNextHandleForReading** in an **EnumRec**.

```
MFile.EnumerateState: TYPE = LONG POINTER TO EnumRec;

MFile.EnumerationType: TYPE = {filesOnly, directoriesOnly, fileAndDirectories};
```

EnumerationType controls which types of files will be enumerated by **EnumerateDirectory** and **GetNextHandleForReading**. Only the distinction between directory and non-directory files is supported.

```
MFile.FileAcquireProc: TYPE = PROCEDURE [
  access: MFile.Access , release: MFile.ReleaseData] RETURNS [MFile.Handle];
```

A **FileAcquireProc** is provided by **EnumerateDirectory** to obtain a **Handle** on the currently enumerated file if the client requires one The **access** parameter is the desired access on the file. **fileProc** can raise **Error[..., conflictingAccess, protectionFault, volumeNotOpen, noSuchFile, ...]**.

```
MFile.Filter: TYPE = RECORD [
  name: LONG STRING ← NIL, type: MFile.Type ← unknown, access: MFile.Access];
```

A **Filter** is used by the file-notification mechanism to indicate which files a client is interested in. **name** is a pattern that is matched against the name of the file (with all directory information stripped). If name is **NIL**, all files match. **type** is the type of file the client is interested in; if type is **null**, all types match. The client is notified only when access **access** becomes available after having been unavailable.

```
MFile.Handle: TYPE = LONG POINTER TO MFile.Object;

MFile.InitialLength: TYPE = MFile.ByteCount;
```

InitialLength is used by **Acquire** to specify the minimum physical size of the file. It is a byte length.

```
MFile.NotifyProc: TYPE = PROCEDURE [
  name: LONG STRING, file: Handle, clientInstanceData: LONG POINTER]
  RETURNS [removeNotifyProc: BOOLEAN ← FALSE];
```

A **NotifyProc** is provided to the file-notification mechanism to be called when a file of interest to the client changes state. **name** contains the name of the file of interest; **file** contains a **Handle** on that file if the file exists. If the file does not exist, **file** is **NIL**. The client should check that the handle is not **NIL** before using it. This handle belongs to the file system. If the client wants a handle on the file, it must call **MFile.CopyFileHandle** on the handle passed in, and it must explicitly specify the access required (the **access** parameter

to **CopyFileHandle** cannot be **null**). The file system does not guarantee that a client can obtain the desired access; the notification should be viewed as a strong hint. The **NotifyProc** returns TRUE if it wishes to be removed and FALSE if it wishes to remain on the file system's notification list. The procedures **AddNotifyProc** and **RemoveNotifyProc** must not be called from within a **NotifyProc** or the file system will deadlock. (See also **AddNotifyProc** and **RemoveNotifyProc**.)

```
MFile.Object: TYPE = ...;

MFile.PleaseReleaseProc: TYPE = PROCEDURE [
  file: MFile.Handle, instanceData: LONG POINTER] RETURNS [MFile.ReleaseChoice];
```

Whenever a client attempts to acquire a file in a way that conflicts with its current uses, each of the file's current owners is asked to give up ownership by calling the owner's **PleaseReleaseProc** with its instance data. The **PleaseReleaseProc** and instance data are registered when an agent calls **Acquire** and passes in the **ReleaseData** parameter. The **PleaseReleaseProc** can take steps to relinquish ownership. (Maintaining proper synchronization and data integrity can be quite difficult when **PleaseReleaseProc**s are used. Invoking certain file system operations from a **PleaseReleaseProc** will cause the file system to deadlock. Clients using this facility should carefully read the discussion and examples at the end of this chapter. See also **Acquire** and **SetLogReadLength**.)

To avoid deadlock, a **PleaseReleaseProc** must not call any of the following procedures on the file for which it has been called: **Acquire**, **AcquireID**, **CopyFileHandle**, **Delete**, **DeleteWhenReleased**, **Log**, **ReadOnly**, **ReadWrite**, **Release**, **SetAccess**, **SetRelease**, **WriteOnly**. In addition, the client should not perform an enumeration that lists the file. The file system guarantees that once a handle has been released, it will not invoke its **PleaseReleaseProc**. If a client must invoke one of these actions under the above circumstances, it must fork a separate process. Note that the file system calls **PleaseReleaseProc** asynchronously; clients using these facilities should be aware that they must deal with all the problems of a multi-process system, even though the rest of their program may be a simple, single process. Thus the client must carefully monitor its own data, particularly that manipulated by **PleaseReleaseProc**. The client must take care that it does not attempt to release a file twice, once from the mainline code of the program and once from the **PleaseReleaseProc**. Because the call to release a file may be blocked on a file system monitor when the **PleaseReleaseProc** on that file is called, the client must carefully maintain its state so that the **PleaseReleaseProc** knows whether to release the file by returning **goAhead** or indicate that it is already being released by returning **later**.

```
MFile.Property: TYPE = RECORD [property: CARDINAL]
```

Clients may add properties to the property list of files. A **Property** is a registered value that is allocated by the Manager of System Development. (See the interface **MFileProperty** for the currently allocated client file properties. See also **RegisteredProperty** and **UnregisteredProperty**.)

```
MFile.RegisteredProperty: TYPE = CARDINAL [0..77777B];
```

This is the range of **Property** that is administrated by the Manager of System Development. A client should not use values in this range without making suitable arrangements.

MFile.ReleaseChoice: TYPE = {later, no, goAhead, allowRename}

later — the client is not ready to release the file, but promises to do so shortly. The file system will delay the **Acquire** until the conflict caused by this handle has been removed. This result should be returned when a client wishes to release the file but cannot do so directly from the **PleaseReleaseProc** because of the file system's synchronization restrictions. The operations that might remove the conflict are **Release**, **SetAccess**, or **Rename**.

no — the client refuses to release the file.

goAhead — the client gives up all claim to the file; the file system releases the MFile.Handle. The client should behave as if the last statement of its **PleaseReleaseProc** were MFile.Release; it must guarantee not to use the handle again after returning from the **PleaseReleaseProc**

allowRename — the client refuses to release the file. However, if the requested access is **rename**, the client has no objections to having the file renamed.

```
MFile.ReleaseData: TYPE = RECORD [
  proc: MFile.PleaseReleaseProc ← NIL, clientInstanceData: LONG POINTER ← NIL];
```

If the **ReleaseData.proc** is NIL, the file is not relinquished on an attempt to acquire it in a conflicting way.

```
MFile.SearchPath: TYPE = LONG POINTER TO MFile.SearchPathObject;
```

A search path is a sequence of directories used for looking up files that are not fully specified. The file system looks up a file name on the search path by searching for it in each successive directory until a match is found.

```
MFile.SearchPathObject: TYPE = RECORD [
  length: CARDINAL, directories: SEQUENCE I: CARDINAL OF LONG STRING];
```

length is the number of items in the sequence that are elements of the search path. **directories** is the sequence of strings containing the names of the search path directories. The search path represented by this object is the first **length** search path element of **directories**. Note that **length** is less than or equal to **I**. The first element of the search path is indexed by 0. The directory strings in a search path must be fully qualified names of existing directory files.

Directories on the search path may be write-protected, in which case it is not possible to change, add, or delete any of the files in the directory. If a file looked up on the search path is to be created or written into, two problems can occur: no match could be found, or the first match might occur in a search path directory that is write-protected. In this case, the file is created in the first directory in the search path that is not write-protected. This directory acts somewhat like a working directory. There must always be at least one directory in a search path that is not write-protected. If the search path contains directories that are all write-protected or on read-only volumes and a file must be created, **Error [illegalSearchPath]** is raised. If the first directory in the search path is write-

protected, anomalies (to the client) may result, such as the file that is written may not necessarily be the file that is subsequently read.

```
MFile.Type: TYPE = MACHINE DEPENDENT {unknown(0), text, binary, directory, null(255)};
```

Files of type **unknown** have no known type; they were not created with one of the other file system types. Files of type **text** should contain characters. Files of type **binary** may contain any data. Files of type **directory** are special files containing part of the directory structure of a file system.

```
MFile.UnregisteredProperty: TYPE = CARDINAL [100000B..177777B];
```

This is the range of **Property** for which no administrative conflict resolution is done.

47.2 Constants and data objects

```
MFile.dontCare: MFile.InitialLength = ...;
```

If **dontCare** is specified as the **initialLength** to **Acquire**, the physical size of an existing file is not changed. If a new file is created, it has an initial physical size of 512 bytes.

```
MFile.maxNameLength: CARDINAL = 100;
```

A file or directory name can be no more than **maxNameLength** characters long; path names can be longer, of course

```
MFile.noSearchPathUsed:CARDINAL = LAST[CARDINAL];
```

noSearchPathUsed is returned in an enumeration if the file enumerated was not obtained by using a search path entry.

```
MFile.dontRelease: MFile.ReleaseData = [];
```

dontRelease is the **ReleaseData** that refuses to release the file.

47.3 Signals and errors

```
MFile.Error: ERROR [file: MFile.Handle, code: ErrorCode];
```

MFile.Error is raised to indicate all file system errors that are not a result of manipulating client-defined file properties. **file** is the handle of the file causing the error. It may be **NIL** if the error occurs in the process of creating a file. **code** describes the error condition. All file system procedures can be invoked from the catch phrase of **MFile.Error**, subject to the deadlock restrictions imposed on the surrounding block or procedure (see **PleaseReleaseProc** and **NotifyProc**). See **MFile.AppendErrorMsg** for an easy way to construct a string containing an MFile error message.

```
MFile.ErrorCode: TYPE = MACHINE DEPENDENT {noSuchFile(0), conflictingAccess,
insufficientAccess, directoryFull, directoryNotEmpty, illegalName, noSuchDirectory,
noRootDirectory, nullAccess, protectionFault, directoryOnSearchPath, illegalSearchPath,
volumeNotOpen, noRoomOnVolume, noSuchVolume, crossingVolumes,
```

fileAlreadyExists, fileIsRemote, fileIsDirectory, invalidHandle, courierError, addressTranslationError, connectionSuspended, other(377B)}

noSuchFile	you are trying to access a file that does not exist.
conflictingAccess	you are trying to read a file that someone else is writing, for example.
insufficientAccess	you are trying to read a file with **writeOnly** access, for example.
directoryFull	you are trying to create a file in a directory with no more room.
directoryNotEmpty	you are trying to delete a directory that contains files.
illegalName	the given file name contains illegal characters.
noSuchDirectory	you are attempting to access a directory that does not exist.
noRootDirectory	you are trying to access a volume that has no development environment directory. The only file system action that can be taken on such a volume is to create a root directory using **MFile.CreateDir**.
nullAccess	you are trying to use a file with null access.
protectionFault	you are trying to access a file in a way conflicting with its protection or the protection of its directory.
directoryOnSearchPath	you are trying to delete a directory that is on the current search path.
illegalSearchPath	you haven't included a directory that is not write-protected in the search path.
volumeNotOpen	you are trying to write on a volume opened read-only or read an unopened volume.
noRoomOnVolume	you are trying to create a file, but there is no room on the volume.
noSuchVolume	you are attempting to access a logical volume that does not exist.
crossingVolumes	the current operation would cause a file that had been created on one logical volume to be added to a directory on a different logical volume.
fileAlreadyExists	you are trying to rename a file, but there is already one by that name.

fileIsRemote	you are performing an operation on a remote file that is not supported by the current implementation. *This error code is intended for future use and will not be seen by standard users of Mesa 11.0.*
fileIsDirectory	the current operation is not permitted on a directory.
invalidHandle	the **Handle** parameter to an operation is invalid; it has probably been released already.
courierError	a courier error has occurred while manipulating a remote file.*This error code is intended for future use and will not be seen by standard users of Mesa 11.0.*
addressTranslationError	an address translation error has occurred while acquiring a remote file. *This error code is intended for future use and will not be seen by standard users of Mesa 11.0.*
connectionSuspended	the connection to the remote machine has been suspended while manipulating a remote file.*This error code is intended for future use and will not be seen by standard users of Mesa 11.0.*
other(377B)	is raised by other errors, in particular implementation errors.

MFile.NameForError: SIGNAL RETURNS [errorName: LONG STRING];

If an **MFile.Error** is raised while acquiring a file (so that the file parameter of **Error** is **NIL**), the name of the desired file can be obtained by raising **NameForError** in the catch phrase for **MFile.Error**. The **LONG STRING** returned by **NameForError** belongs to the file system and should not be deallocated by the client.

MFile.PropertyError: ERROR [code: MFile.PropertyErrorCode];

The error **PropertyError** can be raised when calling operations that manipulate a file property list.

```
MFile.PropertyErrorCode: {
  noSuchProperty, noRoomInPropertyList, insufficientSpaceForProperty, wrongSize};
```

noSuchProperty	the system can't find a property of this type in the property list.
noRoomInPropertyList	the property list is full.
insufficientSpaceForProperty	the property can't be copied into the space provided.
wrongSize	an **AddProperty** has been attempted with a **maxLength** different than the previous one.

47.4 Procedures

```
MFile.Acquire: PROCEDURE [
  name: LONG STRING, access: MFile.Access, release: MFile.ReleaseData, mightWrite:
  BOOLEAN ← FALSE, initialLength: MFile.InitialLength ← MFile.dontCare, type: MFile.Type ←
  unknown]
  RETURNS [MFile.Handle];
```

The **Acquire** procedure obtains a file handle with the requested access rights to file **name**. The search path may be used to look up **name** (see the description of **MFILE.SearchPathObject** for an explanation of how a file name is looked up). This procedure can raise **MFile.Error[..., addressTranslationError, connectionSuspended, courierError, noSuchFile, conflictingAccess, directoryFull, illegalName, volumeNotOpen, noRootDirectory, nullAccess, protectionFault, noRoomOnVolume, noSuchDirectory, noSuchVolume, other, ...]**.

If **access** is **anchor**, **readOnly**, **delete**, or **rename**, the file must already exist or the error **MFile.Error[noSuchFile]** will be raised. If **access** is **readWrite**, **writeOnly**, or **log**, the file system first checks to see if the file already exists. If it does, **Acquire** ensures that the number of bytes in the file is at least as large as **initialLength**, although it does not set the logical length of the file. If it does not exist, a new file of size **initialLength** and type **type** will be created.

The parameter **mightWrite** is significant only if **access** is **anchor** or **readOnly**. If **mightWrite** is **TRUE**, **Acquire** will not return a handle on a file in a write-protected directory. It will skip write-protected directories in the search path if the search path is used to resolve **name**, and it will raise **MFile.Error[protectionFault]** if the search path is not used but would otherwise return a handle in a write-protected directory.

Table 47.1 defines which accesses conflict on an **Acquire**. Each row is the old access (already held by some other client) and each column is the new access (requested in the **Acquire**).

A client may wish to gain use of a file in a way that conflicts with current access rights held by other clients. Whenever a client attempts to acquire a file in a way that conflicts with its current uses, each of the file's current owners is asked to give up ownership by calling the owner's **PleaseReleaseProc** with its instance data. If all clients with conflicting accesses relinquish ownership, the new use is granted. Otherwise, the access is refused. Access may also be refused if the file has been protected against the access required; for example, **readWrite** access will be denied to a file that is write-protected.

If a client requests **readOnly** access to a file for which some other agent has **log** access, no conflict occurs. However, the **PleaseReleaseProc** of the log file is called to make as much as possible of the file readable (see **MStream.SetLogReadLength**). The client will be able to read only as much of the file as is available when it is granted **readOnly** access.

```
MFile.AcquireTemp: PROCEDURE [
  type: MFile.Type, initialLength: MFile.InitialLength ← MFile.dontCare]
  RETURNS [MFile.Handle];
```

Old Access	New Access: anchor	read Only	read Write	write Only	log	delete	rename	null
anchor	ok	ok	ok	ok	ok	no	no	no
readOnly	ok	ok	no	no	no	no	no	no
readWrite	ok	no	no	no	no	no	no	no
writeOnly	ok	no	no	no	no	no	no	no
log	ok	*	no	no	no	no	no	no
delete	no	no	no	no	no	no	no	no
rename	no	no	no	no	no	no	no	no

Table 47.1: **Acquire** accesses

The **AcquireTemp** procedure returns a handle with **readWrite** access on a Pilot temporary file created on the volume containing the first non-write-protected directory on the search path. If all the directories on the search path are protected, it is created on the system volume. This file is not in the file system directory, and it will be deleted when the last handle on it is released. Its name is the empty string. It is not possible to generate an access conflict or protection conflict with **AcquireTemp**. This procedure can raise **MFile.Error[...,nullAccess, noRoomOnVolume, other, ...]**.

```
MFile.AddNotifyProc: PROCEDURE [
  proc: MFile.NotifyProc, filter: MFile.Filter, clientInstanceData: LONG POINTER];
```

The **AddNotifyProc** procedure adds a notification request to the file system notification list. **proc** will be called when the event specified by **filter** occurs; see **NotifyProc** and **Filter**. **clientInstanceData** will be passed to **proc**. The file system will deadlock if this procedure is called from within a **NotifyProc**.

```
MFile.AddProperty: PROCEDURE [
  file: MFile.Handle, property: MFile.Property, maxLength: CARDINAL];
```

The **AddProperty** procedure is used to add a client property to the property list. **maxLength** is the maximum number of bytes that the property will need. If there is insufficient room for the property to be added, **PropertyError[noRoomInPropertyList]** will result. If the property already exists for this file and if the **maxLength** is equal to the existing version, this operation is a no-op. Otherwise, **PropertyError[wrongSize]** is raised. This procedure can be called on a file handle with any access. It can also raise **MFile.Error[addressTranslationError, connectionSuspended, courierError]**.

```
MFile.AppendErrorMessage: PROCEDURE [
  msg:LONG STRING, code: MFile.ErrorCode, file: MFile.Handle];
```

The **AppendErrorMessage** procedure fills into the string **msg** a description of the error **code** on the file **file**. If the description is too long to fit into **msg**, it is truncated.

```
MFile.CompleteFilename: PROCEDURE [name, addedPart: LONG STRING]
  RETURNS [exactMatch: BOOLEAN, matches: CARDINAL];
```

The **CompleteFilename** procedure attempts to "complete" a file name: * is appended to **name** and the files in the file system are searched for matches. The number of matches is returned in **matches**, and the common prefix of the extensions (which may be empty) is returned in **addedPart**. If **addedPart** is too short, a **String.StringBoundsFault** will be raised. If the concatenation of **name** and **addedPart** yields a unique file name that could be used to acquire a file, **exactMatch** will be **TRUE**.

```
MFile.ComputeFileType: PROCEDURE [ file: MFile.Handle] RETURNS [type: MFile.Type];
```

The **ComputeFileType** procedure implements a heuristic for calculating the type of a file of unknown type. It reads all the bytes of **file**; if all represent character codes, it returns the type **text**. Otherwise, it returns the type **binary**.

```
MFile.Copy: PROCEDURE [file: MFile.Handle, newName: LONG STRING];
```

The **Copy** procedure copies a file into another file. The client must have **readOnly** or **readWrite** access to **file**, and it must be able to open file **newName** for **writeOnly**. This procedure can raise **MFile.Error[..., noSuchFile, directoryFull, fileIsRemote, file IsDirectory, insufficientAccess, volumeNotOpen, noRootDirectory, noRoomOnVolume, addressTranslationError, connectionSuspended, courierError, other, ...]**.

```
MFile.CopyFileHandle: PROCEDURE [
  file: MFile.Handle, release: MFile.ReleaseData, access: MFile.Access ← null]
  RETURNS [MFile.Handle];
```

The **CopyFileHandle** procedure produces a new **MFile.Handle** on the same file as **file**. This operation is an accelerator for **Acquire** that avoids looking up the file in the directory again. It may obtain the new handle with a different access. If the **access** parameter is **null**, the new handle has the same access as the old handle; otherwise, it has the requested access. (Note that **null** access cannot be used when copying the handle passed to a **NotifyProc**; see the discussion of the type **NotifyProc**.) Because it can change the access, **CopyFileHandle** can raise **Error[..., conflictingAccess, protectionFault...]**.

This operation provides an escape hatch for some of the file system's access control. If the access requested for the copy is no stronger than the original access, the file system will make the copy even though it would not permit another client to gain that access to the file. For instance, if a client already has a file handle with **readWrite** access, it can obtain a copy with **readOnly** access or **readWrite** access, although another client requesting a handle with either of these accesses would be refused. It is assumed that a client that produces such conflicting handles is responsible for the potential chaos that might result if those handles are misused.

Table 47.2 defines the relative strengths of accesses; < means that the new access is weaker than the old, and > means that the new access is stronger than the old. If the requested access is stronger than the access on the file, the usual access checking is performed.

Old Access	New Access: anchor	read Only	read Write	write Only	log	delete	rename	null
anchor	<	>	>	>	>	>	>	<
readOnly	<	<	>	>	>	>	>	<
readWrite	<	<	<	<	>	>	>	<
writeOnly	<	>	>	>	>	>	>	<
log	<	>	>	>	>	>	>	<
delete	<	>	>	>	>	>	>	<
rename	<	>	>	>	>	>	>	<

Table 47.2: Access strengths

MFile.CopyProperties: PROCEDURE [from, to: MFile.Handle];

The **CopyProperties** procedure is typically used by a utility to copy all of the existing properties of some base file into a new version. It can raise **MFile.Error[addressTranslationError, connectionSuspended, courierError]**. Client properties are preserved while a file stays in the development environment file system. However, they may be lost if, for instance, the file is stored on a remote file server. This procedure can be called on file handles with any access.

MFile.CreateDirectory: PROCEDURE [dir: LONG STRING];

The **CreateDirectory** procedure ensures that a directory exists, creating new directories if necessary. All the intermediate subdirectories on the path will be created as necessary; for example, if **dir** is *<Tajo>Defs>Source* and subdirectory *Defs* does not exist, it as well as subdirectory *Source* will be created. The root directory is *not* created automatically, and an error will be raised if it does not exist. If **dir** is not completely specified, the search path will be used. Trailing >s are stripped from **dir**. Hence **CreateDirectory["<Tajo>Temp>"L]** and **CreateDirectory["<Tajo>Temp"L]** both create a subdirectory in the root directory on a volume named **Tajo**. **CreateDirectory[">Temp"L]** creates a subdirectory in the first writeable directory of the current search path. If a file named **dir** already exists and is a directory, this procedure is a no-op. If **dir** is not a directory, the error code **fileAlreadyExists** is raised. This procedure can raise **Error[..., directoryFull, fileIsRemote, illegalName, volumeNotOpen, noRootDirectory, fileAlreadyExists ...]**.

MFile.Delete: PROCEDURE [file: MFile.Handle];

The **Delete** procedure deletes a file. As directories are just files,this procedure can be used to delete directories. **Delete** can only be called with a handle with delete access. Directories can be deleted only if they are empty and they are not on the search path. **Delete** always releases the MFile.Handle passed in if it is successful. **Delete** can raise **Error[..., conflictingAccess, directoryNotEmpty, insufficientAccess, fileIsRemote, volumeNotOpen, noRootDirectory, directoryOnSearchPath, courierError, addressTranslationError, connectionSuspended, ...]**. (See also **DeleteWhenReleased**).

MFile.DeleteWhenReleased: PROCEDURE [file: MFile.Handle];

The **DeleteWhenReleased** procedure arranges for a file to be deleted when all its current uses of the file. If no other client is currently accessing the file, **DeleteWhenReleased** has the same semantics as **Delete**. If the file is in use by other agents, **DeleteWhenReleased** removes it from the directory and makes it a temporary file, sets its name in the leader page to the empty string, and marks it to be deleted when all other agents have released it. **DeleteWhenReleased** can be called with a handle with *any* access. Directories can be deleted only if they are empty and they are not on the search path. **DeleteWhenReleased** always releases the MFile.Handle passed in if it is successful. **DeleteWhenReleased** can raise **Error[..., directoryNotEmpty, fileIsRemote. fileIsDirectory, volumeNotOpen, noRootDirectory, directoryOnSearchPath, ...]**. (See also **Delete**.)

MFile.EnumerateDirectory: PROCEDURE [
  name: LONG STRING, proc: MFile.EnumerateProc, which: MFile.EnumerationType];

The **EnumerateDirectory** procedure enumerates the files in the file system. Enumerations can be performed on files, directories, or both, depending on the parameter **which**. The procedure **proc** is called for every file matching the pattern **name**. The enumeration can be terminated early by returning TRUE from **proc**. It is possible to enumerate only within a directory or within a directory and all its offspring. A # in **name** matches any single character in a file name except >. A single * occurring in **name** matches zero or more characters in a file name, but does not match >. Hence, enumerating * in a directory lists all the files in that directory but not in its subdirectories. Multiple consecutive *s do match >, so enumerating ** matches all files in a directory and in the entire directory tree below it. **EnumerateDirectory** does not guarantee to enumerate the files in any particular order (that is, they will not necessarily be alphabetical). If the pattern is not completely specified (if it does not start with <VolumeName>) **EnumerateDirectory** uses the search path. It enumerates from every directory in the search path successively. It is possible to enumerate a file several times if it occurs below several search path directories. See also **FileAcquireProc** and **EnumerateProc**. **EnumerateDirectory** can raise **MFile.Error[..., fileIsRemote , volumeNotOpen, illegalName, noRootDirectory, other, ...]**.

MFile.FreeSearchPath: PROCEDURE [MFile.SearchPath];

The **FreeSearchPath** procedure frees a search path allocated by **GetSearchPath**. Note that the search path is not allocated from the system heap, so the client must be careful not to free search paths that contain strings allocated from the system heap.

```
MFile.GetAccess: PROCEDURE [file: MFile.Handle] RETURNS [access: MFile.Access];
```

The **GetAccess** procedure returns the current access associated with a **Handle**.It can raise **MFile.Error** with the error codes **addressTranslationError, connectionSuspended,** and **courierError.**

```
MFile.GetCreateDate: PROCEDURE [ file: MFile.Handle] RETURNS [create: Time.Packed];
```

The **GetCreateDate** procedure returns the create time of **file**, which is updated when a file is acquired with **readWrite, writeOnly**, or **log** access. This procedure can be called on a file handle with any access.

```
MFile.GetDirectoryName: PROCEDURE [file: MFile.Handle, name: LONG STRING];
```

The **GetDirectoryName** procedure appends to the string **name** as much of the directory portion of the fully qualified name of **file** as will fit. The name has a trailing > if **file** was not the top level directory. **PropertyError[insufficientSpaceForProperty]** is raised if it does not fit. This procedure can be called on a file handle with any access. It also raises **MFile.Error[addressTranslationError, connectionSuspended, courierError]**.

```
MFile.GetFullName: PROCEDURE [file: MFile.Handle, name: LONG STRING];
```

The **GetFullName** procedure appends to the string **name** as much of the fully qualified name of file as will fit. **PropertyError[insufficientSpaceForProperty]** is raised if it does not fit. **Error[addressTranslationError, connectionSuspended, courierError]** can also be raised. This procedure can be called on a file handle with any access.

```
MFile.GetLength: PROCEDURE [file: MFile.Handle] RETURNS [MFile.ByteCount];
```

The **GetLength** procedure returns the length of **file** in bytes. It can raise **Mfile.Error[addressTranslationError, connectionSuspended, courierError]**. This procedure can be called on a file handle with any access.

```
MFile.GetNextHandleForReading: PROCEDURE [
  filter, name: LONG STRING, release: ReleaseData, lastState: EnumerateState, stopNow:
  BOOLEAN ← FALSE]
  RETURNS [file: Handle, state: EnumerateState];
```

The **GetNextHandleForReading** procedure provides a restricted form of "stateless" enumeration. A **Handle** with **ReadOnly** access is created for each file in the directory that is not **ReadProtected** and that matches the filter. (See **EnumerateDirectory** for a description of when a file matches a filter.) The current state of the enumeration is passed back and forth on each call. **lastState** must be **NIL** on the initial call, and **filter** should contain the same value for each call in the stateless enumeration. **name** is a client-provided string that will be filled in with the name of the file **file**. When the enumeration terminates, **name.length** will be 0 and **state** will be **NIL**. If the enumeration is to be terminated early, a final call with **stopNow = TRUE** must be made, permitting the file system to free its enumeration state. This procedure can raise **MFile.Error** with error code **fileIsRemote**.

```
MFile.GetProperties: PROCEDURE [file: MFile.Handle, name: LONG STRING ← NIL]
  RETURNS [create, write, read: Time.Packed, length: MFile.ByteCount, type: MFile.Type,
  deleteProtected, writeProtected, readProtected: BOOLEAN];
```

The **GetProperties** procedure returns the values of the built-in properties of a file. If **name** is not **NIL**, it is filled in with the name of the file. If **readProtected** is **TRUE**, all other information is invalid. This procedure can be called on a file handle with any access. It can raise **MFile.Error[addressTranslationError, connectionSuspended, courierError]**.

```
MFile.GetProperty: PROCEDURE [
  file: MFile.Handle, property: MFile.Property, block: Environment.Block]
  RETURNS [length: CARDINAL];
```

The **GetProperty** procedure gets the value of a client property. As much of the property as fits will be placed into block. The actual number of bytes copied is returned. The error **MFile.PropertyError[wrongSize]** will be raised if the number of bytes in block is smaller than the number of bytes of information stored in this property value. If the property is not found, the error **PropertyError[noSuchProperty]** is raised. **MFile.Error** can also be raised with the error codes **addressTranslationError, connectionSuspended,** and **courierError**. This procedure can be called on a file handle with any access.

```
MFile.GetProtection: PROCEDURE [file: MFile.Handle]
  RETURNS [deleteProtected, writeProtected, readProtected: BOOLEAN];
```

The **GetProtection** procedure returns the protection status of **file**. It can raise **MFile.Error[addressTranslationError, connectionSuspended, courierError]** . This procedure can be called on a file handle with any access.

```
MFile.GetReleaseData: PROCEDURE [
  file: MFile.Handle] RETURNS [release: MFile.ReleaseData];
```

The **GetReleaseData** procedure returns the current release data associated with a **Handle**. It can raise **MFile.Error[addressTranslationError, connectionSuspended, courierError]**. This procedure can be called on a file handle with any access.

```
MFile.GetSearchPath: PROCEDURE RETURNS [MFile.SearchPath];
```

The **GetSearchPath** procedure returns a copy of the file system search path. The client is responsible for deallocating the returned search path by calling **FreeSearchPath**.

```
MFile.GetTimes: PROCEDURE [
  file: MFile.Handle] RETURNS [create, write, read: Time.Packed];
```

The **GetTimes** procedure returns the create, read, and write times of **file**. The create and write times of a file are updated when a file is acquired with **readWrite**, **writeOnly**, or **log** access. The read time of a file is updated when a file is acquired with **readOnly** or **readWrite** access. **GetTimes** can raise **Error[addressTranslationError, connectionSuspended, courierError]** . This procedure can be called on a file handle with any access.

```
MFile.GetType: PROCEDURE [file: MFile.Handle] RETURNS [type: MFile.Type];
```

The **GetType** procedure returns the type of **file**. It can raise **MFile.Error [addressTranslationError, connectionSuspended, courierError]** .

```
MFile.GetVolume: PROCEDURE [ file: MFile.Handle] RETURNS [Volume.ID];
```

The **GetVolume** procedure returns the **Volume.ID** of the logical volume containing **file**. If **file** is a remote file, it returns **Volume.nullID**.

```
MFile.InitializeFileSystem: PROCEDURE;
```

The **InitializeFileSystem** procedure starts the file system; it should only be called by clients that include the file system. It causes a top-level directory to be created on the volume from which it is called. Clients of this procedure must be prepared to catch the resumable **SIGNAL AboutToScavenge**, defined in the friends interface **MScavenge**.

```
MFile.Log: PROCEDURE [
  name: LONG STRING, release: MFile.ReleaseData,
  initialLength: MFile.InitialLength ← MFile.dontCare]
  RETURNS [MFile.Handle];
```

The **Log** procedure acquires the file name with **log** access. It ensures that the file is at least as large as **initialLength**, although it will not set the file's logical length. If file **name** does not exist, a new file of size **initialLength** and type **text** is created. See **Acquire** and **PleaseReleaseProc** for a discussion of access conflicts. See also **MStream.SetLogReadLength**. This procedure can raise **MFile.Error[..., noSuchFile, conflictingAccess, directoryFull, illegalName, volumeNotOpen, noRootDirectory, nullAccess, protectionFault, noRoomOnVolume, addressTranslationError, connectionSuspended, courierError, other ...]**,

```
MFile.ReadOnly: PROCEDURE [
  name: LONG STRING, release: MFile.ReleaseData, mightWrite: BOOLEAN ← FALSE] RETURNS
  [MFile.Handle];
```

The **ReadOnly** procedure acquires the file name with **readOnly** access. See **Acquire** and **PleaseReleaseProc** for a discussion of access conflicts and the meaning of **mightWrite**. This procedure can raise **MFile.Error[..., noSuchFile, conflictingAccess, directoryFull, illegalName, volumeNotOpen, noRootDirectory, noSuchDirectory, noSuchVolume,nullAccess, protectionFault, noRoomOnVolume, addressTranslationError, connectionSuspended, courierError, other, ...]**.

```
MFile.ReadWrite: PROCEDURE [
  name: LONG STRING, release: MFile.ReleaseData, type: MFile.Type,
  initialLength: MFile.InitialLength ← MFile.dontCare]
  RETURNS [MFile.Handle];
```

The **ReadWrite** procedure acquires the file name with **readWrite** access. It ensures that the file is at least as large as **initialLength**, although it will not set the file's logical length. If file **name** does not exist, a new file of size **initialLength** and type **type** is created. See **Acquire** and **PleaseReleaseProc** for a discussion of access conflicts. This procedure can raise **MFile.Error[..., noSuchFile, conflictingAccess, directoryFull, illegalName,**

volumeNotOpen, noRootDirectory, nullAccess, protectionFault, noRoomOnVolume, addressTranslationError, connectionSuspended, courierError, other, ...].

MFile.Release: PROCEDURE [file: MFile.Handle];

When a client is through using a handle, it returns it to the file system by calling **Release**. This procedure can raise **MFile.Error[addressTranslationError, connectionSuspended, courierError, invalidHandle]**.

MFile.RemoveNotifyProc: PROCEDURE [
proc: MFile.NotifyProc, filter: MFile.Filter, clientInstanceData: LONG POINTER];

The **RemoveNotifyProc** procedure removes a client notification request from the file system notification list. A call on **RemoveNotifyProc** that finds no match is a no-op, and **NIL clientInstanceData** matches anything. If this procedure is called from a **NotifyProc**, the file system will deadlock. (See also **NotifyProc**.)

MFile.RemoveProperties: PROCEDURE [file: MFile.Handle];

The **RemoveProperties** procedure removes all client properties from **file**. It can raise **MFile.Error [addressTranslationError, connectionSuspended, courierError]**. This procedure can be called on a file handle with any access.

MFile.RemoveProperty: PROCEDURE [file: MFile.Handle, property: MFile.Property];

The **RemoveProperty** procedure removes a client property. It can raise **MFile.Error [addressTranslationError, connectionSuspended, courierError]**. This procedure can be called on a file handle with any access.

MFile.Rename: PROCEDURE [file: MFile.Handle, newName: LONG STRING];

The **Rename** procedure changes the name of a file, potentially moving it between directories but not between volumes. If **Rename** is called with a temporary file, the file will be made permanent and given the specified name; if **Rename** is called with an empty string, the file will be made temporary. The client must have **rename** access to **file**. This procedure can raise **MFile.Error[..., noSuchFile, directoryFull, insufficientAccess, volumeNotOpen, noRootDirectory, fileAlreadyExists, fileIsDirectory, fileIsRemote, addressTranslationError, connectionSuspended, courierError, other, ...]**

MFile.SameFile: PROCEDURE [file1, file2: MFile.Handle] RETURNS [BOOLEAN];

The **SameFile** procedure returns **TRUE** if **file1** and **file2** are **Handles** on the same underlying file.

MFile.SetAccess: PROCEDURE [file: MFile.Handle, access: MFile.Access];

The **SetAccess** procedure changes the access associated with a **Handle**. As with **Acquire**, the **PleaseReleaseProc** of other clients with conflicting access may be called. Because it changes the access, **SetAccess** can raise **Error[..., conflictingAccess addressTranslationError, connectionSuspended, courierError, protectionFault...]**.

```
MFile.SetDeleteProtect: PROCEDURE [file: MFile.Handle, deleteProtected: BOOLEAN];
```

The **SetDeleteProtect** procedure changes the **deleteProtection** attribute of **file**. It can raise **MFile.Error[addressTranslationError, connectionSuspended, courierError]**. This procedure can be called on a file handle with any access.

```
MFile.SetLength: PROCEDURE [file: MFile.Handle, length: MFile.ByteLength];
```

The **SetLength** procedure changes the length of a file, where **length** is specified in bytes. The file is grown or shrunk as necessary. If it must be grown, this happens immediately; however, it will not be shrunk until all users of the file **Release** their **Handles**. This procedure can raise **MFile.Error[addressTranslationError, connectionSuspended, courierError]**.

```
MFile.SetProperties: PROCEDURE [
  file: MFile.Handle, create, write, read: Time.Packed ← System.gmtEpoch, length:
  MFile.ByteLength, type: Type, deleteProtected, writeProtected, readProtected:
  BOOLEAN ← FALSE];
```

The **SetProperties** procedure sets many of the built-in properties of **file**. The **name** property must be changed by calling **Rename**. The type cannot be changed from or to **directory**. If **SetType** would change the type of the file from **directory**, the operation is ignored but no error is raised; if it would change the type of the file to **directory**, the error **MFile.Error[other]** is raised. The procedure can also raise **[addressTranslationError, connectionSuspended, courierError]**. It can be called on a file handle with any access.

```
MFile.SetProperty: PROCEDURE [
  file: MFile.Handle, property: MFile.Property, block: Environment.Block];
```

The **SetProperty** procedure sets the value of a client property. The error **MFile.PropertyError[noSuchProperty]** will be raised if the property is not associated with file. The error **MFile.PropertyError[wrongSize]** will be raised if the number of bytes in the block is greater than the maximum associated with this property. The actual number of bytes is recorded so that, for instance, it is not necessary to pad string properties with **NUL**s. This procedure can also raise **MFile.Error** with error codes **addressTranslationError, connectionSuspended, courierError.** It can be called on a file handle with any access.

```
MFile.SetProtection: PROCEDURE [
  file: MFile.Handle, deleteProtected, writeProtected, readProtected: BOOLEAN ← FALSE];
```

The **SetProtection** procedure changes the protection attributes of **file**. (See also **SetReadProtect**, **SetWriteProtect**, and **SetDeleteProtect**.) This procedure can raise **MFile.Error [addressTranslationError, connectionSuspended, courierError]**. It can be called on a file handle with any access.

```
MFile.SetReadProtect: PROCEDURE [file: MFile.Handle, readProtected: BOOLEAN];
```

The **SetReadProtect** procedure changes the read-protect attribute of **file**. Read protection may be used by a client to mark a file as inconsistent; for example, the compiler read-protects the object file if the corresponding source failed to compile. By not deleting the file, the user's directory structure is preserved. If a file is read-protected, its contents, including its other properties, are assumed to be invalid. This procedure can raise

MFile.Error[addressTranslationError, connectionSuspended, courierError]. It can be called on a file handle with any access.

```
MFile.SetReleaseData: PROCEDURE [file: MFile.Handle, release: MFile.ReleaseData];
```

The SetReleaseData procedure changes the release data associated with a Handle. It can raise MFile.Error [addressTranslationError, connectionSuspended, courierError]. This procedure can be called on a file handle with any access.

```
MFile.SetSearchPath: PROCEDURE [
  MFile.SearchPath] RETURNS [succeeded: BOOLEAN ← TRUE];
```

The SetSearchPath procedure sets the file system search path. Before setting the search path, the file system raises the Supervisor event aboutToChangeSearchPath, which can be aborted by clients wishing to forbid the change. If a client aborts the event aboutToChangeSearchPath, the file system raises the Supervisor event abortedSearchPathChange and returns. SetSearchPath copies the search path object and does not consume it. After successfully changing the search path, the file system raises the Supervisor event newSearchPath.The volumes named in the search path must be open and the directories named must already exist; SetSearchPath will not create them automatically. SetSearchPath can raise Error[..., noSuchFile, illegalName, volumeNotOpen, noRootDirectory, illegalSearchPath, ...]. If the error code is illegalSearchPath, the search path is unchanged. Any other error causes the search path to be set to NIL. (See also filesystem, aboutToChangeSearchPath, and newSearchPath.)

```
MFile.SetTimes: PROCEDURE [
  file: MFile.Handle, create, read, write: Time.Packed ← System.gmtEpoch];
```

The SetTimes procedure changes the read, write and/or create dates of file. Defaulted values are not changed. The procedure can raise MFile.Error[addressTranslationError, connectionSuspended, courierError]. It can be called on a file handle with any access.

```
MFile.SetType: PROCEDURE [file: MFile.Handle, type: MFile.Type];
```

The SetType procedure changes the type of file. The type cannot be changed from or to directory. If SetType would change the type of the file from directory, the operation is ignored but no error is raised; if it would change the type of the file to directory, the error MFile.Error[other] is raised. This procedure can be called on a file handle with any access.

```
MFile.SetWriteProtect: PROCEDURE [file: MFile.Handle, writeProtected: BOOLEAN];
```

The SetWriteProtect procedure changes the write-protect attribute of the file. This procedure can be called on a file handle with any access. MFile.Error[addressTranslationError, connectionSuspended, courierError] can be raised .

```
MFile.SwapNames: PROCEDURE [f1, f2: MFile.Handle];
```

The SwapNames procedure swaps the contents for a pair of files; they may be temporary files or in different directories, but they must be on the same volume. This is a very cheap operation, and the contents of the files are not copied. The client must have rename access to both f1 and f2. This procedure can raise MFile.Error[..., addressTranslationError,

connectionSuspended, courierError, noSuchFile, directoryFull, insufficientAccess, volumeNotOpen, noRootDirectory, fileIsDirectory, other, ...].

```
MFile.ValidFilename: PROCEDURE [name: LONG STRING] RETURNS [ok: BOOLEAN];
```

The **ValidFilename** procedure returns TRUE if **name** contains a syntactically valid file name.

```
MFile.WriteOnly: PROCEDURE [
    name: LONG STRING, release: MFile.ReleaseData, type: MFile.Type, initialLength:
    MFile.InitialLength ← MFile.dontCare]
    RETURNS [MFile.Handle];
```

The **WriteOnly** procedure acquires the file name with **writeOnly** access. It ensures that the file is at least as large as **initialLength**, although it will not set the file's logical length. If file **name** does not exist, a new file of size **initialLength** and type **type** is created. (See **Acquire** and **PleaseReleaseProc** for a discussion of access conflicts.) This procedure can raise MFile.Error[..., noSuchFile, conflictingAccess, directoryFull, illegalName, volumeNotOpen, noRootDirectory, nullAccess, protectionFault, noRoomOnVolume, addressTranslationError, connectionSuspended, courierError, other, ...].

47.5 Discussion and examples

The following contains discussion and examples of **PleaseReleaseProc** and **Notification**. This material may be skipped by the casual client of **MFile**.

47.5.1 Release procedures

A client should provide a **PleaseReleaseProc** for a file if it is making relatively passive use of the file and might be willing to relinquish it. For example, a file window is willing to release a file if it is not open for edit, and a file cache releases an old version of a file so that a new version may be retrieved.

The call on a client's **PleaseReleaseProc** is made from the process that is requesting the file in a conflicting way, such as when (1) the file system has locked some of the data structures associated with the file, and (2) the client's processes are running at the same time. Special care must be taken in writing **PleaseReleaseProc** both to avoid synchronization problems in the client's code and to avoid deadlock in the file system.

To protect itself, the client must monitor data accessed by the **PleaseReleaseProc** and also carefully synchronize which process has actually released the file. The **PleaseReleaseProc** should not wait for a monitor that may be held by a process that might be waiting for the file system. For example, the actual release of the file should not be done from within the client monitor, since the release may be blocked, waiting for the **PleaseReleaseProc**.

The **PleaseReleaseProc** may determine that it can release the file, in which case the client process must not access the file handle again. The **PleaseReleaseProc** does not actually perform the release but returns the value **goAhead**, asking the file system to do the release. The **PleaseReleaseProc** may determine that the file will be released in the near future, either because the client process is already releasing it or because the **PleaseReleaseProc** will fork another process to actually release the file. In this case, the **PleaseReleaseProc** returns the value **later**, and the file system delays the conflicting

request for the file until it has been released. The **PleaseReleaseProc** may determine that it cannot release the file, in which case it returns **no**. If the client does not wish to release the file but does not care if it is renamed, it returns the value **allowRename**.

To avoid deadlock with the file system, the **PleaseReleaseProc** should not call any of the following procedures on the file requested: **Acquire, AcquireId** (a friends-level procedure), **CopyFileHandle, Delete, DeleteWhenReleased, Log, ReadOnly, ReadWrite, Release, SetAccess, SetRelease, WriteOnly**. In addition, the **PleaseReleaseProc** should not perform an enumeration that lists the file. If a **PleaseReleaseProc** must invoke one of these actions, it must fork a separate process and not wait for that process, since the procedures will not be executed until after the **PleaseReleaseProc** returns. The file system guarantees that once a handle has been released, it will not invoke its **PleaseReleaseProc**.

The following simple example of a **PleaseReleaseProc** shows a simple-minded module managing a single file that it is always willing to release.

```
FileNotAvailable: ERROR = CODE;
f: MFile.Handle ← NIL;
busy, pleaseFree: BOOLEAN ← FALSE;

Acquire: ENTRY PROCEDURE = {
  busy ← FALSE;
  pleaseFree ← FALSE;
  f ← MFile.Acquire[name: "Some.File"L, release: [proc: MyReleaseProc], ... ]};

DoneWith: PROCEDURE RETURNS [file: MFile.Handle] = {
  FileToFree: ENTRY PROCEDURE RETURNS [file: MFile.Handle] = {file ← f; f ← NIL};
  localF: MFile.Handle = FileToFree[];
  IF localF # NIL THEN MFile.Release[localF]};

MyReleaseProc: ENTRY MFile.PleaseReleaseProc = {
  SELECT TRUE FROM
    busy = > {pleaseFree ← TRUE; RETURN[later]};
    f = NIL = > RETURN[later];
    ENDCASE = > {f ← NIL; RETURN[goAhead]}};

DoSomeWorkUsingFile: PROCEDURE = {
  MakeBusy: ENTRY PROCEDURE RETURNS [file: MFile.Handle] = {
    busy ← TRUE; RETURN[f]};
  MakeUnbusy: ENTRY PROCEDURE RETURNS [file: MFile.Handle] = {
    busy ← FALSE;
    IF pleaseFree THEN {file ← f; f ← NIL; pleaseFree ← FALSE}
    ELSE file ← NIL};
  file: MFile.Handle = MakeBusy[];
  IF file = NIL THEN ERROR FileNotAvailable;
  -- do the work using file
  IF (file ← MakeUnbusy[]) # NIL THEN MFile.Release[file]};

Acquire[];
DO
  -- do some computing
  DoSomeWorkUsingFile[ ! FileNotAvailable = > EXIT];
```

```
    -- do some more computing
    ENDLOOP;
DoneWith[] ;
```

47.5.2 Notification

Some clients wish to be notified whenever a file becomes available for access. For instance, a cache may wish to know whenever there is a new version of one of its files; that is, whenever one of its files becomes available for **readOnly** access. If a client gives up a file because its **PleaseReleaseProc** was called, it may wish to be notified when the file is available again so it can resume using it. Clients ask to be notified by calling **AddNotifyProc** with the file name and access of interest, and add a **NotifyProc** to be called when the file becomes available.

Notification is performed by a special process in the file system. The file system maintains a list of files that are eligible for notification, and the notification process examines each file in the list. The notification process first checks whether a **NotifyProc** is interested in the file; that is, whether the file name matches the name in the filter and whether the access in the filter corresponds to a recent access transition on the file. If the **NotifyProc** matches, the notification process checks whether it can obtain the filter's access on the file. (It may not be possible because some previous **NotifyProc** has created a conflicting handle on the file; also, if several **NotifyProc**s want to know when they can get **writeOnly** access to a file, only one of them will actually succeed).

There is no guarantee about the order of notification; in particular, files may be released in one order and notification may take place in the other. There is also no guarantee about how quickly notification will take place after a file is released, since the notification takes place in another process. Because the notification process checks whether the filter access is available before calling a **NotifyProc**, a **NotifyProc** may not be called for every transition it is interested in.

Like **PleaseReleaseProc**, **NotifyProc**s are called by a separate process from the client process, so the client must protect itself from the effects of concurrent processing. Common data must be monitored. Furthermore, the client must not make any assumptions about the relative timing of file system manipulations. If a client releases a file in one statement and adds a **NotifyProc** on that file in the next, the file may in fact have been acquired and released between the two statements, and the client will miss the notification of this state change.

To avoid deadlock with the file system, the **NotifyProc** should not directly or indirectly call **AddNotifyProc** or **RemoveNotifyProc**. The boolean result of the **NotifyProc** may be used to allow the **NotifyProc** to remove itself from the notify list.

The following simple example of a **NotifyProc** and a **PleaseReleaseProc** shows a simple-minded module managing a single file. It it is willing to release the file if it is not in use, but wishes to be notified when the file is available again.

```
fileName: LONG STRING: = ...;
f: MFile.Handle;
useCount: CARDINAL;
```

```
Acquire: PUBLIC ENTRY PROCEDURE RETURNS [MFile.Handle] = {
  IF f # NIL THEN useCount ← useCount + 1;
  RETURN[f]};

Release: PUBLIC ENTRY PROCEDURE = {useCount ← useCount - 1};

Initialize: PUBLIC ENTRY PROCEDURE = {
  f ← MFile.Acquire[
    name: fileName, access: readOnly, release: [proc: MyReleaseProc], !
      MFile.Error = > {
        MFile.AddNotifyProc[
          proc: MyNotifyProc,
          filter: [name: fileName, access: readOnly]];
        f ← NIL;
        CONTINUE}]];
  useCount ← 0};

MyReleaseProc: ENTRY MFile.PleaseReleaseProc = {
  IF useCount # 0 THEN RETURN[no];
  f ← NIL;
  MFile.AddNotifyProc[
    proc: MyNotifyProc, filter: [name: fileName, access: readOnly]];
  RETURN[goAhead]}};

MyNotifyProc: ENTRY MFile.NotifyProc = {
  removeNotifyProc ← TRUE;
  f ← MFile.CopyFileHandle[
    file: file, access: readOnly, release: [proc: MyReleaseProc]!
      MFile.Error = > {
        f ← NIL;
        removeNotifyProc ← FALSE;
        CONTINUE}]};

-- main line code
Initialize[];
DO
   Acquire[];
   -- do some computing
   Release[]
   ENDLOOP;
```

48

MFileProperty

The **MFileProperty** interface is a constants-only definitions file that contains the list of the registered client file property numbers.

48.1 Types

None.

48.2 Constants and data objects

MFileProperty.AdobeReportSortTime: CARDINAL = ...;

MFileProperty.Checksum: CARDINAL = ...;

MFileProperty.PropagationDate: CARDINAL = ...;

MFileProperty.RemoteName: CARDINAL = ...;

48.3 Signals and errors

None.

48.4 Procedures

None.

49

MLoader

The **MLoader** interface allows clients to load and start programs stored in files in the development environment file system. This facility is used in place of the Pilot loader facility because clients do not have direct access to file capabilities.

49.1 Types

```
MLoader.Handle: TYPE = LONG POINTER TO Object;

MLoader.Object: TYPE = ...;

MLoader.Options: TYPE = RECORD [codeLinks: BOOLEAN];
```

49.2 Constants and data objects

```
MLoader.defaultOptions: MLoader.Options = [codeLinks: TRUE];
```

49.3 Signals and errors

```
MLoader.Error: ERROR = [code: ErrorCode, string: LONG STRING];

MLoader.ErrorCode: TYPE = {
  invalidParameters, missingCode, badCode, exportedTypeClash, lookupFailure, gftFull,
  loadStateFull,insufficientAccess, alreadyStarted, other};
```

invalidParameters	the file is an invalid configuration.
missingCode	code was not copied into the file when it was bound.
badCode	code was for the wrong machine.
exportedTypeClash	code contains conflicting exported type implementation.
lookupFailure	reserved for future use.
gftFull	no room in the Global Frame Table.

loadStateFull	reserved for future use.
insufficientAccess	file does not have readOnly access.
alreadyStarted	handle has already been started.
other	implementation error.

```
MLoader.VersionMismatch: SIGNAL [module: LONG STRING];
```

The VersionMismatch signal is raised when an interface is exported with one version and imported with another. The parameter is the name of the interface. If this signal is resumed, the item from the imported version remains unbound.

49.4 Procedures

```
MLoader.HandleFromProgram: PROCEDURE [PROGRAM] RETURNS [MLoader.Handle];
```

The HandleFromProgram procedure returns the handle for a loaded program that was loaded by Load or Run. It returns NIL if no handle can be found.

```
MLoader.Load: PROCEDURE [
  file: MFile.Handle, options: MLoader.Options ← MLoader.defaultOptions]
  RETURNS [MLoader.Handle];
```

The Load procedure requires an MFile.Handle with (exactly) readOnly access. It loads the file with the options passed in and returns a handle that can be used by Start or Unload. If options = MLoader.defaultOptions, any module for which code links were requested during binding will be loaded with external links in its code rather than its frame. Ownership of the MFile.Handle is transfered to the MLoader package. If the client wishes to maintain control of the file, it must call MFile.CopyFileHandle before calling Load. This procedure may raise MLoader.VersionMismatch or MLoader.Error[..., insufficientAccess, gftFull, badCode, invalidParameters, missingCode, exportedTypeClash, other, ...].

```
MLoader.Run: PROCEDURE [
  file: MFile.Handle, options: MLoader.Options ← MLoader.defaultOptions]
  RETURNS [MLoader.Handle];
```

The Run procedure is equivalent to a Load followed by a Start. This procedure may raise MLoader.VersionMismatch or MLoader.Error[..., insufficientAccess, gftFull, badCode, invalidParameters, missingCode, exportedTypeClash, alreadyStarted, other, ...].

```
MLoader.Start: PROCEDURE [MLoader.Handle];
```

The Start procedure starts a handle that has been loaded by Load. This procedure may raise MLoader.Error[..., alreadyStarted, other, ...].

```
MLoader.Unload: PROCEDURE [MLoader.Handle];
```

The Unload procedure unloads a loaded file that has been loaded by Load or Run. This procedure may raise MLoader.Error[other].

50

MSegment

The **MSegment** interface supports file mapping to spaces in virtual memory called *segments*. Although most of its operations have direct counterparts in the **Space** interface, **MSegment** is used because clients of the Xerox Development Environment file system do not have access to **File.Files**. For more information on these operations, consult the documentation on **Space** in the *Pilot Programmer's Manual*.

Addressing data pages through the **MSegment** interface is zero-origin. Only files in the Xerox Development Environment file system can have segments created on them.

A segment is created and associated with a portion of a file by the **Create** operation (see its declaration below). The new segment can be used to read and modify the contents of the file (depending on the **MFile.Access** of the file handle passed to **Create**) because the file is the "backing store" for the segment.

Nothing in the **MSegment** interface will change the size of the backing file. If a client wishes to change the size of a file, it should first call **MFile.SetLength**. One situation in which this must be done, for example, is when a client creates a file via **MFile.Acquire** with a large physical size hint and uses **MSegment** to initialize its contents. Since the file is physically large (although logically empty), a segment can be created on it and written into. However, if **MFile.SetLength** is not called, the logical length of the file does not change, and it will appear to later users as if the file were empty.

50.1 Types

```
MSegment.Handle: TYPE = LONG POINTER TO MSegment.Object;

MSegment.Object: TYPE;

MSegment.PleaseReleaseProc: TYPE = PROCEDURE [
  segment: MSegment.Handle, instanceData: LONG POINTER]
  RETURNS [MFile.ReleaseChoice];
```

Note that these types are different than those in **MFile**; in particular, the **Handle** is an **MSegment.Handle**, not an **MFile.Handle**. Each owner of an **MSegment** is notified when some other client wishes to have access to the **MFile.Handle** in a way that conflicts with the original use. If the **ReleaseData.proc** is **NIL**, the new agent is denied access to the file. As

with **MFile**, **MSegment.Delete** cannot be issued from the **PleaseReleaseProc** directly, and the client must synchronize carefully. (See **MFile** for more discussion on **PleaseReleaseProcs** and **Space** in the *Pilot Programmer's Manual* for discussion of **SwapUnitOption**

```
MSegment.ReleaseData: TYPE = RECORD [
  proc: MSegment.PleaseReleaseProc ← NIL,
  clientInstanceData: LONG POINTER ← NIL];

MSegment.SwapUnitOption: TYPE = RECORD [
  body: SELECT tag: MSegment.SwapUntilType FROM
    unitary => NULL,
    uniform => [size: MSegment.SwapUnitSize],
    irregular => [sizes: MSegment.SwapUnitSequence]
    ENDCASE];

MSegment.SwapUnitSequence: TYPE = LONG POINTER TO MSegment.SwapUnitSequenceObject;

MSegment.SwapUnitSequenceObject: TYPE = RECORD [
  swap: SEQUENCE length: CARDINAL OF MSegment.SwapUnitSize];

MSegment.SwapUnitSize: TYPE = Environment.PageCount;
```

A **SwapUnitSize** specifies the size in pages of the uniform swap units to be used.

```
MSegment.SwapUnitType: TYPE = {unitary, uniform, irregular};
```

50.2 Constants and data objects

```
MSegment.defaultPages: Environment.PageCount = ...;

MSegment.defaultSwapUnits: MSegment.SwapUnitOption = ...;
```

If the **defaultSwapUnits** value is used, the swap unit size defaults to 1,2, or 4 pages, depending on whether the size of the segment is less than 11 pages, between 11 and 50 pages, or greater than 50 pages.

```
MSegment.dontChangeFile: MFile.Handle = ...;

MSegment.dontChangeFileBase: File.PageNumber = ...;

MSegment.dontChangePages: Environment.PageCount = ...;

MSegment.dontChangeReleaseData: ReleaseData = ...;
```

50.3 Signals and errors

```
MSegment.Error: SIGNAL [segment: MSegment.Handle, code: MSegment.ErrorCode];

MSegment.ErrorCode: TYPE = MACHINE DEPENDENT {
  zeroLength(0), insufficientVM, noSuchSegment,
  sharedSegment, baseOutOfRange, conflictingAccess,
```

illegalAccess, invalidFile, dataSegmentNeedsPages,noRoomOnVolume,
other(LAST[CARDINAL])}

zeroLength(0)	a zero-length segment cannot be created.
insufficientVM	there is not enough VM left to create the desired segment.
noSuchSegment	there is no segment containing the address or base requested, or the segment is invalid.
sharedSegment	the segment you are resetting is shared with some other client.
baseOutOfRange	a segment cannot have a base larger than File.lastPageNumber.
conflictingAccess	the requested access of the file cannot be obtained.
illegalAccess	the file access is illegal for the operation.
invalidFile	an invalid MFile.Handle has been used.
dataSegmentNeedsPages	the pages parameter may not be defaulted when creating a data segment.
noRoomOnVolume	there is not enough free space on the volume to map the segment.
other	implementation error.

50.4 Procedures

```
MSegment.Activate: PROCEDURE [segment: MSegment.Handle];
```

The **Activate** procedure is called to indicate that the segment is likely to be referenced soon and that Pilot should begin swapping it in. (See **Space.Activate**.)

```
MSegment.Address: PROCEDURE [segment: MSegment.Handle] RETURNS [LONG POINTER];
```

The **Address** procedure returns the virtual memory address of the start of the segment. **Address** should be called after the segment is modified by **MSegment.Reset**, as well as when it is created. This procedure may raise **MSegment.Error[noSuchSegment]**.

```
MSegment.AddresstoSegment: PROCEDURE [pointer: LONG POINTER] RETURNS
  [MSegment.Handle];
```

The **AddresstoSegment** procedure returns the smallest segment containing the virtual memory address. This procedure may raise **MSegment.Error[noSuchSegment]**.

```
MSegment.Base: PROCEDURE [
    segment: MSegment.Handle] RETURNS [Environment.PageNumber];
```

The **Base** procedure returns the virtual memory page number containing the start of the segment. **Base** should be called after the segment is modified by **MSegment.Reset**, as well as when it is created. This procedure may raise **MSegment.Error[noSuchSegment]**.

```
MSegment.BasetoSegment: PROCEDURE [
  page: Environment.PageNumber] RETURNS [MSegment.Handle];
```

The **BasetoSegment** procedure returns the smallest segment containing the virtual memory page number. This procedure may raise **MSegment.Error[noSuchSegment]**.

```
MSegment.CopyIn: PROCEDURE [
  segment: MSegment.Handle, file: MFile.Handle, fileBase: File.PageNumber, count:
  Environment.PageCount];
```

The **CopyIn** procedure copies data into the **segment** from the **file** starting at page **fileBase** for **count** pages. Unlike **Create**, **CopyIn** does not own the **file** when it is done. This procedure may raise **MSegment.Error[..., zeroLength, noSuchSegment, baseOutOfRange, illegalAccess, invalidFile, ...]**

```
MSegment.CopyOut: PROCEDURE [
  segment: MSegment.Handle, file: MFile.Handle, fileBase: File.PageNumber, count:
  Environment.PageCount];
```

The **CopyOut** procedure copies data from the **segment** into the **file** starting at page **fileBase** for **count** pages. It does not own the **file** when it is done. This procedure may raise **MSegment.Error[...,zeroLength, noSuchSegment, baseOutOfRange, illegalAccess, invalidFile, ...]**.

```
MSegment.CopySegment: PROCEDURE [
  segment: MSegment.Handle] RETURNS [newSegment: MSegment.Handle];
```

The **CopySegment** procedure permits a segment to be shared by different programs. Shared segments cannot be modified by **MSegment.Reset**. This procedure may raise **MSegment.Error[noSuchSegment]**.

```
MSegment.Create: PROCEDURE [
  file: MFile.Handle ← NIL, release: MSegment.ReleaseData,
  fileBase: File.PageNumber ← 0, pages: MSegment.PageCount ← defaultPages,
  swapInfo: MSegment.SwapUnitOption ← defaultSwapUnits]
  RETURNS [segment: MSegment.Handle];
```

The **Create** procedure creates a segment. Operations on it are restricted by the **MFile.Access** associated with the file that is passed in. To create a segment, **readOnly** or **readWrite** access to the file is needed. If this operation succeeds, ownership of the file is passed to the **MSegment** package. If the client wishes to maintain control of the file, it must call **MFile.CopyFileHandle** before calling **MSegment.Create**. The segment will be **pages** long; if **pages** is **defaultPages**, the segment will be the logical size of the file. An important special case: if **file** is **NIL**, the segment will be a data segment backed by a temporary file. It is possible to create a segment on nonexistent file pages; that is, **fileBase** + **pages** may be

larger than the number of pages in the file. However, if the client tries to reference such pages, an address fault will result. This procedure may raise MSegment.Error[..., zeroLength, insufficientVM, baseOutOfRange, illegalAccess, invalidFile, dataSegmentNeedsPages, noRoomOnVolume, other, ...].

```
MSegment.Deactivate: PROCEDURE [segment: MSegment.Handle]; (
```

The Deactivate procedure is called to indicate that the segment is not likely to be referenced soon and that Pilot can swap it out. (See Space.Deactivate.)

```
MSegment.Delete: PROCEDURE [segment: MSegment.Handle];
```

The Delete procedure deletes the segment created by MSegment.Create or MSegment.CopySegment. The virtual memory occupied by this segment is freed and the segment object is released. This procedure may raise MSegment.Error[noSuchSegment].

```
MSegment.EquivalentSegments: PROCEDURE [seg1, seg2: MSegment.Handle] RETURNS [BOOLEAN];
```

The EquivalentSegments procedure checks whether two segments refer to the same pages of the same file. It returns TRUE if both arguments are NIL, or if both are segments on the same span of pages of the same file.

```
MSegment.ForceOut: PROCEDURE [segment: MSegment.Handle];
```

The ForceOut procedure forces out the segment; that is, writes its dirty pages to disk. It does not return until all output is complete. (See Space.ForceOut.) This procedure may raise MSegment.Error[noSuchSegment].

```
MSegment.FreePages: PROCEDURE [base: LONG POINTER];
```

The FreePages procedure deallocates a page-aligned block allocated with MSegment.GetPages.

```
MSegment.FreeWords: PROCEDURE [base: LONG POINTER];
```

The FreeWords procedure deallocates a page-aligned block allocated with MSegment.GetWords.

```
MSegment.GetFile: PROCEDURE [segment: MSegment.Handle] RETURNS [MFile.Handle];
```

The GetFile procedure returns the file handle on which this segment was created. This procedure may raise MSegment.Error[noSuchSegment].

```
MSegment.GetFileBase: PROCEDURE [
  segment: MSegment.Handle] RETURNS [File.PageNumber];
```

The GetFileBase procedure returns the starting page in the file of this segment. This procedure may raise MSegment.Error[noSuchSegment].

```
MSegment.GetFilePages: PROCEDURE [
  segment: MSegment.Handle] RETURNS [File.PageCount];
```

The **GetFilePages** procedure returns the number of physical data pages in the file on which this segment was created. It may raise **MSegment.Error[noSuchSegment]**.

```
MSegment.GetPages: PROCEDURE [npages: CARDINAL] RETURNS [base: LONG POINTER];
```

The **GetPages** procedure allocates a page-aligned block containing a specified number of pages. This block must later be freed by **MSegment.FreePages**.

```
MSegment.GetReleaseData: PROCEDURE [
  segment: MSegment.Handle] RETURNS [MSegment.ReleaseData];
```

The **GetReleaseData** procedure returns the release data associated with this segment. This procedure may raise **MSegment.Error[noSuchSegment]**.

```
MSegment.GetWords: PROCEDURE [nwords: CARDINAL] RETURNS [base: LONG POINTER];(
```

The **GetWords** procedure allocates a page-aligned block containing at least a specified number of words. **MSegment.FreeWords** is used to free this block. (An integral number of pages will actually be allocated.)

```
MSegment.Kill: PROCEDURE [segment: MSegment.Handle];
```

The **Kill** procedure, which kills the mapped pages of a segment, is used when the current contents of the segment are not needed. If a word is read from a killed page, the page is not read from backing store. This is useful when the segment has just been created and the backing file does not contain any useful information. If the killed segment is deleted or reset, its pages are not written to disk. (See **Space.Kill**.) This procedure may raise **MSegment.Error[noSuchSegment]**.

```
MSegment.MakeReadOnly: PROCEDURE [segment: MSegment.Handle];
```

The **MakeReadOnly** procedure makes the segment read-only. (See **Space.ReadOnly**.) This procedure may raise **MSegment.Error[noSuchSegment]**.

```
MSegment.MakeWritable: PROCEDURE [segment: MSegment.Handle];
```

The **MakeWritable** procedure makes the segment writable. (See **Space.Writable**.) This procedure may raise **MSegment.Error[..., noSuchSegment, illegalAccess, ...]**.

```
MSegment.Pages: PROCEDURE [
    segment: MSegment.Handle] RETURNS [Environment.PageCount];
```

The **Pages** procedure returns the number of pages in the segment. This procedure may raise **MSegment.Error[noSuchSegment]**.

```
MSegment.PagesForWords: PROCEDURE [nwords: CARDINAL] RETURNS [CARDINAL];
```

The **PagesForWords** procedure returns the number of pages needed to hold **nwords** words.

```
MSegment.Reset: PROCEDURE [
  segment: MSegment.Handle,
  file: MFile.Handle ← MSegment.dontChangeFile,
  release: MSegment.ReleaseData ← MSegment.dontChangeReleaseData,
  fileBase: File.PageNumber ← MSegment.dontChangeFileBase,
  pages: Environment.PageCount ← MSegment.defaultPages,
  swapInfo: MSegment.SwapUnitOption ← MSegment.defaultSwapUnits];
```

The **Reset** procedure changes the properties of the segment without creating a new segment object. Parameters that receive the default values are not changed. Since the segment's virtual memory location might be different, **MSegment.Address** should be called again to obtain the new starting address and **MSegment.Base** should be called for the new base. It is possible to reset to a segment on nonexistent file pages; that is, **fileBase + pages** may be larger than the number of pages in the file. If a client tries to reference such pages, however, an address fault results. This procedure may raise **MSegment.Error[..., insufficientVM, noSuchSegment, sharedSegment, illegalAccess, baseOutOfRange, invalidFile, noRoomOnVolume, ...]**.

```
MSegment.SetReleaseData: PROCEDURE [
  segment: MSegment.Handle, release: MSegment.ReleaseData];
```

The **SetReleaseData** procedure sets the release data associated with the segment. It may raise **MSegment.Error[noSuchSegment]**.

50.5 Examples

The following program fragment reads and updates a data structure stored on the file "MyFile":

```
f: MFile.Handle;
seg: MSegment.Handle;
data: LONG POINTER TO MyData;
-- create a read/write segment on the file
f ← MFile.ReadWrite[name: "MyFile", release: [], type: binary];
seg ← MSegment.Create[file: f, release: []];
data ← MSegment.Address[seg];
-- now manipulate the data structure
data.updateCount ← data.updateCount + 1;
...
MSegment.Delete[seg];
```

Ownership of the file handle is passed to the segment by **Create**. Consequently, the file is released when the segment is deleted. If the client needs to retain access to the file, it must call **MFile.CopyFileHandle** before creating the segment.

It is also possible to create "data segments" that have temporary backing files by passing a **NIL** file handle to **Create**. Data segments are most often used with the **CopyIn** and **CopyOut** operations. These procedures copy data between a segment and a file, much like the read and write operations of traditional file systems. They do not create a permanent association between a segment and a file window, and are relatively fast.

In the program fragment that follows, a data segment is used as a buffer. It is created at the beginning of the program, and data is copied into it from several different files during program execution. Note that ownership of the file handle is not passed to the segment by **CopyIn**; each input file must be explicitly released by the client.

```
buffer: MSegment.Handle;
data: LONG POINTER TO ARRAY [0..0) OF Environment.Byte;
source: MFile.Handle;
buffer ← MSegment.Create[file: NIL, release: [], pages: 20];
data ← MSegment.Address[buffer];
source ← MFile.ReadOnly[name: "Input.data", release: []];
MSegment.CopyIn[segment: buffer, file: source, fileBase: 0, count: 20]; -- read one file's
data
MFile.Release[source]; -- done with input file now
-- process the data
FOR i: CARDINAL IN [0..(20*Environment.bytesPerPage)) DO
 IF data[i] = 0 THEN ...
 ENDLOOP;
...
MSegment.Delete[buffer];
```

51

MStream

The **MStream** interface implements a Pilot transducer for accessing a file as a positionable byte stream. Only files in the Xerox Development Environment directory may have **MStream**s created on them.

51.1 Types

```
MStream.Handle: TYPE = Stream.Handle;
```

An **MStream.Handle** is the same type as a **Stream.Handle**. Clients may pass streams obtained from some other source to the procedures; the error **MStream.Error[invalidHandle]** is raised in most instances.

```
MStream.PleaseReleaseProc: TYPE = PROCEDURE [
  stream: MStream.Handle, instanceData: LONG POINTER]
  RETURNS [MFile.ReleaseChoice];
```

The **PleaseReleaseProc** is similar to that of **MFile**. Each user of an **MStream** is notified when some other agent wishes to have access to the **MFile.Handle** in a way that conflicts with the original use. If the **ReleaseData.proc** is **NIL**, the new agent is denied access to the file. As with **MFile**, the stream cannot be destroyed from the **PleaseReleaseProc** directly, and the client must synchronize carefully. (See the documentation of **MFile.PleaseReleaseProc** and the discussion at the end of **MFile** for the semantics of release procedures.) A stream is released by calling **Stream.Delete**.

```
MStream.ReleaseData: TYPE = RECORD [
  proc: MStream.PleaseReleaseProc ← NIL, clientInstanceData: LONG POINTER ← NIL];
```

51.2 Constants and data objects

None.

51.3 Signals and errors

```
MStream.Error: ERROR [stream: Stream.Handle, code: MStream.ErrorCode];

MStream.ErrorCode: TYPE = MACHINE DEPENDENT {
  invalidHandle(0), indexOutOfRange, invalidOperation, fileTooLong, fileNotAvailable,
  invalidFile, other(LAST[CARDINAL])};
```

invalidHandle	an invalid stream handle has been passed to a file stream procedure.
indexOutOfRange	a client tried to extend a file without the proper access.
invalidOperation	a client tried to operate on a stream in a way conflicting with its access; for example, to write on a read-only stream.
fileTooLong	a client tried to extend a stream beyond 65,535 pages.
fileNotAvailable	a file cannot be acquired from **MFile** with the requested access.
invalidFile	an invalid **MFile.Handle** was used.

51.4 Procedures

```
MStream.BackupLog: PROCEDURE [
  stream: MStream.Handle, count: MFile.ByteCount] RETURNS [backedUp: MFile.ByteCount];
```

The **BackupLog** procedure permits a client to back up in a file of type **log**. The number of characters to be backed over is given by **count**; the number actually backed over is returned by **backedUp. BackupLog** may not back the file up past the point made available for reading by **SetLogReadLength**. (See also **SetLogReadLength**.) This procedure may raise **MStream.Error[..., invalidHandle, indexOutOfRange, invalidOperation,...]**.

```
MStream.Copy: PROCEDURE [from, to: Stream.Handle, bytes: MFile.ByteCount]
  RETURNS [bytesCopied: MFile.ByteCount];
```

The **Copy** procedure copies bytes either to or from an **MStream**. Either **from** or **to** must be a **Stream.Handle** obtained from the **MStream**, or the error **MStream.Error[...invalidHandle]** is raised. It is legal for **endOfStream** to be reached before **bytes** bytes can be copied. To copy the rest of a file, you might call **[] ← Copy[from, to, LAST[LONG CARDINAL]]**. If **from** is not an **MStream**, and if a call of **Stream.GetBlock[from, ...]** returns a **why** of **sstChange**, this procedure may raise **MStream.Error[other]**. It raises **MFile.Error[noRoomOnVolume]** if there is not enough room on the logical volume for the copied file.

```
MStream.Create: PROCEDURE [
  file: MFile.Handle, release: MStream.ReleaseData,
  options: Stream.InputOptions ← Stream.defaultInputOptions,
  streamBase: File.PageNumber ← 0]
  RETURNS [stream: MStream.Handle];
```

The **Create** procedure creates an **MStream**. If the PleaseReleaseProc **release.proc** passed in to **Create** is **NIL**, the stream and underlying file are not released. Note that any

MFile.PleaseReleaseProc previously associated with this MFile.Handle is discarded. Operations on the stream are restricted by the MFile.Access associated with the file that is passed in. Ownership of file is passed to the MStream package. If the client wishes to maintain control of the file, it must call MFile.CopyFileHandle before calling MStream.Create. The streamBase parameter indicates the starting page number for the stream. It permits a client to have a segment and a stream open on a file simultaneously, the segment on the first portion of the file and the stream on the remainder. This procedure may raise MStream.Error[..., invalidOperation, fileTooLong, invalidFile,...].

```
MStream.EndOf: PROCEDURE [stream: MStream.Handle] RETURNS [BOOLEAN];
```

The EndOf procedure returns TRUE if an MStream is at the end of the file. This procedure may raise MStream.Error[invalidHandle].

```
MStream.GetFile: PROCEDURE [stream: MStream.Handle] RETURNS [MFile.Handle];
```

The GetFile procedure returns MFile.Handle underlying an MStream. It can be used to examine properties of the file, etc. The file is still owned by the MStream. This procedure may raise MStream.Error[invalidHandle].

```
MStream.GetLength: PROCEDURE [
  stream: MStream.Handle] RETURNS [fileLength: MFile.ByteCount];
```

The GetLength procedure returns the current length of an MStream. The result is the current length of the file in bytes; it does not change the position of the stream. This procedure may raise MStream.Error[invalidHandle].

```
MStream.GetReleaseData: PROCEDURE [
  stream: MStream.Handle] RETURNS [release: MStream.ReleaseData];
```

The GetReleaseData procedure returns the ReleaseData associated with a stream. This procedure may raise MStream.Error[invalidHandle].

```
MStream.IsIt: PROCEDURE [stream: Stream.Handle] RETURNS [BOOLEAN];
```

The procedure IsIt returns TRUE if stream is a stream created by MStream that can be used in operations that require an MStream stream.

```
MStream.Log: PROCEDURE [name: LONG STRING, release: MStream.ReleaseData] RETURNS
[MStream.Handle];
```

The Log procedure acquires the file name with log access and then creates the stream. This procedure may raise MStream.Error[..., invalidOperation, fileTooLong, fileNotAvailable, ...]. (See also BackupLog and SetLogReadLength.)

```
MStream.ReadOnly: PROCEDURE [name: LONG STRING, release: MStream.ReleaseData] RETURNS
[MStream.Handle];
```

The ReadOnly procedure acquires the file name with readOnly access and then creates the stream. This procedure may raise MStream.Error[..., invalidOperation, fileTooLong, fileNotAvailable, ...].

```
MStream.ReadWrite: PROCEDURE [
  name: LONG STRING, release: MStream.ReleaseData, type: MFile.Type ← unknown]
  RETURNS [MStream.Handle];
```

The **ReadWrite** procedure acquires the file **name** with **readWrite** access and then creates the stream. This procedure may raise **MStream.Error[..., invalidOperation, fileTooLong, fileNotAvailable, ...]**.

```
MStream.SetLength: PROCEDURE [
  stream: MStream.Handle, fileLength: MFile.ByteCount];
```

The **SetLength** procedure changes the length of a file (access permitting). This operation sets the current position only if the file is made shorter than the old position. In that event, the current position is set to be the new end of the file. (See also **BackupLog**.)This procedure may raise **MStream.Error[..., invalidHandle, indexOutOfRange, invalidOperation, fileTooLong,...]**.

```
MStream.SetLogReadLength: PROCEDURE [
  stream: MStream.Handle, position: MFile.ByteCount];
```

The **SetLogReadLength** procedure makes parts of a file of type log available for reading. **Position** is the last position in the file that other clients may read. Owners of log files are encouraged to call this procedure in their **PleaseReleaseProc**s because it enables other clients to read the log file. **BackupLog** may not back the file up past the point made available for reading by **SetLogReadLength**. (See also **BackupLog**.) This procedure may raise **MStream.Error[..., invalidHandle, indexOutOfRange, invalidOperation, ...]**.

```
MStream.SetReleaseData: PROCEDURE [
  stream: MStream.Handle, release: MStream.ReleaseData];
```

The **SetReleaseData** procedure changes the **ReleaseData** associated with a stream. This procedure may raise **MStream.Error[invalidHandle]**.

```
MStream.ShareBlock: PROCEDURE [
  stream: MStream.Handle, start: MFile.ByteCount, length: CARDINAL]
  RETURNS [block: Environment.Block];
```

The **ShareBlock** procedure permits a client to use the mapped buffers of a file directly. To minimize mapping operations, the entire requested block may not be returned. However, at least one byte will be in the block returned if **start** is less than the current length of the file. Subsequent calls with updated values for **start** can be used to get at all the desired addresses of the file. The current position is set to one past the position of the last character in the block returned. The block that is returned is valid only until the next stream operation is executed on this stream. This procedure may raise **MStream.Error[..., invalidHandle, indexOutOfRange, ...]**.

```
MStream.WriteOnly: PROCEDURE [name: LONG STRING, release: MStream.ReleaseData, type:
MFile.Type] RETURNS [MStream.Handle];
```

The **WriteOnly** procedure acquires the file **name** with **writeOnly** access and then creates the stream. This procedure may raise **MStream.Error[..., invalidOperation, fileTooLong, fileNotAvailable, ...]**.

51.5 Stream-specific operations

delete: Stream.DeleteProcedure;

When this procedure is invoked and the stream has **writeOnly** access, the file may be shortened according to the following algorithm:

```
IF access = writeOnly AND Stream.GetPosition[s] # 0 THEN
  MStream.SetLength[s, Stream.GetPosition[s]];
```

get: Stream.GetProcedure;

This procedure may raise **Stream.EndOfStream** if the stream is positioned at the end of the file and the options specify **signalEndOfStream**. It may raise **Stream.ShortBlock** if the stream is positioned at the end of the file and the options specify **signalShortBlock**. This procedure raises **MStream.Error[invalidOperation]** if the stream was created with **writeOnly** access.

getByte: Stream.GetByteProcedure;

getWord: Stream.GetWordProcedure;

These procedures may raise **Stream.EndOfStream** if the stream is positioned at the end of the file. They raise **MStream.Error[invalidOperation]** if the stream was created with **writeOnly** access.

put: Stream.PutProcedure;

putByte: Stream.PutByteProcedure;

putWord: Stream.PutWordProcedure;

These procedures raise **MStream.Error[invalidOperation]** if the stream was created with **readOnly** access. They may raise **MFile.Error[noRoomOnVolume]** if the file needs to be grown and there is no room on the logical volume.

setPosition: Stream.SetPositionProcedure;

This procedure may raise **MStream.Error[indexOutOfRange]** if the stream was created with **readOnly** access and the new position is past the end of the file, or for any stream if the new position is greater than 33,553,920 (65, 535 pages).

setSST: Stream.SetSSTProcedure;

sendAttention: Stream.SendAttentionProcedure;

waitAttention: Stream.WaitAttentionProcedure;

These procedures have no effect.

52

MVolume

The **MVolume** interface exports an error raised by the **Supervisor** (see the **Supervisor** chapter of the *Pilot Programmer's Manual*).

52.1 Types

None.

52.2 Constants and data objects

None.

52.3 Signals and errors

MVolume.CloseAborted: ERROR;

Closing a volume (**Volume.Close**, as defined in the *Pilot Programmer's Manual*) under XDE raises the supervisor event **EventTypes.aboutToCloseVolume**, which can be vetoed. If a client vetos this event, the error **MVolume.CloseAborted** is raised by the implementation of **Volume.Close**.

52.4 Procedures

None.

VI

Sorting and searching

The sorting and searching interfaces, **BTree**, **GSort**, and **StringLookUp**, are largely self-explanatory and are of interest to most programmers.

The **BTree** package was used in implementing the XDE file and directory system. Programmers designing file management tools may want to study it.

VI.1 Interface abstracts

BTree implements a B-tree whose keys are **LONG STRING**s and values are **ARRAY**s of **CARDINAL**.(see D. Knuth, "Sorting and Searching," in *The Art of Computer Programming*, vol. 3, 473-79)

GSort provides a general package for sorting arbitrarily large amounts of data.

StringLookUp provides a facility for looking up an identifier in a list of names. It is particularly useful to programs that process `User.cm` sections and that permit users to abbreviate commands.

53

BTree

The **BTree** package implements a searching algorithm based on multiway tree branching called *B-trees* (see D. Knuth, "Sorting and Searching," in *The Art of Computer Programming*, vol. 3, 473-79). The keys are **LONG STRING**s and the values associated with them are **LONG DESCRIPTOR**s **FOR ARRAY OF CARDINAL**. The directories in the file system are implemented using **BTree**.

53.1 Types

```
BTree.Tree: TYPE = LONG POINTER TO BTree.TreeObject;

BTree.TreeObject: TYPE;

BTree.Value: TYPE = LONG DESCRIPTOR FOR ARRAY OF CARDINAL;

BTree.ValueSize: TYPE = [1..32);
```

A value is an uninterpreted array of words. Its size, specified in words and fixed for any given B-tree, must be in the range **ValueSize**.

53.2 Constants and data objects

```
BTree.defaultValueSize: BTree.ValueSize = 6;

BTree.maxNameLength: CARDINAL = 100;
```

A name is a string with a maximum length of 100 characters.

53.3 Signals and errors

```
BTree.ValueTooSmall: ERROR [tree: BTree.Tree];
```

If the **BTree** implementation needs to copy values to or from the B-tree and there is insufficient room in the destination, **ValueTooSmall** is raised. (See the individual procedures for more details.)

53.4 Procedures

```
BTree.Delete: PROCEDURE [tree: BTree.Tree];
```

Delete deletes a B-tree as well as the data space associated with it, causing the B-tree to be saved in its associated file.

```
BTree.Empty: PROCEDURE [tree: BTree.Tree] RETURNS [BOOLEAN];
```

Empty returns FALSE if there are any entries in the B-tree, and TRUE otherwise.

```
BTree.Find: PROCEDURE [tree: BTree.Tree, name: LONG STRING, value: BTree.Value]
  RETURNS [ok: BOOLEAN];
```

Find locates the B-tree entry corresponding to **name** and returns the associated value in **value**. If the entry does not exist, **ok** is returned FALSE. If **value** is too small to hold the associated value, BTree.**ValueTooSmall** is raised after the leftmost words of the value have been transferred into **value**.

```
BTree.GetInfo: PROCEDURE [
  tree: BTree.Tree] RETURNS [valueSize: BTree.ValueSize, file: MFile.Handle];
```

GetInfo returns the size of the values stored in a B-tree and the file associated with it. If there is no file associated with the B-tree, **file** is returned NIL.

```
BTree.GetNext: PROCEDURE [
  tree: BTree.Tree, name: LONG STRING, nextName: LONG STRING, value: BTree.Value,
  mask: LONG STRING ← NIL];
```

GetNext is a stateless enumerator of a B-tree. The string **nextName** is set to the name following **name** in alphabetical order in the B-tree. The value associated with **nextName** is returned in **value**. If the length of the string **name** is zero, **nextName** is set to the the first entry in the B-tree and the corresponding value is returned. If **name** is the last entry in the B-tree, the length of **nextName** is set to zero. If **name** is not in the B-tree, **nextName** is set to the name in the B-tree that would logically follow **name**. An optional mask may be specified with this procedure. This mask uses the standard syntax in which a # matches any one character and a * matches any string of characters, including the empty string. If **mask** is specified, **nextName** is set to the first name following **name** that matches **mask**, and the corresponding value is returned. If **value** is too small to hold the associated value, BTree.**ValueTooSmall** is raised after the leftmost words of the value have been transferred into **value**.

```
BTree.Insert: PROCEDURE [tree: BTree.Tree, name: LONG STRING, value: BTree.Value] RETURNS
[ok, noRoom: BOOLEAN];
```

Insert inserts the <**name**, **value**> pair into a B-tree. If the name is successfully inserted into the B-tree, **ok** is returned TRUE and **noRoom** is returned FALSE. If an entry with that name already exists in the B-tree, both **ok** and **noRoom** are returned FALSE. If the B-tree is too full to add the new entry, **ok** is returned FALSE and **noRoom** is returned TRUE. If **value** is larger than the value size associated with this B-tree, BTree.**ValueTooSmall** is raised after the leftmost words of the value have been inserted into the B-tree.

```
BTree.Make: PROCEDURE [
  file: MFile.Handle ← NIL, valueSize: BTree.ValueSize ← BTree.defaultValueSize ,
  reset: BOOLEAN ← FALSE] RETURNS [tree: BTree.Tree];
```

Make creates a B-tree. The file passed to **Make** is the file in which a B-tree is located. If either the file is newly created (that is, if the first word of the file is a zero) or if **reset** is **TRUE**, this file is initialized to an empty B-tree that contains values of size **valueSize**; otherwise, the file is assumed to be a previously created B-tree (and **valueSize** is ignored). The file is enlarged as the B-tree grows, up to 256 pages. The initial file size may be zero.

If the default value for the file is used, a data space of 256 pages is used to store a new B-tree. If the client wishes to create a temporary B-tree but does not wish to have the overhead of a 256-page space, the client should pass in to **Make** a Pilot temporary file created with **MFile.AcquireTemp**.

```
BTree.Remove: PROCEDURE PROCEDURE [
tree: BTree.Tree, name: LONG STRING, value: BTree.Value] RETURNS [ok: BOOLEAN];
```

Remove removes the entry **name** from a B-tree. If the entry was successfully removed, **ok** is returned **TRUE** and the associated value is returned in **value**. If the entry was not found in the B-tree, **ok** is returned **FALSE**. If **value** is smaller than the value size associated with this B-tree, **BTree.ValueTooSmall** is raised after the leftmost words of the value have been transferred into **value**.

```
BTree.SwapValue: PROCEDURE [
  tree: BTree.Tree, name: LONG STRING, oldValue, newValue: BTree.Value]
  RETURNS [ok: BOOLEAN];
```

The **SwapValue** procedure replaces the value associated with **name** with **newValue**. If the entry was found, **ok** is returned **TRUE** and the previous value is returned in **oldValue**. If the entry was not found in the B-tree, **ok** is returned **FALSE**. If **newValue** is larger than the value size associated with this B-tree, the inserted value will be truncated on the right. If **oldValue** is smaller than the value size associated with this B-tree, only the leftmost words of the previous value have been transferred into **oldValue**. If either or both of these conditions occur, **BTree.ValueTooSmall** is raised.

54

GSort

The **GSort** interface provides a handy package for sorting arbitrarily large amounts of data. The client program tells the sort package the maximum and expected size of records and provides procedures that the sort package can call for reading, writing, and comparing records. If all of the records can fit in memory, the sort algorithm is an *n log n* in-core sort–if not, up to three scratch files are created and a polyphase merge sort is used (see D. Knuth, "Sorting and Searching," in *The Art of Computer Programming*, vol. 3, 473-79). This interface is not exported from Tajo; it is implemented by file `Basics>GSortImpl.bcd` on the release directory.

54.1 Types

```
GSort.CompareProcType: TYPE = PROCEDURE [
  p1: LONG POINTER, p2: LONG POINTER] RETURNS [INTEGER];
```

One of the parameters passed to the sort package is a procedure that can be called with pointers to two records to compare them. The client program is expected to know how to compare them and to return an integer value in the following range.

Condition	value returned
Record at **p1** > Record at **p2**	>0
Record at **p1** < Record at **p2**	<0
Record at **p1** = Record at **p2**	0

```
GSort.GetProcType: TYPE = PROCEDURE [p: LONG POINTER] RETURNS [CARDINAL];
```

Another of the parameters passed to the sort package is a procedure that will be called to get the input records. The procedure is called with a pointer to a buffer area. It is the responsibility of the client program to place the input record into the buffer and then return to the sort package the actual size (in words) of the record read. The maximum allowable size of input record (and hence the size of the buffer) is specified by the client program when it calls the procedure **GSort.Sort**. If the **get** procedure returns a length larger than the maximum, the signal **GSort.RecordTooLong** is raised. (Of course, the client

program has already smashed the sort package's heap by writing outside the buffer, so you should not write programs that routinely expect this signal.)

```
GSort.PutProcType: TYPE = PROCEDURE [p: LONG POINTER, len: CARDINAL];
```

Another of the parameters passed to the sort package is a procedure that can be called with the records in sorted order. This procedure is passed a pointer to the record and the length that was returned by the **get** procedure when the record was first introduced to the sort process. It's not clear how useful the **len** parameter is, because the client program probably needs to know the length of records from the pointer to compare them properly.

```
GSort.Port: TYPE = MACHINE DEPENDENT RECORD [in, out: UNSPECIFIED];
```

The current version of Mesa does not have a suitable lightweight co-routine mechanism. The Mesa instruction set, on the other hand, allows co-routine operation of the sort package. This type is used with appropriate LOOPHOLEs for setting up the co-routine linkage. While the sort package does not require them, co-routines may simplify some programs that use **GSort**.

```
GSort.SortItemPort: TYPE = PORT [len: CARDINAL] RETURNS [p: LONG POINTER];
```

When the sort package is being run as a co-routine, the **SortItemPort** is called with the length of the current input record and returns a pointer to the buffer in which to build the next input record. To get started, the **SortItemPort** is LOOPHOLEd into a GSort.Port and its **out** field is set to GSort.Sort.

```
GSort.SortStarter: TYPE = PORT [
  nextItem: POINTER TO GSort.SortItemPort,
  put: GSort.PutProcType,
  compare: GSort.CompareProcType,
  expectedItemSize: CARDINAL ← 30, maxItemSize: CARDINAL ← 1000,
  pagesInHeap: CARDINAL ← 100] RETURNS [p: LONG POINTER];
```

When the sort package is being run as a co-routine with the producer of input records, the **SortItemPort** is LOOPHOLEd into a **SortStarter** to get the sort package started (the **out** field of the port must be initialized first). The **nextItem** parameter is a pointer to the same port being LOOPHOLEd. This call returns with a pointer to a buffer where the client procedure is to place the first input record. (The meaning of the other parameters is the same as for GSort.Sort.)

```
GSort.SortStopper: TYPE = PORT [len: CARDINAL ← 0];
```

When the sort package is being run as a co-routine, the **SortItemPort** is LOOPHOLEd into a GSort.SortStopper and called to tell the sort package that there are no more input records. After this call, the **put** procedure is called with the sorted output records.

54.2 Constants and data objects

None.

54.3 Signals and errors

```
GSort.RecordTooLong: ERROR;
```

This signal is raised when the **get** procedure returns a length that is greater than the size of the buffer. With any luck, the client only clobbered the sort package's heap.

54.4 Procedures

```
GSort.Sort: PROCEDURE [
  get: GSort.GetProcType, put: GSort.PutProcType,
  compare: GSort.CompareProcType,
  expectedItemSize: CARDINAL ← 30, maxItemSize: CARDINAL ← 1000, pagesInHeap:
  CARDINAL ← 100];
```

The **Sort** procedure is called to start up the sort package. The client passes in procedures that are called to obtain input records (**get**), to compare two records (**compare**), and to receive the sorted records (**put**). The sort package knows nothing about the contents of the records; it only knows that they will all be not greater than **maxItemSize** in length (they need not be of equal length). The amount of memory used for buffers and the tournament area is specified by **pagesInHeap**. The package uses the **expectedItemSize** hint to decide how to partition its use of memory.

In operation, the **get** procedure is called for each input record. These records are maintained in a sorted heap by calling **compare**. If the heap fills up before **get** returns a length of zero, runs of sorted records are written on temporary disk files. When there are no more input records, **get** returns a length of zero; **put** is then called with the sorted records, which are obtained either from the heap or by merging the scratch files into a single run. The maximum number of scratch files is three; they are deleted when the procedure **GSort.Sort** returns.

The re-entrant procedure **GSort.Sort** is also suitably protected by **UNWIND** catch phrases so that it cleans up (destroys its heap, deletes its scratch files, etc.) if it terminates abnormally. For example, the **put** procedure could raise a signal that is caught by the call on **Sort**, which does a **CONTINUE**.

54.5 Examples

The program fragment below shows how to use **GSort** to sort a text file alphabetically by lines.

```
SortLines: PROCEDURE [in, out: Stream.Handle] =
    BEGIN
    maxLineLength: CARDINAL = 1000;
    maxRecordSize: CARDINAL = String.WordsForString[maxLineLength];

    GetLine: GSort.GetProcType =
```

```
        BEGIN
        s: LONG STRING = p;
        s↑ ← [length: 0, maxlength: maxLineLength, text:];
        DO
            c: CHARACTER = Stream.GetChar[in ! Stream.EndOfStream = > EXIT];
            String.AppendChar[s, c];
            IF c = Ascii.CR THEN EXIT;
            ENDLOOP;
        IF s.length = 0 THEN RETURN [0]
        ELSE RETURN [String.WordsForString[s.length]];
        END;

    PutLine: GSort.PutProcType =
        BEGIN
        s: LONG STRING = p;
        block: Environment.Block = [
            blockPointer: LOOPHOLE[@s.text],
            startIndex: 0,
            stopIndexPlusOne: s.length];
        Stream.PutBlock[out, block];
        END;

    CompareLines: GSort.CompareProcType = {
        RETURN [String.CompareStrings[p1, p2]]};

    GSort.Sort [
        get: GetLine, put: PutLine, compare: CompareLines,
        expectedItemSize: String.WordsForString[80],
        maxItemSize: maxRecordSize];
    END; -- SortLines
```

It is sometimes inconvenient to write a procedure that can be called for the next input (a **GSort.GetProcType**), such as when the data to be sorted might be obtained by enumerating some complicated tree structure. Such enumerators are easy to write in a recursive descent manner in which the enumerator calls the sort package whenever it finds a record. The interface **GSort** contains type declarations that allow reasonably simple co-routine execution of the sort package. Below, the same **SortLines** procedure is written running the sort package as a co-routine:

```
SortLines: PROCEDURE [in, out: Stream.Handle] =
    BEGIN
    maxLineLength: CARDINAL = 1000;
    maxRecordSize: CARDINAL = String.WordsForString[maxLineLength];
    buffer: LONG STRING;

    OutToSort: GSort.SortItemPort;

    PutLine: GSort.PutProcType =
        BEGIN
        s: LONG STRING = p;
        block: Environment.Block = [
            blockPointer: LOOPHOLE[@s.text],
```

```
        startIndex: 0,
        stopIndexPlusOne: s.length];
    Stream.PutBlock[out, block];
    END;

CompareLines: GSort.CompareProcType = {
    RETURN [String.CompareStrings[p1, p2]]};

-- initialization of the port for co-routine linkage
LOOPHOLE[OutToSort, GSort.Port].out ← GSort.Sort;

-- get the sort package started. Its first call of "get" looks like a return of this call
buffer ← LOOPHOLE [OutToSort, GSort.SortStarter] [
    nextItem: @OutToSort,
    put: PutLine, compare: CompareLines,
    expectedItemSize: String.WordsForString[80],
    maxItemSize: maxRecordSize];
DO
    buffer↑ ← [length: 0, maxlength: maxLineLength, text:];
    DO
        c: CHARACTER = Stream.GetChar[in ! Stream.EndOfStream = > EXIT];
        String.AppendChar[buffer, c];
        IF c = Ascii.CR THEN EXIT;
        ENDLOOP;
    IF buffer.length = 0 THEN EXIT;

    -- the sort package thinks this call is a return from "get." We think its next call
    -- of "get" is the return of this call
    buffer ← OutToSort[String.WordsForString[buffer.length]];
    ENDLOOP;

-- the sort package thinks this call is a return of 0 from "get." This call returns when
-- the sort package returns from the "Sort " procedure
LOOPHOLE[OutToSort, GSort.SortStopper] [ ];
END; -- SortLines
```

55

StringLookUp

The **StringLookUp** interface provides a facility for looking up an identifier in a list of names. It is particularly useful for programs that process `User.cm` sections or that permit users to abbreviate commands.

55.1 Types

```
StringLookUp.GeneratorProcType: TYPE = PROCEDURE [buffer: LONG STRING];
```

A **GeneratorProcType** provides a way to get the elements of a list of names, one by one. The implementor of a **GeneratorProcType** can assume that **buffer** is not **NIL** and that it is long enough to hold any of the names it generates. Each time the generator is called, **buffer** contains the previous name generated; the generator should replace the contents of **buffer** with the next name in its list. On the initial call, **buffer.length** is zero. When the generator's list is exhausted, it should set **buffer.length** to zero. The string **buffer** is owned by the caller of the generator and should not be deallocated by the generator.

```
StringLookUp.Table: TYPE = ARRAY OF LONG STRING;
```

A **Table** provides a list of names.

```
StringLookUp.TableDesc: TYPE = LONG DESCRIPTOR FOR StringLookUp.Table;
```

55.2 Constants and data objects

```
StringLookUp.ambiguous: CARDINAL = StringLookUp.emptyKey - 1;
```

ambiguous is the index indicating that the look-up key matched more than one of the names in the list.

```
StringLookUp.emptyKey: CARDINAL = StringLookUp.noMatch - 1;
```

emptyKey is the index indicating that the look-up key was **NIL** or had zero length.

```
StringLookUp.noMatch: CARDINAL = ...;
```

noMatch is the index indicating that the look-up key did not match any of the names.

55.3 Signals and errors

None.

55.4 Procedures

```
StringLookUp.Initial: PROCEDURE [key, entry: LONG STRING, caseFold: BOOLEAN ← TRUE]
  RETURNS [matchLength: CARDINAL];
```

The **Initial** procedure compares two strings and returns the length of their common prefix. If **casefold** is TRUE, the case-shift of characters is ignored; that is, lower-case and upper-case instances of the same character are considered to be equivalent. If **casefold** is FALSE, they are considered to be different.

```
StringLookUp.InitialInternal: PROCEDURE [
  key, entry: LONG STRING, maxLength: CARDINAL, caseFold: BOOLEAN ← TRUE]
  RETURNS [matchLength: CARDINAL];
```

This procedure is the internal procedure used to implement **Initial**. It is provided as an accelerator in case the client can meet the requirements on the input: the caller must guarantee that neither **key** nor **entry** is NIL and that neither string is longer than **maxLength**. If these conditions are not met, the result is undefined and an address fault may occur. **InitialInternal** compares two strings and returns the length of their common prefix. If **casefold** is TRUE, the case-shift of characters is ignored; that is, lower-case and upper-case instances of the same character are considered to be equivalent. If **casefold** is FALSE, they are considered to be different.

```
StringLookUp.InTable: PROCEDURE [
  key: LONG STRING, table: StringLookUp.TableDesc, caseFold: BOOLEAN ← TRUE,
  noAbbreviation: BOOLEAN ← FALSE]
  RETURNS [index: CARDINAL];
```

The **InTable** procedure returns the index of the matching table entry if **key** matches exactly one of the entries in **table**. Otherwise, it returns **ambiguous**, **emptyKey**, or **noMatch**, as appropriate. If **casefold** is TRUE, the case-shift of characters is ignored; that is, lower-case and upper-case instances of the same character are considered to be equivalent. If **casefold** is FALSE, they are considered to be different. If **noAbbreviation** is TRUE, the comparison must yield an exact match. If **noAbbreviation** is FALSE, **key** may be a prefix of the table entry and still match it. If **noAbbreviation** is FALSE and **key** is both the prefix of one table entry and the exact match of another, the index of the exact match is returned rather than **ambiguous**.

```
StringLookUp.IsPrefix: PROCEDURE [
  maybePrefix, string: LONG STRING, caseFold: BOOLEAN ← TRUE]
  RETURNS [yes: BOOLEAN];
```

The **IsPrefix** procedure indicates whether **maybePrefix** is a prefix of **string**. If **casefold** is TRUE, the case-shift of characters is ignored; that is, lower-case and upper-case

instances of the same character are considered to be equivalent. If **casefold** is FALSE, they are considered to be different. Either **maybePrefix** or **string** may be NIL.

```
StringLookUp.UsingGenerator: PROCEDURE [
  key: LONG STRING, generator: GeneratorProcType, caseFold: BOOLEAN ← TRUE,
  noAbbreviation: BOOLEAN ← FALSE, bufferBytes: CARDINAL ← 500]
  RETURNS [index: CARDINAL];
```

The **UsingGenerator** procedure returns the index of the matching entry if **key** matches exactly one of the entries in the list generated by **generator**. Otherwise, it returns **ambiguous**, **emptyKey**, or **noMatch**, as appropriate. If **casefold** is TRUE, the case-shift of characters is ignored; that is, lower-case and upper-case instances of the same character are considered to be equivalent. If **casefold** is FALSE, they are considered to be different. If **noAbbreviation** is TRUE, the comparison must yield an exact match. If **noAbbreviation** is FALSE, **key** may be a prefix of the table entry and still match it. If **noAbbreviation** is FALSE and **key** is both the prefix of one table entry and the exact match of another, the index of the exact match is returned rather than **ambiguous**. **bufferBytes** indicates the maxiumum number of characters in any name that will be generated by **generator**. If **generator** generates a name longer than **bufferBytes** characters, the results are unspecified and are dependent on the implementation of the client-provided procedure **generator**.

```
StringLookUp.UsingGeneratorWithBuffer: PROCEDURE [
  key: LONG STRING, generator: GeneratorProcType, caseFold: BOOLEAN ← TRUE,
  noAbbreviation: BOOLEAN ← FALSE, buffer: LONG STRING]
  RETURNS [index: CARDINAL];
```

The **UsingGeneratorWithBuffer** procedure returns the index of the matching entry if **key** matches exactly one of the entries in the list generated by **generator**. Otherwise, it returns **ambiguous**, **emptyKey**, or **noMatch**, as appropriate. If **casefold** is TRUE, the case-shift of characters is ignored; that is, lower-case and upper-case instances of the same character are considered to be equivalent. If **casefold** is FALSE, they are considered to be different. If **noAbbreviation** is TRUE, the comparison must yield an exact match. If **noAbbreviation** is FALSE, **key** may be a prefix of the table entry and still match it. If **noAbbreviation** is FALSE and **key** is both the prefix of one table entry and the exact match of another, the index of the exact match is returned rather than **ambiguous**. **buffer** is the client-provided storage to be passed to **generator** for buffering the name. If **buffer** is NIL or **generator** generates a name longer than **buffer.length** characters, the results are unspecified and are dependent on the implementation of the client-provided procedure **generator**.

55.5 Examples

The following example demonstrates how the **StringLookUp** facilities can be used to look up a (possibly abbreviated) command in a command table:

```
MyCommands: TYPE = MACHINE DEPENDENT{
    alpha(0), beta, gamma, noMatch(StringLookUp.noMatch)};
commandTable: ARRAY MyOptions OF LONG STRING ← [
    alpha: "Alpha"L, beta: "Beta"L, gamma: "Gamma"L];
commands: StringLookUp.TableDesc = LOOPHOLE[DESCRIPTOR[commandTable]];
```

```
GetCommand: PROCEDURE [command: LONG STRING] RETURNS [index: MyOptions] =
    BEGIN
    index ← StringLookUp.InTable[key: command, table: commands];
    IF index = StringLookUp.ambiguous OR index = StringLookUp.emptyKey THEN
        index ← StringLookUp.noMatch;
    END;
```

For a use of the **StringLookUp** facilities for parsing `User.cm` entries, see the example at the end of the **CmFile** chapter.

VII

Program analysis

There is currently one interface to aid in program analysis: **DebugUsefulDefs**. It is a public interface that provides access to debugger information. Tools such as the Performance Tools use **DebugUsefulDefs** to get information about such things as stack allocation. A programmer designing a system-monitoring tool may want to use this interface.

VII.1 Interface abstract

DebugUsefulDefs provides access to some of CoPilot's basic debugging utilities and data structures. It also allows a client to display variables itself instead of using the debugger's default display routines. This interface is exported only by CoPilot.

56

DebugUsefulDefs

The **DebugUsefulDefs** interface provides access to some of CoPilot's basic debugging utilities and data structures. Many of these facilities act on data structures in the Mesa processor. They are provided in this interface for special debugging tools (such as the performance tools and **DebugHeap**); contact your local support group to discuss any application involving these procedures.

In addition, it is the interface to a facility known as **Printer**s. By writing and registering a **Printer**, clients can take over the debugger's display of variables of client-defined types instead of using the default display routines. The client can thus deal with situations that CoPilot cannot, such as interpreting **OVERLAID** variant records. It also allows the client to supress printing of irrelevant record fields.

The client's state can change considerably between invocations of the debugger. Tools that live inside the debugger should be careful about cacheing information between calls to its operations; there may be several days between calls, and the debugee need not even be in the same boot file. The most direct course is to validate the debugee's state on each operation; for example, call **DebugUsefulDefs.Frame** each time rather than save the global frame address.

56.1 Types

```
DebugUsefulDefs.Bits: TYPE = [0..Environment.bitsPerWord);
```

This type is used for describing the bit offset in a word for a client variable.

```
DebugUsefulDefs.FrameList: TYPE = LONG POINTER TO FrameSeq;

DebugUsefulDefs.FrameSeq: TYPE = RECORD [
  SEQUENCE COUNT: NATURAL OF DebugUsefulDefs.GFHandle];

DebugUsefulDefs.GFHandle: TYPE = PrincOps.GlobalFrameHandle;

DebugUsefulDefs.Handle: TYPE = LONG POINTER TO Object;
```

A **Handle** is a pointer to an object that describes a variable in the client's core image. Variables have the following interesting properties: type, address, size, and bit offset. There are procedures in this interface that return a **Handle**.

```
DebugUsefulDefs.Object: TYPE;
```

The storage for **Object**s and any values copied into them by **ReadValue** is owned by the debugger; it is freed between commands. The **BOOLEAN** returned is whether or not the **Printer** actually displayed the variable; if it is **FALSE**, CoPilot displays it. A **Printer** may be called as either the result of invoking the interpreter or the `Display Stack` command.

```
DebugUsefulDefs.Printer: TYPE PROCEDURE [DebugUsefulDefs.Handle] RETURNS [BOOLEAN];
```

56.2 Constants and data objects

```
DebugUsefulDefs.fileSW: READONLY Window.Handle;
```

This is the file subwindow containing `Debug.log`; it is set to **NIL** when CoPilot is deactivated.

```
DebugUsefulDefs.window: READONLY Window.Handle;
```

This is the tool window for CoPilot.

56.3 Signals and errors

```
DebugUsefulDefs.InvalidAddress: ERROR [address: LONG POINTER];
```

This is raised if a reference to the debugger client's memory is made using one of the **READ** or **WRITE** procedures that would have caused an address fault in the client program.

```
DebugUsefulDefs.InvalidFrame: ERROR [f: POINTER];
```

This error may be raised by any procedure that takes either a **GlobalFrameHandle** or **LocalFrameHandle**.

```
DebugUsefulDefs.InvalidNumber: ERROR [p: LONG POINTER];
```

This error may be raised by the interpreter when it has an expression it cannot convert to a number. The parameter is used internally by CoPilot.

```
DebugUsefulDefs.MultipleFrames: ERROR [list: DebugUsefulDefs.FrameList];
```

This error is raised only by **Frame** if there is more than one instance of a module; the argument is a sequence of **GlobalFrameHandle**s. The procedure **ConfigForFrame** may be used to determine which configurations the frames are in.

```
DebugUsefulDefs.NotFound: ERROR [s: LONG STRING];
```

This is raised whenever the debugger fails in an attempt to look up an identifier; the argument is the identifier.

```
DebugUsefulDefs.UserAborted: SIGNAL;
```

This is treated the same way as the predefined error, **ABORTED**; it may be raised by any procedure if you strike the **ABORT** key.

```
DebugUsefulDefs.WriteProtected: ERROR [page: CARDINAL];
```

This is raised if a **WRITE** to the debugger's client memory is made that would have caused a write-protect fault in the client program.

56.4 Procedures

```
DebugUsefulDefs.AddPrinter: PROCEDURE [
  type: LONG STRING, proc: DebugUsefulDefs.Printer];
```

This procedure allows you to take over display of all variables of a known type; **proc** is called whenever a variable of type **type** is about to be displayed. CoPilot's interpreter evaluates **type** at the beginning of each session and remembers the target type of the result. Unfortunately, **type** is not a simple type expression, but rather a statement evaluated by the interpreter; the type is extracted from the result. Any additional information, such as the address of a variable used when evaluating the statement, is ignored.

A good technique for debugging the string used in the call to **AddPrinter** is to actually try it out using the interpreter. For example, all **REAL**s could be intercepted by supplying the following **STRING** to **AddPrinter**: **0%(REAL)**

```
DebugUsefulDefs.ConfigForFrame: PROCEDURE [
  gf: DebugUsefulDefs.GFHandle, config: LONG STRING];
```

This procedure fills in the name of the configuration containing the **GlobalFrameHandle** passed in; it is useful for finding a particular instance of a module that has multiple copies. It may raise the error **InvalidFrame**.

```
DebugUsefulDefs.Copied: PROCEDURE [DebugUsefulDefs.GFHandle] RETURNS [BOOLEAN];
```

This procedure returns **TRUE** if the frame corresponding to the **GFHandle** was copied and **FALSE** otherwise.

```
DebugUsefulDefs.Enumerate: PROCEDURE[
  proc: PROCEDURE [DebugUsefulDefs.GFHandle] RETURNS [BOOLEAN]] RETURNS [
  gf: DebugUsefulDefs.GFHandle];
```

This procedure enumerates all **GFHandle**s. The client procedure (**proc**) can halt the enumeration by returning **TRUE**. **Enumerate** returns the **GFHandle** that the client stopped the enumeration on. If the client doesn't stop the enumeration before all **GFHandle**s are enumerated, **Enumerate** returns **NIL**..

```
DebugUsefulDefs.Frame: PROCEDURE [
  name: LONG STRING] RETURNS [DebugUsefulDefs.GFHandle];
```

This procedure looks up the **GFHandle** that corresponds to **name**; it is extremely useful for printers that need things like a **BASE POINTER** to display complicated data structures. The following code reads a **LONG BASE POINTER** out of CoPilot's client, **SomeProg**:

```
DIRECTORY
    SomeProg USING [basePtr],
    ...
MyPrinterImpl: PROGRAM ... SHARES SomeProg =
    BEGIN
    GetBase: PROCEDURE RETURNS [myBase: LONG POINTER] = {
        frame: POINTER TO FRAME[SomeProg] ←
            LOOPHOLE[DebugUsefulDefs.Frame["SomeProg"L]];
        DebugUsefuleDefs.ShortCopyRead[
            from: @frame.basePtr, nwords: SIZE[LONG BASE POINTER],
            to: @myBase]};
    ...
    END.
```

Frame may raise the errors **MultipleFrames** and **NotFound**.

```
DebugUsefulDefs.GetAddress: PROCEDURE [DebugUsefulDefs.Handle]
  RETURNS [base: LONG POINTER, offset: DebugUsefulDefs.Bits, there: BOOLEAN];
```

This procedure returns the address of a variable. **base** is its location in memory and **offset** may be non-zero if the variable is less than a word long (e.g., inside a record). If **there** is **TRUE**, the base is a pointer in the debugee's core image. The client can ensure that **there** will be **FALSE** by first calling **DebugUsefulDefs.ReadValue**, but should only do so if the variable is reasonably small.

```
DebugUsefulDefs.GetSize: PROCEDURE [DebugUsefulDefs.Handle]
  RETURNS [words: CARDINAL, bits: DebugUsefulDefs.Bits];
```

This procedure returns the size of a variable; **bits** may be non-zero only if **words** is zero.

```
DebugUsefulDefs.Interpreter: PROCEDURE [
  exp: LONG STRING, results: PROC [DebugUsefulDefs.Handle]];
```

This procedure invokes CoPilot's interpreter on a given string. Any resulting variable is described by a **Handle** passed to the **results** procedure. It may raise any of the errors defined in this interface.

```
DebugUsefulDefs.Lengthen: PROCEDURE [POINTER] RETURNS [LONG POINTER];
```

This procedure lengthens a short (MDS-relative) pointer in the debuggee's core image.

```
DebugUsefulDefs.LongCopyREAD: PROCEDURE [
  from: LONG POINTER, nwords: CARDINAL, to: LONG POINTER];
```

This procedure copies a block of memory from the debuggee's core image. It may raise the error **InvalidAddress** or **UserAborted**.

```
DebugUsefulDefs.LongCopyWRITE: PROCEDURE [
  from: LONG POINTER, nwords: CARDINAL, to: LONG POINTER];
```

This procedure writes a block of memory into the debuggee's core image. It may raise the error **InvalidAddress**, **WriteProtected**, or **UserAborted**.

```
DebugUsefulDefs.LongREAD: PROCEDURE [loc: LONG POINTER] RETURNS [val: UNSPECIFIED];
```

This procedure returns one word from the debuggee's core image. It may raise the error **InvalidAddress** or **UserAborted**.

```
DebugUsefulDefs.LongWRITE: PROCEDURE [loc: LONG POINTER, val: UNSPECIFIED];
```

This procedure writes one word into the debuggee's core image. It may raise the error **InvalidAddress**, **WriteProtected**, or **UserAborted**.

```
DebugUsefulDefs.Name: PROCEDURE [
  name: LONG STRING, gf: DebugUsefulDefs.GFHandle];
```

This procedure appends **name** with the module name whose **GlobalFrameHandle** is **gf**. It may raise the error **InvalidFrame** or the signal **String.StringBoundsFault**.

```
DebugUsefulDefs.Original: PROCEDURE [
  new: DebugUsefulDefs.GFHandle] RETURNS [old: DebugUsefulDefs.GFHandle];
```

This procedure returns a handle for the original (uncopied) frame of which this frame is a copy. If this frame was not copied, **Original** returns the **GFHandle** it was passed.

```
DebugUsefulDefs.ReadValue: PROCEDURE [DebugUsefulDefs.Handle];
```

This procedure copies a variable described by the **Handle** from the debuggee's core image into CoPilot's; it also fixes up the **Object** to reflect this copying. It should only be used on variables that are relatively small; large data structures should only be accessed with the **LongREAD** and **LongCopyREAD** procedures. CoPilot keeps these copies in its own heap; any variable up to about a page in size is safe to copy. The space is freed between commands.

```
DebugUsefulDefs.ShortCopyREAD: PROCEDURE [
  from: POINTER, nwords: CARDINAL, to: LONG POINTER];
```

This procedure operates like **LongCopyREAD** except that it takes a short (MDS-relative) pointer as the source. CoPilot lengthens the pointer and then performs the operation.

```
DebugUsefulDefs.ShortCopyWRITE: PROCEDURE [
  from: LONG POINTER, nwords: CARDINAL, to: POINTER];
```

This procedure operates like **LongCopyWRITE** except that it takes a short (MDS-relative) pointer as the destination. CoPilot lengthens the pointer and then performs the operation.

```
DebugUsefulDefs.ShortREAD: PROCEDURE [loc: POINTER] RETURNS [val: UNSPECIFIED];
```

This procedure operates like **LongREAD** except that it takes a short (MDS-relative) pointer as the source. CoPilot lengthens the pointer and then performs the operation.

```
DebugUsefulDefs.ShortWRITE: PROCEDURE [loc: POINTER, val: UNSPECIFIED];
```

This procedure operates like **LongWRITE** except that it takes a short (MDS-relative) pointer as the destination. CoPilot lengthens the pointer and then performs the operation.

```
DebugUsefulDefs.Started: PROCEDURE [GDebugUsefulDefs.FHandle] RETURNS [BOOLEAN];
```

This procedure returns **TRUE** if the module corresponding to the **GFHandle** has been started and **FALSE** otherwise.

```
DebugUsefulDefs.StringExpToDecimal: PROCEDURE [exp: LONG STRING]
  RETURNS [INTEGER];
```

This procedure converts an expression to an **INTEGER**. Any expression may be passed in. CoPilot invokes its interpreter if necessary. It may raise the error **InvalidNumber**.

```
DebugUsefulDefs.StringExpToLDecimal: PROCEDURE [exp: LONG STRING]
  RETURNS [LONG INTEGER];
```

This procedure converts an expression to a **LONG INTEGER**. It may raise the error **InvalidNumber**.

```
DebugUsefulDefs.StringExpToLNum: PROCEDURE [
  exp: LONG STRING, radix: CARDINAL] RETURNS [LONG UNSPECIFIED];
```

This procedure converts an expression to a long number; it uses **radix** as the default radix if one is not explicitly contained in **exp**. It may raise the error **InvalidNumber**.

```
DebugUsefulDefs.StringExpToNum: PROCEDURE [
  exp: LONG STRING, radix: CARDINAL] RETURNS [UNSPECIFIED];
```

This procedure converts an expression to a short number. It may raise the error **InvalidNumber**.

```
DebugUsefulDefs.StringExpToLOctal: PROCEDURE [exp: LONG STRING]
  RETURNS [LONG CARDINAL];
```

This procedure converts an expression to a **LONG CARDINAL**. It may raise the error **InvalidNumber**.

```
DebugUsefulDefs.StringExpToOctal: PROCEDURE [exp: LONG STRING] RETURNS [CARDINAL];
```

This procedure converts an expression to a **CARDINAL**. It may raise the error **InvalidNumber**.

```
DebugUsefulDefs.Text: Format.StringProc ;
```

This procedure displays text in CoPilot's window (**DebugUsefulDefs.fileSW**). It may raise the error **UserAborted**.

```
DebugUsefulDefs.Valid: PROCEDURE [DebugUsefulDefs.GFHandle] RETURNS [BOOLEAN];
```

This procedure returns TRUE if the **GFHandle** describes a valid global frame and FALSE otherwise.

56.5 Sample Printer

Once **StackPrinter** is loaded in CoPilot, **PrintStack** is called whenever the debugger wants to display a **StackObject**. Since **PrintStack** understands the format of **StackObjects**, it can show the complete contents of a **stack**, which CoPilot is unable to do because of the zero-length array. Note the type passed into **AddPrinter**:

```
LOOPHOLE[200000B, StackFormat$Stack]↑
```

The constant 200000B is simply a location that is always mapped; **AddPrinter**'s evaluation of this type does not actually read or write that location.

```
-- StackFormat.mesa - Last edit: Keith, October 21, 1980 10:30 PM
StackFormat: DEFINITIONS = {
    Stack: TYPE = LONG POINTER TO StackObject;
    StackObject: TYPE = RECORD [
        top: CARDINAL ← 0,
        max: CARDINAL ← 0,
        overflowed: BOOLEAN ← FALSE,
        stack: ARRAY [0..0) OF CARDINAL]}.

-- StackPrinter.mesa - Last Edited:
    -- Keith, October 21, 1980 10:38 PM
    -- Bruce, February 26, 1982 4:05 PM
DIRECTORY
Ascii USING [CR, SP],
DebugUsefulDefs USING [AddPrinter, GetAddress, Handle, LongREAD, ReadValue, Text],
Format USING [Char, Octal, StringProc],
StackFormat USING [Stack];

StackPrinter: PROGRAM IMPORTS DebugUsefulDefs, Format =
    BEGIN
    PrintRecord: PROC [here, there: StackFormat.Stack] = {
        out: Format.StringProc = DebugUsefulDefs.Text;
        lpStack: LONG POINTER TO CARDINAL ← LOOPHOLE[@there.stack];
        IF here.top = 0 THEN out["empty "L]
        ELSE
            FOR i: CARDINAL DECREASING IN [0..here.top) DO
                Format.Octal[out, DebugUsefulDefs.LongREAD[lpStack + i]];
                Format.Char[out, Ascii.SP];
                ENDLOOP;
        IF here.overflowed THEN out["(overflow!) "L];
        IF here.max = here.top THEN out["(full!)"L];
        Format.Char[out, Ascii.CR]};

    PrintStack: PROC[h: DebugUsefulDefs.Handle] RETURNS[BOOLEAN] = {
        address: StackFormat.Stack = DebugUsefulDefs.GetAddress[h].base;
```

```
        DebugUsefulDefs.ReadValue[h];
        PrintRecord[DebugUsefulDefs.GetAddress[h].base, address];
        RETURN[TRUE]};

    DebugUsefulDefs.AddPrinter[
        type: "LOOPHOLE[200000B, StackFormat$Stack] ↑ ",
        proc: PrintStack];
    END.
```

VIII

Miscellaneous

The TajoMisc chapter describes various facilities, including those to determine whether the ToolDriver is currently running and to wait a specified number of milliseconds. The **Version** interface supports determining the running boot file's version number. These interfaces are straightforward to use.

VIII.2 Interface abstracts

TajoMisc is a catch-all for public and semi-public Tajo utilities that did not fit logically into other interfaces.

Version provides the single procedure **Append**, used by various tools to construct heralds.

57

TajoMisc

The **TajoMisc** interface is a catch-all for public and semi-public Tajo utilities that did not fit logically into any of the other interfaces.

57.1 Types

None.

57.2 Constants and data objects

```
TajoMisc.toolDriverRunning: READONLY BOOLEAN;
```

toolDriverRunning can be polled to determine if the Tool Driver is currently running. Tools that can cause destructive changes to a large data base may wish to use this to restrict their operations when run by the Tool Driver instead of interactively by a user.

57.3 Signals and errors

None.

57.4 Procedures

```
TajoMisc.FindClippingWindow: PROCEDURE [
  Window.Handle] RETURNS [Window.Handle];
```

The **FindClippingWindow** procedure returns the clipping window of the tool window associated with some window. This is the only safe way to get at the clipping, because the clipping window will not be a child of the tool window if the tool is tiny. The argument may be a tool window, a clipping window, a subwindow, or the root window. If the argument is the root window, the root is returned.

```
TajoMisc.GetWindowManagerMenu: PROCEDURE RETURNS [Menu.Handle];
```

The **GetWindowManagerMenu** procedure returns the handle for the Window Manager menu.

```
TajoMisc.Quit: PROCEDURE [powerOff: BOOLEAN ← FALSE];
```

The **Quit** procedure lets a client stop Tajo and all other tools safely. Since access to this procedure is through the HeraldWindow's menu, it is expected that calls on this procedure will be rare. Consult your support personnel before using it.

```
TajoMisc.SetState: PROCEDURE [
  new: UserTerminal.State] RETURNS [old: UserTerminal.State];
```

The **SetState** procedure must be used rather than **UserTerminal.SetState** to change the state of the display bitmap because **UserTerminal.SetState** bypasses Tajo with disastrous consequences.

```
TajoMisc.SetToolDriverRunning:PROCEDURE [BOOLEAN];
```

The **SetToolDriverRunning** procedure allows you to change the value of **toolDriverRunning**. This procedure is provided for use by the Tool Driver. Other clients should not call this procedure.

```
TajoMisc.StartClient: PROCEDURE;
```

The **StartClient** procedure is an outward call that Tajo makes before starting the Notifier. Tajo checks to make sure that the procedure is bound before making the call. Clients may build their own boot files containing a procedure that will satisfy the **IMPORT** and have their modules started by Tajo as part of the normal startup sequence.

```
TajoMisc.WaitMilliSecs: PROCEDURE [msec: CARDINAL];
```

The **WaitMilliSecs** procedure allows a process to do a **WAIT** for a period of milliseconds without having to be in a convenient **MONITOR**. It returns within 1 second if **UserInput.UserAbort[NIL]** is **TRUE**.

```
TajoMisc.WaitSecs: PROCEDURE [secs: CARDINAL];
```

The **WaitSecs** procedure allows a process to do a **WAIT** for a period of seconds without having to be in a convenient **MONITOR**. It returns within 1 second if **UserInput.UserAbort[NIL]** is **TRUE**.

58

Version

The **Version** interface, which provides the single procedure **Append**, is used by various tools to construct heralds.

58.1 Types

None.

58.2 Constants and data objects

None.

58.3 Signals and errors

None.

58.4 Procedures

Version.Append: PROCEDURE [LONG STRING];

The **Append** procedure appends a version number (five characters long, with no leading or trailing blanks) to the string passed in. Tools that wish to provide more precise information about when they were built should use **Runtime.GetBcdTime** as well. If the string argument is **NIL**, no actions will be performed. If the length of the string passed in is not at least 5 less than the **maxLength** (that is, if there is not enough storage allocated for appending 5 more characters) the signal **String.StringBoundsFault** is raised. It will not be caught in **Version.Append**.

A

ExampleTool

ExampleTool is a tool that illustrates the features and techniques used in writing tools that run in the Xerox Development Environment. This appendix presents a description of the important features in ExampleTool as well as a code listing of the ExampleTool program.

A.1 Creation and start-up of ExampleTool

The user interacts with ExampleTool either via the Executive's command line or the standard tool window interface. To be able to type input to the Executive's command line, the tool must register a command with the Executive. The call to **Exec.AddCommand** is invoked in ExampleTool's initialization procedure (**Init**). The **proc** parameter, **ExampleToolCommand**, is responsible for eventually creating the tool window itself. Although no Executive command line processing is done in **ExampleToolCommand**, this would be the logical place to handle it. Some applications look first for command line input; if there is any, they do not create the tool window. Such applications are operating under the assumption that the user simply wants to type to the Executive and has no need for the tool interface.

The **unload** parameter sent to **Exec.AddCommand** must destroy the tool window before the program is unloaded; thus the tool writer cannot use the Executive's **defaultUnloadProc**.

To create the tool window interface, the procedure **Tool.Create** is called. Two important parameters in this call are **ClientTransition** and **MakeSWs**. **ClientTransition** is a procedure (of type **ToolWindow.TransitionProcType**) that is called whenever the tool changes state. **MakeSWs** is the procedure (of type **Tool.MakeSWsProc**) responsible for creating the subwindows and menus provided by the tool.

Since users may wish to use the tool with the ToolDriver, the necessary ToolDriver registration is performed in the **MakeSWs** procedure. All tool writers should incorporate this feature.

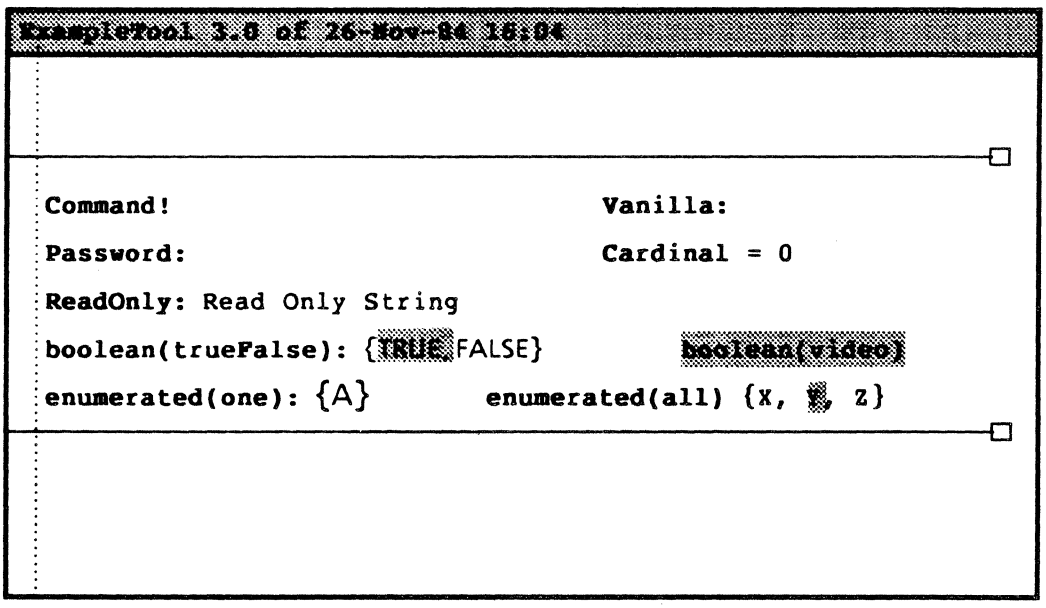


Figure 13.1: Example Tool

A.2 Tool states and storage management

A tool is always in one of three states: inactive, tiny, or active.

inactive the user is no longer interested in any of the tool's functions. When a tool enters the inactive state, all the resources it utilized should be released. Deactivating a tool causes a menu entry for that tool to be placed on the *Inactive* menu.

tiny the user is not interested in what the tool displays. When the tiny state is entered, the resources associated with the display state should be freed.

active the user wants access to the functions provided by the tool. Typically, all the storage for tool data is allocated when the active state is entered.

Whenever ExampleTool enters a new state, **ClientTransition** is called; the old state and the new state are passed to **ClientTransition**. If the old state is inactive, **ClientTransition** must perform the necessary reallocation of storage. Since ExampleTool uses its own heap, resource allocation involves re-creating ExampleTool's heap and re-allocating its data. When going to the inactive state, all items allocated from the heap are deallocated in **ClientTransition**. Those items include storage allocated for ExampleTool's form subwindow, its menu, and other global tool data. Once these items are deallocated, the heap itself is destroyed.

A.3 Data

The data needed by ExampleTool is contained within a single record structure. This enables the necessary storage to be allocated with a single call to **heap.NEW**. When the tool

is deactivated, the storage is freed by a call to **heap.FREE**, where **heap** is ExampleTool's private heap. This technique minimizes the memory used by the tool when it is inactive.

The fields within the record structure storing the tool data must be aligned at word boundaries. Therefore, this data must be stored in a **MACHINE DEPENDENT RECORD**. Addresses must be generated for the locations used to store values of enumerated and boolean items that are displayed in form subwindows (see "value" in the Enumerated items section and "switch" in the Boolean items section). Normally, the compiler allocates the minimum amount of space necessary for these items, and they are not word aligned. This **MACHINE DEPENDENT RECORD** allocates an entire word for each boolean and enumerated item.

A.4 Subwindows

ExampleTool uses three types of subwindows: file, message, and form. There are three other types of subwindows: text, string, and TTY.

text subwindows provide a way to view text from a wide variety of sources. The **TextSW** interface contains a comprehensive set of facilities for viewing and manipulating text independent of its source. File and string subwindows are specific types of text subwindows.

file subwindows are text subwindows whose backing store is a disk file. When creating a file subwindow, the client must specify the name of the file to be used as the backing store. (Backing store refers to the data object used to hold the information, typically text, that is displayed.)

string subwindows are text subwindows whose backing store is a **LONG STRING**. When creating a string subwindow, the client specifies the address of the **LONG STRING** used as the backing store.

message subwindows provide a simple way of posting feedback messages to the user.

form subwindows allow the user to indicate parameters, options, and commands for the tool to process. When creating a form subwindow, the client must specify a procedure (of type **FormSW.ClientItemsProcType**) that sets up the items within the form subwindow (see the next section on Form subwindows).

TTY subwindows provide traditional teletype interaction with the user.

When a tool is created, the specified **MakeSWsProc** is called. Within ExampleTool's **MakeSwProc**, subwindows are created by calls to the following **Tool** interface procedures: **MakeFileSW**, **MakeMsgSW** and **MakeFormSW**.

A.5 Form subwindows

The call to **Tool.MakeFormSW** must supply the procedure parameter (of type **FormSW.ClientItemsProcType**) specifying the items to be included in the form. In ExampleTool this procedure is called **MakeForm**. A procedure of this type must return an array descriptor; each element within this array is a record describing one of the items in the form subwindow. In ExampleTool, the variable **formItems** is declared as a **LONG POINTER TO ARRAY**, which eventually is **LOOPHOLE**d into the base of the descriptor returned by **MakeForm**. Although at first glance this seems like a confusing way to build an array

descriptor, ExampleTool was written this way to allow for indexing with an enumerated type (**FormIndex**), which is not only more readable but also simplifies the addition or deletion of items.

Items within a form subwindow have a "**tag**" and a "**place**" (in addition to other fields within the variant record describing that item). A **tag** is a client-supplied **LONG STRING** used as a label for an item. **place** is a record that specifies where in the form subwindow an item is located. **place** contains two integer fields: **x** and **y**. **x** specifies the number of bits from the left side of the subwindow that an item is shifted to the right; **y** specifies the number of lines from the top of the subwindow that an item is shifted down. **[0, 0]** places an item in the upper-left corner of the form subwindow. Each item in a form subwindow must be located below or to the right of the previous item.

Initial values for items in the form subwindow may be set by reading from the tool's section of the User.cm. The procedure **ProcessUserDotCM** in ExampleTool looks for initial values for **enumOne** and **enumAll** and is called from **ClientTransition** each time the tool is activated.

ExampleTool illustrates the use and construction of five principal types of items: command, string, enumerated, number, and boolean.

A.5.1 Command items

Command items enable the tool user to invoke a desired operation by selecting an item with the mouse. A command item can be created by a call to **FormSW.CommandItem**. An important parameter of **FormSW.CommandItem** is **proc**. **proc** is the procedure to be invoked when the user selects the comand item.

The first element in **formItems** is a command. When the user selects this command, the procedure **FormSWCommandRoutine** is executed. **FormSWCommandRoutine** forks a process for CommandRoutine, which is the procedure that does the actual work associated with the command. In general, command procedures that take a fair amount of time to execute should be forked. However, for short procedures it is not always worth the extra synchronization overhead.

Part of the synchronization is accomplished by using facilities in the **Supervisor**, **Event** and **EventTypes** interfaces. If the command process has been forked, there are two dangers to guard against while the process is running: a request to deactivate the tool and a request to swap to or from a debugger volume. ExampleTool registers an agent procedure on the supervisor event **Event.toolWindow**. This notifies ExampleTool whenever any active or tiny tool window is about to deactivate. **CheckDeactive**, which is the agent procedure that gets called when this event happens, first checks to see if the window that is about to deactivate is ExampleTool's. If so, a check on the boolean **toolData.commandIsRunning** (maintained in **CommandRoutine**) is made to determine whether the command process is currently running. If the command process is running, the request to deactivate is aborted; otherwise it is allowed to proceed. Since ExampleTool is interested only in being notified about deactivation when it is active, the dependency on **Event.toolWindow** is added and removed when ExampleTool changes states in **ClientTransition**.

Protecting the command process from an untimely swap to or from a debugger volume is accomplished by making a call to **Event.StartingProcess** upon entry to the process and by calling **Event.DoneWithProcess** when the command process has finished doing its work.

This will ensure that any request to swap will abort and display the string passed as a parameter to Event.StartingProcess if the swap request comes during execution of the command.

A.5.2 String items

String items allow a user to provide textual input. A string item can be created by a call to FormSW.StringItem. Five important parameters of FormSW.StringItem are string, readOnly, feedback, z , and inHeap.

string is a LONG POINTER to the LONG STRING to be used as the backing store for this item.

readOnly is a BOOLEAN. If readOnly is TRUE, the user cannot alter this item. If readOnly is FALSE, the user can edit this string item.

feedback describes the displayed appearance of characters within the string, either normal or password. normal means that the characters themselves are displayed. password means that a "*" is displayed in place of each character.

z signifies the heap from which the storage for the string must be allocated. The value of z should match the zone initially passed to Tool.MakeFormSW.

inHeap is a BOOLEAN used to indicate whether the FormSW interface should automatically grow the backing string when necessary. If inHeap is TRUE, FormSW takes care of allocating storage dynamically for the string. ExampleTool's strings are grown by FormSW.

A.5.3 Enumerated items

Enumerated items allow a user to select from a list of values. An enumerated item can be created by a call to FormSW.EnumeratedItem. Four important parameters of FormSW.EnumeratedItem are feedback, value, proc , and choices.

feedback describes how the item is to be displayed; the options are one and all.

all displays all the enumerated item's options (e.g., "tag: {a, b, c}"). Selecting an item within the curly brackets video-inverts that item.

one diplays only the selected item (e.g., "tag: {c}"). Depressing a mouse chord over the tag displays a menu containing all the allowable options; selecting an entry from this menu displays that selection.

value is the LONG POINTER indicating where the current value of the enumerated item is to be placed (see choices below).

choices specifies the options available for an enumerated item. The records decribing the enumerated options (of type FormSW.Enumerated) contain two fields: a LONG STRING and a value (of UNSPECIFIED type). The LONG STRING is used to describe the option; when the option is selected, the value is stored in the location pointed to by the value parameter described above.

In ExampleTool, a SEQUENCE of FormSW.Enumerated is created. Sequences are used so that backing storage for the enumerated items can be allocated from ExampleTool's private heap. Note also that the string associated with a particular choice is allocated locally, thus minimizing storage management overhead. To avoid passing the entire array of records, the **choices** parameter is actually a DESCRIPTOR of the ARRAY OF FormSW.Enumerated; therefore the sequence is made into a descriptor to satisify the type constraints of the FormSW.EnumeratedItem procedure.

The item with the tag "Boolean(TRUE FALSE)" in ExampleTool is an enumerated item, but a **choices** array descriptor is not constructed. Instead, a call to FormSW.BooleanChoices[] is used. **BooleanChoices** is a procedure that returns "**choices**" appropriate for creating an enumerated item whose options are TRUE and FALSE.

proc permits the client to specify a procedure that will be called when the user selects a value for an enumerated item. ExampleTool does not illustrate this feature.

A.5.4 Number items

Number items provide a way for clients to solicit arithmetic input. The item whose tag is "Cardinal" illustrates this feature.

A.5.5 Boolean items

Boolean items are form subwindow items with two possible states: TRUE and FALSE. The state is toggled when the user selects the item. When a boolean item is TRUE, it is displayed in inverse video. A boolean item can be created by a call to FormSW.BooleanItem. Two important parameters of **BooleanItem** are **switch** and **proc**.

switch is the LONG POINTER indicating where the current value of the boolean item is to be placed.

proc permits the client to specify a procedure that will be called when the user toggles the value of a boolean item. The item in ExampleTool whose tag is "boolean(video)" illustrates a boolean item that has a **proc** parameter.

A.6 Menus

Menus that appear when the user holds both mouse buttons down can be created by a call to Menu.Make. Menus are normally created in the client procedure that is resposible for creating subwindows (in ExampleTool this is procedure **MakeSWs**).

Three important parameters of Menu.Make are **name**, **strings**, and **mcrProc**.

name is the name to be placed at the top of the menu.

strings is a LONG DESCRIPTOR FOR ARRAY OF LONG STRINGS. The array contains the strings used as menu options. Procedure **MakeSWs** in ExampleTool

illustrates a simple way of creating the array of **LONG STRING**s and passing its descriptor.

mcrProc is the procedure that will be called when the user selects a menu item.

After a menu is created, the client must indicate the window(s) in which it should be present. This is accomplished by procedure **Menu.Instantiate**. Menus may be attached to the tool window or any of its subwindows. ExampleTool attaches its menu to the form subwindow.

When a window containing a menu is deactivated, the menu should be removed from that window and its storage freed. These operations generally occur in the transition procedure for a tool window before deallocating the tool data record structure. **Menu.Uninstantiate** removes a menu from a window, and **Menu.Free** deallocates the storage associated with it.

A.7 The ExampleTool program

```
DIRECTORY
  CmFile USING [
    Close, Error, FindSection, Handle, NextValue,
    TableError, UserDotCmOpen],
  Event USING [DoneWithProcess, Handle, StartingProcess, toolWindow],
  EventTypes USING [deactivate],
  Exec USING [AddCommand, ExecProc, OutputProc, RemoveCommand],
  Format USING [StringProc],
  FormSW USING [
    AllocateItemDescriptor, BooleanChoices, BooleanItem, ClientItemsProcType,
    CommandItem, Destroy, Enumerated, EnumeratedItem, ItemHandle, line0, line1,
    line2, line3, line4, LongNumberItem, NotifyProcType, ProcType, StringItem],
  Heap USING [Create, Delete],
  Menu USING [Free, Handle, Instantiate, Make, MCRType, Uninstantiate],
  Process USING [Detach, Pause, SecondsToTicks],
  Put USING [Line],
  Runtime USING [GetBcdTime],
  Supervisor USING [
    AddDependency, AgentProcedure, CreateSubsystem, EnumerationAborted,
    RemoveDependency, SubsystemHandle],
  String USING [AppendString, CopyToNewString],
  StringLookUp USING [InTable, noMatch, TableDesc],
  Time USING [Append, Unpack],
  Token USING [FreeTokenString, Item],
  Tool USING [
    Create, Destroy, MakeFileSW, MakeFormSW, MakeMsgSW,
    MakeSWsProc, UnusedLogName],
  ToolDriver USING [Address, NoteSWs, RemoveSWs],
  ToolWindow USING [Activate, TransitionProcType],
  Version USING [Append],
  Window USING [Handle];
```

```
ExampleTool: MONITOR

  IMPORTS
    CmFile, Event, Exec, FormSW, Heap, Menu, Process, Put, Runtime,
    Supervisor, String, StringLookUp, Time, Tool, Token, ToolDriver, ToolWindow,
    Version =
  BEGIN

  --TYPEs
  FormIndex: TYPE = {
    command, vanilla, password, readOnly, number, boolTF, boolVideo, enumOne,
    enumAll};
  StringNames: TYPE = {vanilla, password, readOnly};
  MenuIndex: TYPE = {postMessage, aCommand, bCommand};
  DataHandle: TYPE = LONG POINTER TO Data;

  Data: TYPE = MACHINE DEPENDENT RECORD [
    msgSW(0): Window.Handle ← NIL,
    fileSW(2): Window.Handle ← NIL,
    formSW(4): Window.Handle ← NIL,
    << Note: enumerateds and booleans must be word-boundary
    aligned as addresses for them must be generated>>
    commandIsRunning(6): BOOLEAN ← FALSE,
    switch1(7): BOOLEAN ← TRUE,
    switch2(8): BOOLEAN ← TRUE,
    enum1(9): Enum1[a..c] ← a,
    enum1Seq(10): LONG POINTER TO EnumSeq ← NIL,
    enum2(12): Enum2[x..z] ← y,
    enum2Seq(13): LONG POINTER TO EnumSeq ← NIL,
    number(15): LONG CARDINAL ← 0,
    menu(17): Menu.Handle ← NIL,
    strings(19): ARRAY StringNames OF LONG STRING ← ALL[NIL]];

  Enum1: TYPE = MACHINE DEPENDENT
    {a(0), b, c, noMatch(StringLookUp.noMatch)};
  Enum1Options: TYPE = Enum1[a..c];
  Enum2: TYPE = MACHINE DEPENDENT
    {x(0), y, z, noMatch(StringLookUp.noMatch)};
  Enum2Options: TYPE = Enum2[x..z];
  EnumSeq: TYPE = RECORD [seq: SEQUENCE n: CARDINAL OF FormSW.Enumerated];

  --Variable declarations

  --This data is for minimizing memory use when this tool is inactive
  toolData: DataHandle ← NIL;
  wh: Window.Handle ← NIL;
  heap: UNCOUNTED ZONE ← NIL;
  heraldName: LONG STRING ← NIL;
  inactive: BOOLEAN ← TRUE;
  agent: Supervisor.SubsystemHandle = Supervisor.CreateSubsystem[CheckDeactivate];
```

```
  CheckDeactivate: Supervisor.AgentProcedure =
    BEGIN
    IF event = EventTypes.deactivate AND
      wh # NIL AND wh = eventData
      AND toolData.commandIsRunning THEN {
      Put.Line[toolData.msgSW, "The tool is still processsing a command: aborting
deactivation"L];
      ERROR Supervisor.EnumerationAborted};
    END;

  --Example Tool Menu support routines

  MenuCommandRoutine: Menu.MCRType =
    BEGIN
    -- Do the tasks necessary to execute a menu command.
    mx: MenuIndex = VAL[index];
    SELECT mx FROM
      postMessage = > Put.Line[toolData.msgSW, "Message posted."L];
      aCommand = > Put.Line[toolData.fileSW, "A Menu command called."L];
      ENDCASE = > Put.Line[toolData.fileSW, "B Menu command called."L]
    END;

  --Example Tool FormSW support routines
  CommandRoutine: ENTRY PROCEDURE =
    BEGIN
    handle: Event.Handle ← Event.StartingProcess[
      "CommandRoutine in SampleTool is running"L];
    toolData.commandIsRunning ← TRUE;
    << The following statement represents 10 seconds of work done
    outside the monitor >>
    Process.Pause[Process.SecondsToTicks[10]];
    Put.Line[toolData.fileSW, "The Command Procedure has been called."L];
    toolData.commandIsRunning ← FALSE;
    Event.DoneWithProcess[handle];
    END;

  FormSWCommandRoutine: FormSW.ProcType =
    BEGIN
    --Do the tasks necessary to execute a form subwindow command.
    Process.Detach[FORK CommandRoutine];
    END;

  NotifyClientOfBooleanAction: FormSW.NotifyProcType =
    BEGIN
    << This procedure is called whenever a state change
    (user action) occurs to the boolean item of the Form subwindow. >>
    Put.Line[toolData.fileSW, "The Boolean Notify Procedure has been called."L]
    END;

  ProcessUserDotCM: PROCEDURE =
    BEGIN
    CMOption: TYPE = MACHINE DEPENDENT {
```

```
EnumOne(0), EnumAll, noMatch(StringLookUp.noMatch)};
DefinedOption: TYPE = CMOption[EnumOne..EnumAll];
cmOptionTable: ARRAY DefinedOption OF LONG STRING ← [
  EnumOne: "EnumOne"L, EnumAll: "EnumAll"L];
cmIndex: CMOption;

CheckType: PROCEDURE[h: CmFile.Handle, table: StringLookUp.TableDesc]
  RETURNS[index: CARDINAL] = CmFile.NextValue;

MyNextValue: PROCEDURE[
  h: CmFile.Handle,
  table: LONG DESCRIPTOR FOR ARRAY DefinedOption OF LONG STRING]
  RETURNS [index: CMOption] = LOOPHOLE[CheckType];

TranslateValueToEnum1: PROCEDURE[
  key: LONG STRING, table: LONG DESCRIPTOR FOR ARRAY
  Enum1Options OF LONG STRING, caseFold: BOOLEAN ← TRUE,
  noAbbreviation: BOOLEAN ← FALSE]
  RETURNS [index: CARDINAL] = LOOPHOLE[StringLookUp.InTable];

TranslateValueToEnum2: PROCEDURE[
  key: LONG STRING, table: LONG DESCRIPTOR FOR ARRAY
  Enum2Options OF LONG STRING, caseFold: BOOLEAN ← TRUE,
  noAbbreviation: BOOLEAN ← FALSE]
  RETURNS [index: CARDINAL] = LOOPHOLE[StringLookUp.InTable];

cmFile: CmFile.Handle ← CmFile.UserDotCmOpen[
  !CmFile.Error = > IF code = fileNotFound THEN GOTO return];
IF CmFile.FindSection[cmFile, "ExampleTool"L] THEN
DO
  SELECT
    (cmIndex ← MyNextValue[h: cmFile, table: DESCRIPTOR[cmOptionTable]
      ! CmFile.TableError = > RESUME]) FROM
    noMatch = > EXIT;
    EnumOne = > BEGIN
      enum1Table: ARRAY Enum1Options OF LONG STRING ← [
        a: "a"L, b: "b"L, c: "c"L];
      e1Index: Enum1Options;
      value: LONG STRING = Token.Item[cmFile];
      SELECT e1Index ← VAL[TranslateValueToEnum1[
        value, DESCRIPTOR[enum1Table], FALSE, TRUE]] FROM
        noMatch = > NULL;
        ENDCASE = > toolData.enum1 ← e1Index;
      [] ← Token.FreeTokenString[value];
      END;

    EnumAll = > BEGIN
      enum2Table: ARRAY Enum2Options OF LONG STRING ← [
        x: "x"L, y: "y"L, z: "z"L];
      e2Index: Enum2Options;
      value: LONG STRING = Token.Item[cmFile];
      SELECT e2Index ← VAL[TranslateValueToEnum2[
```

```
            value, DESCRIPTOR[enum2Table], FALSE, TRUE]] FROM
                noMatch => NULL;
                ENDCASE => toolData.enum2 ← e2Index;
            [] ← Token.FreeTokenString[value];
            END;
        ENDCASE;
    ENDLOOP;

  [] ← CmFile.Close[cmFile];
  EXITS return => NULL;
  END;

ClientTransition: Toolwindow.TransitionProcType =
  << This procedure is called whenever the tool's state is undergoing a
  user-invoked transition.
  This procedure demonstrates a technique that minimizes the memory
  requirements for an inactive tool. >>
  BEGIN
  SELECT TRUE FROM
    old = inactive =>
  BEGIN
      IF heap = NIL THEN InitHeap[];
      IF toolData = NIL THEN toolData ← heap.NEW[Data ← []];
      ProcessUserDotCM[];
      inactive ← FALSE;
      END;

    new = inactive =>
  BEGIN
      Supervisor.RemoveDependency[client: agent, implementor: Event.toolWindow];
      IF toolData # NIL THEN BEGIN
        FormSW.Destroy[toolData.formSW];
        Menu.Uninstantiate[menu: toolData.menu, window: toolData.formSW];
        Menu.Free[toolData.menu];
        heap.FREE[@toolData];
        heap.FREE[@heraldName];
        END;
      IF heap # NIL THEN KillHeap[];
      ToolDriver.RemoveSWs[tool: "ExampleTool"L];
      inactive ← TRUE;
      END;
    ENDCASE
  END;

Help: Exec.ExecProc =
  BEGIN
  OutputProc: Format.StringProc ← Exec.OutputProc[h];
  OutputProc[
    "This command activates the ExampleTool window. The ExampleTool is an
example of a 'Tool' that runs in Tajo. It demonstrates the use of a comprehensive set of
```

commonly used Tajo facilities. Specifically we present examples of the definition, creation, use, and destruction of the following:

Windows and subwindows, Menus, Msg subwindows, Form subwindows and File subwindows"L];
END;

```
Unload: Exec.ExecProc =
  BEGIN
  IF wh # NIL THEN Tool.Destroy [wh];
  wh ← NIL;
  [] ← Exec.RemoveCommand[h, "ExampleTool.~"L];
  END;

InitHeap: PROCEDURE = INLINE
  BEGIN
  heap ← Heap.Create[initial: 1];
  END;

KillHeap: PROCEDURE = INLINE
  BEGIN
  Heap.Delete[heap];
  heap ← NIL;
  END;

Init: PROCEDURE =
  BEGIN
  Exec.AddCommand["ExampleTool.~"L, ExampleToolCommand, Help, Unload];
  END;

MakeHeraldName: PROCEDURE =
  BEGIN
  tempName: LONG STRING ← heap.NEW[StringBody [60]];
  String.AppendString [tempName, "ExampleTool "L];
  Version.Append[tempName];
  String.AppendString[tempName, " of "L];
  Time.Append[tempName, Time.Unpack[Runtime.GetBcdTime []]];
  tempName.length ← tempName.length - 3; -- gun the seconds
  heraldName ← String.CopyToNewString[tempName, heap];
  heap.FREE[@tempName];
  END;

MakeTool: PROCEDURE RETURNS[wh: Window.Handle] =
  BEGIN
  RETURN[Tool.Create[
  makeSWsProc: MakeSWs, initialState: default,
  clientTransition: ClientTransition, name: heraldName,
  cmSection: "ExampleTool"L, tinyName1: "Example"L, tinyName2: "Tool"L]]
  END;
```

```
ExampleToolCommand: Exec.ExecProc =
  BEGIN
  IF heap = NIL THEN InitHeap[];
  IF heraldName = NIL THEN MakeHeraldName[];
  IF (wh # NIL) AND inactive THEN ToolWindow.Activate[wh]
  ELSE IF wh = NIL THEN wh ← MakeTool[];
  END;

MakeForm: FormSW.ClientItemsProcType =
  BEGIN
  OPEN FormSW;
  --This procedure creates a sample FormSW.
  formItems: LONG POINTER TO ARRAY FormIndex OF FormSW.ItemHandle ← NIL;
  toolData.enum1Seq ← heap.NEW[EnumSeq[3]];
  toolData.enum2Seq ← heap.NEW[EnumSeq[3]];
  toolData.enum1Seq[0] ← ["A"L, Enum1.a];
  toolData.enum1Seq[1] ← ["B"L, Enum1.b];
  toolData.enum1Seq[2] ← ["C"L, Enum1.c];
  toolData.enum2Seq[0] ← ["X"L, Enum2.x];
  toolData.enum2Seq[1] ← ["Y"L, Enum2.y];
  toolData.enum2Seq[2] ← ["Z"L, Enum2.z];
  toolData.strings[vanilla] ← toolData.strings[password] ← NIL;
  toolData.strings[readOnly] ← String.CopyToNewString[
    "Read Only String"L, heap];
  items ← AllocateItemDescriptor[nItems: FormIndex.LAST.ORD + 1, z: heap];
  formItems ← LOOPHOLE[BASE[items]];
  formItems^ ← [
    command: CommandItem[
      tag: "Command"L, place: [0, line0], z: heap, proc: FormSWCommandRoutine],
    vanilla: StringItem[
      tag: "Vanilla"L, place: [200, line0], z: heap, string: @toolData.strings[vanilla],
      inHeap: TRUE],
    password: StringItem[
      tag: "Password"L, place: [0, line1], z: heap,
      string: @toolData.strings[password],
      feedback: password, inHeap: TRUE],
    readOnly: StringItem[
      tag: "ReadOnly"L, place: [0, line2], z: heap,
      string: @toolData.strings[readOnly],
      readOnly: TRUE, inHeap: TRUE],
    number: LongNumberItem[
      tag: "Cardinal"L, place: [200, line2], z: heap, value: @toolData.number,
      notNegative: TRUE, signed: FALSE],
    boolTF: EnumeratedItem[
      tag: "boolean(trueFalse)"L, place: [0, line3], z: heap, feedback: all,
      value: @toolData.switch1, copyChoices: FALSE, choices: BooleanChoices[]],
    boolVideo: BooleanItem[
      tag: "boolean(video)"L, place: [250, line3], z: heap,
      switch: @toolData.switch2,
      proc: NotifyClientOfBooleanAction],
    enumOne: EnumeratedItem[
      tag: "enumerated(one)"L, place: [0, line4], z: heap, feedback: one,
```

```
            value: @toolData.enum1, copyChoices: TRUE,
            choices: DESCRIPTOR[toolData.enum1Seq^]],
          enumAll: EnumeratedItem[
            tag: "enumerated(all)"L, place: [175, line4], z: heap, feedback: all,
            value: @toolData.enum2, copyChoices: TRUE,
            choices: DESCRIPTOR[toolData.enum2Seq^]]];

      heap.FREE[@toolData.enum1Seq];
      heap.FREE[@toolData.enum2Seq];
      RETURN[items: items, freeDesc: TRUE]
      END;

    MakeSWs: Tool.MakeSWsProc =
      BEGIN
      logName: STRING ← [40];
      addresses: ARRAY [0..3) OF ToolDriver.Address;
      menuStrings: ARRAY MenuIndex OF LONG STRING ← [
      postMessage: "Post message"L, aCommand: "A Command"L,
      bCommand: "B Command"L];
      toolData.menu ← Menu.Make[
      name: "Tests"L, strings: DESCRIPTOR[menuStrings.BASE, menuStrings.LENGTH],
      mcrProc: MenuCommandRoutine];
      Tool.UnusedLogName[unused: logName, root: "Example.log"L];
      toolData.msgSW ← Tool.MakeMsgSW[window: window];
      toolData.formSW ← Tool.MakeFormSW[window: window, formProc: MakeForm,
        zone: heap];
      toolData.fileSW ← Tool.MakeFileSW[window: window, name: logName];
      Menu.Instantiate[toolData.menu, toolData.formSW];
      Supervisor.AddDependency[client: agent, implementor: Event.toolWindow];
      --do the ToolDriver stuff
      addresses ← [
        [name: "msgSW"L, sw: toolData.msgSW],
        [name: "formSW"L, sw: toolData.formSW],
        [name: "fileSW"L, sw: toolData.fileSW]];
      ToolDriver.NoteSWs[tool: "ExampleTool"L, subwindows: DESCRIPTOR[addresses]]
      END;

    --Mainline code
    Init[];
    END.
```

B

References

The following documents should be studied before or in conjunction with this manual:

- *Mesa Language Manual, Version 3.0.* [November 1984].
- *XDE User's Guide.* [November 1984].
- *Pilot Programmer's Manual, Version 3.0.* [November 1984].

In addition, any other documentation accompanying a release of Mesa should be consulted before you write Mesa programs. A list of this documentation can be found in the release change summary.

C

Listing of Public Symbols

-- PUBLIC SYMBOLS FOR
-- AddressTranslation Answer Ascii AsciiSink Atom Authenticator Backstop BackstopNub BandBLT BitBlt BlockSource BodyDefs BTree ByteBlt Caret CH Checksum CHLookup CHPIDs CmFile CommOnlineDiagnostics CommonSoftwareFileTypes Context Courier Cursor Date DebuggerHacks DebugUsefulDefs Device DeviceTypes Dialup DiskSource Display Environment Event EventTypes Exec Expand ExpeditedCourier ExtendedString File FileExtras FileName FileSW FileTransfer FileTypes FileWindow Floppy FloppyChannel Fonts Format FormatPilotDisk FormatPilotDiskExtras FormSW GSort Heap HeraldWindow Inline JLevelIVKeys Keys KeyStations LevelIIIKeys LevelIVKeys LexiconDefs LibrarianUtility Log LogFile LsepFace MailParse MDSStorage MemoryStream Menu MFile MFileProperty MLoader MoreCH MSegment MsgSW MStream MVolume NetworkStream NSAddr NSAssignedTypes NSConstants NSDataStream NSFile NSName NSPrint NSSegment NSSessionControl NSString NSTimeServer NSVolumeControl ObjAlloc OnlineDiagnostics OthelloOps PacketExchange PageScavenger PerformancePrograms PerformanceToolFileTypes PhysicalVolume PieceSource PilotClient PilotSwitches PilotSwitchesExtraExtraExtras PilotSwitchesExtraExtras PilotSwitchesExtras PrincOps Process Profile ProtocolCertification Put RavenFace Real RealFns RemoteCommDiags RetrieveDefs Router RS232C RS232CControl RS232CCorrespondents RS232CEnvironment Runtime Scavenger ScavengerExtras ScratchSource ScratchSW Scrollbar Selection SendDefs Space SpaceUsage SpaceUsageExtras SpyClient Storage Stream String StringLookUp StringSource StringSW Supervisor SupervisorEventIndex SupervisorEventIndexExtras System TajoMisc TemporaryBooting TextBlt TextData TextSink TextSource TextSW Time TimeServerLog TIP Token Tool ToolDriver ToolFont ToolWindow TTY TTYPort TTYPortEnvironment TTYSW UserInput UserTerminal UserTerminalExtras Version Volume VolumeConversion Window WindowFont Zone

A10: --*KeyStations*-- Bit = 108;
A11: --*KeyStations*-- Bit = 110;
A12: --*KeyStations*-- Bit = 111;
A1: --*KeyStations*-- Bit = 50;
A2: --*KeyStations*-- Bit = 31;
A3: --*KeyStations*-- Bit = 72;
A4: --*KeyStations*-- Bit = 60;
A5: --*KeyStations*-- Bit = 57;
A6: --*KeyStations*-- Bit = 76;
A7: --*KeyStations*-- Bit = 73;
A8: --*KeyStations*-- Bit = 88;
A9: --*KeyStations*-- Bit = 86,

```
Abort: --Exec-- PROCEDURE RETURNS [error: ERROR];
Abort: --NSDataStream-- PROCEDURE [stream: Handle];
Abort: --Process-- PROCEDURE [process: UNSPECIFIED];
AbortCall: --Dialup-- PROCEDURE [dialerNumber: CARDINAL];
Aborted: --NSDataStream- ERROR;
abortedSearchPathChange: - EventTypes-- Supervisor.Event;
AbortPending: --Process-- PROCEDURE RETURNS [abortPending: BOOLEAN];
AbortProcType: --Expand-- TYPE = PROCEDURE RETURNS [BOOLEAN];
abortSession: --EventTypes-- Supervisor.Event;
aboutToAbortSession: --EventTypes-- Supervisor.Event;
aboutToBoot: --EventTypes-- Supervisor.Event;
aboutToBootPhysicalVolume. --EventTypes-- Supervisor.Event;
aboutToChangeSearchPath: --EventTypes-- Supervisor.Event;
aboutToCloseVolume: --EventTypes-- Supervisor.Event;
aboutToOpenVolume: --EventTypes-- Supervisor.Event;
aboutToResume: --EventTypes-- Supervisor.Event;
aboutToSwap: --Event-- READONLY Supervisor.SubsystemHandle;
Access: --FileSW-- TYPE = TextSource.Access;
Access: --MFile-- TYPE = MACHINE DEPENDENT{
    anchor, readOnly, readWrite, writeOnly, log, delete, rename, null};
Access: --NSFile-- TYPE = PACKED ARRAY AccessType OF BooleanFalseDefault;
Access: --TextSource-- TYPE = {read, append, edit};
Access: --TextSW-- TYPE = TextSource.Access;
AccessEntries: --NSFile-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
    AccessEntry;
AccessEntry: --NSFile-- TYPE = MACHINE DEPENDENT RECORD [
    key(0:0..63): String,
    type(4:0..15): AccessEntryType,
    access(5:0..15): Access];
AccessEntryType: --NSFile-- TYPE = {individual, alias, group, other};
accessList: --NSAssignedTypes-- AttributeType = 19;
AccessList: --NSFile-- TYPE = MACHINE DEPENDENT RECORD [
    entries(0:0..47): AccessEntries ← NIL, defaulted(3:0..15): BOOLEAN ← FALSE];
accessOffset: --PrincOps-- CARDINAL = 2;
AccessProblem: --NSFile-- TYPE = MACHINE DEPENDENT{
    accessRightsInsufficient, accessRightsIndeterminate, fileChanged, fileDamaged,
    fileInUse, fileNotFound, fileOpen, fileNotLocal, volumeNotFound};
AccessProcs: --RetrieveDefs-- TYPE = RECORD [
    nextMessage: PROCEDURE [handle: Handle]
        RETURNS [msgExists: BOOLEAN, archived: BOOLEAN, deleted: BOOLEAN],
    nextItem: PROCEDURE [handle: Handle] RETURNS [BodyDefs.ItemHeader],
    nextBlock: PROCEDURE [handle: Handle, buffer: Environment.Block]
        RETURNS [bytes: CARDINAL],
    accept: PROCEDURE [handle: Handle],
    extra: SELECT type: ServerType FROM
        MTP = > NULL,
        GV = > [
            readTOC: PROCEDURE [handle: Handle, text: LONG STRING],
            startMessage: PROCEDURE [
                handle: Handle, postmark: LONG POINTER TO BodyDefs.Timestamp ← NIL,
                sender: BodyDefs.RName ← NIL, returnTo: BodyDefs.RName ← NIL],
            writeTOC: PROCEDURE [handle: Handle, text: LONG STRING],
            deleteMessage PROCEDURE [handle: Handle]],
        ENDCASE];
AccessType: --NSFile - TYPE = MACHINE DEPENDENT{
    read, write, owner, add, remove};
```

```
ACLFlavor: --MoreCH-- TYPE = MACHINE DEPENDENT{
    readers, value, administrators, selfControllers, (177777B)};
Acquire: --Context-- PROCEDURE [type: Type, window: Window.Handle]
    RETURNS [Data];
Acquire: --MFile-- PROCEDURE [
    name: LONG STRING, access: Access, release: ReleaseData,
    mightWrite: BOOLEAN ← FALSE, initialLength: InitialLength ← dontCare,
    type: Type ← unknown] RETURNS [Handle];
AcquireBcd: --DebuggerHacks-- PROCEDURE [
    info: LoadStateFormat.BcdInfo, space: LONG POINTER TO Space.Interval]
    RETURNS [success: BOOLEAN];
AcquireTemp: --MFile-- PROCEDURE [
    type: Type, initialLength: InitialLength ← dontCare,
    volume: Volume.ID ← Volume.nullID] RETURNS [Handle];
Action: --Caret-- TYPE = MACHINE DEPENDENT{
    clear, mark, invert, start, stop, reset, firstFree, last(255)};
Action: --PageScavenger-- TYPE = {
    fixDataCRCError, fixHardware, boot, lvScavenge, pvScavenge};
Action: --TextSink-- TYPE = {destroy, sleep, wakeup};
Action: --TextSource-- TYPE = {destroy, mark, sleep, truncate, wakeup};
ActionResult: --TextSink-- TYPE = {ok, bad};
actionToWindow: --TIP-- PACKED ARRAY Keys.KeyName OF BOOLEAN;
activate: --EventTypes-- Supervisor.Event;
Activate: --MSegment-- PROCEDURE [segment: Handle];
Activate: --ToolWindow-- PROCEDURE [window: Handle];
ActOn: --Caret-- PROCEDURE [Action];
ActOn: --TextSink-- ActOnProc;
ActOn: --TextSource-- ActOnProc;
ActOnProc: --TextSink-- TYPE = PROCEDURE [sink: Handle, action: Action]
    RETURNS [ActionResult];
ActOnProc: --TextSource-- TYPE = PROCEDURE [source: Handle, action: Action];
Add: --NSSegment-- PROCEDURE [
    file: NSFile.Handle, segment: ID, size: PageCount,
    session: Session ← nullSession];
AddAlias: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
    newAliasName: Name, distingName: Name] RETURNS [rc: ReturnCode];
AddCommand: --Exec-- PROCEDURE [
    name: LONG STRING, proc: ExecProc, help: ExecProc ← NIL,
    unload: ExecProc ← DefaultUnloadProc, clientData: LONG POINTER ← NIL];
AddDistinguishedName: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
    distingName: Name] RETURNS [rc: ReturnCode];
AddDomainAccessMember: --MoreCH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
    element: CH.Element, domain: CH.Name, acl: ACLFlavor]
    RETURNS [rc: CH.ReturnCode],
AddGroupMember: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
    element: Element, name: Name, pn: PropertyID, distingName: Name]
    RETURNS [rc: ReturnCode];
AddGroupProperty: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
    pn: PropertyID, elementEnumerator: EnumerateNewGroupElements ← NIL,
    distingName: Name] RETURNS [rc: ReturnCode];
AddInfinityNaN: --Real-- LONG CARDINAL = 3;
```

```
AddNotifyProc: --MFile-- PROCEDURE [
    proc: NotifyProc, filter: Filter, clientInstanceData: LONG POINTER];
AddOrgAccessMember: --MoreCH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
    element: CH.Element, org: CH.Name, acl: ACLFlavor]
    RETURNS [rc: CH.ReturnCode];
AddPrinter: --DebugUsefulDefs-- PROCEDURE [type: LONG STRING, proc: Printer];
AddProperty: --MFile-- PROCEDURE [
    file: Handle, property: Property, maxLength: CARDINAL];
AddPropertyAccessMember: --MoreCH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
    element: CH.Element, name: CH.Name, pn: CH.PropertyID, acl: ACLFlavor,
    distingName: CH.Name] RETURNS [rc: CH.ReturnCode];
Address: --MSegment-- PROCEDURE [segment: Handle] RETURNS [LONG POINTER];
Address: --NSAddr-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
    System.NetworkAddress;
Address: --ToolDriver-- TYPE = RECORD [name: LONG STRING, sw: Window.Handle];
AddressDescriptor: --ToolDriver-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
    Address;
AddressToRhs: --NSAddr-- PROCEDURE [address: Address] RETURNS [rhs: CH.Buffer];
AddresstoSegment: --MSegment-- PROCEDURE [pointer: LONG POINTER]
    RETURNS [Handle];
AddrList: --ExpeditedCourier-- TYPE = LONG POINTER TO AddrObject;
AddrObject: --ExpeditedCourier-- TYPE = RECORD [
    next: AddrList, address: System.NetworkAddress];
AddSegment: --Zone-- PROCEDURE [
    zH: Handle, storage: LONG POINTER, length: BlockSize]
    RETURNS [sH: SegmentHandle, s: Status];
AddSelf: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
    pn: PropertyID, distingName: Name] RETURNS [rc: ReturnCode];
AddString: --LexiconDefs-- PROCEDURE [LONG STRING];
AddThisSW: --Tool-- PROCEDURE [
    window: Window.Handle, sw: Window.Handle, swType: SWType ← predefined,
    nextSW: Window.Handle ← NIL, h: INTEGER ← 0];
AddToListOfIDs: --LibrarianUtility-- PROCEDURE [
    id: Librarian.LibjectID, ids: IDArrayHandle] RETURNS [BOOLEAN];
AddValueProperty: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
    pn: PropertyID, rhs: Buffer, distingName: Name] RETURNS [rc: ReturnCode];
Adjust: --FormSW-- ToolWindow.AdjustProcType;
Adjust: --TextSW-- ToolWindow.AdjustProcType;
AdjustProcType: --ToolWindow-- TYPE = PROCEDURE [
    window: Handle, box: Box, when: When];
AdobeReportSortTime: --MFileProperty-- MFile.Property;
AdvanceBand: --RavenFace-- PROCEDURE [currentBand: Index]
    RETURNS [nextBand: Index, nextBandAddress: BandPointer];
alias: --CHPIDs-- CH.PropertyID = 1;
AliasCommand: --Exec-- PROCEDURE [old: LONG STRING, new: LONG STRING]
    RETURNS [ok: BOOLEAN];
aliases: --CHPIDs-- CH.PropertyID = 2;
AlignedBandBLTTable: --BandBLT-- PROCEDURE [ip: POINTER TO BBTableSpace]
    RETURNS [b: BandBLTTablePtr];
AlignedBBTable: --BitBlt-- PROCEDURE [ip: POINTER TO BBTableSpace]
    RETURNS [b: BBptr];
```

```
AlignedTextBltArg: --TextBlt-- PROCEDURE [ip: POINTER TO TextBltArgSpace]
    RETURNS [p: POINTER TO TextBltArg];
Alignment: --Zone-- TYPE = {a1, a2, a4, a8, a16};
AList: --Atom-- TYPE = LONG POINTER TO DPCell ← NIL;
all: --CHPIDs-- CH.PropertyID = 0;
allControlSelections: --NSFile-- ControlSelections;
AllExceptions: --Real-- ExceptionFlags;
allExtendedSelections: --NSFile- READONLY ExtendedSelections;
allExtendedSelectionsRepresentation: --NSFile-- ARRAY [0..0] OF
    ExtendedAttributeType;
allInterpretedSelections: --NSFile-- InterpretedSelections;
Allocate: --ObjAlloc-- PROCEDURE [
    pool: AllocPoolDesc, count: ItemCount, willTakeSmaller: BOOLEAN ← FALSE]
    RETURNS [interval: Interval];
AllocateBands: --RavenFace-- PROCEDURE [
    bandVirtualPageNumber: Environment.PageNumber, nBands: BandBufferCount,
    sizeEachBand: Environment.PageCount, slop: Environment.PageCount];
AllocateItemDescriptor: --FormSW-- PROCEDURE [
    nItems: CARDINAL, z: UNCOUNTED ZONE ← NIL] RETURNS [ItemDescriptor];
AllocateListOfIDs: --LibrarianUtility-- PROCEDURE [maxIDs: CARDINAL]
    RETURNS [IDArrayHandle];
AllocationPool: --ObjAlloc-- TYPE = PACKED ARRAY [0..0) OF AllocFree;
AllocationVector: --PrincOps-- TYPE = ARRAY FSIndex OF AVItem;
AllocFree: --ObjAlloc-- TYPE = MACHINE DEPENDENT{free, alloc};
AllocPoolDesc: --ObjAlloc-- TYPE = RECORD [
    allocPool: LONG POINTER TO AllocationPool, poolSize: ItemCount];
AllocTag: --PrincOps-- TYPE = {frame, empty, indirect, unused};
AllocVFN: --FileName-- PROCEDURE [LONG STRING] RETURNS [VFN];
allSelections: --NSFile-- READONLY Selections;
AlmostEqual: --RealFns-- PROCEDURE [y: REAL, x: REAL, distance: [-126..0]]
    RETURNS [BOOLEAN];
AlmostZero: --RealFns-- PROCEDURE [x: REAL, distance: [-126..127]]
    RETURNS [BOOLEAN];
Alphabetic: --Token-- FilterProcType;
AlphaNumeric: --Token-- FilterProcType;
AlreadyFormatted: --Floppy-- SIGNAL [labelString: LONG STRING];
AlreadyFreed: --ObjAlloc-- ERROR [item: ItemIndex];
altL0: --ProtocolCertification-- Stage;
altL1: --ProtocolCertification-- Stage;
AlwaysConfirm: --HeraldWindow-- ConfirmProcType;
AnnounceStream: --NSDataStream-- PROCEDURE [cH: Courier.Handle];
AnonymousBackingFileSize: --PilotSwitches-- TYPE = PilotDomainC [173C..175C];
anyEthernet: --DeviceTypes-- Device.Type;
anyPilotDisk: --DeviceTypes-- Device.Type;
Append: --Time-- PROCEDURE [
    s: LONG STRING, unpacked: Unpacked, zone: BOOLEAN ← FALSE,
    zoneStandard: TimeZoneStandard ← ANSI];
Append: --Version-- PROCEDURE [LONG STRING];
AppendBrokenMessage: --HeraldWindow-- PROCEDURE [
    msg1: LONG STRING ← NIL, msg2: LONG STRING ← NIL, msg3: LONG STRING ← NIL,
    newLine: BOOLEAN ← TRUE, clearOld: BOOLEAN ← TRUE];
AppendChar: --MDSStorage-- PROCEDURE [p: POINTER TO STRING, c: CHARACTER];
AppendChar: --TTYSW-- PROCEDURE [sw: Window.Handle, char: CHARACTER];
AppendCharacter: --NSString-- PROCEDURE [to: String, from: Character]
    RETURNS [String];
AppendCommands: --Exec-- PROCEDURE [h: Handle, command: LONG STRING];
```

```
AppendCurrent: --Time-- PROCEDURE [
    s: LONG STRING, zone: BOOLEAN ← FALSE, ltp: LTP ← useSystem,
    zoneStandard: TimeZoneStandard ← ANSI];
AppendDecimal: --ExtendedString-- PROCEDURE [
    field: LONG POINTER, size: CARDINAL, string: LONG STRING];
AppendDecimal: --NSString-- PROCEDURE [s: String, n: INTEGER] RETURNS [String];
AppendErrorMessage: --MFile-- PROCEDURE [
    msg: LONG STRING, code: ErrorCode, file: Handle];
AppendExtensionIfNeeded: --MDSStorage-- PROCEDURE [
    to: POINTER TO STRING, extension: LONG STRING] RETURNS [BOOLEAN];
AppendLogicalVolumeName: --HeraldWindow-- PROCEDURE [
    s: LONG STRING, id: Volume.ID ← Volume.systemID];
AppendLongDecimal: --NSString-- PROCEDURE [s: String, n: LONG INTEGER]
    RETURNS [String];
AppendLongNumber: --NSString-- PROCEDURE [
    s: String, n: LONG UNSPECIFIED, radix: CARDINAL ← 10] RETURNS [String];
AppendMessage: --HeraldWindow-- PROCEDURE [
    msg: LONG STRING ← NIL, newLine: BOOLEAN ← TRUE, clearOld: BOOLEAN ← TRUE];
AppendNameToString: --NSName-- PROCEDURE [
    s: String, name: Name, resetLengthFirst: BOOLEAN ← FALSE]
    RETURNS [newS: String];
AppendNumber: --ExtendedString-- PROCEDURE [
    field: LONG POINTER, size: CARDINAL, base: CARDINAL, string: LONG STRING];
AppendNumber: --NSString-- PROCEDURE [
    s: String, n: UNSPECIFIED, radix: CARDINAL ← 10] RETURNS [String];
AppendOctal: --ExtendedString-- PROCEDURE [
    field: LONG POINTER, size: CARDINAL, string: LONG STRING];
AppendOctal: --NSString-- PROCEDURE [s: String, n: UNSPECIFIED]
    RETURNS [String];
AppendPhysicalVolumeName: --HeraldWindow-- PROCEDURE [s: LONG STRING];
AppendReal: --Real-- PROCEDURE [
    s: LONG STRING, r: REAL, precision: CARDINAL ← DefaultSinglePrecision,
    forceE: BOOLEAN ← FALSE];
AppendString: --MDSStorage-- PROCEDURE [
    to: POINTER TO STRING, from: LONG STRING, extra: CARDINAL ← 0];
AppendString: --MsgSW-- UserInput.StringProcType;
AppendString: --NSString-- PROCEDURE [to: String, from: String]
    RETURNS [String];
AppendString: --TTYSW-- UserInput.StringProcType;
AppendSubString: --NSString-- PROCEDURE [to: String, from: SubString]
    RETURNS [String];
AppendSwitches: --HeraldWindow-- PROCEDURE [s: LONG STRING];
AppendToMesaString: --NSString-- PROCEDURE [to: MesaString, from: String];
Block: --Environment-- TYPE = RECORD [
    blockPointer: LONG POINTER TO PACKED ARRAY [0..0) OF Byte,
    startIndex: CARDINAL,
    stopIndexPlusOne: CARDINAL];
Block: --Format-- PROCEDURE [
    proc: StringProc, block: Environment.Block, clientData: LONG POINTER ← NIL];
Block: --Put-- PROCEDURE [h: Window.Handle ← NIL, block: Environment.Block];
BlockSize: --Zone-- TYPE = CARDINAL;
Boolean: --Token-- PROCEDURE [h: Handle, signalOnError: BOOLEAN ← TRUE]
    RETURNS [true: BOOLEAN];
BooleanChoices: --FormSW-- PROCEDURE RETURNS [EnumeratedDescriptor];
BooleanDefaultFalse: --Volume-- TYPE = BOOLEAN ← FALSE;
BooleanFalseDefault: --NSFile-- TYPE = BOOLEAN ← FALSE;
```

```
BooleanFalseDefault: --NSSessionControl-- TYPE = BOOLEAN ← FALSE;
BooleanHandle: --FormSW-- TYPE = LONG POINTER TO boolean ItemObject;
BooleanItem: --FormSW-- PROCEDURE [
    tag: LONG STRING ← NIL, readOnly: BOOLEAN ← FALSE, invisible: BOOLEAN ← FALSE,
    drawBox: BOOLEAN ← FALSE, hasContext: BOOLEAN ← FALSE,
    place: Window.Place ← nextPlace, proc: NotifyProcType ← NopNotifyProc,
    switch: LONG POINTER TO BOOLEAN, z: UNCOUNTED ZONE ← NIL]
    RETURNS [BooleanHandle];
BootFilePointer: --Floppy-- TYPE = RECORD [file: FileID, page: PageNumber];
BootFileType: --OthelloOps-- TYPE = {hardMicrocode, softMicrocode, germ, pilot};
BootFromFile: --HeraldWindow-- PROCEDURE [
    name: LONG STRING, bootSwitches: System.Switches ← switches,
    postProc: Format.StringProc ← DefaultPost,
    confirmProc: ConfirmProcType ← DefaultConfirm];
BootFromVolumeID: --HeraldWindow-- PROCEDURE [
    id: Volume.ID, bootSwitches: System.Switches ← switches,
    postProc: Format.StringProc ← DefaultPost,
    confirmProc: ConfirmProcType ← DefaultConfirm];
BootFromVolumeName: --HeraldWindow-- PROCEDURE [
    name: LONG STRING, bootSwitches: System.Switches ← switches,
    postProc: Format.StringProc ← DefaultPost,
    confirmProc: ConfirmProcType ← DefaultConfirm];
bootPhysicalVolume: --EventTypes-- Supervisor.Event;
bootPhysicalVolumeCancelled: --EventTypes-- Supervisor.Event;
bootServerSocket: --NSConstants-- System.SocketNumber;
Bounds: --TextSW-- TYPE = RECORD [
    from: Position, to: Position, delta: LONG INTEGER];
Box: --ToolWindow-- TYPE = Window Box;
Box: --Window-- TYPE = RECORD [place: Place, dims: Dims];
BoxesAreDisjoint: --Window-- PROCEDURE [a: Box, b: Box] RETURNS [BOOLEAN];
boxFlags: --Display-- BitBltFlags;
BoxHandle: --Window-- TYPE = LONG POINTER TO Box;
BoxProcType: --ToolWindow-- TYPE = PROCEDURE RETURNS [box: Box];
Brackets: --Token-- QuoteProcType;
BracketType: --MailParse-- TYPE = RECORD [
    group: BOOLEAN ← FALSE, routeAddr: BOOLEAN ← FALSE];
break0: --PilotSwitches-- PilotDomainA = 60C;
break1: --PilotSwitches-- PilotDomainA = 61C;
break2: --PilotSwitches-- PilotDomainA = 62C;
breakFileMgr: --PilotSwitches-- PilotDomainA = 72C;
BreakReason: --Display-- TYPE = {normal, margin, stop};
BreakReason: --TextSink-- TYPE = {eol, consumed, margin};
breakVMMgr: --PilotSwitches-- PilotDomainA = 73C;
Brick: --Display-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF CARDINAL;
BS: --Ascii-- CHARACTER = 10C;
Buffer: --CH-- TYPE = LONG POINTER TO BufferArea;
BufferArea: --CH-- TYPE = MACHINE DEPENDENT RECORD [
    maxlength(0:0..15): CARDINAL [0..7777B],
    length(1:0..15): CARDINAL [0..7777B],
    data(2): SEQUENCE COMPUTED CARDINAL OF WORD];
BufferTooSmall: --CH-- SIGNAL [offender: Buffer, lengthNeeded: CARDINAL]
    RETURNS [newBuffer: Buffer];
BuildPropertyArray: --LibrarianUtility-- PROCEDURE [fileName: LONG STRING]
    RETURNS [PropertyArray];
bypassDebuggerSubstitute: --PilotSwitchesExtras-- PilotSwitches.PilotDomainC =
    370C;
```

```
Byte: --Environment-- TYPE = [0..255];
BYTE: --PrincOps-- TYPE = [0..255];
ByteBlt: --ByteBlt-- PROCEDURE [
    to: Environment.Block, from: Environment.Block,
    overLap: OverLapOption ← ripple] RETURNS [nBytes: CARDINAL];
ByteCount: --MFile-- TYPE = LONG CARDINAL;
ByteCount: --NSSegment-- TYPE = LONG CARDINAL;
BytePair: --Inline-- TYPE = MACHINE DEPENDENT RECORD [
    high(0:0..7): [0..255], low(0:8..15): [0..255]];
BytePC: --PrincOps-- TYPE = RECORD [CARDINAL];
bytesPerPage: --Environment-- CARDINAL = 512;
bytesPerWord: --Environment-- CARDINAL = 2;
CADFileType: --FileTypes-- TYPE = CARDINAL [22400B..22777B];
CalculateIncrementalEthernetStats: --RemoteCommDiags-- PROCEDURE [
    host: System.NetworkAddress,
    baseEthernetStatistics: LONG POINTER TO
        CommOnlineDiagnostics.EtherStatsResult,
    currentEthernetStatistics: LONG POINTER TO
        CommOnlineDiagnostics.EtherStatsResult]
    RETURNS [CommOnlineDiagnostics.EtherStatsResult];
Call: --Courier-- PROCEDURE [
    cH: Handle, procedureNumber: CARDINAL, arguments: Parameters ←
nullParameters,
    results: Parameters ← nullParameters,
    timeoutInSeconds: LONG CARDINAL ← 37777777777B,
    requestDataStream: BOOLEAN ← FALSE,
    streamCheckoutProc: PROCEDURE [cH: Handle] ← NIL] RETURNS [sH: Stream.Handle];
Call: --ExpeditedCourier-- PROCEDURE [
    programNumber: LONG CARDINAL, versionNumber: CARDINAL,
    procedureNumber: CARDINAL,
    arguments: Courier.Parameters ← Courier.nullParameters,
    address: System.NetworkAddress, response: ResponseProc];
CallToAddresses: --ExpeditedCourier-- PROCEDURE [
    programNumber: LONG CARDINAL, versionNumber: CARDINAL,
    procedureNumber: CARDINAL,
    arguments: Courier.Parameters ← Courier.nullParameters,
    socket: System.SocketNumber, addresses: AddrList, response: ResponseProc,
    responseBufferCount: CARDINAL ← 5];
CallToInternetRing: --ExpeditedCourier-- PROCEDURE [
    programNumber: LONG CARDINAL, versionNumber: CARDINAL,
    procedureNumber: CARDINAL,
    arguments: Courier.Parameters ← Courier.nullParameters, ring: RingBound,
    socket: System.SocketNumber, action: ExpandingRingAction,
    eachResponse: ResponseProc, newRadiusNotify: NewRadiusNotifyProc ← NIL,
    responseBufferCount: CARDINAL ← 5];
CancelAbort: --Process-- PROCEDURE [process: UNSPECIFIED];
CancelPeriodicNotify: --UserInput-- PROCEDURE [PeriodicNotifyHandle]
    RETURNS [nil: PeriodicNotifyHandle];
CancelTicket: --NSDataStream-- PROCEDURE [ticket: Ticket, cH: Courier.Handle];
cannotExpand: --TextSource-- CARDINAL = 177777B;
CantInstallUCodeOnThisDevice: --FormatPilotDisk-- ERROR;
CaretProcType: --UserInput-- TYPE = PROCEDURE [
    window: Window.Handle, startStop: StartStop];
caretRate: --UserInput-- Process.Ticks;
Cause: --Authenticator-- TYPE = MACHINE DEPENDENT{
    userKeyNotFound, serverKeyNotFound, authServerDown,
```

```
    remoteAuthServerDown,
        communicationError, protocolViolation, weakAndStrongNotImplemented,
        (177777B)};
  cdc9730: --DeviceTypes-- Device.Type;
  CedarFileType: --FileTypes-- TYPE = CARDINAL [23000B..23377B];
  ch3chs: --CHPIDs-- CH.PropertyID = 25;
  ch3ciu: --CHPIDs-- CH.PropertyID = 22;
  ch3ecs: --CHPIDs-- CH.PropertyID = 20;
  ch3fileserver: --CHPIDs-- CH.PropertyID = 10;
  ch3gws: --CHPIDs-- CH.PropertyID = 24;
  ch3its: --CHPIDs-- CH.PropertyID = 23;
  ch3mailserver: --CHPIDs-- CH.PropertyID = 15;
  ch3printserver: --CHPIDs-- CH.PropertyID = 11;
  ch3remote: --CHPIDs-- CH.PropertyID = 16;
  ch3router: --CHPIDs-- CH.PropertyID = 12;
  ch3rs232cport: --CHPIDs-- CH.PropertyID = 21;
  ch3user: --CHPIDs-- CH.PropertyID = 14;
  ch3workstation: --CHPIDs-- CH.PropertyID = 17;
  ch4ciu: --CHPIDs-- CH.PropertyID = 28;
  ch4rs232cPort: --CHPIDs-- CH.PropertyID = 27;
  ch5ecs: --CHPIDs-- CH.PropertyID = 46;
  ch5fileserver: --CHPIDs-- CH.PropertyID = 49;
  ch5gws: --CHPIDs-- CH.PropertyID = 48;
  ch5ibm3270host: --CHPIDs-- CH.PropertyID = 54;
  ch5irs: --CHPIDs-- CH.PropertyID = 45;
  ch5its: --CHPIDs-- CH.PropertyID = 47;
  ch5printserver: --CHPIDs-- CH.PropertyID = 50;
  ch5server: --CHPIDs-- CH.PropertyID = 42;
  ch5starUser: --CHPIDs-- CH.PropertyID = 44;
  ch5starWorkstation: --CHPIDs-- CH.PropertyID = 41;
  ch5user: --CHPIDs-- CH.PropertyID = 43;
  ch5workstation: --CHPIDs-- CH.PropertyID = 40;
  ChangeAttributes: --NSFile-- PROCEDURE [
      file: Handle, attributes: AttributeList, session: Session ← nullSession];
  ChangeAttributesByName: --NSFile-- PROCEDURE [
      directory: Handle, path: String, attributes: AttributeList,
      session: Session ← nullSession];
  ChangeAttributesChild: --NSFile-- PROCEDURE [
      directory: Handle, id: ID, attributes: AttributeList,
      session: Session ← nullSession];
  ChangeControls: --NSFile-- PROCEDURE [
      file: Handle, controlSelections: ControlSelections, controls: Controls,
      session: Session ← nullSession];
  ChangeLabelString: --Volume-- PROCEDURE [volume: ID, newLabel: LONG STRING];
  ChangeName: --PhysicalVolume-- PROCEDURE [pvID: ID, newName: LONG STRING];
  ChangeSessionRestrictions: --NSSessionControl-- PROCEDURE [
      selections: SessionRestrictionSelections, restrictions: SessionRestrictions,
      terminateRestrictedSessions: BOOLEAN ← FALSE];
  ChangeValueProperty: --CH-- PROCEDURE [
      cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
      pn: PropertyID, newRhs: Buffer, distingName: Name] RETURNS [rc: ReturnCode];
  ChannelAlreadyExists: --TTYPort-- ERROR;
  ChannelHandle: --RS232C-- TYPE [2];
  ChannelHandle: --TTYPort-- TYPE = LONG POINTER;
  ChannelInUse: --RS232C-- ERROR;
  ChannelQuiesced: --TTYPort-- ERROR;
```

```
ChannelSuspended: --RS232C-- ERROR;
Char: --Format-- PROCEDURE [
    proc: StringProc, char: CHARACTER, clientData: LONG POINTER ← NIL];
Char: --Put-- PROCEDURE [h: Window.Handle ← NIL, char: CHARACTER];
Character: --Display-- PROCEDURE [
    window: Handle, char: CHARACTER, place: Window.Place,
    font: WindowFont.Handle ← NIL, flags: BitBltFlags ← textFlags,
    bounds: Window.BoxHandle ← NIL] RETURNS [Window.Place];
Character: --NSString-- TYPE = MACHINE DEPENDENT RECORD [
    chset(0:0..7): Environment.Byte, code(0:8..15): Environment.Byte];
CharacterLength: --TTYPort-- TYPE = TTYPortEnvironment.CharacterLength;
CharacterLength: --TTYPortEnvironment-- TYPE = {
    lengthIs5bits, lengthIs6bits, lengthIs7bits, lengthIs8bits};
Characters: --NSString-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF Character;
characterSetChangeOverhead: --NSName-- CARDINAL = 2;
charCmd: --BandBLT-- CARDINAL = 0;
CharEntry: --Fonts-- TYPE = MACHINE DEPENDENT RECORD [
    leftKern(0:0..0): BOOLEAN,
    rightKern(0:1..1): BOOLEAN,
    offset(0:2..15): CARDINAL [0..37777B],
    mica(1:0..15): CARDINAL];
CharIsDefined: --WindowFont-- PROCEDURE [
    char: CHARACTER, font: Handle ← defaultFont] RETURNS [BOOLEAN];
CharLength: --RS232C-- TYPE = RS232CEnvironment.CharLength;
CharLength: --RS232CEnvironment-- TYPE = [5..8];
CharsAvailable: --TTY-- PROCEDURE [h: Handle] RETURNS [number: CARDINAL];
CharsAvailable: --TTYPort-- PROCEDURE [channel: ChannelHandle]
    RETURNS [number: CARDINAL];
CharsAvailable: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [CARDINAL];
charsPerPage: --Environment-- CARDINAL = 512;
charsPerWord: --Environment-- CARDINAL = 2;
CharStatus: --TTY-- TYPE = {ok, stop, ignore};
CharWidth: --WindowFont-- PROCEDURE [
    char: CHARACTER, font: Handle ← defaultFont] RETURNS [NATURAL];
CheckAbortProc: --Exec-- TYPE = PROCEDURE [h: Handle] RETURNS [abort: BOOLEAN];
CheckAbortProc: --FileTransfer-- TYPE = PROCEDURE [clientData: LONG POINTER]
    RETURNS [abort: BOOLEAN];
CheckChanges: --RetrieveDefs-- PROCEDURE [handle: Handle];
CheckCredentialsProc: --NSSessionControl-- TYPE = PROCEDURE [
    credentials: NSFile.Credentials, verifier: NSFile.Verifier,
    privileged: BOOLEAN]
    RETURNS [status: AuthenticationStatus, fullName: NSString.String];
CheckForAbort: --Exec-- CheckAbortProc;
CheckOwner: --Heap-- PROCEDURE [p: LONG POINTER, z: UNCOUNTED ZONE];
CheckOwnerMDS: --Heap-- PROCEDURE [p: POINTER, z: MDSZone];
Checksum: --MFileProperty-- MFile.Property;
checksum: --NSAssignedTypes-- AttributeType = 0;
CheckVerifier: --NSSessionControl-- PROCEDURE [
    verifier: NSFile.Verifier, session: NSFile.Session]
    RETURNS [AuthenticationStatus];
childrenUniquelyNamed: --NSAssignedTypes-- AttributeType = 1;
CHLookupProblem: --AddressTranslation-- ERROR [rc: CH.ReturnCode];
Circle: --Display-- PROCEDURE [
    window: Handle, place: Window.Place, radius: INTEGER,
    bounds: Window.BoxHandle ← NIL];
```

```
CIU: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    location(0:0..63): NSString.String,
    owningECS(4:0..63): NSString.String,
    model(8:0..15): CIUModel,
    timeStamp(9:0..31): System.GreenwichMeanTime,
    address(11:0..31): NSAddr.NSAddr];
CIUDescribe: --CHLookup-- Courier.Description;
CIUModel: --CHLookup-- TYPE = MACHINE DEPENDENT{
    oneBoard, twoBoards, threeBoards, fourBoards};
CIUPt: --CHLookup-- TYPE = LONG POINTER TO CIU;
Clarity: --Window-- TYPE = {isClear, isDirty};
Class: --TextSource-- TYPE = {none, eol, alpha, space, other};
ClassOfService: --NetworkStream-- TYPE = {bulk, transactional};
Clear: --MsgSW-- PROCEDURE [sw: Window.Handle];
ClearAttributeList: --NSFile-- PROCEDURE [attributeList: AttributeList];
ClearAttributes: --NSFile-- PROCEDURE [attributes: Attributes];
clearingHouseSocket: --NSConstants-- System.SocketNumber;
ClearInputFocusOnMatch: --UserInput-- PROCEDURE [w: Window.Handle];
ClearName: --NSName-- PROCEDURE [z: UNCOUNTED ZONE, name: Name];
clickTimeout: --TIP-- System.Pulses;
ClientData: --Caret-- TYPE = LONG POINTER;
ClientDest: --DebugUsefulDefs-- TYPE = POINTER;
clientDirectoryWords: --NSAssignedTypes-- AttributeType = 10373B;
clientFileWords: --NSAssignedTypes-- AttributeType = 10372B;
ClientItemsProcType: --FormSW-- TYPE = PROCEDURE [sw: Window.Handle]
    RETURNS [items: ItemDescriptor, freeDesc: BOOLEAN];
ClientProc: --FileTransfer-- TYPE = PROCEDURE [clientData: LONG POINTER];
clientSize: --NSAssignedTypes-- AttributeType = 10375B;
ClientSource: --DebugUsefulDefs-- TYPE = POINTER TO READONLY UNSPECIFIED;
clientStatus: --NSAssignedTypes-- AttributeType = 10374B;
Close: --CmFile-- PROCEDURE [h: Handle] RETURNS [nil: Handle];
Close: --FileTransfer-- PROCEDURE [conn: Connection];
Close: --Floppy-- PROCEDURE [volume: VolumeHandle];
Close: --Log-- PROCEDURE;
Close: --NetworkStream-- PROCEDURE [sH: Stream.Handle] RETURNS [CloseStatus];
Close: --NSFile-- PROCEDURE [file: Handle, session: Session ← nullSession];
Close: --NSVolumeControl-- PROCEDURE [volume: Volume.ID];
Close: --Volume-- PROCEDURE [volume: ID];
CloseAborted: --MVolume-- ERROR;
CloseReply: --NetworkStream-- PROCEDURE [sH: Stream.Handle]
    RETURNS [CloseStatus];
closeReplySST: --NetworkStream-- Stream.SubSequenceType = 255;
closeSST: --NetworkStream-- Stream.SubSequenceType = 254;
CloseStatus: --NetworkStream-- TYPE = {good, noReply, incomplete};
cmcII: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
Code: --CH-- TYPE = MACHINE DEPENDENT{
    done, notAllowed, rejectedTooBusy, allDown, (4), badProtocol,
    illegalPropertyID(10), illegalOrgName, illegalDomainName, illegalLocalName,
    noSuchOrg, noSuchDomain, noSuchLocal, propertyIDNotFound(20),
    wrongPropertyType, noChange(30), outOfDate, overflowOfName,
    overflowOfDataBase, (50), (60), wasUpNowDown(70), (177777B)};
codebaseHighOffset: --PrincOps-- CARDINAL = 1;
codebaseLowOffset: --PrincOps-- CARDINAL = 2;
CodeSegment: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    header(0:0..63): PrefixHeader];
CodeToString: --FileTransfer-- PROCEDURE [ErrorCode, LONG STRING];
```

```
Command: --BandBLT-- TYPE = CARDINAL [0..15];
CommandHandle: --FormSW-- TYPE = LONG POINTER TO command ItemObject;
CommandItem: --FormSW-- PROCEDURE [
    tag: LONG STRING ← NIL, readOnly: BOOLEAN ← FALSE, invisible: BOOLEAN ← FALSE,
    drawBox: BOOLEAN ← FALSE, hasContext: BOOLEAN ← FALSE,
    place: Window.Place ← nextPlace, proc: ProcType, z: UNCOUNTED ZONE ← NIL]
    RETURNS [CommandHandle];
CommError: --CommOnlineDiagnostics-- ERROR [reason: CommErrorCode];
CommErrorCode: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT{
    transmissionMediumProblem, noAnswerOrBusy, noRouteToSystemElement,
    remoteSystemElementNotResponding, tooManyConnections, noSuchDiagnostic,
    communicationError};
ApproveConnection: --NetworkStream-- PROCEDURE [
    listenerH: ListenerHandle, streamTimeout: WaitTime ← infiniteWaitTime,
    classOfService: ClassOfService ← bulk] RETURNS [sH: Stream.Handle];
Arc: --Display-- PROCEDURE [
    window: Handle, place: Window.Place, radius: INTEGER, startSector: CARDINAL,
    stopSector: CARDINAL, start: Window.Place, stop: Window.Place,
    bounds: Window.BoxHandle ← NIL];
ArcTan: --RealFns-- PROCEDURE [y: REAL, x: REAL] RETURNS [radians: REAL];
ArcTanDeg: --RealFns-- PROCEDURE [y: REAL, x: REAL] RETURNS [degrees: REAL];
ArgumentProblem: --NSFile-- TYPE = MACHINE DEPENDENT{
    illegal, disallowed, unreasonable, unimplemented, duplicated, missing};
Arguments: --Courier-- TYPE = PROCEDURE [
    argumentsRecord: Parameters ← nullParameters];
ascendingPositionOrdering: --NSFile-- key Ordering;
AsciiAppend: --TextSource-- PROCEDURE [
    source: Handle, string: LONG STRING, start: Position, n: CARDINAL];
AsciiDeleteSubString: --TextSource-- PROCEDURE [
    ss: String.SubString, keepTrash: BOOLEAN] RETURNS [trash: LONG STRING];
AsciiDoEditAction: --TextSource-- DoEditActionProc;
AsciiInsertBlock: --TextSource-- PROCEDURE [
    string: LONG POINTER TO LONG STRING, position: CARDINAL,
    toAdd: Environment.Block, extra: CARDINAL];
AsciiScanText: --TextSource-- ScanTextProc;
AsciiTestClass: --TextSource-- PROCEDURE [char: CHARACTER, class: Class]
    RETURNS [equal: BOOLEAN];
AsciiTextSearch: --TextSource-- PROCEDURE [
    source: Handle, string: LONG STRING, start: Position ← 0,
    stop: Position ← 37777777777B, ignoreCase: BOOLEAN ← FALSE]
    RETURNS [lineStart: Position, left: Position];
AssertLocal: --NSDataStream-- PROCEDURE [stream: Handle];
AssertNotAPilotVolume: --PhysicalVolume-- PROCEDURE [instance: Handle];
AssertPilotVolume: --PhysicalVolume-- PROCEDURE [instance: Handle] RETURNS [ID];
AssignAddress: --Router-- PROCEDURE RETURNS [System.NetworkAddress];
AssignDestinationRelativeAddress: --Router-- PROCEDURE [System.NetworkNumber]
    RETURNS [System.NetworkAddress];
AssignedType: --NSAssignedTypes-- TYPE = LONG CARDINAL;
AssignNetworkAddress: --NetworkStream-- PROCEDURE
    RETURNS [System.NetworkAddress];
AssignServiceID: --NSSessionControl-- PROCEDURE RETURNS [ServiceID];
ATOM: --Atom-- TYPE = LONG STRING ← NIL;
AttentionProcType: --UserInput-- TYPE = PROCEDURE [window: Window.Handle];
Attribute: --NSFile-- TYPE = MACHINE DEPENDENT RECORD [
    var(0:0..111): SELECT type(0:0..15): AttributeType FROM
        fileID = > [value(1:0..79): ID],
```

```
        parentID => [value(1:0..79): ID],
        checksum => [value(1:0..15): CARDINAL],
        type => [value(1:0..31): Type],
        position => [value(1:0..47): Position],
        systemElement => [value(1:0..95): SystemElement],
        volumeID => [value(1:0..79): Volume],
        ordering => [value(1:0..79): Ordering],
        accessList => [value(1:0..63): AccessList],
        defaultAccessList => [value(1:0..63): AccessList],
        backedUpOn => [value(1:0..31): Time],
        createdOn => [value(1:0..31): Time],
        filedOn => [value(1:0..31): Time],
        modifiedOn => [value(1:0..31): Time],
        readOn => [value(1:0..31): Time],
        createdBy => [value(1:0..63): String],
        filedBy => [value(1:0..63): String],
        modifiedBy => [value(1:0..63): String],
        name => [value(1:0..63): String],
        pathname => [value(1:0..63): String],
        readBy => [value(1:0..63): String],
        childrenUniquelyNamed => [value(1:0..15): BOOLEAN],
        isDirectory => [value(1:0..15): BOOLEAN],
        isTemporary => [value(1:0..15): BOOLEAN],
        version => [value(1:0..15): CARDINAL],
        numberOfChildren => [value(1:0..15): CARDINAL],
        sizeInBytes => [value(1:0..31): LONG CARDINAL],
        sizeInPages => [value(1:0..31): LONG CARDINAL],
        subtreeSize => [value(1:0..31): LONG CARDINAL],
        subtreeSizeLimit => [value(1:0..31): LONG CARDINAL],
        extended => [type(1:0..31): ExtendedAttributeType, value(3:0..47): Words],
        ENDCASE];
AttributeList: --NSFile-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
    Attribute;
Attributes: --FloppyChannel-- TYPE = RECORD [
    deviceType: {SA800, SA850},
    numberOfCylinders: [0..255],
    numberOfHeads: [0..255],
    trackLength: CARDINAL];
Attributes: --Heap-- TYPE = RECORD [
    SELECT tag: Type FROM
    normal => [
        largeNodePages: Environment.PageCount,
        threshold: NWords,
        largeNodeThreshold: NWords],
    uniform => [objectSize: NWords],
    ENDCASE];
Attributes: --NSFile-- TYPE = LONG POINTER TO AttributesRecord;
AttributesProc: --NSFile-- TYPE = PROCEDURE [attributes: Attributes]
    RETURNS [continue: BOOLEAN ← TRUE];
AttributesRecord: --NSFile-- TYPE = RECORD [
    fileID: ID,
    systemElement: SystemElement,
    volumeID: Volume,
    name: String,
    pathname: String,
    version: CARDINAL,
```

```
        checksum: CARDINAL,
        type: Type,
        isDirectory: BOOLEAN,
        isTemporary: BOOLEAN,
        parentID: ID,
        position: Position,
        backedUpOn: Time,
        createdOn: Time,
        filedOn: Time,
        modifiedOn: Time,
        readOn: Time,
        createdBy: String,
        filedBy: String,
        modifiedBy: String,
        readBy: String,
        sizeInBytes: LONG CARDINAL,
        sizeInPages: LONG CARDINAL,
        accessList: AccessList,
        defaultAccessList: AccessList,
        ordering: Ordering,
        childrenUniquelyNamed: BOOLEAN,
        subtreeSizeLimit: LONG CARDINAL,
        subtreeSize: LONG CARDINAL,
        numberOfChildren: CARDINAL,
        extended: ExtendedAttributeList];
AttributeType: --NSAssignedTypes-- TYPE = NSFile.ExtendedAttributeType;
AttributeType: --NSFile-- TYPE = MACHINE DEPENDENT{
    checksum, childrenUniquelyNamed, createdBy, createdOn, fileID, isDirectory,
    isTemporary, modifiedBy, modifiedOn, name, numberOfChildren, ordering,
    parentID, position, readBy, readOn, sizeInBytes, type, version, accessList,
    defaultAccessList, pathname, volumeID, backedUpOn, filedBy, filedOn,
    sizeInPages, subtreeSize, subtreeSizeLimit, systemElement, extended};
Authenticate: --Authenticator-- PROCEDURE [
    serverKey: Key, credentials: Credentials, verifier: Verifier, userName: Name]
    RETURNS [flavor: Flavor, status: Status];
AuthenticationProblem: --NSName-- TYPE = MACHINE DEPENDENT{
    credentialsInvalid, verifierInvalid};
AuthenticationStatus: --NSSessionControl-- TYPE = {
    valid, noSuchUser, incorrectPassword, cannotAuthenticate, invalidCredentials,
    invalidVerifier};
AutoRecognitionOutcome: --RS232C-- TYPE =
    RS232CEnvironment.AutoRecognitionOutcome;
AutoRecognitionWait: --RS232C-- PROCEDURE [channel: ChannelHandle]
    RETURNS [outcome: AutoRecognitionOutcome];
AV: --PrincOps-- AVHandle;
AVHandle: --PrincOps-- TYPE = POINTER TO AllocationVector;
AVHeap: --PrincOps-- TYPE = ARRAY [0..31] OF AVItem;
AVHeapSize: --PrincOps-- CARDINAL = 32;
AVItem: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
    data = > [fsi(0:0..13): [0..37777B], tag(0:14..15): AllocTag],
    link = > [link(0:0..15): POINTER TO AVItem],
    frame = > [frame(0:0..15): LocalFrameHandle],
    ENDCASE];
```

```
AwaitStateChange: --PhysicalVolume-- PROCEDURE [
    changeCount: CARDINAL, index: CARDINAL ← nullDeviceIndex]
    RETURNS [currentChangeCount: CARDINAL];
backedUpOn: --NSAssignedTypes-- AttributeType = 23;
Background: --OnlineDiagnostics-- TYPE = {white, black};
Background: --UserTerminal-- TYPE = {white, black};
BackingStream: --TTY-- PROCEDURE [h: Handle] RETURNS [stream: Stream.Handle];
BackingStream: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [Stream.Handle];
BackupLog: --MStream-- PROCEDURE [stream: Handle, count: MFile.ByteCount]
    RETURNS [backedUp: MFile.ByteCount];
BadFormatSnapShot: --LibrarianUtility-- SIGNAL [badLine: LONG STRING];
BadPage: --FormatPilotDisk-- SIGNAL [p: DiskPageNumber];
BadSwitches: --OthelloOps-- ERROR;
BadSyntax: --AddressTranslation-- ERROR [field: Field];
balanceBeamChoice: --Profile-- READONLY BalanceBeamChoice;
BalanceBeamChoice: --Profile-- TYPE = {never, notForCharacter, always};
Band: --LsepFace-- TYPE = LONG POINTER;
BandBLT: --BandBLT-- PROCEDURE [BandBLTTablePtr] RETURNS [LONG POINTER];
BandBLTTable: --BandBLT-- TYPE = MACHINE DEPENDENT RECORD [
    readLO(0:0..15): PageNumber,
    bandlist(1:0..31): LONG POINTER,
    writeLO(3:0..15): PageNumber,
    bandbuf(4:0..31): LONG POINTER,
    fontPtrTbl(6:0..15): PageNumber,
    fontRasters(7:0..15): PageNumber,
    inkwells(8:0..15): PageNumber];
BandBLTTableAlignment: --BandBLT-- CARDINAL = 16;
BandBLTTablePtr: --BandBLT-- TYPE = POINTER TO BandBLTTable;
BandBufferCount: --RavenFace-- TYPE = CARDINAL [1..8];
bandBufferSize: --BandBLT-- Environment.PageCount = 16;
BandFull: --RavenFace-- PROCEDURE [band: Index] RETURNS [bandBusy: BOOLEAN];
BandListItemLongPointer: --BandBLT-- TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
    ptr => [ptr(0:0..31): LONG POINTER],
    char => [char(0:0..31): LONG POINTER TO char BlItem],
    leftOverChar => [leftOverChar(0:0..31): LONG POINTER TO leftOverChar BlItem],
    rectangle => [rectangle(0:0..31): LONG POINTER TO rectangle BlItem],
    setLevel => [setLevel(0:0..31): LONG POINTER TO setLevel BlItem],
    setInk => [setInk(0:0..31): LONG POINTER TO setInk BlItem],
    endOfBand => [endOfBand(0:0..31): LONG POINTER TO endOfBand BlItem],
    endOfPage => [endOfPage(0:0..31): LONG POINTER TO endOfPage BlItem],
    rulette => [rulette(0:0..31): LONG POINTER TO rulette BlItem],
    nop1 => [nop1(0:0..31): LONG POINTER TO nop1 BlItem],
    nop2 => [nop2(0:0..31): LONG POINTER TO nop2 BlItem],
    ENDCASE];
BandOverrun: --RavenFace-- PROCEDURE RETURNS [BOOLEAN];
BandPointer: --RavenFace-- TYPE = LONG POINTER;
Base: --Environment-- TYPE = LONG BASE POINTER;
Base: --MSegment-- PROCEDURE [segment: Handle] RETURNS [Environment.PageNumber];
Base: --Zone-- TYPE = Environment.Base;
BaseDirectoryProc: --NSVolumeControl-- TYPE = PROCEDURE [
    baseString: NSString.String, baseReference: NSFile.Reference]
    RETURNS [status: BaseDirectoryStatus ← valid];
BaseDirectoryStatus: --NSVolumeControl-- TYPE = {
    cannotDetermine, invalid, invalidSyntax, valid};
```

```
BasetoSegment: --MSegment-- PROCEDURE [page: Environment.PageNumber]
    RETURNS [Handle];
BBptr: --BitBlt-- TYPE = POINTER TO BBTable;
BBTable: --BitBlt-- TYPE = MACHINE DEPENDENT RECORD [
    dst(0:0..47): BitAddress,
    dstBpl(3:0..15): INTEGER,
    src(4:0..47): BitAddress,
    srcDesc(7:0..15): SrcDesc,
    width(8:0..15): CARDINAL,
    height(9:0..15): CARDINAL,
    flags(10:0..15): BitBltFlags,
    reserved(11:0..15): UNSPECIFIED ← 0];
BBTableAlignment: --BitBlt-- CARDINAL = 16;
BBTableSpace: --BandBLT-- TYPE = ARRAY [1..24] OF WORD;
BBTableSpace: --BitBlt-- TYPE = ARRAY [1..27] OF UNSPECIFIED;
Beep: --UserTerminal-- PROCEDURE [
    frequency: CARDINAL ← 1000, duration: CARDINAL ← 500];
BEL: --Ascii-- CHARACTER = 7C;
Bit: --JLevelIVKeys-- TYPE = KeyStations.Bit;
Bit: --Keys-- TYPE = KeyStations.Bit;
Bit: --KeyStations-- TYPE = KeyStation;
Bit: --LevelIIIKeys-- TYPE = KeyStations.Bit;
Bit: --LevelIVKeys-- TYPE = KeyStations.Bit;
BitAddress: --BitBlt-- TYPE = Environment.BitAddress;
BitAddress: --Display-- TYPE = BitBlt.BitAddress;
BitAddress: --Environment-- TYPE = MACHINE DEPENDENT RECORD [
    word(0:0..31): LONG POINTER,
    reserved(2:0..11): [0..7776B] ← 0,
    bit(2:12..15): [0..15]];
BitAddressFromPlace: --Display-- PROCEDURE [
    base: BitAddress, x: NATURAL, y: NATURAL, raster: CARDINAL]
    RETURNS [BitAddress];
BITAND: --Inline-- BitOp;
BITBLT: --BitBlt-- PROCEDURE [ptr: BBptr];
BitBltFlags: --BitBlt-- TYPE = MACHINE DEPENDENT RECORD [
    direction(0:0..0): Direction ← forward,
    disjoint(0:1..1): BOOLEAN ← FALSE,
    disjointItems(0:2..2): BOOLEAN ← FALSE,
    gray(0:3..3): BOOLEAN ← FALSE,
    srcFunc(0:4..4): SrcFunc ← null,
    dstFunc(0:5..6): DstFunc ← null,
    reserved(0:7..15): [0..511] ← 0];
BitBltFlags: --Display-- TYPE = BitBlt.BitBltFlags;
BitBltTable: --BitBlt-- TYPE = BBTable;
BitBltTablePtr: --BitBlt-- TYPE = BBptr;
bitFlags: --Display-- BitBltFlags;
Bitmap: --Display-- PROCEDURE [
    window: Handle, box: Window.Box, address: BitAddress,
    bitmapBitWidth: CARDINAL, flags: BitBltFlags ← paintFlags];
BitmapIsDisconnected: --UserTerminal-- ERROR;
BitmapPlace: --Window-- PROCEDURE [window: Handle, place: Place ← [0, 0]]
    RETURNS [Place];
BitmapPlaceToWindowAndPlace: --Window-- PROCEDURE [bitmapPlace: Place]
    RETURNS [window: Handle, place: Place];
BITNOT: --Inline-- PROCEDURE [UNSPECIFIED] RETURNS [UNSPECIFIED];
```

```
BitOp: --Inline-- TYPE = PROCEDURE [UNSPECIFIED, UNSPECIFIED]
    RETURNS [UNSPECIFIED];
BITOR: --Inline-- BitOp;
BITROTATE: --Inline-- PROCEDURE [value: UNSPECIFIED, count: INTEGER]
    RETURNS [UNSPECIFIED];
Bits: --DebugUsefulDefs-- TYPE = [0..15];
BITSHIFT: --Inline-- PROCEDURE [value: UNSPECIFIED, count: INTEGER]
    RETURNS [UNSPECIFIED];
bitsPerByte: --Environment-- CARDINAL = 8;
bitsPerCharacter: --Environment-- CARDINAL = 8;
bitsPerWord: --Environment-- CARDINAL = 16;
BITXOR: --Inline-- BitOp;
Black: --Display-- PROCEDURE [window: Handle, box: Window.Box];
Blank: --Format-- PROCEDURE [
    proc: StringProc, n: CARDINAL ← 1, clientData: LONG POINTER ← NIL];
Blank: --Put-- PROCEDURE [h: Window.Handle ← NIL, n: CARDINAL ← 1];
Blanks: --Format-- PROCEDURE [
    proc: StringProc, n: CARDINAL ← 1, clientData: LONG POINTER ← NIL];
Blanks: --Put-- PROCEDURE [h: Window.Handle ← NIL, n: CARDINAL ← 1];
BlinkDisplay: --TTY-- PROCEDURE [h: Handle];
BlinkDisplay: --UserTerminal-- PROCEDURE;
BlinkingCaret: --TextSW-- PROCEDURE [sw: Window.Handle, state: OnOff];
BltItem: --BandBLT-- TYPE = MACHINE DEPENDENT RECORD [
    tag(0:0..47): SELECT OVERLAID * FROM
        char = > [
            type(0:0..0): [0..1] ← charCmd,
            font(0:1..7): Font,
            cc(0:8..15): [0..255],
            xloc(1:0..3): [0..15],
            yloc(1:4..15): [0..7777B]],
        leftOverChar = > [
            type(0:0..0): [0..1] ← charCmd,
            font(0:1..7): Font,
            cc(0:8..15): [0..255],
            mustBeZero(1:0..3): [0..15] ← 0,
            yloc(1:4..15): [0..7777B],
            alsoMustBeZero(2:0..3): [0..15] ← 0,
            scansToSkip(2:4..15): [0..7777B]],
        rectangle = > [
            type(0:0..3): Command ← rectangleCmd,
            yloc(0:4..15): [0..7777B],
            mustBeZero(1:0..3): [0..15] ← 0,
            bitsPerScan(1:4..15): [0..7777B],
            nScans(2:0..11): [0..7777B],
            xloc(2:12..15): [0..15]],
        setLevel = > [
            type(0:0..3): Command ← setLevelCmd,
            mustBeZero(0:4..4): [0..1] ← 0,
            pad(0:5..7): [0..7] ← 0,
            levelnum(0:8..15): [0..255]],
        setInk = > [
            type(0:0..3): Command ← setInkCmd,
            srcFunc(0:4..4): BitBlt.SrcFunc ← null,
            dstFunc(0:5..6): BitBlt.DstFunc ← null,
            unused(0:7..7): BOOLEAN ← NULL,
            inknum(0:8..15): [0..255]],
```

```
            endOfBand => [
                type(0:0..3): Command ← endOfBandCmd, pad(0:4..15): [0..7777B] ← 0],
            endOfPage => [
                type(0:0..3): Command ← endOfPageCmd, pad(0:4..15): [0..7777B] ← 0],
            rulette => [
                type(0:0..3): Command ← ruletteCmd,
                yloc(0:4..15): [0..7777B],
                length(1:0..11): [0..7777B],
                xloc(1:12..15): [0..15]],
            nop1 => [type(0:0..3): Command ← nopCmd1, pad(0:4..15): [0..7777B] ←
0],
            nop2 => [type(0:0..3): Command ← nopCmd2, pad(0:4..15): [0..7777B] ←
0],
            ENDCASE];
    Block: --BlockSource-- TYPE = Environment.Block;
    Block: --Display-- PROCEDURE [
        window: Handle, block: Environment.Block, lineLength: INTEGER ← infinity,
        place: Window.Place, font: WindowFont.Handle ← NIL,
        flags: BitBltFlags ← textFlags, bounds: Window.BoxHandle ← NIL]
        RETURNS [newPlace: Window.Place, positions: CARDINAL, why: BreakReason];
    CommonSoftwareFileType: --CommonSoftwareFileTypes-- TYPE =
        FileTypes.CommonSoftwareFileType;
    CommonSoftwareFileType: --FileTypes-- TYPE = CARDINAL [4000B..5777B];
    CommParamHandle: --RS232C-- TYPE = RS232CEnvironment.CommParamHandle;
    CommParamHandle: --RS232CEnvironment-- TYPE = POINTER TO CommParamObject;
    CommParamObject: --RS232C-- TYPE = RS232CEnvironment.CommParamObject;
    CommParamObject: --RS232CEnvironment-- TYPE = MACHINE DEPENDENT RECORD [
        duplex(0:0..15): Duplexity,
        lineType(1:0..15): LineType,
        lineSpeed(2:0..15): LineSpeed,
        accessDetail(3:0..63): SELECT netAccess(3:0..15): NetAccess FROM
            directConn => NULL,
            dialConn => [
                dialMode(4:0..15): DialMode,
                dialerNumber(5:0..15): CARDINAL,
                retryCount(6:0..15): RetryCount],
            ENDCASE];
    Compact: --Floppy-- PROCEDURE [volume: VolumeHandle];
    CompareAddresses: --NSAddr-- PROCEDURE [
        a: Address, b: Address, ignoreSockets: BOOLEAN, ignoreNets: BOOLEAN]
        RETURNS [similar: BOOLEAN];
    CompareNames: --NSName-- PROCEDURE [
        n1: Name, n2: Name, ignoreOrg: BOOLEAN ← FALSE, ignoreDomain: BOOLEAN ←
FALSE,
        ignoreLocal: BOOLEAN ← FALSE] RETURNS [NSString.Relation];
    CompareNSAddrs: --NSAddr-- PROCEDURE [
        a: NSAddr, b: NSAddr, ignoreSockets: BOOLEAN, ignoreNets: BOOLEAN]
        RETURNS [similar: BOOLEAN];
    CompareProcType: --GSort-- TYPE = PROCEDURE [p1: LONG POINTER, p2: LONG POINTER]
        RETURNS [INTEGER];
    CompareStrings: --NSString-- PROCEDURE [
        s1: String, s2: String, ignoreCase: BOOLEAN ← TRUE] RETURNS [Relation];
    CompareStringsAndStems: --NSString-- PROCEDURE [
        s1: String, s2: String, ignoreCase: BOOLEAN ← TRUE]
        RETURNS [relation: Relation, equalStems: BOOLEAN];
```

```
CompareStringsTruncated: --NSString-- PROCEDURE [
    s1: String, s2: String, trunc1: BOOLEAN ← FALSE, trunc2: BOOLEAN ← FALSE,
    ignoreCase: BOOLEAN ← TRUE] RETURNS [Relation];
CompareSubStrings: --NSString-- PROCEDURE [
    s1: SubString, s2: SubString, ignoreCase: BOOLEAN ← TRUE] RETURNS [Relation];
compatibility: --NSAssignedTypes-- AttributeType = 10376B;
CompleteFilename: --MFile-- PROCEDURE [
    name: LONG STRING, addedPart: LONG STRING]
    RETURNS [exactMatch: BOOLEAN, matches: CARDINAL];
CompletionHandle: --RS232C-- TYPE = RS232CEnvironment.CompletionHandle;
CompletionHandle: --RS232CEnvironment-- TYPE [2];
ComputeChecksum: --Checksum-- PROCEDURE [
    cs: CARDINAL ← 0, nWords: CARDINAL, p: LONG POINTER]
    RETURNS [checksum: CARDINAL];
ComputeFileType: --MFile-- PROCEDURE [file: Handle] RETURNS [type: Type];
ConfigForFrame: --DebugUsefulDefs-- PROCEDURE [
    gf: GFHandle, config: LONG STRING];
Confirm: --Exec-- PROCEDURE [h: Handle] RETURNS [yes: BOOLEAN];
Confirmation: --FileTransfer-- TYPE = MACHINE DEPENDENT{
    do, skip, abort, firstPrivate(8), null(255)};
ConfirmProcType: --HeraldWindow-- TYPE = PROCEDURE [
    post: Format.StringProc, cleanup: BOOLEAN ← TRUE] RETURNS [okay: BOOLEAN];
Conic: --Display-- PROCEDURE [
    window: Handle, a: LONG INTEGER, b: LONG INTEGER, c: LONG INTEGER,
    d: LONG INTEGER, e: LONG INTEGER, errorTerm: LONG INTEGER,
    start: Window.Place, stop: Window.Place, errorRef: Window.Place,
    sharpCornered: BOOLEAN, unboundedStart: BOOLEAN, unboundedStop: BOOLEAN,
    bounds: Window.BoxHandle ← NIL];
Connect: --BodyDefs-- TYPE = LONG STRING;
Connection: --FileTransfer-- TYPE = LONG POINTER TO ConnectionObject;
ConnectionFailed: --NetworkStream-- SIGNAL [why: FailureReason];
ConnectionID: --NetworkStream-- TYPE [1];
ConnectionObject: --FileTransfer-- TYPE;
ConnectionProblem: --NSFile-- TYPE = MACHINE DEPENDENT{
    noRoute, noResponse, transmissionHardware, transportTimeout,
    tooManyLocalConnections, tooManyRemoteConnections, missingCourier,
    missingProgram, missingProcedure, protocolMismatch, parameterInconsistency,
    invalidMessage, returnTimedOut, otherCallProblem(177777B)};
ConnectionProblem: --NSPrint-- TYPE = MACHINE DEPENDENT{
    noRoute, noResponse, transmissionHardware, transportTimeout,
    tooManyLocalConnections, tooManyRemoteConnections, missingCourier,
    missingProgram, missingProcedure, protocolMismatch, parameterInconsistency,
    invalidMessage, returnTimedOut};
ConnectionSuspended: --NetworkStream-- ERROR [why: SuspendReason];
ConsoleCharacter: --RavenFace-- TYPE = MACHINE DEPENDENT{
    (0), F(47), zero, one, two, three, four, five, six, seven, eight, nine, A, L,
    C, D, E, blank};
Context: --FloppyChannel-- TYPE = RECORD [
    protect: BOOLEAN,
    format: {IBM, Troy},
    density: {single, double},
    sectorLength: CARDINAL [0..1023]];
ContextFromItem: --FormSW-- PROCEDURE [ItemHandle] RETURNS [LONG POINTER];
ContinueStop: --FileWindow-- TYPE = {continue, stop};
Control: --ProtocolCertification-- TYPE = LONG POINTER TO ControlPacket;
ControlA: --Ascii-- CHARACTER = 1C;
```

```
ControlB: --Ascii-- CHARACTER = 2C;
ControlC: --Ascii-- CHARACTER = 3C;
ControlD: --Ascii-- CHARACTER = 4C;
ControlE: --Ascii-- CHARACTER = 5C;
ControlError: --NSSessionControl-- ERROR [type: ControlErrorType];
ControlErrorType: --NSSessionControl-- TYPE = {
    alreadySet, invalidRestrictions, notEntered, notLocal, notRemote, notSet};
ControlF: --Ascii-- CHARACTER = 6C;
ControlG: --Ascii-- CHARACTER = 7C;
ControlH: --Ascii-- CHARACTER = 10C;
ControlI: --Ascii-- CHARACTER = 11C;
ControlJ: --Ascii-- CHARACTER = 12C;
ControlK: --Ascii-- CHARACTER = 13C;
ControlL: --Ascii-- CHARACTER = 14C;
ControllerLinkType: --CHLookup-- TYPE = MACHINE DEPENDENT{sdlc, bsc, (177777B)};
ControlLink: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID ControlLinkTag FROM
    frame => [frame(0:0..15): LocalFrameHandle, fill(1:0..15): WORD ← 0],
    procedure => [gf(0:0..15): GlobalFrameHandle, pc(1:0..15): BytePC],
    indirect => [
        SELECT OVERLAID * FROM
        port => [port(0:0..15): PortHandle, fill(1:0..15): WORD ← 0],
        link => [link(0:0..15): POINTER TO ControlLink, fill(1:0..15): WORD ← 0],
        ENDCASE],
    rep => [
        fill0(0:0..13): [0..37777B],
        indirect(0:14..14): BOOLEAN,
        proc(0:15..15): BOOLEAN,
        fill(1:0..15): WORD],
    ENDCASE];
ControlLink: --Runtime-- TYPE = LONG UNSPECIFIED;
ControlLinkTag: --PrincOps-- TYPE = {frame, procedure, indirect, rep};
ControlM: --Ascii-- CHARACTER = 15C;
ControlN: --Ascii-- CHARACTER = 16C;
ControlO: --Ascii-- CHARACTER = 17C;
ControlP: --Ascii-- CHARACTER = 20C;
ControlPacket: --ProtocolCertification-- TYPE = MACHINE DEPENDENT RECORD [
    checksum(0:0..15): CARDINAL,
    pktLength(1:0..15): CARDINAL,
    transportControl(2:0..7): NSTypes.TransportControl,
    packetType(2:8..15): NSTypes.PacketType,
    destination(3:0..95): System.NetworkAddress,
    source(9:0..95): System.NetworkAddress,
    operation(15:0..15): OperationType,
    stage(16:0..31): Stage,
    results(18:0..15): CARDINAL];
ControlQ: --Ascii-- CHARACTER = 21C;
ControlR: --Ascii-- CHARACTER = 22C;
ControlS: --Ascii-- CHARACTER = 23C;
Controls: --NSFile-- TYPE = RECORD [
    lock: Lock ← none,
    timeout: Timeout ← defaultTimeout,
    access: Access ← fullAccess];
ControlSelections: --NSFile- TYPE = PACKED ARRAY ControlType OF
    BooleanFalseDefault;
ControlT: --Ascii-- CHARACTER = 24C;
```

```
ControlType: --NSFile-- TYPE = MACHINE DEPENDENT{lock, timeout, access};
ControlU: --Ascii-- CHARACTER = 25C;
ControlV: --Ascii-- CHARACTER = 26C;
ControlW: --Ascii-- CHARACTER = 27C;
ControlX: --Ascii-- CHARACTER = 30C;
ControlY: --Ascii-- CHARACTER = 31C;
ControlZ: --Ascii-- CHARACTER = 32C;
ConversionLog: --VolumeConversion-- TYPE = MACHINE DEPENDENT RECORD [
    seal(0:0..15): CARDINAL ← Seal,
    version(1:0..15): CARDINAL ← currentVersion,
    date(2:0..31): System.GreenwichMeanTime,
    numberOfFiles(4:0..31): LONG CARDINAL,
    logState(6:0..15): LogState,
    reserved(7:0..3983): Reserved,
    files(256): ARRAY [0..0) OF LogEntry];
Coordinate: --OnlineDiagnostics-- TYPE = MACHINE DEPENDENT RECORD [
    x(0:0..15): INTEGER, y(1:0..15): INTEGER];
Coordinate: --UserTerminal-- TYPE = MACHINE DEPENDENT RECORD [
    x(0:0..15): INTEGER, y(1:0..15): INTEGER];
Copied: --DebugUsefulDefs-- PROCEDURE [GFHandle] RETURNS [BOOLEAN];
Copy: --FileTransfer-- PROCEDURE [
    sourceFile: FileName.VFN, destFile: FileName.VFN,
    sourceConn: Connection ← NIL, destConn: Connection ← NIL,
    veto: VetoProc ← NIL, showDates: BOOLEAN ← FALSE];
COPY: --Inline-- PROCEDURE [from: POINTER, nwords: CARDINAL, to: POINTER];
Copy: --MFile-- PROCEDURE [file: Handle, newName: LONG STRING];
Copy: --MStream-- PROCEDURE [from: Handle, to: Handle, bytes: MFile.ByteCount]
    RETURNS [bytesCopied: MFile.ByteCount];
Copy: --NSFile-- PROCEDURE [
    file: Handle, destination: Handle,
    attributes: AttributeList ← nullAttributeList, controls: Controls ← [],
    session: Session ← nullSession] RETURNS [newFile: Handle];
CopyAccessList: --NSFile-- PROCEDURE [list: AccessList] RETURNS [AccessList];
CopyAttributes: --NSFile-- PROCEDURE [attributes: Attributes]
    RETURNS [Attributes];
CopyByName: --NSFile-- PROCEDURE [
    directory: Handle, path: String, destination: Handle,
    attributes: AttributeList ← nullAttributeList, session: Session ← nullSession]
    RETURNS [ID];
CopyChild: --NSFile-- PROCEDURE [
    directory: Handle, id: ID, destination: Handle,
    attributes: AttributeList ← nullAttributeList, session: Session ← nullSession]
    RETURNS [ID];
CopyExtendedAttributes: --NSFile-- PROCEDURE [
    extendedAttributes: ExtendedAttributeList] RETURNS [ExtendedAttributeList];
CopyFileHandle: --MFile-- PROCEDURE [
    file: Handle, release: ReleaseData, access: Access ← null] RETURNS [Handle];
CopyFromPilotFile: --Floppy-- PROCEDURE [
    pilotFile: File.File, floppyFile: FileHandle, firstPilotPage: File.PageNumber,
    firstFloppyPage: PageNumber, count: PageCount ← defaultPageCount];
CopyIn: --MSegment-- PROCEDURE [
    segment: Handle, file: MFile.Handle, fileBase: File.PageNumber,
    count: Environment.PageCount];
CopyIn: --NSSegment-- PROCEDURE [
    pointer: LONG POINTER, origin: Origin, session: Session ← nullSession]
    RETURNS [countRead: PageCount];
```

```
CopyName: --NSName-- PROCEDURE [z: UNCOUNTED ZONE, name: Name] RETURNS [Name];
CopyNameFields: --NSName-- PROCEDURE [
    z: UNCOUNTED ZONE, source: Name, destination: Name];
CopyOut: --MSegment-- PROCEDURE [
    segment: Handle, file: MFile.Handle, fileBase: File.PageNumber,
    count: Environment.PageCount];
CopyOut: --NSSegment-- PROCEDURE [
    pointer: LONG POINTER, origin: Origin, session: Session ← nullSession]
    RETURNS [countWritten: PageCount];
CopyProperties: --MFile-- PROCEDURE [from: Handle, to: Handle];
CopySegment: --MSegment-- PROCEDURE [segment: Handle]
    RETURNS [newSegment: Handle];
CopyString: --MDSStorage-- PROCEDURE [s: LONG STRING, longer: CARDINAL ← 0]
    RETURNS [newS: STRING];
CopyString: --NSString-- PROCEDURE [z: UNCOUNTED ZONE, s: String]
    RETURNS [String];
CopyToPilotFile: --Floppy-- PROCEDURE [
    floppyFile: FileHandle, pilotFile: File.File, firstFloppyPage: PageNumber,
    firstPilotPage: File.PageNumber, count: PageCount ← defaultPageCount];
CopyWords: --NSFile-- PROCEDURE [words: Words] RETURNS [Words];
Correspondent: --RS232C-- TYPE = RS232CEnvironment.Correspondent;
Correspondent: --RS232CEnvironment-- TYPE = RECORD [[0..255]];
Cos: --RealFns-- PROCEDURE [radians: REAL] RETURNS [cos: REAL];
CosDeg: --RealFns-- PROCEDURE [degrees: REAL] RETURNS [cos: REAL];
CountType: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT RECORD [
    sendOk(0:0..31): LONG CARDINAL,
    bytesSent(2:0..31): LONG CARDINAL,
    recOk(4:0..31): LONG CARDINAL,
    bytesRec(6:0..31): LONG CARDINAL,
    deviceError(8:0..31): LONG CARDINAL,
    dataLost(10:0..31): LONG CARDINAL,
    checkSum(12:0..31): LONG CARDINAL,
    parity(14:0..31): LONG CARDINAL,
    invalidChar(16:0..31): LONG CARDINAL,
    invalidFrame(18:0..31): LONG CARDINAL,
    asynchFrame(20:0..31): LONG CARDINAL,
    breakDetected(22:0..31): LONG CARDINAL,
    frameTimeout(24:0..31): LONG CARDINAL,
    badSeq(26:0..31): LONG CARDINAL,
    missing(28:0..31): LONG CARDINAL,
    sendErrors(30:0..31): LONG CARDINAL];
Couple: --NSDataStream-- TYPE = RECORD [sink: SinkStream, source: SourceStream];
CourierError: --RemoteCommDiags-- ERROR [reason: Courier.ErrorCode];
courierSocket: --NSConstants-- System.SocketNumber;
CR: --Ascii-- CHARACTER = 15C;
CR: --Format-- PROCEDURE [proc: StringProc, clientData: LONG POINTER ← NIL];
CR: --Put-- PROCEDURE [h: Window.Handle ← NIL];
Create: --AsciiSink-- PROCEDURE [font: WindowFont.Handle]
    RETURNS [TextSink.Handle];
Create: --BlockSource-- PROCEDURE [block: Block] RETURNS [source: Handle];
Create: --Context-- PROCEDURE [
    type: Type, data: Data, proc: DestroyProcType, window: Window.Handle];
Create: --Courier-- PROCEDURE [
    remote: SystemElement, programNumber: LONG CARDINAL, versionNumber:
CARDINAL,
```

```
    zone: UNCOUNTED ZONE, classOfService: NetworkStream.ClassOfService]
    RETURNS [cH: Handle];
Create: --DiskSource-- PROCEDURE [
    name: LONG STRING, access: TextSource.Access, s: Stream.Handle ← NIL]
    RETURNS [source: TextSource.Handle];
Create: --File-- PROCEDURE [
    volume: System.VolumeID, initialSize: PageCount, type: Type]
    RETURNS [file: File];
Create: --FileSW-- PROCEDURE [
    sw: Window.Handle, name: LONG STRING, options: Options ← defaultOptions,
    s: Stream.Handle ← NIL, position: TextSource.Position ← 0,
    allowTypeIn: BOOLEAN ← TRUE, resetLengthOnNewSession: BOOLEAN ← FALSE];
Create: --FileTransfer-- PROCEDURE RETURNS [Connection];
Create: --FileWindow-- PROCEDURE [
    box: Window.Box, options: TextSW.Options ← defaultOptions,
    initialState: ToolWindow.State ← active] RETURNS [sw: Window.Handle];
Create: --FormSW-- PROCEDURE [
    sw: Window.Handle, clientItemsProc: ClientItemsProcType,
    readOnlyNotifyProc: ReadOnlyProcType ← IgnoreReadOnlyProc,
    options: Options ← [], initialState: ToolWindow.State ← active,
    zone: UNCOUNTED ZONE ← NIL];
Create: --Heap-- PROCEDURE [
    initial: Environment.PageCount,
    maxSize: Environment.PageCount ← unlimitedSize,
    increment: Environment.PageCount ← 4, swapUnitSize: Space.SwapUnitSize ← 0,
    threshold: NWords ← minimumNodeSize, largeNodeThreshold: NWords ← 128,
    ownerChecking: BOOLEAN ← FALSE, checking: BOOLEAN ← FALSE]
    RETURNS [UNCOUNTED ZONE];
Create: --MemoryStream-- PROCEDURE [b: Environment.Block]
    RETURNS [sH: Stream.Handle];
Create: --Menu-- PROCEDURE [
    items: Items, name: LONG STRING, permanent: BOOLEAN ← FALSE] RETURNS [Handle];
Create: --MSegment-- PROCEDURE [
    file: MFile.Handle ← NIL, release: ReleaseData, fileBase: File.PageNumber ← 0,
    pages: Environment.PageCount ← defaultPages,
    swapInfo: SwapUnitOption ← defaultSwapUnitOption, usage: Space.Usage ← 0]
    RETURNS [segment: Handle];
Create: --MsgSW-- PROCEDURE [
    sw: Window.Handle, lines: CARDINAL ← 1,
    options: TextSW.Options ← defaultOptions];
Create: --MStream-- PROCEDURE [
    file: MFile.Handle, release: ReleaseData,
    options: Stream.InputOptions ← Stream.defaultInputOptions,
    streamBase: File.PageNumber ← 0] RETURNS [stream: Handle];
Create: --NetworkStream-- PROCEDURE [
    remote: System.NetworkAddress,
    connectData: Environment.Block ← Environment.nullBlock,
    timeout: WaitTime ← defaultWaitTime, classOfService: ClassOfService ← bulk]
    RETURNS [Stream.Handle];
Create: --NSFile-- PROCEDURE [
    directory: Handle, attributes: AttributeList ← nullAttributeList,
    controls: Controls ← [], session: Session ← nullSession]
    RETURNS [file: Handle];
Create: --PieceSource-- PROCEDURE [
    original: TextSource.Handle, scratch: TextSource.Handle]
    RETURNS [source: TextSource.Handle];
```

```
  Create: --RetrieveDefs-- PROCEDURE [
      pollingInterval: CARDINAL ← 300,
      reportChanges: PROCEDURE [MBXState, LONG POINTER] ← NIL,
      clientData: LONG POINTER ← NIL] RETURNS [Handle];
  Create: --RS232C-- PROCEDURE [
      lineNumber: CARDINAL, commParams: CommParamHandle, preemptOthers:
  ReserveType,
      preemptMe: ReserveType] RETURNS [channel: ChannelHandle];
  Create: --TextSW-- PROCEDURE [
      sw: Window.Handle, source: TextSource.Handle, sink: TextSink.Handle ← NIL,
      options: Options ← defaultOptions, position: Position ← 0,
      allowTypeIn: BOOLEAN ← TRUE, resetLengthOnNewSession: BOOLEAN ← FALSE];
  Create: --Tool-- PROCEDURE [
      name: LONG STRING, makeSWsProc: MakeSWsProc, initialState: State ← default,
      clientTransition: ToolWindow.TransitionProcType ← NIL,
      movableBoundaries: BOOLEAN ← TRUE,
      initialBox: Window.Box ← ToolWindow.nullBox, cmSection: LONG STRING ← NIL,
      tinyName1: LONG STRING ← NIL, tinyName2: LONG STRING ← NIL,
      named: BOOLEAN ← TRUE] RETURNS [window: Window.Handle];
  Create: --ToolFont-- PROCEDURE [MFile.Handle] RETURNS [WindowFont.Handle];
  Create: --ToolWindow-- PROCEDURE [
      name: LONG STRING, adjust: AdjustProcType, transition: TransitionProcType,
      box: Box ← nullBox, limit: LimitProcType ← StandardLimitProc,
      initialState: State ← active, named: BOOLEAN ← TRUE,
      gravity: Window.Gravity ← nw] RETURNS [Handle];
  Create: --TTY-- PROCEDURE [
      name: LONG STRING ← NIL, backingStream: Stream.Handle ← NIL,
      ttyImpl: Stream.Handle ← NIL] RETURNS [h: Handle];
  Create: --TTYPort-- PROCEDURE [lineNumber: CARDINAL] RETURNS [ChannelHandle];
  Create: --TTYSW-- PROCEDURE [
      sw: Window.Handle, backupFile: LONG STRING, s: Stream.Handle ← NIL,
      newFile: BOOLEAN ← TRUE, options: TextSW.Options ← defaultOptions,
      resetLengthOnNewSession: BOOLEAN ← FALSE];
  Create: --Volume-- PROCEDURE [
      pvID: System.PhysicalVolumeID, size: PageCount, name: LONG STRING, type: Type,
      minPVPageNumber: PhysicalVolume.PageNumber ← 1] RETURNS [volume: ID];
  Create: --Zone-- PROCEDURE [
      storage: LONG POINTER, length: BlockSize, zoneBase: Base,
      threshold: BlockSize ← minimumNodeSize, checking: BOOLEAN ← FALSE]
      RETURNS [zH: Handle, s: Status];
  CreateBackstopLog: --Backstop-- PROCEDURE [
      size: CARDINAL, file: File.File, firstPageNumber: File.PageNumber ← 0];
  CreateClient: --TIP-- PROCEDURE [
      window: Window.Handle, table: Table ← NIL, notify: NotifyProc ← NIL];
  CreateCouple: --NSDataStream-- PROCEDURE RETURNS [Couple];
  createdBy: --NSAssignedTypes-- AttributeType = 2;
  CreateDirectory: --MFile-- PROCEDURE [dir: LONG STRING];
  createdOn: --NSAssignedTypes-- AttributeType = 3;
  CreateFile: --Floppy-- PROCEDURE [
      volume: VolumeHandle, size: PageCount, type: File.Type]
      RETURNS [file: FileHandle];
  CreateFloppyFromImage: --Floppy-- PROCEDURE [
      floppyDrive: CARDINAL ← 0, imageFile: File.File,
      firstImagePage: File.PageNumber, reformatFloppy: BOOLEAN,
      floppyDensity: Density ← default, floppySides: Sides ← default,
      numberOfFiles: CARDINAL ← 0, newLabelString: LONG STRING ← NIL];
```

```
  CreateIndirectStringIn: --UserInput-- PROCEDURE [
      from: Window.Handle, to: Window.Handle];
  CreateIndirectStringInOut: --UserInput-- PROCEDURE [
      from: Window.Handle, to: Window.Handle];
  CreateIndirectStringOut: --UserInput-- PROCEDURE [
      from: Window.Handle, to: Window.Handle];
  CreateInitialMicrocodeFile: --Floppy-- PROCEDURE [
      volume: VolumeHandle, size: PageCount, type: File.Type,
      startingPageNumber: PageNumber ← 1] RETURNS [file: FileHandle];
  CreateListener: --NetworkStream-- PROCEDURE [addr: System.NetworkAddress]
      RETURNS [ListenerHandle];
  CreateMCR: --FileWindow-- Menu.MCRType;
  CreateMDS: --Heap-- PROCEDURE [
      initial: Environment.PageCount,
      maxSize: Environment.PageCount ← unlimitedSize,
      increment: Environment.PageCount ← 4, swapUnitSize: Space.SwapUnitSize ← 0,
      threshold: NWords ← minimumNodeSize, largeNodeThreshold: NWords ← 128,
      ownerChecking: BOOLEAN ← FALSE, checking: BOOLEAN ← FALSE] RETURNS
  [MDSZone];
  CreatePeriodicNotify: --UserInput-- PROCEDURE [
      proc: PeriodicProcType, window: Window.Handle, rate: Process.Ticks]
      RETURNS [PeriodicNotifyHandle];
  CreatePhysicalVolume: --PhysicalVolume-- PROCEDURE [
      instance: Handle, name: LONG STRING] RETURNS [ID];
  CreateProcType: --Context-- TYPE = PROCEDURE RETURNS [Data, DestroyProcType];
  CreateReplier: --PacketExchange-- PROCEDURE [
      local: System.NetworkAddress, requestCount: CARDINAL ← 1,
      waitTime: WaitTime ← defaultWaitTime,
      retransmissionInterval: WaitTime ← defaultRetransmissionInterval]
      RETURNS [ExchangeHandle];
  CreateRequestor: --PacketExchange-- PROCEDURE [
      waitTime: WaitTime ← defaultWaitTime,
      retransmissionInterval: WaitTime ← defaultRetransmissionInterval]
      RETURNS [ExchangeHandle];
  CreateScrollWindow: --UserTerminalExtras-- PROCEDURE [
      locn: UserTerminal.Coordinate, width: CARDINAL, height: CARDINAL];
  CreateServer: --NSTimeServer-- PROCEDURE;
  CreateStringInOut: --UserInput-- PROCEDURE [
      window: Window.Handle, in: StringProcType, out: StringProcType,
      caretProc: CaretProcType ← NopCaretProc];
  CreateSubwindow: --ToolWindow-- PROCEDURE [
      parent: Handle, display: DisplayProcType ← NIL, box: Box ← nullBox,
      gravity: Window.Gravity ← nw] RETURNS [Handle];
  CreateTable: --TIP-- PROCEDURE [
      file: LONG STRING ← NIL, opaque: BOOLEAN ← FALSE, z: UNCOUNTED ZONE ← NIL,
      contents: LONG STRING ← NIL] RETURNS [table: Table];
  createTool: --EventTypes-- Supervisor.Event;
  CreateTransducer: --NetworkStream-- PROCEDURE [
      local: System.NetworkAddress, remote: System.NetworkAddress,
      connectData: Environment.Block ← Environment.nullBlock,
      localConnID: ConnectionID, remoteConnID: ConnectionID,
      activelyEstablish: BOOLEAN, timeout: WaitTime ← defaultWaitTime,
      classOfService: ClassOfService ← bulk] RETURNS [Stream.Handle];
  CreateTTYInstance: --TTY-- PROCEDURE [
      name: LONG STRING, backingStream: Stream.Handle, tty: Handle]
      RETURNS [ttyImpl: Stream.Handle, backing: Stream.Handle];
```

```
CreateUniform: --Heap-- PROCEDURE [
    initial: Environment.PageCount,
    maxSize: Environment.PageCount ← unlimitedSize,
    increment: Environment.PageCount ← 4, swapUnitSize: Space.SwapUnitSize ← 0,
    objectSize: NWords, ownerChecking: BOOLEAN ← FALSE, checking: BOOLEAN ←
FALSE]
    RETURNS [UNCOUNTED ZONE];
createWindow: --EventTypes-- Supervisor.Event;
CredentialEvents: --EventTypes-- TYPE = [400..499];
Credentials: --Authenticator-- TYPE = NSName.Credentials;
Credentials: --NSFile-- TYPE = NSName.Credentials;
Credentials: --NSName-- TYPE = RECORD [
    type: CredentialsType, value: CredentialsContent];
CredentialsContent: --NSName-- TYPE [3];
CredentialsType: --NSName-- TYPE = MACHINE DEPENDENT{
    superWeak, weak, strong, (177777B)};
Current: --Time-- PROCEDURE RETURNS [time: System.GreenwichMeanTime];
CurrentSelection: --Put-- PROCEDURE [h: Window.Handle ← NIL];
currentVersion: --VolumeConversion-- CARDINAL = 0;
cursor: --UserTerminal-- READONLY LONG POINTER TO READONLY Coordinate;
CursorArray: --OnlineDiagnostics-- TYPE = ARRAY [0..15] OF WORD;
CursorArray: --UserTerminal-- TYPE = ARRAY [0..15] OF WORD;
CursorState: --HeraldWindow-- TYPE = {invert, negative, positive};
D1: --KeyStations-- Bit = 96;
D2: --KeyStations-- Bit = 95;
DamageStatus: --PhysicalVolume-- TYPE = {okay, damaged, lost};
Data: --Context-- TYPE = LONG POINTER;
DataError: --Floppy-- ERROR [
    file: FileHandle, page: PageNumber, vm: LONG POINTER];
Date: --Format-- PROCEDURE [
    proc: StringProc, pt: Time.Packed, format: DateFormat ← noSeconds,
    zone: Time.TimeZoneStandard ← ANSI, clientData: LONG POINTER ← NIL];
Date: --Put-- PROCEDURE [
    h: Window.Handle ← NIL, pt: Time.Packed,
    format: Format.DateFormat ← noSeconds];
DateFormat: --Format-- TYPE = {dateOnly, noSeconds, dateTime, full, mailDate};
DateFormat: --TTY-- TYPE = Format.DateFormat;
DBITAND: --Inline-- DBitOp;
DBITNOT: --Inline-- PROCEDURE [LONG UNSPECIFIED] RETURNS [LONG UNSPECIFIED];
DBitOp: --Inline-- TYPE = PROCEDURE [LONG UNSPECIFIED, LONG UNSPECIFIED]
    RETURNS [LONG UNSPECIFIED];
DBITOR: --Inline-- DBitOp;
DBITSHIFT: --Inline-- PROCEDURE [value: LONG UNSPECIFIED, count: INTEGER]
    RETURNS [LONG UNSPECIFIED];
DBITXOR: --Inline-- DBitOp;
DCSFileType: --FileTypes-- TYPE = CARDINAL [512..767];
deactivate: --EventTypes-- Supervisor.Event;
Deactivate: --MSegment-- PROCEDURE [segment: Handle];
Deactivate: --ToolWindow-- PROCEDURE [window: Handle]
    RETURNS [aborted: BOOLEAN];
DeallocateBands: --RavenFace-- PROCEDURE;
DeallocateListOfIDs: --LibrarianUtility-- PROCEDURE [array: IDArrayHandle]
    RETURNS [IDArrayHandle];
DebugEvents: --EventTypes-- TYPE = [0..99];
debuggerVolumeID: --Volume-- READONLY ID;
debugging: --EventTypes-- Supervisor.Event;
```

```
debugging: --Profile-- READONLY BOOLEAN;
debuggingOnUtilityPilot: --PilotSwitches-- PilotDomainA = 77C;
Decimal: --Format-- PROCEDURE [
    proc: StringProc, n: INTEGER, clientData: LONG POINTER ← NIL];
Decimal: --Put-- PROCEDURE [h: Window.Handle ← NIL, n: INTEGER];
Decimal: --Token-- PROCEDURE [h: Handle, signalOnError: BOOLEAN ← TRUE]
    RETURNS [i: INTEGER];
DecimalFormat: --Format-- NumberFormat;
DecodeBoolean: --NSFile-- PROCEDURE [Words] RETURNS [b: BOOLEAN];
DecodeCardinal: --NSFile-- PROCEDURE [Words] RETURNS [c: CARDINAL];
DecodeInteger: --NSFile-- PROCEDURE [Words] RETURNS [i: INTEGER];
DecodeLongCardinal: --NSFile-- PROCEDURE [Words] RETURNS [lc: LONG CARDINAL];
DecodeLongInteger: --NSFile-- PROCEDURE [Words] RETURNS [li: LONG INTEGER];
DecodeParameters: --NSName-- PROCEDURE [
    z: UNCOUNTED ZONE,
    encoding: LONG DESCRIPTOR FOR ARRAY CARDINAL OF UNSPECIFIED,
    parameters: Courier.Parameters];
DecodeSimpleCredentials: --NSName-- PROCEDURE [credentials: Credentials]
    RETURNS [SimpleCredentials];
DecodeSimpleVerifier: --NSName-- PROCEDURE [verifier: Verifier]
    RETURNS [SimpleVerifier];
DecodeString: --NSFile-- PROCEDURE [Words] RETURNS [s: String];
DecodeSwitches: --OthelloOps-- PROCEDURE [switchString: LONG STRING]
    RETURNS [switches: System.Switches];
defaultAccessList: --NSAssignedTypes-- AttributeType = 20;
defaultBaseDirectoryProc: --NSVolumeControl-- READONLY BaseDirectoryProc;
defaultBoxWidth: --FormSW-- CARDINAL = 0;
DefaultCheckCredentialsProc: --NSSessionControl-- CheckCredentialsProc;
DefaultConfirm: --HeraldWindow-- ConfirmProcType;
defaultExpirationTime: --Authenticator-- Seconds = 250600B;
defaultFileServerProtocol: --Profile-- READONLY FileServerProtocol;
defaultFont: --WindowFont-- READONLY Handle;
DefaultGetCredentialsProc: --NSSessionControl-- GetCredentialsProc;
DefaultHeight: --Tool-- INTEGER = 0;
defaultID: --NSSegment-- ID = 0;
defaultMask: --Expand-- Mask;
DefaultMembershipProc: --NSSessionControl-- MembershipProc;
defaultName: --NSVolumeControl-- READONLY NSString.String;
defaultOptions: --FileSW-- Options;
defaultOptions: --FileWindow-- TextSW.Options;
defaultOptions: --MLoader-- Options;
defaultOptions: --MsgSW-- TextSW.Options;
defaultOptions: --TextSW-- Options;
defaultOptions: --TTYSW-- TextSW.Options;
defaultOrdering: --NSFile-- key Ordering;
defaultPageCount: --Floppy-- PageNumber = 37777777777B;
defaultPages: --MSegment-- Environment.PageCount = 37777777777B;
DefaultPost: --HeraldWindow-- Format.StringProc;
DefaultPutback: --Real-- PROCEDURE [CHARACTER];
defaultRetransmissionInterval: --PacketExchange-- WaitTime = 72460B;
DefaultSinglePrecision: --Real-- CARDINAL = 7;
defaultSwapUnitOption: --MSegment-- SwapUnitOption;
defaultSwapUnitSize: --MSegment-- SwapUnitSize = 0;
defaultTime: --Time-- System.GreenwichMeanTime;
defaultTimeout: --NSFile-- Timeout = 177777B;
defaultTimeout: --NSVolumeControl-- READONLY NSFile.Timeout;
```

```
DefaultUnloadProc: --Exec-- ExecProc;
defaultValueSize: --BTree-- ValueSize = 3;
defaultVolume: --NSVolumeControl-- READONLY Volume.ID;
defaultWaitTime: --NetworkStream-- WaitTime = 165140B;
defaultWaitTime: --PacketExchange-- WaitTime = 165140B;
Defined: --Cursor-- TYPE = Type [activate..groundedText];
DEL: --Ascii-- CHARACTER = 177C;
Delete: --BTree-- PROCEDURE [tree: Tree];
Delete: --Courier-- PROCEDURE [cH: Handle];
Delete: --File-- PROCEDURE [file: File];
Delete: --FileTransfer-- PROCEDURE [
    conn: Connection, file: FileName.VFN, veto: VetoProc ← NIL];
Delete: --Heap-- PROCEDURE [z: UNCOUNTED ZONE, checkEmpty: BOOLEAN ← FALSE];
Delete: --MFile-- PROCEDURE [file: Handle];
Delete: --MSegment-- PROCEDURE [segment: Handle];
Delete: --NSFile-- PROCEDURE [file: Handle, session: Session ← nullSession];
Delete: --NSSegment-- PROCEDURE [
    file: NSFile.Handle, segment: ID, session: Session ← nullSession];
Delete: --PacketExchange-- PROCEDURE [h: ExchangeHandle];
Delete: --RS232C-- PROCEDURE [channel: ChannelHandle];
Delete: --TTYPort-- PROCEDURE [channel: ChannelHandle];
DeleteAlias: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, aliasName: Name,
    distingName: Name] RETURNS [rc: ReturnCode];
DeleteByName: --NSFile-- PROCEDURE [
    directory: Handle, path: String, session: Session ← nullSession];
DeleteChild: --NSFile-- PROCEDURE [
    directory: Handle, id: ID, session: Session ← nullSession];
DeleteDistinguishedName: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
    distingName: Name] RETURNS [rc: ReturnCode];
DeleteDomainAccessMember: --MoreCH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
    element: CH.Element, domain: CH.Name, acl: ACLFlavor]
    RETURNS [rc: CH.ReturnCode];
DeleteFile: --Floppy-- PROCEDURE [file: FileHandle];
DeleteGroupMember: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
    element: Element, name: Name, pn: PropertyID, distingName: Name]
    RETURNS [rc: ReturnCode];
DeleteListener: --NetworkStream-- PROCEDURE [listenerH: ListenerHandle];
DeleteLog: --VolumeConversion-- PROCEDURE [volume: Volume.ID];
DeleteMDS: --Heap-- PROCEDURE [z: MDSZone, checkEmpty: BOOLEAN ← FALSE];
DeleteOrgAccessMember: --MoreCH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
    element: CH.Element, org: CH.Name, acl: ACLFlavor]
    RETURNS [rc: CH.ReturnCode];
DeleteProperty: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
    pn: PropertyID, distingName: Name] RETURNS [rc: ReturnCode];
DeletePropertyAccessMember: --MoreCH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
    element: CH.Element, name: CH.Name, pn: CH.PropertyID, acl: ACLFlavor,
    distingName: CH.Name] RETURNS [rc: CH.ReturnCode];
DeleteScrollWindow: --UserTerminalExtras-- PROCEDURE;
```

```
DeleteSelf: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
    pn: PropertyID, distingName: Name] RETURNS [rc: ReturnCode];
DeleteServer: --NSTimeServer-- PROCEDURE;
DeleteSubString: --NSString-- PROCEDURE [s: SubString] RETURNS [String];
DeleteTempFiles: --OthelloOps-- PROCEDURE [Volume.ID];
DeleteText: --TextSW-- PROCEDURE [
    sw: Window.Handle, pos: Position, count: LONG CARDINAL,
    keepTrash: BOOLEAN ← TRUE];
DeleteThisSW: --Tool-- PROCEDURE [sw: Window.Handle];
DeleteWhenReleased: --MFile-- PROCEDURE [file: Handle];
Delimited: --Token-- FilterProcType;
DelinkSubwindow: --ToolWindow-- PROCEDURE [child: Handle];
Density: --Floppy-- TYPE = {single, double, default};
descendingPositionOrdering: --NSFile-- key Ordering;
Describe: --CHLookup-- PROCEDURE [
    sH: Stream.Handle, op: Operation, type: CHPIDs.PropertyID,
    buffer: LONG POINTER];
DescribeAddress: --NSAddr-- Courier.Description;
DescribeCredentials: --NSName-- Courier.Description;
DescribeHeader: --ExpeditedCourier-- Courier.Description;
DescribeNameRecord: --NSName-- Courier.Description;
DescribeNSAddr: --NSAddr-- Courier.Description;
DescribeString: --NSString-- Courier.Description;
DescribeTicket: --NSDataStream-- Courier.Description;
DescribeVerifier: --NSName-- Courier.Description;END.
Description: --Courier-- TYPE = PROCEDURE [notes: Notes];
Deserialize: --NSFile-- PROCEDURE [
    directory: Handle, source: Source,
    attributes: AttributeList ← nullAttributeList, controls: Controls ← [],
    session: Session ← nullSession] RETURNS [file: Handle];
DeserializeFromBlock: --CH-- PROCEDURE [
    parms: Courier.Parameters, heap: UNCOUNTED ZONE, blk: Environment.Block]
    RETURNS [succeeded: BOOLEAN];
DeserializeFromBlock: --ExpeditedCourier-- PROCEDURE [
    parms: Courier.Parameters, heap: Heap.Handle, blk: Environment.Block]
    RETURNS [succeeded: BOOLEAN];
DeserializeFromRhs: --CH-- PROCEDURE [
    parms: Courier.Parameters, heap: UNCOUNTED ZONE, rhs: Buffer]
    RETURNS [succeeded: BOOLEAN];
DeserializeParameters: --Courier-- PROCEDURE [
    parameters: Parameters, sH: Stream.Handle, zone: UNCOUNTED ZONE];
DesiredProperties: --FileTransfer-- TYPE = PACKED ARRAY ValidProperties OF
    BOOLEAN ← ALL[FALSE];
Destroy: --Context-- PROCEDURE [type: Type, window: Window.Handle];
destroy: --EventTypes-- Supervisor.Event;
Destroy: --FileSW-- PROCEDURE [sw: Window.Handle];
Destroy: --FileTransfer-- PROCEDURE [Connection];
Destroy: --FileWindow-- PROCEDURE [sw: Window.Handle];
Destroy: --FormSW-- PROCEDURE [Window.Handle];
Destroy: --MemoryStream-- PROCEDURE [sH: Stream.Handle];
Destroy: --Menu-- PROCEDURE [Handle];
Destroy: --MsgSW-- PROCEDURE [sw: Window.Handle];
Destroy: --RetrieveDefs-- PROCEDURE [Handle];
Destroy: --TextSW-- PROCEDURE [sw: Window.Handle];
Destroy: --Tool-- PROCEDURE [window: Window.Handle];
```

```
Destroy: --ToolFont-- PROCEDURE [WindowFont.Handle];
Destroy: --ToolWindow-- PROCEDURE [window: Handle];
Destroy: --TTY-- PROCEDURE [h: Handle, deleteBackingFile: BOOLEAN ← FALSE];
Destroy: --TTYSW-- PROCEDURE [sw: Window.Handle];
DestroyAll: --Context-- PROCEDURE [window: Window.Handle];
DestroyClient: --TIP-- PROCEDURE [window: Window.Handle];
DestroyFromBackgroundProcess: --TTYSW-- PROCEDURE [sw: Window.Handle];
DestroyIndirectStringIn: --UserInput-- PROCEDURE [Window.Handle];
DestroyIndirectStringInOut: --UserInput-- PROCEDURE [Window.Handle];
DestroyIndirectStringOut: --UserInput-- PROCEDURE [Window.Handle];
DestroyMCR: --FileWindow-- Menu.MCRType;
DestroyProcType: --Context-- TYPE = PROCEDURE [Data, Window.Handle];
DestroyStringInOut: --UserInput-- PROCEDURE [Window.Handle];
DestroySW: --Tool-- PROCEDURE [window: Window.Handle];
Detach: --Process-- PROCEDURE [process: PROCESS];
Detail: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT RECORD [
    msec(0:0..15): CARDINAL, count(1:0..15): CARDINAL];
DevelopmentEnvironmentDomain: --PilotSwitches-- TYPE = SwitchName
[101C..132C];
DeviceIndex: --PageScavenger-- TYPE = CARDINAL;
DeviceStatus: --RS232C-- TYPE = RECORD [
    statusAborted: BOOLEAN,
    dataLost: BOOLEAN,
    breakDetected: BOOLEAN,
    clearToSend: BOOLEAN,
    dataSetReady: BOOLEAN,
    carrierDetect: BOOLEAN,
    ringHeard: BOOLEAN,
    ringIndicator: BOOLEAN,
    deviceError: BOOLEAN];
DeviceStatus: --TTYPort-- TYPE = RECORD [
    aborted: BOOLEAN,
    breakDetected: BOOLEAN,
    dataTerminalReady: BOOLEAN,
    readyToGet: BOOLEAN,
    readyToPut: BOOLEAN,
    requestToSend: BOOLEAN];
DiagnosticsFileType: --FileTypes-- TYPE = CARDINAL [22300B..22377B];
diagnosticsServerSocket: --NSConstants-- System.SocketNumber;
Dial: --Dialup-- PROCEDURE [
    dialerNumber: CARDINAL, number: LONG POINTER TO Number, retries: RetryCount]
    RETURNS [Outcome];
DialMode: --RS232C-- TYPE = RS232CEnvironment.DialMode;
DialMode: --RS232CEnvironment-- TYPE = {manual, auto};
DialupOutcome: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT{
    success, failure, aborted, formatError, transmissionError, dataLineOccupied,
    dialerNotPresent, dialingTimeout, transferTimeout, otherError, noHardware,
    noSuchLine, channelInUse, unimplementedFeature, invalidParamater};
DialupTest: --CommOnlineDiagnostics-- PROCEDURE [
    rs232ClineNumber: CARDINAL, phoneNumber: LONG POINTER TO Dialup.Number,
    host: System.NetworkAddress ← System.nullNetworkAddress]
    RETURNS [outcome: DialupOutcome];
DialupTest: --RemoteCommDiags-- PROCEDURE [
    host: System.NetworkAddress, rs232ClineNumber: CARDINAL,
    phoneNumber: LONG POINTER TO Dialup.Number]
    RETURNS [outcome: CommOnlineDiagnostics.DialupOutcome];
```

```
DifferentType: --LogFile-- ERROR;
Dims: --Window-- TYPE = RECORD [w: INTEGER, h: INTEGER];
DirectedBroadcastCall: --ExpeditedCourier-- PROCEDURE [
    programNumber: LONG CARDINAL, versionNumber: CARDINAL,
    procedureNumber: CARDINAL, arguments: Courier.Parameters,
    address: System.NetworkAddress, action: ExpandingRingAction,
    eachResponse: ResponseProc, responseBufferCount: CARDINAL ← 5];
Direction: --BitBlt-- TYPE = {forward, backward};
Direction: --NSFile-- TYPE = MACHINE DEPENDENT{forward, backward};
directoryCreated: --EventTypes-- Supervisor.Event;
directoryDeleted: --EventTypes-- Supervisor.Event;
Disable: --Log-- PROCEDURE RETURNS [State];
DisableAborts: --Process-- PROCEDURE [condition: LONG POINTER TO CONDITION];
disableMapLog: --PilotSwitches-- PilotDomainA = 67C;
DisableTimeout: --Process-- PROCEDURE [condition: LONG POINTER TO CONDITION];
DiskAddress: --FloppyChannel-- TYPE = MACHINE DEPENDENT RECORD [
    cylinder(0:0..15): CARDINAL,
    head(1:0..7): [0..255],
    sector(1:8..15): [0..255]];
DiskPageNumber: --FormatPilotDisk-- TYPE = PhysicalVolume.PageNumber;
DiskStatus: --PageScavenger-- TYPE = {
    goodCompletion, noSuchPage, labelDoesNotMatch, seekFailed, checkError,
    dataError, hardwareError, notReady, labelError};
Dispatcher: --Courier-- TYPE = PROCEDURE [
    cH: Handle, procedureNumber: CARDINAL, arguments: Arguments,
    results: Results];
DispatcherProc: --ExpeditedCourier-- TYPE = PROCEDURE [
    programNumber: LONG CARDINAL, version: CARDINAL, procedureNumber: CARDINAL,
    serializedRequest: Environment.Block, replyMemoryStream: Stream.Handle,
    callWasABroadcast: BOOLEAN] RETURNS [sendReply: BOOLEAN];
Display: --FormSW-- PROCEDURE [sw: Window.Handle, yOffset: CARDINAL ← 0];
Display: --RavenFace-- PROCEDURE [char: ConsoleCharacter];
displayedPages: --HeraldWindow-- READONLY LONG CARDINAL;
DisplayEvents: --EventTypes-- TYPE = [800..999];
DisplayFieldsProc: --OnlineDiagnostics-- TYPE = PROCEDURE [
    fields: DESCRIPTOR FOR ARRAY CARDINAL OF Field, title: FloppyMessage ← tFirst,
    fieldType: FieldDataType, numberOfColumns: CARDINAL ← 3];
DisplayItem: --FormSW-- PROCEDURE [sw: Window.Handle, index: CARDINAL];
DisplayLibjectID: --LibrarianUtility-- PROCEDURE [
    sw: Window.Handle, id: Librarian.LibjectID];
DisplayNumberedTableProc: --OnlineDiagnostics-- TYPE = PROCEDURE [
    values: LONG DESCRIPTOR FOR ARRAY CARDINAL OF UNSPECIFIED,
    rowNameHeader: FloppyMessage ← tFirst, title: FloppyMessage ← tFirst,
    numOfColumns: CARDINAL, startNum: INTEGER, fieldType: FieldDataType];
displayOff: --EventTypes-- Supervisor.Event;
displayOn: --EventTypes-- Supervisor.Event;
DisplayProcType: --ToolWindow-- TYPE = PROCEDURE [window: Handle];
DisplayPropertyList: --LibrarianUtility-- PROCEDURE [
    sw: Window.Handle, plist: Librarian.PropertyList, properties: PropertyArray,
    leader: LONG STRING, outputTags: BOOLEAN];
DisplayPropertyPair: --LibrarianUtility-- PROCEDURE [
    sw: Window.Handle, pp: Librarian.PropertyPair, properties: PropertyArray,
    leader: LONG STRING, outputTags: BOOLEAN];
displayState: --Event-- READONLY Supervisor.SubsystemHandle;
DisplayTableProc: --OnlineDiagnostics-- TYPE = PROCEDURE [
    headers: DESCRIPTOR FOR ARRAY CARDINAL OF FloppyMessage,
```

```
    rowNames: DESCRIPTOR FOR ARRAY CARDINAL OF FloppyMessage,
    values: DESCRIPTOR FOR ARRAY CARDINAL OF DESCRIPTOR FOR ARRAY CARDINAL OF
      UNSPECIFIED, title: FloppyMessage ← tFirst, fieldType: FieldDataType];
DisplayVersion: --LibrarianUtility-- PROCEDURE [
    sw: Window.Handle, version: Librarian.LibjectVersion];
DistinguishSegmentedFileType: --NSVolumeControl-- PROCEDURE [type: NSFile.Type];
DivideCheck: --Runtime-- SIGNAL;
DivideInfinityNaN: --Real-- LONG CARDINAL = 5;
DIVMOD: --Inline-- PROCEDURE [num: CARDINAL, den: CARDINAL]
    RETURNS [quotient: CARDINAL, remainder: CARDINAL];
DocProcFileType: --FileTypes-- TYPE = CARDINAL [6000B..7777B];
DoEditAction: --TextSW-- PROCEDURE [
    sw: Window.Handle, action: TextSource.EditAction]
    RETURNS [delta: LONG INTEGER];
DoesNotExist: --TextSW-- SIGNAL;
domain: --EventTypes-- Supervisor.Event;
Domain: --NSName-- TYPE = String ← NSString.nullString;
DomainName: --CH-- TYPE = NSName.Domain;
DoneWithProcess: --Event-- PROCEDURE [Handle];
dontCare: --MFile-- InitialLength = 37777777777B;
dontChangeFile: --MSegment-- MFile.Handle;
dontChangeFileBase: --MSegment-- File.PageNumber = 37777777777B;
dontChangePages: --MSegment-- Environment.PageCount = 37777777776B;
dontChangeReleaseData: --MSegment-- ReleaseData;
dontChangeUsage: --MSegment-- Space.Usage = 255;
dontRelease: --MFile-- ReleaseData;
DownUp: --JLevelIVKeys-- TYPE = KeyStations.DownUp;
DownUp: --Keys-- TYPE = KeyStations.DownUp;
DownUp: --KeyStations-- TYPE = {down, up};
DownUp: --LevelIIIKeys-- TYPE = KeyStations.DownUp;
DownUp: --LevelIVKeys-- TYPE = KeyStations.DownUp;
DownUp: --TIP-- TYPE = Keys.DownUp;
DozeOff: --RavenFace-- PROCEDURE;
DPCell: --Atom-- TYPE = RECORD [first: LONG STRING, rest: AList];
DrawNameFrame: --ToolWindow-- DisplayProcType;
DrawRectangle: --ToolWindow-- PROCEDURE [
    window: Handle, box: Box, width: CARDINAL ← 1];
Drive: --FloppyChannel-- TYPE = CARDINAL;
DstFunc: --BitBlt-- TYPE = {null, and, or, xor};
DstFunc: --Display-- TYPE = BitBlt.DstFunc;
DumpObject: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
    eachBlock: PROCEDURE [LONG POINTER, CARDINAL], distingName: Name]
    RETURNS [rc: ReturnCode];
Duplexity: --RS232C-- TYPE = RS232CEnvironment.Duplexity;
Duplexity: --RS232CEnvironment-- TYPE = {full, half};
eatGerm: --PilotSwitches-- PilotDomainC = 376C;
ebcdicByteSync: --RS232CCorrespondents--
    RS232CEnvironment.AutoRecognitionOutcome;
EchoClass: --TTY-- TYPE = {none, plain, stars};
echoerSocket: --NSConstants-- System.SocketNumber;
EchoEvent: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT{
    success, late, timeout, badDataGoodCRC, sizeChange, unexpected};
EchoParams: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT RECORD [
    totalCount(0:0..15): CARDINAL ← 177777B,
    safetyTOInMsecs(1:0..31): LONG CARDINAL ← 165140B,
```

```
    minPacketSizeInBytes(3:0..15): CARDINAL ← 2,
    maxPacketSizeInBytes(4:0..15): CARDINAL ← 512,
    wordContents(5:0..15): WordsInPacket ← incrWords,
    constant(6:0..15): CARDINAL ← 125252B,
    waitForResponse(7:0..15): BOOLEAN ← TRUE,
    minMsecsBetweenPackets(8:0..15): CARDINAL ← 0,
    checkContents(9:0..15): BOOLEAN ← TRUE,
    showMpCode(10:0..15): BOOLEAN ← FALSE];
EchoResults: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT RECORD [
    totalTime(0:0..31): LONG CARDINAL,
    totalAttempts(2:0..31): LONG CARDINAL,
    successes(4:0..31): LONG CARDINAL,
    timeouts(6:0..31): LONG CARDINAL,
    late(8:0..31): LONG CARDINAL,
    unexpected(10:0..31): LONG CARDINAL,
    bad(12:0..31): LONG CARDINAL,
    avgDelayInMsecs(14:0..31): LONG CARDINAL,
    okButDribble(16:0..31): LONG CARDINAL,
    badAlignmentButOkCrc(18:0..31): LONG CARDINAL,
    packetTooLong(20:0..31): LONG CARDINAL,
    overrun(22:0..31): LONG CARDINAL,
    idleInput(24:0..31): LONG CARDINAL,
    tooManyCollisions(26:0..31): LONG CARDINAL,
    lateCollisions(28:0..31): LONG CARDINAL,
    underrun(30:0..31): LONG CARDINAL,
    stuckOutput(32:0..31): LONG CARDINAL,
    spare(34:0..31): LONG CARDINAL];
echoServer: --ProtocolCertification-- Stage;
echoUser: --ProtocolCertification-- Stage;
ECS: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    address(0:0..95): System.NetworkAddress, location(6:0..63): NSString.String];
ECSDescribe: --CHLookup-- Courier.Description;
ECSPt: --CHLookup-- TYPE = LONG POINTER TO ECS;
edit: --EventTypes-- Supervisor.Event;
ElapseTime: --ExpeditedCourier-- TYPE = NSTypes.WaitTime;
electronicMailFirstSocket: --NSConstants-- System.SocketNumber;
electronicMailLastSocket: --NSConstants-- System.SocketNumber;
Element: --CH-- TYPE = LONG POINTER TO ThreePartName;
Ellipse: --Display-- PROCEDURE [
    window: Handle, center: Window.Place, xRadius: INTEGER, yRadius: INTEGER,
    bounds: Window.BoxHandle ← NIL];
Empty: --BTree-- PROCEDURE [tree: Tree] RETURNS [BOOLEAN];
EmptyString: --MDSStorage-- PROCEDURE [s: LONG STRING] RETURNS [BOOLEAN];
EnableAborts: --Process-- PROCEDURE [condition: LONG POINTER TO CONDITION];
EncodeBoolean: --NSFile-- PROCEDURE [b: BOOLEAN] RETURNS [Words];
EncodeCardinal: --NSFile-- PROCEDURE [c: CARDINAL] RETURNS [Words];
EncodeInteger: --NSFile-- PROCEDURE [i: INTEGER] RETURNS [Words];
EncodeLongCardinal: --NSFile-- PROCEDURE [lc: LONG CARDINAL] RETURNS [Words];
EncodeLongInteger: --NSFile-- PROCEDURE [li: LONG INTEGER] RETURNS [Words];
EncodeParameters: --NSName-- PROCEDURE [
    z: UNCOUNTED ZONE, parameters: Courier.Parameters]
    RETURNS [LONG DESCRIPTOR FOR ARRAY CARDINAL OF UNSPECIFIED];
EncodeSimpleCredentials: --NSName-- PROCEDURE [
    z: UNCOUNTED ZONE, simpleCredentials: SimpleCredentials]
    RETURNS [Credentials];
```

```
EncodeSimpleVerifier: --NSName-- PROCEDURE [simpleVerifier: SimpleVerifier]
    RETURNS [Verifier];
EncodeString: --NSFile-- PROCEDURE [s: String] RETURNS [Words];
endEnumeration: --Router-- READONLY System.NetworkNumber;
EndOf: --MStream-- PROCEDURE [stream: Handle] RETURNS [BOOLEAN];
EndOf: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
endOfBandCmd: --BandBLT-- CARDINAL = 8;
EndOfCommandLine: --Exec-- PROCEDURE [h: Handle] RETURNS [BOOLEAN];
endOfInput: --MailParse-- CHARACTER = 203C;
endOfList: --MailParse-- CHARACTER = 204C;
endOfPageCmd: --BandBLT-- CARDINAL = 9;
EnlinkSubwindow: --ToolWindow-- PROCEDURE [
    parent: Handle, child: Handle, youngerSibling: Handle];
Enter: --NSSessionControl-- PROCEDURE [session: NSFile.Session, id: ServiceID];
Entry: --NSVolumeControl-- TYPE = MACHINE DEPENDENT RECORD [
    file(0:0..79): NSFile.ID,
    type(5:0..31): NSFile.Type,
    numberOfProblems(7:0..31): LONG CARDINAL];
EntryPointer: --NSVolumeControl-- TYPE = LONG POINTER TO Entry;
EntryType: --Scavenger-- TYPE = MACHINE DEPENDENT{
    unreadable, missing, duplicate, orphan};
Enumerate: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Pattern,
    pn: PropertyID, eachName: NameStreamProc] RETURNS [rc: ReturnCode];
Enumerate: --DebugUsefulDefs-- PROCEDURE [
    proc: PROCEDURE [GFHandle] RETURNS [BOOLEAN]] RETURNS [gf: GFHandle];
Enumerate: --FileSW-- PROCEDURE [proc: EnumerateProcType];
Enumerate: --FileTransfer-- PROCEDURE [
    conn: Connection, files: FileName.VFN, proc: ListProc];
Enumerate: --FileWindow-- PROCEDURE [proc: EnumerateProcType];
Enumerate: --Menu-- PROCEDURE [
    window: Window.Handle, which: EnumerateFor, proc: EnumerateProcType];
EnumerateAliases: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Pattern,
    eachAlias: NameStreamProc] RETURNS [rc: ReturnCode];
EnumerateCommands: --Exec-- PROCEDURE [
    userProc: PROCEDURE [
        name: LONG STRING, proc: ExecProc, help: ExecProc, unload: ExecProc,
        clientData: LONG POINTER] RETURNS [stop: BOOLEAN]];
Enumerated: --FormSW-- TYPE = RECORD [string: LONG STRING, value: UNSPECIFIED];
EnumeratedDescriptor: --FormSW-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
    Enumerated;
EnumeratedFeedback: --FormSW-- TYPE = {all, one};
EnumeratedHandle: --FormSW-- TYPE = LONG POINTER TO enumerated ItemObject;
EnumerateDirectory: --MFile-- PROCEDURE [
    name: LONG STRING, proc: EnumerateProc, which: EnumerationType];
EnumeratedItem: --FormSW-- PROCEDURE [
    tag: LONG STRING ← NIL, readOnly: BOOLEAN ← FALSE, invisible: BOOLEAN ← FALSE,
    drawBox: BOOLEAN ← FALSE, hasContext: BOOLEAN ← FALSE,
    place: Window.Place ← nextPlace, feedback: EnumeratedFeedback ← one,
    proc: EnumeratedNotifyProcType ← NopEnumeratedNotifyProc,
    copyChoices: BOOLEAN ← TRUE, choices: EnumeratedDescriptor,
    value: LONG POINTER, z: UNCOUNTED ZONE ← NIL] RETURNS [EnumeratedHandle];
EnumeratedNotifyProcType: --FormSW-- TYPE = PROCEDURE [
    sw: Window.Handle ← NIL, item: ItemHandle ← NIL, index: CARDINAL ← nullIndex,
    oldValue: UNSPECIFIED ← nullEnumeratedValue];
```

```
EnumerateDomains: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Pattern,
    eachDomain: NameStreamProc] RETURNS [rc: ReturnCode];
EnumerateExports: --Courier-- PROCEDURE
    RETURNS [enum: LONG DESCRIPTOR FOR Exports];
EnumerateFileType: --LibrarianUtility-- TYPE = {id, name};
EnumerateFor: --Menu-- TYPE = {all, inSW, availableInSW};
EnumerateInactiveWindows: --ToolWindow-- PROCEDURE [proc: EnumerateProcType];
EnumerateInvalidBoxes: --Window-- PROCEDURE [
    window: Handle, proc: PROCEDURE [Handle, Box]];
EnumerateLibjectProc: --LibrarianUtility-- TYPE = PROCEDURE [
    Librarian.Handle, Librarian.LibjectID, Librarian.PropertyList, CARDINAL]
    RETURNS [continue: BOOLEAN];
EnumerateLibjectStructure: --LibrarianUtility-- PROCEDURE [
    Librarian.Handle, LONG STRING, Librarian.SnapShotHandle,
    Librarian.PropertyList, BOOLEAN, EnumerateLibjectProc]
    RETURNS [Librarian.Handle];
EnumerateLibjectVersions: --LibrarianUtility-- PROCEDURE [
    Librarian.Handle, Librarian.LibjectID, Librarian.SnapShotHandle,
    Librarian.PropertyList, EnumerateVersionProc];
EnumerateNearbyDomains: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
    eachDomain: NameStreamProc] RETURNS [rc: ReturnCode];
EnumerateNewGroupElements: --CH-- TYPE = PROCEDURE [NameStreamProc];
EnumerateObjects: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Name,
    eachName: NameStreamProc] RETURNS [rc: ReturnCode];
EnumerateOrganizations: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
    orgPattern: Pattern, eachOrg: NameStreamProc] RETURNS [rc: ReturnCode];
EnumerateProc: --MFile-- TYPE = PROCEDURE [
    name: LONG STRING, fullName: LONG STRING, fileProc: FileAcquireProc,
    type: Type, spIndex: CARDINAL] RETURNS [done: BOOLEAN ← FALSE];
EnumerateProcType: --FileSW-- TYPE = PROCEDURE [
    sw: Window.Handle, name: LONG STRING, access: Access] RETURNS [done: BOOLEAN];
EnumerateProcType: --FileWindow-- TYPE = PROCEDURE [sw: Window.Handle]
    RETURNS [continue: ContinueStop];
EnumerateProcType: --Menu-- TYPE = PROCEDURE [
    window: Window.Handle, menu: Handle] RETURNS [stop: BOOLEAN];
EnumerateProcType: --ToolWindow-- TYPE = PROCEDURE [window: Window.Handle]
    RETURNS [done: BOOLEAN];
EnumerateRoutingTable: --Router-- PROCEDURE [
    previous: System.NetworkNumber, delay: CARDINAL]
    RETURNS [net: System.NetworkNumber];
EnumerateSecondarySelections: --TextSW-- PROCEDURE,[
    sw: Window.Handle, proc: PROCEDURE [TextData.Selection] RETURNS [BOOLEAN]];
EnumerateSessionAttributes: --NSSessionControl-- PROCEDURE [
    procedure: PROCEDURE [SessionAttributes]];
EnumerateSplits: --TextSW-- PROCEDURE [
    sw: Window.Handle, proc: SplitInfoProcType];
EnumerateState: --MFile-- TYPE = LONG POINTER TO EnumRec;
EnumerateSWProcType: --ToolWindow-- TYPE = PROCEDURE [
    window: Window.Handle, sw: Window.Handle] RETURNS [done: BOOLEAN];
EnumerateSWs: --ToolWindow-- PROCEDURE [
    window: Window.Handle, proc: EnumerateSWProcType];
```

```
EnumerateTree: --Window-- PROCEDURE [
    root: Handle, proc: PROCEDURE [window: Handle]];
EnumerateUsingFile: --LibrarianUtility-- PROCEDURE [
    Librarian.Handle, LONG STRING, Librarian.SnapShotHandle,
    Librarian.PropertyList, BOOLEAN, EnumerateFileType, EnumerateLibjectProc];
EnumerateUsingFileServer: --LibrarianUtility-- PROCEDURE [
    LONG STRING, Librarian.SnapShotHandle, Librarian.PropertyList, BOOLEAN,
    EnumerateLibjectProc];
EnumerateVersionProc: --LibrarianUtility-- TYPE = PROCEDURE [
    Librarian.Handle, Librarian.FullLibjectIDHandle, Librarian.PropertyList]
    RETURNS [continue: BOOLEAN];
EnumerationType: --MFile-- TYPE = {
    filesOnly, directoriesOnly, fileAndDirectories};
EnumRec: --MFile-- TYPE;
envoySocket: --NSConstants-- System.SocketNumber;
EqualCharacter: --NSString-- PROCEDURE [
    c: Character, s: String, index: CARDINAL] RETURNS [BOOLEAN];
EqualString: --NSString-- PROCEDURE [s1: String, s2: String] RETURNS [BOOLEAN];
EqualStrings: --NSString-- PROCEDURE [s1: String, s2: String] RETURNS [BOOLEAN];
EqualSubString: --NSString-- PROCEDURE [s1: SubString, s2: SubString]
    RETURNS [BOOLEAN];
EqualSubStrings: --NSString-- PROCEDURE [s1: SubString, s2: SubString]
    RETURNS [BOOLEAN];
EqualSystemElements: --NSSessionControl-- PROCEDURE [
    systemElement1: NSFile.SystemElement, systemElement2: NSFile.SystemElement]
    RETURNS [BOOLEAN];
EquivalentNames: --NSName-- PROCEDURE [n1: Name, n2: Name] RETURNS [BOOLEAN];
EquivalentSegments: --MSegment-- PROCEDURE [seg1: Handle, seg2: Handle]
    RETURNS [BOOLEAN];
EquivalentString: --NSString-- PROCEDURE [s1: String, s2: String]
    RETURNS [BOOLEAN];
EquivalentStrings: --NSString-- PROCEDURE [s1: String, s2: String]
    RETURNS [BOOLEAN];
EquivalentSubString: --NSString-- PROCEDURE [s1: SubString, s2: SubString]
    RETURNS [BOOLEAN];
EquivalentSubStrings: --NSString-- PROCEDURE [s1: SubString, s2: SubString]
    RETURNS [BOOLEAN];
Erase: --Volume-- PROCEDURE [volume: ID];
errL1: --ProtocolCertification-- Stage;
Error: --Authenticator-- ERROR [reason: Cause, forWhom: Name];
Error: --CHLookup-- ERROR [reason: CH.ReturnCode];
Error: --CmFile-- SIGNAL [code: ErrorCode];
Error: --Context-- ERROR [code: ErrorCode];
Error: --Courier-- ERROR [errorCode: ErrorCode];
Error: --File-- ERROR [type: ErrorType];
Error: --FileName-- SIGNAL;
Error: --FileSW-- SIGNAL [code: ErrorCode];
Error: --FileTransfer-- SIGNAL [conn: Connection, code: ErrorCode];
Error: --Floppy-- ERROR [error: ErrorType];
Error: --FloppyChannel-- ERROR [type: ErrorType];
Error: --FormSW-- SIGNAL [code: ErrorCode];
Error: --Heap-- ERROR [type: ErrorType];
Error: --Log-- ERROR [reason: ErrorType];
Error: --MailParse-- ERROR [code: ErrorCode, position: CARDINAL];
Error: --Menu-- ERROR [code: ErrorCode];
Error: --MFile-- ERROR [file: Handle, code: ErrorCode];
```

```
  Error: --MLoader-- ERROR [code: ErrorCode, string: LONG STRING];
  Error: --MSegment-- ERROR [segment: Handle, code: ErrorCode];
  Error: --MsgSW-- SIGNAL [code: ErrorCode];
  Error: --MStream-- ERROR [stream: Handle, code: ErrorCode];
  Error: --NSDataStream-- ERROR [errorCode: ErrorCode];
  Error: --NSFile-- ERROR [error: ErrorRecord];
  Error: --NSName-- ERROR [type: ErrorType];
  Error: --NSPrint-- ERROR [why: ErrorRecord];
  Error: --NSSegment-- ERROR [type: ErrorType];
  Error: --NSVolumeControl-- ERROR [type: ErrorType];
  Error: --ObjAlloc-- ERROR [error: ErrorType];
  Error: --PacketExchange-- ERROR [why: ErrorReason];
  Error: --PageScavenger-- ERROR [errorType: ErrorType];
  Error: --PhysicalVolume-- ERROR [error: ErrorType];
  Error: --Scavenger-- ERROR [error: ErrorType];
  Error: --Tool-- SIGNAL [code: ErrorCode];
  Error: --TTYSW-- SIGNAL [code: ErrorCode];
  Error: --UserInput-- ERROR [code: ErrorCode];
  Error: --UserTerminalExtras-- ERROR [type: ErrorType];
  Error: --Volume-- ERROR [error: ErrorType];
  Error: --VolumeConversion-- ERROR [error: ErrorType];
  Error: --Window-- ERROR [code: ErrorCode];
  Error: --WindowFont-- ERROR [code: ErrorCode];
  ErrorCode: --CmFile-- TYPE = {fileNotFound, invalidHandle, other};
  ErrorCode: --Context-- TYPE = {duplicateType, windowIsNIL, tooManyTypes, other};
  ErrorCode: --Courier-- TYPE = {
      transmissionMediumHardwareProblem, transmissionMediumUnavailable,
      transmissionMediumNotReady, noAnswerOrBusy, noRouteToSystemElement,
      transportTimeout, remoteSystemElementNotResponding,
noCourierAtRemoteSite,
      tooManyConnections, invalidMessage, noSuchProcedureNumber,
returnTimedOut,
      callerAborted, unknownErrorInRemoteProcedure, streamNotYours,
      truncatedTransfer, parameterInconsistency, invalidArguments,
      noSuchProgramNumber, protocolMismatch, duplicateProgramExport,
      noSuchProgramExport, invalidHandle, noError};
  ErrorCode: --FileSW-- TYPE = {
      notAFileSW, isAFileSW, notEditable, isEditable, accessDenied, other};
  ErrorCode: --FileTransfer-- TYPE = MACHINE DEPENDENT{
      illegalParameters, invalidObject, notAStream, illegalLogin(4), illegalConnect,
      skip, cantModify, retry, directoryFull, notFound, spare1, spare2,
      unknown(31)};
  ErrorCode: --FormSW-- TYPE = {alreadyAFormSW, notAFormSW, other};
  ErrorCode: --MailParse-- TYPE = {
      illegalCharacter, unclosedBracket, bracketNesting, implementationBug,
      phraseExpected, domainExpected, atomExpected, commaOrColonExpected,
      atExpected, spaceInLocalName, mailBoxExpected, missingSemiColon,
nestedGroup,
      endOfInput, commaExpected, fieldsAreAtoms, colonExpected, lessThanExpected,
      greaterThanExpected, noFromField};
  ErrorCode: --Menu-- TYPE = {
      isInstantiated, alreadyInstantiated, notInstantiated, contextNotAvailable,
      isPermanent, other};
  ErrorCode: --MFile-- TYPE = MACHINE DEPENDENT{
      noSuchFile, conflictingAccess, insufficientAccess, directoryFull,
      directoryNotEmpty, illegalName, noSuchDirectory, noRootDirectory, nullAccess,
```

```
        protectionFault, directoryOnSearchPath, illegalSearchPath, volumeNotOpen,
        volumeReadOnly, noRoomOnVolume, noSuchVolume, crossingVolumes,
        fileAlreadyExists, fileIsRemote, fileIsDirectory, invalidHandle, courierError,
        addressTranslationError, connectionSuspended, other(255)};
    ErrorCode: --MLoader-- TYPE = {
        invalidParameters, missingCode, badCode, exportedTypeClash, lookupFailure,
        gftFull, loadStateFull, insufficientAccess, alreadyStarted, invalidHandle,
        invalidGlobalFrame, other};
    ErrorCode: --MSegment-- TYPE = MACHINE DEPENDENT{
        zeroLength, insufficientVM, noSuchSegment, sharedSegment, baseOutOfRange,
        conflictingAccess, illegalAccess, invalidFile, dataSegmentNeedsPages,
        noRoomOnVolume, volumeReadOnly, other(177777B)};
    ErrorCode: --MsgSW-- TYPE = {appendOnly, notAMsgSW, other};
    ErrorCode: --MStream-- TYPE = MACHINE DEPENDENT{
        invalidHandle, indexOutOfRange, invalidOperation, fileTooLong,
        fileNotAvailable, invalidFile, other(177777B)};
    ErrorCode: --NSDataStream-- TYPE = {
        localEndIncorrect, tooManyLocalConnections, tooManyTickets,
unimplemented};
    ErrorCode: --Tool-- TYPE = {
        notATool, unknownSWType, swNotFound, invalidWindow, invalidParameters,
other};
    ErrorCode: --TTYSW-- TYPE = {notATTYSW, badTTYHandle, other};
    ErrorCode: --UserInput-- TYPE = {
        windowAlreadyHasStringInOut, noStringInOutForWindow,
noSuchPeriodicNotifier,
        other};
    ErrorCode: --Window-- TYPE = {
        illegalBitmap, illegalFloat, windowNotChildOfParent, whosSlidingRoot,
        noSuchSibling, noUnderVariant, windowInTree, sizingWithBitmapUnder,
        illegalStack};
    ErrorCode: --WindowFont-- TYPE = {illegalFormat};
    ErrorEntry: --BackstopNub-- TYPE = MACHINE DEPENDENT RECORD [
        globalFrame(0:0..15): GlobalFrame,
        pc(1:0..15): PC,
        time(2:0..31): System.GreenwichMeanTime,
        options(4:0..287): SELECT error(4:0..15): ErrorType FROM
            signal = > [
                signal(5:0..31): Signal,
                msg(7:0..15): SignalMsg,
                stk(8:0..223): ARRAY [0..13] OF UNSPECIFIED],
            call = > [msg(5:0..31): StringBody],
            unused = > NULL,
            interrupt = > NULL,
            addressfault = > [faultedProcess(5:0..15): PSBIndex],
            writeprotectfault = > [faultedProcess(5:0..15): PSBIndex],
            other = > [reason(5:0..15): SwapReason],
            bug = > [bugtype(5:0..15): CARDINAL],
            ENDCASE];
    ErrorHandling: --OnlineDiagnostics-- TYPE = {
        noChecking, stopOnError, loopOnError, continueOnError};
    ErrorReason: --PacketExchange-- TYPE = {
        blockTooBig, blockTooSmall, noDestinationSocket, noRouteToDestination,
        noReceiverAtDestination, insufficientResourcesAtDestination,
        rejectedByReceiver, hardwareProblem, aborted, timeout};
```

```
ErrorRecord: --NSFile-- TYPE = RECORD [
    SELECT errorType: ErrorType FROM
    access = > [problem: AccessProblem],
    attributeType = > [
        problem: ArgumentProblem,
        type: AttributeType,
        extendedType: ExtendedAttributeType ← 37777777777B],
    attributeValue = > [
        problem: ArgumentProblem,
        type: AttributeType,
        extendedType: ExtendedAttributeType ← 37777777777B],
    authentication = > [problem: NSName.AuthenticationProblem],
    connection = > [problem: ConnectionProblem],
    controlType = > [problem: ArgumentProblem, type: ControlType],
    controlValue = > [problem: ArgumentProblem, type: ControlType],
    handle = > [problem: HandleProblem],
    insertion = > [problem: InsertionProblem],
    scopeType = > [problem: ArgumentProblem, type: ScopeType],
    scopeValue = > [problem: ArgumentProblem, type: ScopeType],
    service = > [problem: ServiceProblem],
    session = > [problem: SessionProblem],
    space = > [problem: SpaceProblem],
    transfer = > [problem: TransferProblem],
    undefined = > [problem: UndefinedProblem],
    ENDCASE];
ErrorRecord: --NSPrint-- TYPE = RECORD [
    SELECT errorType: ErrorType FROM
    busy = > NULL,
    insufficientSpoolSpace = > NULL,
    invalidPrintParameters = > NULL,
    masterTooLarge = > NULL,
    mediumUnavailable = > NULL,
    serviceUnavailable = > NULL,
    spoolingDisabled = > NULL,
    spoolingQueueFull = > NULL,
    systemError = > NULL,
    tooManyClients = > NULL,
    undefinedError = > [undefined: UndefinedProblem],
    transferError = > [transfer: TransferProblem],
    connectionError = > [connection: ConnectionProblem],
    courier = > [courier: Courier.ErrorCode],
    ENDCASE];
errorServer: --ProtocolCertification-- Stage;
errorSocket: --NSConstants-- System.SocketNumber;
ErrorType: --BackstopNub-- TYPE = MACHINE DEPENDENT{
    addressfault, writeprotectfault, signal, call, unused, interrupt, other, bug};
ErrorType: --File-- TYPE = {invalidParameters, reservedType};
ErrorType: --Floppy-- TYPE = {
    badDisk, badSectors, endOfFile, fileListFull, fileNotFound, hardwareError,
    incompatibleSizes, invalidFormat, invalidPageNumber, invalidVolumeHandle,
    insufficientSpace, needsScavenging, noSuchDrive, notReady, onlyOneSide,
    onlySingleDensity, initialMicrocodeSpaceNotAvailable, stringTooShort,
    volumeNotOpen, writeInhibited, zeroSizeFile, fileListLengthTooShort,
    floppyImageInvalid, floppySpaceTooSmall};
ErrorType: --FloppyChannel-- TYPE = {invalidDrive, invalidHandle};
```

```
ErrorType: --Heap-- TYPE = {
    insufficientSpace, invalidHeap, invalidNode, invalidZone, invalidOwner,
    otherError, invalidSize, invalidParameters, maxSizeExceeded};
ErrorType: --Log-- TYPE = MACHINE DEPENDENT{
    illegalLog, invalidFile, logNoEntry, logNotOpened, tooSmallFile};
ErrorType: --NSFile-- TYPE = {
    access, attributeType, attributeValue, authentication, connection,
    controlType, controlValue, handle, insertion, scopeType, scopeValue, service,
    session, space, transfer, undefined};
ErrorType: --NSName-- TYPE = {
    ambiguousSeparators, invalidCredentials, notSuperWeak, tooManySeparators};
ErrorType: --NSPrint-- TYPE = MACHINE DEPENDENT{
    busy, insufficientSpoolSpace, invalidPrintParameters, masterTooLarge,
    mediumUnavailable, serviceUnavailable, spoolingDisabled, spoolingQueueFull,
    systemError, tooManyClients, undefinedError, connectionError, transferError,
    courier};
ErrorType: --NSSegment-- TYPE = {
    illegalForDefault, improperByteCount, invalidSegmentID, noSuchSegment,
    segmentAlreadyExists, tooManySegments};
ErrorType: --NSVolumeControl-- TYPE = {
    alreadyInitialized, alreadyOpen, badPilotLog, cannotScavengeSystemVolume,
    cannotWriteLog, incompatibleVolume, invalidVolume, logVolumeNotOpen,
    needsScavenging, noFileSystem, notMounted, notOpen, openFiles,
    pilotScavengeFailed, pilotScavengerError, unknownPilotVolume};
ErrorType: --ObjAlloc-- TYPE = {insufficientSpace, invalidParameters};
ErrorType: --PageScavenger-- TYPE = {
    driveNotAvailable, driveNotReady, invalidPageNumber, unknownDrive};
ErrorType: --PhysicalVolume-- TYPE = {
    badDisk, badSpotTableFull, containsOpenVolumes, diskReadError,
hardwareError,
    hasPilotVolume, alreadyAsserted, insufficientSpace, invalidHandle,
    nameRequired, notReady, noSuchDrive, noSuchLogicalVolume,
    physicalVolumeUnknown, writeProtected, wrongFormat, needsConversion};
ErrorType: --Scavenger-- TYPE = {
    cannotWriteLog, noSuchPage, orphanNotFound, volumeOpen,
diskHardwareError,
    diskNotReady, needsRiskyRepair, needsConversion};
ErrorType: --UserTerminalExtras-- TYPE = {
    multipleWindows, noScrollWindow, lineCountError, yQuantumError,
    xQuantumError};
ErrorType: --Volume-- TYPE = {
    nameRequired, pageCountTooSmallForVolume,
subvolumeHasTooManyBadPages,
    tooManySubvolumes};
ErrorType: --VolumeConversion-- TYPE = {
    hardwareBroken, lostLog, runPreviousScavenger, volumeVersionTooNew,
    volumeVersionTooOld};
errorUser: --ProtocolCertification-- Stage;
ESC: --Ascii-- CHARACTER = 33C;
ESCTrapTable: --PrincOps-- OpTrapTable;
etherBooteeFirstSocket: --NSConstants-- System.SocketNumber;
etherBooteeLastSocket: --NSConstants-- System.SocketNumber;
etherBootGermSocket: --NSConstants-- System.SocketNumber;
EtherDiagError: --CommOnlineDiagnostics-- ERROR [reason: EtherErrorReason];
EtherDiagError: --RemoteCommDiags-- ERROR [
    reason: CommOnlineDiagnostics.EtherErrorReason];
```

```
  EtherErrorReason: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT{
      echoUserNotThere, noMoreNets, tooManyEchoUsers};
  Ethernet: --Device-- TYPE = CARDINAL [5..15];
  ethernet: --DeviceTypes-- Device.Type;
  ethernetOne: --DeviceTypes-- Device.Type;
  EtherStatsInfo: --CommOnlineDiagnostics-- TYPE = ARRAY StatsIndices OF LONG
      CARDINAL;
  EventReporter: --CommOnlineDiagnostics-- TYPE = PROCEDURE [event: EchoEvent];
  Exception: --Real-- TYPE = MACHINE DEPENDENT{
      fixOverflow, inexactResult, invalidOperation, divisionByZero, overflow,
      underflow};
  ExceptionFlags: --Real-- TYPE = PACKED ARRAY Exception OF Flag;
  ExchangeClientType: --PacketExchange-- TYPE = MACHINE DEPENDENT{
      unspecified, timeService, clearinghouseService, teledebug(8),
      electronicMailFirstPEType(16), electronicMailLastPEType(23),
      remoteDebugFirstPEType, remoteDebugLastPEType(31),
acceptanceTestRegistration,
      performanceTestData, protocolCertification(40), voyeur, dixieDataPEType(65),
      dixieAckPEType, dixieBusyPEType, dixieErrorPEType, outsideXeroxFirst(100000B),
      outsideXeroxLast(177777B)};
  ExchangeHandle: --PacketExchange-- TYPE [2];
  ExchangeID: --PacketExchange-- TYPE = MACHINE DEPENDENT RECORD [
      a(0:0..15): WORD, b(1:0..15): WORD];
  ExchWords: --PacketExchange-- PROCEDURE [LONG UNSPECIFIED]
      RETURNS [LONG UNSPECIFIED];
  ExecProc: --Exec-- TYPE = PROCEDURE [h: Handle, clientData: LONG POINTER ← NIL]
      RETURNS [outcome: Outcome ← normal];
  Exit: --NSSessionControl-- PROCEDURE [session: NSFile.Session, id: ServiceID];
  Exp: --RealFns-- PROCEDURE [REAL] RETURNS [REAL];
  Expand: --Heap-- PROCEDURE [z: UNCOUNTED ZONE, pages: Environment.PageCount];
  Expand: --MDSStorage-- PROCEDURE [pages: CARDINAL];
  ExpandAllocation: --ObjAlloc-- PROCEDURE [
      pool: AllocPoolDesc, where: ItemIndex, count: ItemCount,
      willTakeSmaller: BOOLEAN ← FALSE] RETURNS [extendedBy: ItemCount];
  ExpandingRingAction: --ExpeditedCourier-- TYPE = {
      findMostServersInShortTime, reliablyFindAllServers};
  ExpandMDS: --Heap-- PROCEDURE [z: MDSZone, pages: Environment.PageCount];
  ExpandQ: --Expand-- TYPE [1];
  ExpandQueues: --Expand-- PROCEDURE [
      toQ: ExpandQ, fromQ: ExpandQ, all: BOOLEAN ← FALSE,
      isAborted: AbortProcType ← NIL, mask: Mask ← defaultMask];
  ExpandString: --Expand-- PROCEDURE [
      cmdLine: LONG STRING, isAborted: AbortProcType ← NIL,
      mask: Mask ← defaultMask] RETURNS [LONG STRING];
  ExpandString: --MDSStorage-- PROCEDURE [s: POINTER TO STRING, longer: CARDINAL];
  ExpandString: --NSString-- PROCEDURE [z: UNCOUNTED ZONE, s: String]
      RETURNS [Characters];
  ExpandToTokens: --Expand-- PROCEDURE [
      cmdLine: LONG STRING, proc: PROCEDURE [LONG STRING] RETURNS [BOOLEAN],
      isAborted: AbortProcType ← NIL, mask: Mask ← defaultMask];
  ExpeditedServiceHandle: --ExpeditedCourier-- TYPE [2];
  ExportExpeditedPrograms: --ExpeditedCourier-- PROCEDURE [
      services: Services, socket: System.SocketNumber]
      RETURNS [h: ExpeditedServiceHandle];
  ExportItem: --Courier-- TYPE = MACHINE DEPENDENT RECORD [
      programNumber(0:0..31): LONG CARDINAL,
```

```
        versionRange(2:0..31): VersionRange,
        serviceName(4:0..31): LONG STRING,
        exportTime(6:0..31): System.GreenwichMeanTime];
    ExportRemoteProgram: --Courier-- PROCEDURE [
        programNumber: LONG CARDINAL, versionRange: VersionRange,
        dispatcher: Dispatcher, serviceName: LONG STRING ← NIL, zone: UNCOUNTED ZONE,
        classOfService: NetworkStream.ClassOfService];
    Exports: --Courier-- TYPE = ARRAY CARDINAL OF ExportItem;
    Extended: --Real-- TYPE = RECORD [
        type: NumberType, sign: BOOLEAN, exp: INTEGER, frac: LONG CARDINAL];
    ExtendedAttributeList: --NSFile-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
        extended Attribute;
    ExtendedAttributeType: --NSFile-- TYPE = LONG CARDINAL;
    ExtendedSelections: --NSFile-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
        ExtendedAttributeType;
    ExtractHashedPassword: --Authenticator-- PROCEDURE [verifier: Verifier]
        RETURNS [hash: CARDINAL];
    FAdd: --Real-- PROCEDURE [a: REAL, b: REAL] RETURNS [REAL];
    Failed: --RetrieveDefs-- ERROR [why: FailureReason];
    failure: --RS232CCorrespondents-- RS232CEnvironment.AutoRecognitionOutcome;
    FailureReason: --NetworkStream-- TYPE = {
        timeout, noRouteToDestination, noServiceAtDestination, remoteReject,
        tooManyConnections, noAnswerOrBusy, noTranslationForDestination,
circuitInUse,
        circuitNotReady, noDialingHardware, dialerHardwareProblem};
    FailureReason: --RetrieveDefs-- TYPE = {
        communicationFailure, noSuchServer, connectionRejected, badCredentials,
        unknownFailure};
    FailureType: --FormatPilotDisk-- TYPE = {
        emptyFile, firstPageBad, flakeyPageFound, microcodeTooBig, other};
    FComp: --Real-- PROCEDURE [a: REAL, b: REAL] RETURNS [INTEGER];
    FDiv: --Real-- PROCEDURE [a: REAL, b: REAL] RETURNS [REAL];
    Feed: --RavenFace-- PROCEDURE [
        paperSource: PaperSource, paperStacking: PaperStacking];
    FeedAll: --LsepFace-- PROCEDURE [paperSource: PaperSource];
    FeedbackProc: --Exec-- PROCEDURE [h: Handle] RETURNS [proc: Format.StringProc];
    FeedExit: --LsepFace-- PROCEDURE;
    Fetch: --Cursor-- PROCEDURE [Handle];
    FetchFromType: --Cursor-- PROCEDURE [cursor: Handle, type: Defined];
    FF: --Ascii-- CHARACTER = 14C;
    Field: --AddressTranslation-- TYPE = {net, host, socket, ambiguous};
    Field: --OnlineDiagnostics-- TYPE = RECORD [
        fieldName: FloppyMessage, fieldValue: UNSPECIFIED];
    FieldDataType: --OnlineDiagnostics-- TYPE = {
        boolean, cardinal, character, hexadecimal, hexbyte, integer, octal, string};
    FieldDescriptor: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
        offset(0:0..7): BYTE, posn(0:8..11): [0..15], size(0:12..15): [1..16]];
    fiftyPercent: --Display-- Brick;
    File: --File-- TYPE = RECORD [fileID: ID, volumeID: System.VolumeID];
    FileAcquireProc: --MFile-- TYPE = PROCEDURE [
        access: Access, release: ReleaseData] RETURNS [Handle];
    filedBy: --NSAssignedTypes-- AttributeType = 24;
    filedOn: --NSAssignedTypes-- AttributeType = 25;
    FileEntry: --Scavenger-- TYPE = MACHINE DEPENDENT RECORD [
        file(0:0..31): File.ID,
        sortKey(2:0..31): LONG CARDINAL,
```

```
    numberOfProblems(4:0..15): CARDINAL,
    problems(5): ARRAY [0..0) OF Problem];
FileHandle: --Floppy-- TYPE = RECORD [volume: VolumeHandle, file: FileID];
FileID: --Floppy-- TYPE = PRIVATE MACHINE DEPENDENT RECORD [
    a(0:0..15): WORD, b(1:0..15): WORD];
fileID: --NSAssignedTypes-- AttributeType = 4;
FileInfo: --FileTransfer-- TYPE = LONG POINTER TO FileInfoObject;
FileInfoObject: --FileTransfer-- TYPE = MACHINE DEPENDENT RECORD [
    host(0:0..31): LONG STRING ← NIL,
    directory(2:0..31): LONG STRING ← NIL,
    body(4:0..31): LONG STRING ← NIL,
    version(6:0..31): LONG STRING ← NIL,
    author(8:0..31): LONG STRING ← NIL,
    create(10:0..31): Time.Packed ← System.gmtEpoch,
    read(12:0..31): Time.Packed ← System.gmtEpoch,
    write(14:0..31): Time.Packed ← System.gmtEpoch,
    size(16:0..31): LONG CARDINAL ← 0,
    type(18:0..7): FileType ← unknown,
    oldFile(18:8..8): BOOLEAN ← TRUE,
    readProtect(18:9..9): BOOLEAN ← FALSE,
    pad(18:10..15): [0..63] ← 0];
FileInWindow: --FileWindow-- PROCEDURE [sw: Window.Handle]
    RETURNS [fileName: LONG STRING, s: Stream.Handle];
FileName: --Token-- FilterProcType;
Fileserver: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    address(0:0..95): System.NetworkAddress, location(6:0..63): NSString.String];
FileserverDescribe: --CHLookup-- Courier.Description;
fileServerProtocol: --EventTypes-- Supervisor.Event;
FileServerProtocol: --Profile-- TYPE = {pup, ns};
FileserverPt: --CHLookup-- TYPE = LONG POINTER TO Fileserver;
FileServiceFileType: --FileTypes-- TYPE = CARDINAL [10000B..21777B];
fileSW: --DebugUsefulDefs-- READONLY Window.Handle;
fileSystem: --Event-- READONLY Supervisor.SubsystemHandle;
FileSystemEvents: --EventTypes-- TYPE = [200..299];
FileType: --FileTransfer-- TYPE = MACHINE DEPENDENT{
    unknown, text, binary, directory, null(255)};
FileType: --FileTypes-- TYPE = File.Type;
FileType: --NSAssignedTypes-- TYPE = NSFile.Type;
fileWindow: --Event-- READONLY Supervisor.SubsystemHandle;
FileWindowEvents: --EventTypes-- TYPE = [500..599];
fillMapLog: --PilotSwitches-- PilotDomainC = 375C;
FillRoutingTable: --Router-- PROCEDURE [maxDelay: CARDINAL ← infinity];
Filter: --MFile-- TYPE = RECORD [
    name: LONG STRING ← NIL, type: Type ← null, access: Access];
Filter: --NSFile-- TYPE = RECORD [
    var: SELECT type: FilterType FROM
        less = > [attribute: Attribute, interpretation: Interpretation ← none],
        lessOrEqual = > [
            attribute: Attribute, interpretation: Interpretation ← none],
        equal = > [attribute: Attribute, interpretation: Interpretation ← none],
        notEqual = > [attribute: Attribute, interpretation: Interpretation ← none],
        greaterOrEqual = > [
            attribute: Attribute, interpretation: Interpretation ← none],
        greater = > [attribute: Attribute, interpretation: Interpretation ← none],
        matches = > [attribute: Attribute],
        and = > [list: LONG DESCRIPTOR FOR ARRAY CARDINAL OF Filter],
```

```
        or => [list: LONG DESCRIPTOR FOR ARRAY CARDINAL OF Filter],
        not => [filter: LONG POINTER TO Filter],
        none => NULL,
        all => NULL,
        ENDCASE];
Filtered: --Token-- PROCEDURE [
    h: Handle, data: FilterState, filter: FilterProcType,
    skip: SkipMode ← whiteSpace, temporary: BOOLEAN ← TRUE]
    RETURNS [value: LONG STRING];
FilterProcType: --FormSW-- TYPE = PROCEDURE [
    sw: Window.Handle, item: ItemHandle, insert: CARDINAL, string: LONG STRING];
FilterProcType: --Token-- TYPE = PROCEDURE [c: CHARACTER, data: FilterState]
    RETURNS [inClass: BOOLEAN];
FilterState: --Token-- TYPE = LONG POINTER TO StandardFilterState;
FilterType: --NSFile-- TYPE = MACHINE DEPENDENT{
    less, lessOrEqual, equal, notEqual, greaterOrEqual, greater, and, or, not,
    none, all, matches};
Finalize: --MailParse-- PROCEDURE [h: Handle];
Find: --BTree-- PROCEDURE [tree: Tree, name: LONG STRING, value: Value]
    RETURNS [ok: BOOLEAN];
Find: --Context-- PROCEDURE [type: Type, window: Window.Handle] RETURNS [Data];
Find: --NSFile-- PROCEDURE [
    directory: Handle, scope: Scope ← [], controls: Controls ← [],
    session: Session ← nullSession] RETURNS [file: Handle];
FindAddresses: --NetworkStream-- PROCEDURE [sH: Stream.Handle]
    RETURNS [local: System.NetworkAddress, remote: System.NetworkAddress];
FindData: --ToolDriver-- FindDataProcType;
FindDataProcType: --ToolDriver-- TYPE = PROCEDURE [toolID: ToolID]
    RETURNS [LONG POINTER];
FindDestinationRelativeNetID: --Router-- PROCEDURE [System.NetworkNumber]
    RETURNS [System.NetworkNumber];
FindIndex: --FormSW-- PROCEDURE [sw: Window.Handle, item: ItemHandle]
    RETURNS [CARDINAL];
FindItem: --CmFile-- PROCEDURE [
    h: Handle, title: LONG STRING, name: LONG STRING] RETURNS [found: BOOLEAN];
FindItem: --FormSW-- PROCEDURE [sw: Window.Handle, index: CARDINAL]
    RETURNS [ItemHandle];
FindMCR: --TextSW-- Menu.MCRType;
FindMyHostID: --Router-- PROCEDURE RETURNS [System.HostNumber];
FindOrCreate: --Context-- PROCEDURE [
    type: Type, window: Window.Handle, createProc: CreateProcType] RETURNS [Data];
FindSection: --CmFile-- PROCEDURE [h: Handle, title: LONG STRING]
    RETURNS [opened: BOOLEAN];
FindString: --LexiconDefs-- PROCEDURE [LONG STRING] RETURNS [BOOLEAN];
FindUnused: --NSSegment-- PROCEDURE [
    file: NSFile.Handle, startID: ID ← defaultID, session: Session ← nullSession]
    RETURNS [ID];
finishStage: --ProtocolCertification-- Stage;
FinishWithNonPilotVolume: --PhysicalVolume-- PROCEDURE [instance: Handle];
first64K: --Environment-- Base;
First: --TIP-- PROCEDURE [results: Results] RETURNS [ResultElement];
firstCredentialEvent: --EventTypes-- CARDINAL = 400;
firstDebugEvent: --EventTypes-- CARDINAL = 0;
firstDefaultEvent: --EventTypes-- CARDINAL = 300;
firstDisplayEvent: --EventTypes-- CARDINAL = 800;
firstFileSystem: --EventTypes-- CARDINAL = 200;
```

```
  firstFileWindowEvent: --EventTypes-- CARDINAL = 500;
  FirstNearerThenSecond: --NSAddr-- PROCEDURE [
      first: System.NetworkAddress, second: System.NetworkAddress]
      RETURNS [itIs: BOOLEAN];
  firstOtherEvent: --EventTypes-- CARDINAL = 700;
  firstPageCount: --Environment-- PageCount = 0;
  firstPageCount: --File-- PageCount = 0;
  firstPageCount: --Floppy-- PageNumber = 0;
  firstPageCount: --PhysicalVolume-- PageCount = 0;
  firstPageCount: --Volume-- PageCount = 0;
  firstPageNumber: --Environment-- PageNumber = 0;
  firstPageNumber: --File-- PageNumber = 0;
  firstPageNumber: --PhysicalVolume-- PageNumber = 0;
  firstPageNumber: --Volume-- PageNumber = 0;
  firstPageOffset: --Environment-- PageOffset = 0;
  firstPosition: --NSFile-- READONLY Position;
  firstPositionRepresentation: --NSFile-- ARRAY [0..0] OF UNSPECIFIED;
  FirstQ2000PageForPilot: --FormatPilotDisk-- DiskPageNumber = 128;
  FirstSA1000PageForPilot: --FormatPilotDisk-- DiskPageNumber = 128;
  firstServicesAType: --NSAssignedTypes-- AssignedType = 10000B;
  firstServicesBType: --NSAssignedTypes-- AssignedType = 11000B;
  firstSpare: --EventTypes-- CARDINAL = 1000;
  firstStandardType: --NSAssignedTypes-- AssignedType = 0;
  firstStarType: --NSAssignedTypes-- AssignedType = 10400B;
  Firstt300PageForPilot: --FormatPilotDisk-- DiskPageNumber = 570;
  Firstt80PageForPilot: --FormatPilotDisk-- DiskPageNumber = 150;
  firstToolWindowEvent: --EventTypes-- CARDINAL = 600;
  firstVerifier: --Authenticator-- Verifier;
  firstVetoEvent: --EventTypes-- CARDINAL = 100;
  firstWS860Type: --NSAssignedTypes-- AssignedType = 12000B;
  Fix: --Real-- PROCEDURE [REAL] RETURNS [LONG INTEGER];
  FixC: --Real-- PROCEDURE [REAL] RETURNS [CARDINAL];
  FixI: --Real-- PROCEDURE [REAL] RETURNS [INTEGER];
  Flag: --FormSW-- TYPE = {
      clientOwnsItem, drawBox, hasContext, invisible, readOnly, modified};
  Flag: --Real-- TYPE = BOOLEAN ← FALSE;
  Flags: --Fonts-- TYPE = MACHINE DEPENDENT RECORD [
      pad(0:0..0): BOOLEAN, stop(0:1..1): BOOLEAN];
  Flavor: --Authenticator-- TYPE = NSName.CredentialsType;
  Float: --Real-- PROCEDURE [LONG INTEGER] RETURNS [REAL];
  Float: --Window-- PROCEDURE [
      window: Handle, temp: Handle,
      proc: PROCEDURE [window: Handle] RETURNS [place: Place, done: BOOLEAN]];
  FloppyCleanReadWriteHeads: --OnlineDiagnostics-- PROCEDURE [
      displayFields: DisplayFieldsProc, displayTable: DisplayTableProc,
      displayNumberedTable: DisplayNumberedTableProc, putMessage:
PutMessageProc,
      getConfirmation: GetConfirmationProc, getYesOrNo: GetYesOrNoProc,
      getFloppyChoice: GetFloppyChoiceProc] RETURNS [floppyReturn: FloppyReturn];
  FloppyCommandFileTest: --OnlineDiagnostics-- PROCEDURE [
      density: SingleDouble, sides: SingleDouble, sectorsPerTrack: CARDINAL [8..26],
      sectorLength: SectorLength, errorHandling: ErrorHandling,
      cmdFile: LONG STRING, displayFields: DisplayFieldsProc,
      displayTable: DisplayTableProc,
      displayNumberedTable: DisplayNumberedTableProc, putMessage:
PutMessageProc,
```

```
    getConfirmation: GetConfirmationProc, getYesOrNo: GetYesOrNoProc,
    getFloppyChoice: GetFloppyChoiceProc];
  FloppyDisplayErrorLog: --OnlineDiagnostics-- PROCEDURE [
    displayFields: DisplayFieldsProc, displayTable: DisplayTableProc,
    displayNumberedTable: DisplayNumberedTableProc, putMessage:
PutMessageProc,
    getConfirmation: GetConfirmationProc, getYesOrNo: GetYesOrNoProc,
    getFloppyChoice: GetFloppyChoiceProc];
  FloppyExerciser: --OnlineDiagnostics-- PROCEDURE [
    displayFields: DisplayFieldsProc, displayTable: DisplayTableProc,
    displayNumberedTable: DisplayNumberedTableProc, putMessage:
PutMessageProc,
    getConfirmation: GetConfirmationProc, getYesOrNo: GetYesOrNoProc,
    getFloppyChoice: GetFloppyChoiceProc];
  FloppyFormatDiskette: --OnlineDiagnostics-- PROCEDURE [
    displayFields: DisplayFieldsProc, displayTable: DisplayTableProc,
    displayNumberedTable: DisplayNumberedTableProc, putMessage:
PutMessageProc,
    getConfirmation: GetConfirmationProc, getYesOrNo: GetYesOrNoProc,
    getFloppyChoice: GetFloppyChoiceProc];
  FloppyMessage: --OnlineDiagnostics-- TYPE = {
    cFirst, cCallCSC, cCloseWn, cEnsureReady, cExit, cInsDiffCleanDisk,
    cInsertCleanDisk, cInsertDiagDisk, cInsertWriteable, cNBNotReady,
    cOtherDiskErr, cRemoveCleanDisk, cRemoveDiskette, cLast, hFirst, hBusy,
    hExpec1, hExpec2, hCRC1, hCRC2, hCRCErr, hDelSector, hDiskChng, hErrDetc,
    hGoodComp, hHead, hHeadAddr, hIllglStat, hIncrtLngth, hObser1, hObser2,
    hReadHead, hReadSector, hReadStat, hReady, hRecal, hRecalErr, hSector,
    hSectorAddr, hSectorCntErr, hSectorLgth, hSeekErr, hTimeExc, hTrack, hTrack0,
    hTrackAddr, hTwoSide, hWriteDelSector, hWritePro, hWriteSector, hLast, iFirst,
    iBadContext, iBadLabel, iBadSector, iBadTrack0, iCheckPanel, iCIERec,
    iCleanDone, iCleanProgress, iErrDet, iErrNoCRCErr, iExerWarning, iFormDone,
    iFormProgress, iFormWarning, iHardErr, iHeadDataErr, iInsertDiagDisk,
    iInsertFormDisk, iOneSided, iRunStdTest, iSoftErr, iTnx, iTwoSided,
    iUnitNotReady, iVerDataErr, iLast, tFirst, tByteCnt, tCIERH, tCIERS, tCIEVer,
    tCIEWDS, tCIEWS, tHeadDataErr, tHeadDisp, tHeadErrDisp, tSectorDisp,
    tStatDisp, tSummErrLog, tVerDataErr, tLast, yFirst, yDispSects,
    yDispExpObsData, yDoorJustOpened, yDoorOpenNow, yDoorOpenShut,
yIsItDiagDisk,
    yIsItWrProt, yStillContinue, yStillSure, yLast};
  FloppyReturn: --OnlineDiagnostics-- TYPE = {
    deviceNotReady, notDiagDiskette, floppyFailure, noErrorFound};
  FloppyStandardTest: --OnlineDiagnostics-- PROCEDURE [
    displayFields: DisplayFieldsProc, displayTable: DisplayTableProc,
    displayNumberedTable: DisplayNumberedTableProc, putMessage:
PutMessageProc,
    getConfirmation: GetConfirmationProc, getYesOrNo: GetYesOrNoProc,
    getFloppyChoice: GetFloppyChoiceProc] RETURNS [floppyReturn: FloppyReturn];
  FloppyWhatToDoNext: --OnlineDiagnostics-- TYPE = {
    continueToNextError, loopOnThisError, displayStuff, exit};
  FlowControl: --RS232C-- TYPE = RS232CEnvironment.FlowControl;
  FlowControl: --RS232CEnvironment-- TYPE = MACHINE DEPENDENT RECORD [
    type(0:0..15): MACHINE DEPENDENT{none, xOnXOff},
    xOn(1:0..15): UNSPECIFIED,
    xOff(2:0..15): UNSPECIFIED];
  Flush: --Heap-- PROCEDURE [z: UNCOUNTED ZONE];
  FlushMDS: --Heap-- PROCEDURE [z: MDSZone];
```

```
flushSymbols: --EventTypes-- Supervisor.Event;
FlushUserInput: --TIP-- PROCEDURE;
FMul: --Real-- PROCEDURE [a: REAL, b: REAL] RETURNS [REAL];
FocusTakesInput: --UserInput-- PROCEDURE RETURNS [BOOLEAN];
Font: --BandBLT-- TYPE = CARDINAL [0..127];
FontBitsPtr: --Fonts-- TYPE = LONG POINTER TO ARRAY [0..0) OF UNSPECIFIED;
FontCharPtr: --Fonts-- TYPE = LONG POINTER TO ARRAY CHARACTER OF CharEntry;
FontHeight: --WindowFont-- PROCEDURE [font: Handle ← defaultFont]
    RETURNS [NATURAL];
FontRecord: --Fonts-- TYPE = MACHINE DEPENDENT RECORD [
    fontbits(0:0..31): FontBitsPtr,
    fontwidths(2:0..31): FontWidthsPtr,
    fontchar(4:0..31): FontCharPtr,
    rgflags(6:0..31): RgflagsPtr,
    height(8:0..15): CARDINAL];
FontWidthsPtr: --Fonts-- TYPE = LONG POINTER TO PACKED ARRAY CHARACTER OF
    CARDINAL [0..255];
ForceOut: --MSegment-- PROCEDURE [segment: Handle];
forkAgingProcess: --PilotSwitchesExtraExtraExtras-- PilotSwitches.PilotDomainC =
    365C;
Format: --Floppy-- PROCEDURE [
    drive: CARDINAL, maxNumberOfFileListEntries: CARDINAL,
    labelString: LONG STRING, density: Density ← default, sides: Sides ← default];
Format: --FormatPilotDisk-- PROCEDURE [
    h: PhysicalVolume.Handle, firstPage: DiskPageNumber, count: LONG CARDINAL,
    passes: CARDINAL ← 10, retries: RetryLimit ← noRetries];
FormatBootMicrocodeArea: --FormatPilotDisk-- PROCEDURE [
    h: PhysicalVolume.Handle, passes: CARDINAL, retries: RetryLimit];
Formatter: --NSPrint-- TYPE = MACHINE DEPENDENT{available, busy, disabled};
FormattingMustBeTrackAligned: --FormatPilotDisk-- ERROR;
FormatTracks: --FloppyChannel-- PROCEDURE [
    handle: Handle, start: DiskAddress, trackCount: CARDINAL]
    RETURNS [status: Status, countDone: CARDINAL];
Frame: --Backstop-- TYPE [1];
Frame: --DebugUsefulDefs-- PROCEDURE [name: LONG STRING] RETURNS [GFHandle];
FrameDesc: --DebugUsefulDefs-- TYPE = LONG DESCRIPTOR FOR READONLY ARRAY
    CARDINAL OF GFHandle;
framelink: --PrincOps-- CARDINAL = 0;
FrameSizeIndex: --PrincOps-- TYPE = [0..30];
frameSizeMap: --PrincOps-- ARRAY FrameSizeIndex OF [0..7774B];
FrameVec: --PrincOps-- ARRAY FrameSizeIndex OF [0..7774B];
Free: --Courier-- PROCEDURE [parameters: Parameters, zone: UNCOUNTED ZONE];
Free: --MDSStorage-- PROCEDURE [p: POINTER];
Free: --Menu-- PROCEDURE [menu: Handle, freeStrings: BOOLEAN ← TRUE];
Free: --ObjAlloc-- PROCEDURE [
    pool: AllocPoolDesc, interval: Interval, validate: BOOLEAN ← TRUE];
FreeAccessList: --NSFile-- PROCEDURE [list: AccessList];
FreeAddress: --NSAddr-- PROCEDURE [address: Address]
    RETURNS [nullAddress: Address];
FreeAllItems: --FormSW-- PROCEDURE [sw: Window.Handle];
FreeAttributeList: --NSFile-- PROCEDURE [list: AttributeList];
FreeAttributes: --NSFile-- PROCEDURE [attributes: Attributes];
FreeBadPhosphorList: --Window-- PROCEDURE [window: Handle];
FreeCharacters: --NSString-- PROCEDURE [z: UNCOUNTED ZONE, c: Characters];
FreeCredentials: --Authenticator-- PROCEDURE [
    z: UNCOUNTED ZONE, credentials: LONG POINTER TO Credentials];
```

```
FreeCredentials: --NSName-- PROCEDURE [
    z: UNCOUNTED ZONE, credentials: Credentials];
FreeCursorSlot: --HeraldWindow-- PROCEDURE [slot: Slot] RETURNS [nil: Slot];
FreeEncodedParameters: --NSName-- PROCEDURE [
    z: UNCOUNTED ZONE,
    encoding: LONG DESCRIPTOR FOR ARRAY CARDINAL OF UNSPECIFIED];
FreeEnumeration: --Courier-- PROCEDURE [enum: LONG DESCRIPTOR FOR Exports];
FreeExtendedAttributes: --NSFile-- PROCEDURE [
    extendedAttributes: ExtendedAttributeList];
FreeFilename: --FileName-- PROCEDURE [LONG STRING];
FreeHintsProcType: --FormSW-- TYPE = PROCEDURE [hints: Hints];
FreeHistogram: --CommOnlineDiagnostics-- PROCEDURE [hist: Histogram];
FreeItem: --FormSW-- PROCEDURE [item: ItemHandle, z: UNCOUNTED ZONE ← NIL]
    RETURNS [ItemHandle];
FreeItem: --Menu-- PROCEDURE [ItemObject];
FreeMDSNode: --Heap-- PROCEDURE [z: MDSZone ← systemMDSZone, p: POINTER];
FreeMedia: --NSPrint-- PROCEDURE [media: LONG POINTER TO Media];
FreeName: --NSName-- PROCEDURE [z: UNCOUNTED ZONE, name: Name];
FreeNameFields: --NSName-- PROCEDURE [z: UNCOUNTED ZONE, name: Name];
FreeNode: --Heap-- PROCEDURE [z: UNCOUNTED ZONE ← systemZone, p: LONG POINTER];
FreeNode: --Zone-- PROCEDURE [zH: Handle, p: LONG POINTER] RETURNS [s: Status];
FreeNodeNil: --MDSStorage-- PROCEDURE [p: POINTER] RETURNS [nil: POINTER];
FreeNSAddrStorage: --NSAddr-- PROCEDURE [nsAddr: NSAddr];
FreePages: --MDSStorage-- PROCEDURE [base: POINTER];
FreePages: --MSegment-- PROCEDURE [base: LONG POINTER];
FreePagesNil: --MDSStorage-- PROCEDURE [base: POINTER] RETURNS [nil: POINTER];
FreePrinterProperties: --NSPrint-- PROCEDURE [
    printerProperties: LONG POINTER TO PrinterProperties];
FreePrinterStatus: --NSPrint-- PROCEDURE [
    printerStatus: LONG POINTER TO PrinterStatus];
FreeRequestStatus: --NSPrint-- PROCEDURE [
    requestStatus: LONG POINTER TO RequestStatus];
FreeRhs: --CH-- PROCEDURE [rhs: Buffer, heap: UNCOUNTED ZONE];
FreeRhsStorage: --NSAddr-- PROCEDURE [rhs: CH.Buffer];
FreeSearchPath: --MFile-- PROCEDURE [SearchPath];
FreeString: --CmFile-- PROCEDURE [LONG STRING] RETURNS [nil: LONG STRING];
FreeString: --MDSStorage-- PROCEDURE [s: STRING];
FreeString: --NSPrint-- PROCEDURE [string: LONG POINTER TO String];
FreeString: --NSString-- PROCEDURE [z: UNCOUNTED ZONE, s: String];
FreeStringHandle: --Token-- PROCEDURE [h: Handle] RETURNS [nil: Handle];
FreeStringNil: --MDSStorage-- PROCEDURE [s: STRING] RETURNS [nil: STRING];
FreeTokenString: --Exec-- PROCEDURE [s: LONG STRING] RETURNS [nil: LONG STRING];
FreeTokenString: --Token-- PROCEDURE [s: LONG STRING]
    RETURNS [nil: LONG STRING ← NIL];
FreeVFN: --FileName-- PROCEDURE [VFN];
FreeWords: --MDSStorage-- PROCEDURE [base: POINTER];
FreeWords: --MSegment-- PROCEDURE [base: LONG POINTER];
FreeWords: --NSFile-- PROCEDURE [words: Words];
FRem: --Real-- PROCEDURE [a: REAL, b: REAL] RETURNS [REAL];
FSIndex: --PrincOps-- TYPE = CARDINAL [0..255];
FSub: --Real-- PROCEDURE [a: REAL, b: REAL] RETURNS [REAL];
fullAccess: --NSFile-- Access;
galaxySocket: --NSConstants-- System.SocketNumber;
GenericProgram: --Runtime-- TYPE = LONG UNSPECIFIED;
germExtendedErrorReports: --PilotSwitches-- PilotDomainC = 360C;
```

```
Get: --RS232C-- PROCEDURE [channel: ChannelHandle, rec: PhysicalRecordHandle]
    RETURNS [CompletionHandle];
Get: --TTYPort-- PROCEDURE [channel: ChannelHandle]
    RETURNS [data: CHARACTER, status: TransferStatus];
GetAccess: --MFile-- PROCEDURE [file: Handle] RETURNS [access: Access];
GetAddress: --DebugUsefulDefs-- PROCEDURE [Handle]
    RETURNS [base: LONG POINTER, offset: Bits, there: BOOLEAN];
GetAddress: --NSAddr-- PROCEDURE RETURNS [address: Address];
GetAdjustProc: --ToolWindow-- PROCEDURE [window: Handle]
    RETURNS [AdjustProcType];
GetAttributes: --File-- PROCEDURE [file: File]
    RETURNS [type: Type, temporary: BOOLEAN];
GetAttributes: --Floppy-- PROCEDURE [
    volume: VolumeHandle, labelString: LONG STRING]
    RETURNS [
        freeSpace: PageCount, largestBlock: PageCount, fileList: FileHandle,
        rootFile: FileHandle, density: Density [single..double],
        sides: Sides [one..two], maxFileListEntries: CARDINAL];
GetAttributes: --Heap-- PROCEDURE [z: UNCOUNTED ZONE]
    RETURNS [
        heapPages: Environment.PageCount, maxSize: Environment.PageCount,
        increment: Environment.PageCount, swapUnitSize: Space.SwapUnitSize,
        ownerChecking: BOOLEAN, checking: BOOLEAN, attributes: Attributes];
GetAttributes: --LogFile-- PROCEDURE [
    file: File.File, current: Log.Index, firstPageNumber: File.PageNumber ← 1]
    RETURNS [
        time: System.GreenwichMeanTime, type: Type, level: Log.Level,
        size: CARDINAL];
GetAttributes: --NSFile-- PROCEDURE [
    file: Handle, selections: Selections, attributes: Attributes,
    session: Session ← nullSession];
GetAttributes: --NSVolumeControl-- PROCEDURE [volume: Volume.ID]
    RETURNS [
        used: LONG CARDINAL, available: LONG CARDINAL, index: IndexAttributes,
        root: NSFile.ID];
GetAttributes: --PhysicalVolume-- PROCEDURE [pvID: ID, label: LONG STRING ← NIL]
    RETURNS [instance: Handle, layout: Layout];
GetAttributes: --Volume-- PROCEDURE [volume: ID]
    RETURNS [volumeSize: PageCount, freePageCount: PageCount, readOnly:
BOOLEAN];
GetAttributes: --Zone-- PROCEDURE [zH: Handle]
    RETURNS [
        zoneBase: Base, threshold: BlockSize, checking: BOOLEAN,
        storage: LONG POINTER, length: BlockSize, next: SegmentHandle];
GetAttributesByName: --NSFile-- PROCEDURE [
    directory: Handle, path: String, selections: Selections,
    attributes: Attributes, session: Session ← nullSession];
GetAttributesChild: --NSFile-- PROCEDURE [
    directory: Handle, id: ID, selections: Selections, attributes: Attributes,
    session: Session ← nullSession];
GetAttributesMDS: --Heap-- PROCEDURE [z: MDSZone]
    RETURNS [
        heapPages: Environment.PageCount, largeNodePages: Environment.PageCount,
        maxSize: Environment.PageCount, increment: Environment.PageCount,
        swapUnitSize: Space.SwapUnitSize, threshold: NWords,
        largeNodeThreshold: NWords, ownerChecking: BOOLEAN, checking: BOOLEAN];
```

```
GetBackground: --UserTerminal-- PROCEDURE RETURNS [background: Background];
GetBase: --NSSegment-- PROCEDURE [
    pointer: LONG POINTER, session: Session ← nullSession] RETURNS [PageNumber];
GetBcdTime: --Runtime-- PROCEDURE RETURNS [System.GreenwichMeanTime];
GetBitBltTable: --UserTerminal-- PROCEDURE RETURNS [bbt: BitBlt.BBTable];
GetBitmapUnder: --Window-- PROCEDURE [window: Handle] RETURNS [LONG POINTER];
GetBlock: --LogFile-- PROCEDURE [
    file: File.File, current: Log.Index, place: LONG POINTER,
    firstPageNumber: File.PageNumber ← 1];
GetBootFiles: --Floppy-- PROCEDURE [volume: VolumeHandle]
    RETURNS [
        initialMicrocode: BootFilePointer, pilotMicrocode: BootFilePointer,
        diagnosticMicrocode: BootFilePointer, germ: BootFilePointer,
        pilotBootFile: BootFilePointer];
GetBox: --ToolWindow-- PROCEDURE [window: Handle] RETURNS [Box];
GetBox: --Window-- PROCEDURE [Handle] RETURNS [Box];
GetBuildTime: --Runtime-- PROCEDURE RETURNS [System.GreenwichMeanTime];
GetCaller: --Runtime-- PROCEDURE RETURNS [PROGRAM];
GetChar: --Exec-- GetCharProc;
GetChar: --TTY-- PROCEDURE [h: Handle] RETURNS [c: CHARACTER];
GetChar: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [CHARACTER];
GetCharProc: --Exec-- TYPE = PROCEDURE [h: Handle] RETURNS [char: CHARACTER];
GetCharProcType: --Token-- TYPE = PROCEDURE [h: Handle] RETURNS [c: CHARACTER];
GetChild: --Window-- PROCEDURE [Handle] RETURNS [Handle];
GetClearingRequired: --Window-- PROCEDURE [Handle] RETURNS [BOOLEAN];
GetClientSystemElement: --NSSessionControl-- PROCEDURE [session: NSFile.Session]
    RETURNS [NSFile.SystemElement];
GetClippedDims: --ToolWindow-- PROCEDURE [window: Handle] RETURNS
[Window.Dims];
GetConfirmationProc: --OnlineDiagnostics-- TYPE = PROCEDURE [
    msg: FloppyMessage];
GetContainingPhysicalVolume: --PhysicalVolume-- PROCEDURE [
    lvID: System.VolumeID] RETURNS [pvID: ID];
GetContext: --FloppyChannel-- PROCEDURE [handle: Handle]
    RETURNS [context: Context];
GetControls: --NSFile-- PROCEDURE [
    file: Handle, controlSelections: ControlSelections ← allControlSelections,
    session: Session ← nullSession] RETURNS [Controls];
GetConversationCredentials: --Authenticator-- PROCEDURE [
    z: UNCOUNTED ZONE, flavor: Flavor ← superWeak, userName: Name, userKey: Key,
    serverName: Name, secondsToExpiration: Seconds ← defaultExpirationTime]
    RETURNS [credentials: Credentials, conversationKey: Key];
GetCount: --Log-- PROCEDURE RETURNS [count: CARDINAL];
GetCount: --LogFile-- PROCEDURE [
    file: File.File, firstPageNumber: File.PageNumber ← 1]
    RETURNS [count: CARDINAL];
GetCreateDate: --MFile-- PROCEDURE [file: Handle] RETURNS [create: Time.Packed];
GetCredentials: --NSSessionControl-- PROCEDURE [
    name: NSString.String, password: NSString.String,
    server: NSFile.SystemElement]
    RETURNS [
        status: AuthenticationStatus, credentials: NSFile.Credentials,
        verifier: NSFile.Verifier];
```

```
GetCredentialsProc: --NSSessionControl-- TYPE = PROCEDURE [
    name: NSString.String, password: NSString.String,
    server: NSFile.SystemElement]
    RETURNS [
        status: AuthenticationStatus, credentials: NSFile.Credentials,
        verifier: NSFile.Verifier];
  GetCurrent: --Process-- PROCEDURE RETURNS [process: PROCESS];
  GetCurrentProcess: --Backstop-- PROCEDURE RETURNS [process: Process];
  GetCursorPattern: --UserTerminal-- PROCEDURE
    RETURNS [cursorPattern: CursorArray];
  GetCursorSlot: --HeraldWindow-- PROCEDURE RETURNS [slot: Slot];
  GetDecimal: --TTY-- PROCEDURE [h: Handle] RETURNS [n: INTEGER];
  GetDecimal: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [INTEGER];
  GetDefaultDomain: --Profile-- PROCEDURE [PROCEDURE [String]];
  GetDefaultOrganization: --Profile-- PROCEDURE [PROCEDURE [String]];
  GetDefaultRegistry: --Profile-- PROCEDURE [PROCEDURE [String]];
  GetDefaultSession: --NSFile-- PROCEDURE RETURNS [Session];
  GetDefaultSocketNumber: --ExpeditedCourier-- PROCEDURE
    RETURNS [System.SocketNumber];
  GetDefaultWindow: --UserInput-- PROCEDURE RETURNS [Window.Handle];
  GetDelayToNet: --Router-- PROCEDURE [net: System.NetworkNumber]
    RETURNS [delay: CARDINAL];
  GetDesiredProperties: --FileTransfer-- PROCEDURE [conn: Connection]
    RETURNS [props: DesiredProperties];
  GetDeviceAttributes: --FloppyChannel-- PROCEDURE [handle: Handle]
    RETURNS [attributes: Attributes];
  GetDialerCount: --Dialup-- PROCEDURE RETURNS [numberOfDialers: CARDINAL];
  GetDirectoryName: --MFile-- PROCEDURE [file: Handle, name: LONG STRING];
  GetDisplayProc: --Window-- PROCEDURE [Handle] RETURNS [PROCEDURE [Handle]];
  GetDriveSize: --OthelloOps-- PROCEDURE [h: PhysicalVolume.Handle]
    RETURNS [nPages: LONG CARDINAL];
  GetEcho: --TTY-- PROCEDURE [h: Handle] RETURNS [old: EchoClass];
  GetEcho: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [old: TTY.EchoClass];
  GetEchoCounters: --CommOnlineDiagnostics-- PROCEDURE [
    host: System.NetworkAddress ← System.nullNetworkAddress]
    RETURNS [
        packets: LONG CARDINAL, bytes: LONG CARDINAL,
        time: System.GreenwichMeanTime];
  GetEchoCounters: --RemoteCommDiags-- PROCEDURE [host: System.NetworkAddress]
    RETURNS [
        packets: LONG CARDINAL, bytes: LONG CARDINAL,
        time: System.GreenwichMeanTime];
  GetEchoResults: --CommOnlineDiagnostics-- PROCEDURE [
    stopIt: BOOLEAN, host: System.NetworkAddress ← System.nullNetworkAddress]
    RETURNS [totalsSinceStart: EchoResults, hist: Histogram];
  GetEchoResults: --RemoteCommDiags-- PROCEDURE [
    host: System.NetworkAddress, echoUser: CommOnlineDiagnostics.EchoUserHandle,
    stopIt: BOOLEAN]
    RETURNS [
        totalsSinceStart: CommOnlineDiagnostics.EchoResults,
        hist: CommOnlineDiagnostics.Histogram];
  GetEditedString: --TTY-- PROCEDURE [
    h: Handle, s: LONG STRING,
    t: PROCEDURE [c: CHARACTER] RETURNS [status: CharStatus]]
    RETURNS [c: CHARACTER];
```

```
  GetEditedString: --TTYSW-- PROCEDURE [
      sw: Window.Handle, s: LONG STRING,
      t: PROCEDURE [CHARACTER] RETURNS [TTY.CharStatus]] RETURNS [CHARACTER];
  GetError: --Backstop-- PROCEDURE RETURNS [BackstopNub.ErrorType];
  GetEthernetStats: --CommOnlineDiagnostics-- PROCEDURE [
      physicalOrder: CARDINAL ← 1,
      host: System.NetworkAddress ← System.nullNetworkAddress]
      RETURNS [info: EtherStatsInfo, time: System.GreenwichMeanTime];
  GetEthernetStats: --RemoteCommDiags-- PROCEDURE [
      host: System.NetworkAddress, physicalOrder: CARDINAL ← 1]
      RETURNS [
          info: CommOnlineDiagnostics.EtherStatsResult,
          time: System.GreenwichMeanTime];
  GetExpirationDate: --OthelloOps-- PROCEDURE [
      file: File.File, firstPage: File.PageNumber]
      RETURNS [GetExpirationDateSuccess, System.GreenwichMeanTime];
  GetExpirationDateSuccess: --OthelloOps-- TYPE = SetDebuggerSuccess
      [success..other];
  GetFaultedProcess: --Backstop-- PROCEDURE RETURNS [process: Process];
  GetFieldBody: --MailParse-- PROCEDURE [
      h: Handle, string: LONG STRING, suppressWhiteSpace: BOOLEAN ← FALSE];
  GetFieldName: --MailParse-- PROCEDURE [h: Handle, field: LONG STRING]
      RETURNS [found: BOOLEAN];
  GetFile: --FileSW-- PROCEDURE [sw: Window.Handle]
      RETURNS [name: LONG STRING, s: Stream.Handle];
  GetFile: --MSegment-- PROCEDURE [segment: Handle] RETURNS [MFile.Handle];
  GetFile: --MStream-- PROCEDURE [stream: Handle] RETURNS [file: MFile.Handle];
  GetFileAttributes: --Floppy-- PROCEDURE [file: FileHandle]
      RETURNS [size: PageCount, type: File.Type];
  GetFileBase: --MSegment-- PROCEDURE [segment: Handle] RETURNS
[File.PageNumber];
  GetFilePages: --MSegment-- PROCEDURE [segment: Handle] RETURNS
[File.PageCount];
  GetFloppyChoiceProc: --OnlineDiagnostics-- TYPE = PROCEDURE
      RETURNS [FloppyWhatToDoNext];
  GetFont: --Menu-- PROCEDURE RETURNS [font: WindowFont.Handle];
  GetFullName: --MFile-- PROCEDURE [file: Handle, name: LONG STRING];
  GetGravity: --ToolWindow-- PROCEDURE [window: Handle]
      RETURNS [gravity: Window.Gravity];
  GetGroupPhrase: --MailParse-- PROCEDURE [h: Handle, phrase: LONG STRING];
  GetHandle: --FloppyChannel-- PROCEDURE [drive: Drive] RETURNS [handle: Handle];
  GetHandle: --PhysicalVolume-- PROCEDURE [index: CARDINAL] RETURNS [Handle];
  GetHints: --PhysicalVolume-- PROCEDURE [
      instance: Handle, label: LONG STRING ← NIL]
      RETURNS [pvID: ID, volumeType: VolumeType];
  GetID: --TTY-- PROCEDURE [h: Handle, s: LONG STRING];
  GetID: --TTYSW-- PROCEDURE [sw: Window.Handle, s: LONG STRING];
  GetImageAttributes: --Floppy-- PROCEDURE [
      imageFile: File.File, firstImagePage: File.PageNumber,
      name: LONG STRING ← NIL]
      RETURNS [
          maxNumberOfFiles: CARDINAL, currentNumberOfFiles: CARDINAL,
          density: Density [single..double], sides: Sides [one..two]];
  GetInactiveName: --ToolWindow-- PROCEDURE [window: Handle]
      RETURNS [name: LONG STRING];
  GetIndex: --Log-- PROCEDURE RETURNS [index: Index];
```

```
  GetIndex: --MemoryStream-- PROCEDURE [sH: Stream.Handle]
      RETURNS [position: Stream.Position];
  GetInfo: --BTree-- PROCEDURE [tree: Tree]
      RETURNS [valueSize: ValueSize, file: MFile.Handle, usage: Space.Usage];
  GetInfo: --Cursor-- PROCEDURE RETURNS [Info];
  GetInfo: --FileWindow-- PROCEDURE
      RETURNS [
          ext: LONG STRING, fileMenu: Menu.Handle, sourceMenu: Menu.Handle,
          minimumWindows: CARDINAL];
  GetInputFocus: --UserInput-- PROCEDURE RETURNS [Window.Handle];
  GetLabelString: --Volume-- PROCEDURE [volume: ID, s: LONG STRING];
  GetLength: --MFile-- PROCEDURE [file: Handle] RETURNS [ByteCount];
  GetLength: --MStream-- PROCEDURE [stream: Handle]
      RETURNS [fileLength: MFile.ByteCount];
  GetLibrarian: --Profile-- PROCEDURE [PROCEDURE [String]];
  GetLibrarianNames: --Profile-- PROCEDURE [
      PROCEDURE [prefix: String, suffix: String]];
  GetLimitProc: --ToolWindow-- PROCEDURE [window: Handle] RETURNS
[LimitProcType];
  GetLine: --TTY-- PROCEDURE [h: Handle, s: LONG STRING];
  GetLine: --TTYSW-- PROCEDURE [sw: Window.Handle, s: LONG STRING];
  GetLog: --Scavenger-- PROCEDURE [volume: Volume.ID]
      RETURNS [logFile: File.File];
  GetLog: --VolumeConversion-- PROCEDURE [volume: Volume.ID]
      RETURNS [logFile: File.File];
  GetLogEntry: --BackstopNub-- PROCEDURE [
      log: File.File, current: Log.Index, place: Handle,
      firstPageNumber: File.PageNumber ← 0];
  GetLongDecimal: --TTY-- PROCEDURE [h: Handle] RETURNS [n: LONG INTEGER];
  GetLongDecimal: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [LONG
INTEGER];
  GetLongNumber: --TTY-- PROCEDURE [
      h: Handle, default: LONG UNSPECIFIED, radix: CARDINAL,
      showDefault: BOOLEAN ← TRUE] RETURNS [n: LONG UNSPECIFIED];
  GetLongNumber: --TTYSW-- PROCEDURE [
      sw: Window.Handle, default: LONG UNSPECIFIED, radix: CARDINAL,
      showDefault: BOOLEAN ← TRUE] RETURNS [LONG UNSPECIFIED];
  GetLongOctal: --TTY-- PROCEDURE [h: Handle] RETURNS [n: LONG UNSPECIFIED];
  GetLongOctal: --TTYSW-- PROCEDURE [sw: Window.Handle]
      RETURNS [LONG UNSPECIFIED];
  GetLost: --Log-- PROCEDURE RETURNS [lost: CARDINAL];
  GetLost: --LogFile-- PROCEDURE [
      file: File.File, firstPageNumber: File.PageNumber ← 1]
      RETURNS [count: CARDINAL];
  GetName: --ToolWindow-- PROCEDURE [window: Handle] RETURNS [name: LONG
STRING];
  GetNameandPassword: --Exec-- PROCEDURE [
      h: Handle, name: LONG STRING, password: LONG STRING,
      prompt: LONG STRING ← NIL];
  GetNameStripe: --ToolWindow-- PROCEDURE [window: Handle] RETURNS [OnOff];
  GetNetworkID: --Router-- PROCEDURE [
      physicalOrder: CARDINAL, medium: PhysicalMedium]
      RETURNS [System.NetworkNumber];
  GetNext: --BackstopNub-- PROCEDURE [
      log: File.File, current: Log.Index, firstPageNumber: File.PageNumber ← 0]
      RETURNS [next: Log.Index];
```

```
GetNext: --BTree-- PROCEDURE [
    tree: Tree, name: LONG STRING, nextName: LONG STRING, value: Value,
    mask: LONG STRING ← NIL];
GetNext: --LogFile-- PROCEDURE [
    file: File.File, current: Log.Index, firstPageNumber: File.PageNumber ← 1]
    RETURNS [next: Log.Index];
GetNext: --NSSegment-- PROCEDURE [
    file: NSFile.Handle, currentSegment: ID, session: Session ← nullSession]
    RETURNS [ID];
GetNext: --PhysicalVolume-- PROCEDURE [pvID: ID] RETURNS [ID];
GetNext: --Volume-- PROCEDURE [
    volume: ID, includeWhichVolumes: TypeSet ← onlyEnumerateCurrentType]
    RETURNS [nextVolume: ID];
GetNextBadPage: --PhysicalVolume-- PROCEDURE [
    pvID: ID, thisBadPageNumber: PageNumber]
    RETURNS [nextBadPageNumber: PageNumber];
GetNextBadSector: --Floppy-- PROCEDURE [
    volume: VolumeHandle, oldIndex: CARDINAL]
    RETURNS [newIndex: CARDINAL, file: FileHandle, page: PageNumber];
GetNextDrive: --FloppyChannel-- PROCEDURE [lastDrive: Drive]
    RETURNS [nextDrive: Drive];
GetNextDrive: --PhysicalVolume-- PROCEDURE [index: CARDINAL]
    RETURNS [nextIndex: CARDINAL];
GetNextFile: --Floppy-- PROCEDURE [previousFile: FileHandle]
    RETURNS [nextFile: FileHandle];
GetNextFrame: --Backstop-- PROCEDURE [process: Process, frame: Frame]
    RETURNS [next: Frame];
GetNextHandleForReading: --MFile-- PROCEDURE [
    filter: LONG STRING, name: LONG STRING, release: ReleaseData,
    lastState: EnumerateState, stopNow: BOOLEAN ← FALSE]
    RETURNS [file: Handle, state: EnumerateState];
GetNextLine: --RS232C-- PROCEDURE [lineNumber: CARDINAL]
    RETURNS [nextLineNumber: CARDINAL];
GetNextLogicalVolume: --PhysicalVolume-- PROCEDURE [
    pvID: ID, lvID: System.VolumeID] RETURNS [System.VolumeID];
GetNextProcess: --Backstop-- PROCEDURE [process: Process]
    RETURNS [next: Process];
GetNextRootFile: --Volume-- PROCEDURE [
    lastType: File.Type, volume: ID ← systemID]
    RETURNS [file: File.File, type: File.Type];
GetNextSubVolume: --OthelloOps-- PROCEDURE [
    pvID: PhysicalVolume.ID, thisSv: SubVolume] RETURNS [nextSV: SubVolume];
GetNextVerifier: --Authenticator-- PROCEDURE [
    credentials: Credentials, conversationKey: Key,
    lastVerifier: Verifier ← firstVerifier] RETURNS [nextV: Verifier];
GetNotifier: --Scrollbar-- PROCEDURE [window: Window.Handle, type: Type]
    RETURNS [ScrollProcType];
GetNotifyProc: --TIP-- PROCEDURE [window: Window.Handle] RETURNS [NotifyProc];
GetNotifyProcFromTable: --TIP-- PROCEDURE [table: Table] RETURNS [NotifyProc];
GetNSAddr: --NSAddr-- PROCEDURE [nsAddr: NSAddr];
GetNumber: --TTY-- PROCEDURE [
    h: Handle, default: UNSPECIFIED, radix: CARDINAL, showDefault: BOOLEAN ←
TRUE]
    RETURNS [n: UNSPECIFIED];
```

```
  GetNumber: --TTYSW-- PROCEDURE [
      sw: Window.Handle, default: UNSPECIFIED, radix: CARDINAL,
      showDefault: BOOLEAN ← TRUE] RETURNS [UNSPECIFIED];
  GetOctal: --TTY-- PROCEDURE [h: Handle] RETURNS [n: UNSPECIFIED];
  GetOctal: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [UNSPECIFIED];
  GetPages: --MSegment-- PROCEDURE [npages: CARDINAL]
      RETURNS [base: LONG POINTER];
  GetParent: --Window-- PROCEDURE [Handle] RETURNS [Handle];
  GetPassword: --TTY-- PROCEDURE [h: Handle, s: LONG STRING];
  GetPassword: --TTYSW-- PROCEDURE [sw: Window.Handle, s: LONG STRING];
  GetPhysicalVolumeBootFile: --OthelloOps-- PROCEDURE [
      pvID: PhysicalVolume.ID, type: BootFileType]
      RETURNS [file: File.File, firstPage: File.PageNumber];
  GetPlace: --TIP-- PROCEDURE [window: Window.Handle] RETURNS [Window.Place];
  GetPName: --Atom-- PROCEDURE [atom: ATOM] RETURNS [pName: LONG STRING];
  GetPrinterProperties: --NSPrint-- PROCEDURE [systemElement: SystemElement]
      RETURNS [properties: PrinterProperties];
  GetPrinterStatus: --NSPrint-- PROCEDURE [systemElement: SystemElement]
      RETURNS [status: PrinterStatus];
  GetPrintRequestStatus: --NSPrint-- PROCEDURE [
      printRequestID: RequestID, systemElement: SystemElement]
      RETURNS [status: RequestStatus];
  GetPriority: --Process-- PROCEDURE RETURNS [priority: Priority];
  GetProcs: --FileTransfer-- PROCEDURE [conn: Connection]
      RETURNS [
          clientData: LONG POINTER, messages: MessageProc, login: ClientProc,
          noteProgress: ClientProc, checkAbort: CheckAbortProc];
  GetProcType: --GSort-- TYPE = PROCEDURE [p: LONG POINTER] RETURNS [CARDINAL];
  GetProperties: --CH-- PROCEDURE [
      cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Pattern,
      getArray: PropertiesAllocator, distingName: Name]
      RETURNS [rc: ReturnCode, properties: Properties];
  GetProperties: --MFile-- PROCEDURE [file: Handle, name: LONG STRING ← NIL]
      RETURNS [
          create: Time.Packed, write: Time.Packed, read: Time.Packed,
          length: ByteCount, type: Type, deleteProtected: BOOLEAN,
          writeProtected: BOOLEAN, readProtected: BOOLEAN];
  GetProperty: --MFile-- PROCEDURE [
      file: Handle, property: Property, block: Environment.Block]
      RETURNS [length: CARDINAL];
  GetProtection: --MFile-- PROCEDURE [file: Handle]
      RETURNS [
          deleteProtected: BOOLEAN, writeProtected: BOOLEAN, readProtected:
BOOLEAN];
  GetRandomKey: --Authenticator-- PROCEDURE RETURNS [key: Key];
  GetReference: --NSFile-- PROCEDURE [
      file: Handle, reference: Reference, session: Session ← nullSession];
  GetReleaseData: --MFile-- PROCEDURE [file: Handle]
      RETURNS [release: ReleaseData];
  GetReleaseData: --MSegment-- PROCEDURE [segment: Handle] RETURNS
[ReleaseData];
  GetReleaseData: --MStream-- PROCEDURE [stream: Handle]
      RETURNS [release. ReleaseData];
  GetRemoteName: --FileName-- PROCEDURE [
      file: MFile.Handle, remoteName: LONG STRING];
```

```
GetRestart: --LogFile-- PROCEDURE [
    file: File.File, firstPageNumber: File.PageNumber ← 1]
    RETURNS [restart: Restart];
GetRootNode: --Zone-- PROCEDURE [zH: Handle]
    RETURNS [node: Base RELATIVE POINTER];
GetRouteAddrPhrase: --MailParse-- PROCEDURE [h: Handle, name: LONG STRING];
GetRouterFunction: --Router-- PROCEDURE RETURNS [RoutersFunction];
GetRS232CResults: --CommOnlineDiagnostics-- PROCEDURE [
    stopIt: BOOLEAN, host: System.NetworkAddress ←
System.nullNetworkAddress]
    RETURNS [counters: CountType];
GetSearchPath: --MFile-- PROCEDURE RETURNS [SearchPath];
GetSegmentAttributes: --Zone-- PROCEDURE [zH: Handle, sH: SegmentHandle]
    RETURNS [storage: LONG POINTER, length: BlockSize, next: SegmentHandle];
GetSelection: --FormSW-- PROCEDURE [Window.Handle]
    RETURNS [index: CARDINAL, first: CARDINAL, last: CARDINAL];
GetServerType: --FileTransfer-- PROCEDURE [conn: Connection, host: LONG STRING]
    RETURNS [ServerType];
GetServiceData: --NSSessionControl-- PROCEDURE [
    session: NSFile.Session, id: ServiceID] RETURNS [ServiceData];
GetSessionAttributes: --NSSessionControl-- PROCEDURE [session: NSFile.Session]
    RETURNS [SessionAttributes];
GetSessionRestrictions: --NSSessionControl-- PROCEDURE
    RETURNS [SessionRestrictions];
GetSeverity: --MsgSW-- PROCEDURE [sw: Window.Handle]
    RETURNS [severity: Severity];
GetSibling: --Window-- PROCEDURE [Handle] RETURNS [Handle];
GetSize: --BackstopNub-- PROCEDURE [
    log: File.File, current: Log.Index, firstPageNumber: File.PageNumber ← 0]
    RETURNS [size: CARDINAL];
GetSize: --DebugUsefulDefs-- PROCEDURE [Handle]
    RETURNS [words: CARDINAL, bits: Bits];
GetSize: --File-- PROCEDURE [file: File] RETURNS [size: PageCount];
GetSizeInBytes: --NSSegment-- PROCEDURE [
    file: NSFile.Handle, segment: ID ← defaultID, session: Session ← nullSession]
    RETURNS [ByteCount];
GetSizeInPages: --NSSegment-- PROCEDURE [
    file: NSFile.Handle, segment: ID ← defaultID, session: Session ← nullSession]
    RETURNS [PageCount];
GetSnapShotFromFile: --LibrarianUtility-- PROCEDURE [fileName: LONG STRING]
    RETURNS [Librarian.SnapShotHandle];
GetState: --Log-- PROCEDURE RETURNS [state: State];
GetState: --ToolWindow-- PROCEDURE [window: Handle] RETURNS [state: State];
GetState: --UserTerminal-- PROCEDURE RETURNS [state: State];
GetStatus: --LsepFace-- PROCEDURE RETURNS [status: PrinterStatus];
GetStatus: --RavenFace-- PROCEDURE RETURNS [status: PrinterStatus];
GetStatus: --RS232C-- PROCEDURE [channel: ChannelHandle]
    RETURNS [stat: DeviceStatus];
GetStatus: --TTYPort-- PROCEDURE [channel: ChannelHandle]
    RETURNS [stat: DeviceStatus];
GetStatus: --Volume-- PROCEDURE [volume: ID] RETURNS [Status];
GetStickyFlags: --Real-- PROCEDURE RETURNS [ExceptionFlags];
GetStreamInfo: --FileTransfer-- PROCEDURE [remoteStream: Stream.Handle]
    RETURNS [FileInfo];
GetStreamName: --FileTransfer PROCEDURE [remoteStream: Stream.Handle]
    RETURNS [file: LONG STRING];
```

```
  GetString: --LogFile-- PROCEDURE [
      file: File.File, current: Log.Index, place: LONG STRING,
      firstPageNumber: File.PageNumber ← 1];
  GetString: --TTY-- PROCEDURE [
      h: Handle, s: LONG STRING,
      t: PROCEDURE [c: CHARACTER] RETURNS [status: CharStatus]];
  GetString: --TTYSW-- PROCEDURE [
      sw: Window.Handle, s: LONG STRING,
      t: PROCEDURE [CHARACTER] RETURNS [TTY.CharStatus]];
  GetSwitches: --OthelloOps-- PROCEDURE [
      file: File.File, firstPage: File.PageNumber]
      RETURNS [SetGetSwitchesSuccess, System.Switches];
  GetTable: --TIP-- PROCEDURE [window: Window.Handle] RETURNS [Table];
  GetTableBase: --Runtime-- PROCEDURE [frame: PROGRAM] RETURNS [LONG POINTER];
  GetTabs: --AsciiSink-- PROCEDURE [sink: TextSink.Handle] RETURNS [TabStops];
  GetTimeFromTimeServer: --OthelloOps-- PROCEDURE
      RETURNS [
          serverTime: System.GreenwichMeanTime,
          serverLTPs: System.LocalTimeParameters];
  GetTimes: --MFile-- PROCEDURE [file: Handle]
      RETURNS [create: Time.Packed, write: Time.Packed, read: Time.Packed];
  GetTinyName: --ToolWindow-- PROCEDURE [window: Handle]
      RETURNS [name: LONG STRING, name2: LONG STRING];
  GetTinyPlace: --ToolWindow-- PROCEDURE [window: Handle] RETURNS [place:
Place];
  GetToken: --Exec-- PROCEDURE [h: Handle]
      RETURNS [token: LONG STRING, switches: LONG STRING];
  GetToolsPropertyArray: --LibrarianUtility-- PROCEDURE RETURNS [PropertyArray];
  GetTransitionProc: --ToolWindow-- PROCEDURE [window: Handle]
      RETURNS [TransitionProcType];
  GetTTY: --Exec-- PROCEDURE [h: Handle] RETURNS [tty: TTY.Handle];
  GetTTYHandle: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [tty:
TTY.Handle];
  GetType: --FileExtras-- PROCEDURE [file: File.File] RETURNS [type: File.Type];
  GetType: --MFile-- PROCEDURE [file: Handle] RETURNS [type: Type];
  GetType: --NSFile-- PROCEDURE [file: Handle, session: Session ← nullSession]
      RETURNS [Type];
  GetType: --Volume-- PROCEDURE [volume: ID] RETURNS [type: Type];
  GetTypeIn: --FormSW-- PROCEDURE [Window.Handle]
      RETURNS [index: CARDINAL, position: CARDINAL];
  GetUniqueConnectionID: --NetworkStream-- PROCEDURE RETURNS [iD:
ConnectionID];
  GetUpdate: --Log-- PROCEDURE RETURNS [time: System.GreenwichMeanTime];
  GetUser: --Profile-- PROCEDURE [
      proc: PROCEDURE [name: String, password: String],
      qualification: Qualification ← none];
  GetVolume: --MFile-- PROCEDURE [file: Handle] RETURNS [Volume.ID];
  GetVolumeBootFile: --OthelloOps-- PROCEDURE [
      lvID: Volume.ID, type: BootFileType]
      RETURNS [file: File.File, firstPage: File.PageNumber];
  GetWords: --MSegment-- PROCEDURE [nwords: CARDINAL]
      RETURNS [base: LONG POINTER];
  GetYesOrNoProc: --OnlineDiagnostics-- TYPE = PROCEDURE [msg: FloppyMessage]
      RETURNS [YesOrNo];
  GFHandle: --DebugUsefulDefs-- TYPE = PrincOps.GlobalFrameHandle;
  globalbase: --PrincOps-- CARDINAL = 0;
```

```
GlobalCodebase: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
    code => [codebase(0:0..31): PrefixHandle],
    offset => [offset(0:0..15): CARDINAL, highHalf(1:0..15): CARDINAL],
    either => [
        fill(0:0..14): CARDINAL [0..77777B],
        out(0:15..15): BOOLEAN,
        highByte(1:0..7): BYTE,
        otherByte(1:8..15): BYTE],
    ENDCASE];
GlobalFrame: --BackstopNub-- TYPE [1];
GlobalFrame: --Runtime-- PROCEDURE [link: ControlLink] RETURNS [PROGRAM];
GlobalFrameBase: --PrincOps-- TYPE = POINTER TO GlobalOverhead;
GlobalFrameHandle: --PrincOps-- TYPE = POINTER TO GlobalVariables;
globalOffset: --PrincOps-- CARDINAL = 2;
GlobalOverhead: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    extra(0:0..15): WORD,
    word(1:0..15): GlobalWord,
    codebase(2:0..31): GlobalCodebase,
    global(4): GlobalVariables];
globalTable: --TIP-- READONLY ARRAY GlobalTable OF Table;
GlobalTable: --TIP-- TYPE = {
    root, formSW, textSW, fileWindow, ttySW, executive, spare1, spare2};
GlobalVariables: --PrincOps-- TYPE = ARRAY CARDINAL [0..0) OF UNSPECIFIED;
GlobalWord: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    index(0:0..8): CARDINAL [0..511],
    started(0:9..9): BOOLEAN,
    copy(0:10..10): BOOLEAN,
    copied(0:11..11): BOOLEAN,
    alloced(0:12..12): BOOLEAN,
    shared(0:13..13): BOOLEAN,
    trapxfers(0:14..14): BOOLEAN,
    codelinks(0:15..15): BOOLEAN];
globalWordOffset: --PrincOps-- CARDINAL = 3;
Gravity: --Window-- TYPE = {nil, nw, n, ne, e, se, s, sw, w, c, xxx};
Gray: --Display-- PROCEDURE [
    window: Handle, box: Window.Box, gray: Brick ← fiftyPercent,
    dstFunc: DstFunc ← null];
GrayParm: --BitBlt-- TYPE = MACHINE DEPENDENT RECORD [
    reserved(0:0..3): [0..15] ← 0,
    yOffset(0:4..7): [0..15],
    widthMinusOne(0:8..11): [0..15],
    heightMinusOne(0:12..15): [0..15]];
GWS: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    address(0:0..95): System.NetworkAddress,
    location(6:0..63): NSString.String,
    mailClerk(10:0..31): NSName.Name];
GWSDescribe: --CHLookup-- Courier.Description;
GWSPt: --CHLookup-- TYPE = LONG POINTER TO GWS;
Handle: --BackstopNub-- TYPE = LONG POINTER TO ErrorEntry;
Handle: --BlockSource-- TYPE = TextSource.Handle;
Handle: --CmFile-- TYPE = Token.Handle;
Handle: --Courier-- TYPE = LONG POINTER TO READONLY Object;
Handle: --Cursor-- TYPE = POINTER TO Object;
Handle: --DebugUsefulDefs-- TYPE = LONG POINTER TO Object;
Handle: --Display-- TYPE = Window.Handle;
```

```
  Handle: --Event-- TYPE = LONG POINTER TO Object;
  Handle: --Exec-- TYPE = LONG POINTER TO Object;
  Handle: --FloppyChannel-- TYPE [2];
  Handle: --Heap-- TYPE = UNCOUNTED ZONE;
  Handle: --MailParse-- TYPE = LONG POINTER TO Object;
  Handle: --Menu-- TYPE = LONG POINTER TO Object;
  Handle: --MFile-- TYPE = LONG POINTER TO Object;
  Handle: --MLoader-- TYPE = LONG POINTER TO Object;
  Handle: --MSegment-- TYPE = LONG POINTER TO Object;
  Handle: --MStream-- TYPE = Stream.Handle;
  Handle: --NSDataStream-- TYPE = RECORD [Stream.Handle];
  Handle: --NSFile-- TYPE [2];
  Handle: --PhysicalVolume-- TYPE [3];
  Handle: --RetrieveDefs-- TYPE [2];
  Handle: --SendDefs-- TYPE [2];
  Handle: --Token-- TYPE = LONG POINTER TO Object;
  Handle: --ToolWindow-- TYPE = Window.Handle;
  Handle: --TTY-- TYPE [2];
  Handle: --Window-- TYPE = LONG POINTER TO Object;
  Handle: --WindowFont-- TYPE = LONG POINTER TO Object;
  Handle: --Zone-- TYPE [2];
  HandleFromProgram: --MLoader-- PROCEDURE [PROGRAM] RETURNS [Handle];
  HandleProblem: --NSFile-- TYPE = MACHINE DEPENDENT{
      invalid, nullDisallowed, directoryRequired, obsolete};
  hang: --PilotSwitches-- PilotDomainA = 46C;
  hasBorder: --UserTerminal-- READONLY BOOLEAN;
  HashedPassword: --NSName-- TYPE = CARDINAL;
  HashPassword: --NSName-- PROCEDURE [password: String] RETURNS
[HashedPassword];
  HashSimplePassword: --Authenticator-- PROCEDURE [password: NSString.String]
      RETURNS [hash: CARDINAL];
  HasScrollbar: --Scrollbar-- PROCEDURE [window: Window.Handle, type: Type]
      RETURNS [BOOLEAN];
  Header: --ExpeditedCourier-- TYPE = MACHINE DEPENDENT RECORD [
      protRange(0:0..31): CourierInternal.ProtocolRange ← [protocol3, protocol3],
      body(2:0..95): CourierInternal.Protocol3Body];
  Header: --NSVolumeControl-- TYPE = MACHINE DEPENDENT RECORD [
      volume(0:0..79): Volume.ID,
      date(5:0..31): System.GreenwichMeanTime,
      incomplete(7:0..14): BOOLEAN,
      repaired(7:15..15): BOOLEAN,
      numberOfFiles(8:0..31): LONG CARDINAL];
  Header: --Scavenger-- TYPE = MACHINE DEPENDENT RECORD [
      seal(0:0..15): CARDINAL ← LogSeal,
      version(1:0..15): CARDINAL ← currentLogVersion,
      volume(2:0..79): Volume.ID,
      date(7:0..31): System.GreenwichMeanTime,
      repairMode(9:0..1): RepairType,
      incomplete(9:2..2): BOOLEAN,
      repaired(9:3..3): BOOLEAN,
      bootFilesDeleted(9:4..9): BootFileArray,
      pad(9:10..15): [0..0] ← 0,
      numberOfFiles(10:0..31): LONG CARDINAL];
  HeaderPointer: --NSVolumeControl-- TYPE = LONG POINTER TO Header;
  heapChecking: --PilotSwitches-- PilotDomainB = 136C;
  heapOwnerChecking: --PilotSwitches-- PilotDomainA = 66C;
```

```
  heapParamsFromClient: --PilotSwitches-- PilotDomainC = 374C;
  Hide: --ToolWindow-- PROCEDURE [window: Handle];
  hierachicalLevels: --CH-- CARDINAL = 3;
  hierarchicalLevels: --NSName-- CARDINAL = 3;
  HighByte: --Inline-- PROCEDURE [u: UNSPECIFIED] RETURNS [UNSPECIFIED];
  HighestVersion: --FileTransfer-- PROCEDURE [
      conn: Connection, remote: FileName.VFN] RETURNS [exists: BOOLEAN];
  highestVersion: --NSFile-- CARDINAL = 177777B;
  HighHalf: --Inline-- PROCEDURE [u: LONG UNSPECIFIED] RETURNS [UNSPECIFIED];
  Hints: --FormSW-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF LONG STRING;
  Histogram: --CommOnlineDiagnostics-- TYPE = LONG DESCRIPTOR FOR ARRAY
CARDINAL
      OF Detail;
  Hop: --ExpeditedCourier-- TYPE = [0..15];
  HostNumber: --Format-- PROCEDURE [
      proc: StringProc, hostNumber: System.HostNumber, format: NetFormat,
      clientData: LONG POINTER ← NIL];
  HostNumber: --Put-- PROCEDURE [
      h: Window.Handle ← NIL, host: System.HostNumber, format: NetFormat ←
octal];
  HostNumber: --Token-- PROCEDURE [
      h: Handle, format: NetFormat ← octal, signalOnError: BOOLEAN ← TRUE]
      RETURNS [host: System.HostNumber];
  ibm2770Host: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
  ibm3270Host: --CHPIDs-- CH.PropertyID = 26;
  ibm3270Host: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
  ibm6670: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
  ibm6670Host: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
  IBMHost: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
      description(0:0..63): NSString.String,
      controllers(4:0..47): LONG DESCRIPTOR FOR ARRAY CARDINAL OF
          IBMHostControllerRecord];
  IBMHostControllerRecord: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
      controllerAddress(0:0..15): CARDINAL,
      portsOnController(1:0..15): CARDINAL,
      linkType(2:0..15): ControllerLinkType,
      path(3:0..31): NSName.Name,
      language(5:0..15): IBMLanguages,
      terminalModelType(6:0..127): ARRAY PortRange OF IBMTerminalType ← ALL[
          model2]];
  IBMHostDescribe: --CHLookup-- Courier.Description;
  IBMHostPt: --CHLookup-- TYPE = LONG POINTER TO IBMHost;
  IBMLanguages: --CHLookup-- TYPE = MACHINE DEPENDENT{
      USenglish, Austrian, AustrianAlt, German, GermanAlt, Belgian, Brazilian,
      CanadianFrench, Danish, DanishAlt, Norwegian, NorwegianAlt, Finnish,
      FinnishAlt, Swedish, SwedishAlt, French, International, Italian,
      JapaneseEnglish, JapaneseKana, Portuguese, Spanish, SpanishAlt,
      SpanishSpeaking, UKenglish, unused1, unused2, unused3, unused4,
unused5,
      unused6, (177777B)};
  IBMTerminalType: --CHLookup-- TYPE = MACHINE DEPENDENT{
      (0), model1, model2, model3, model4, model5, (177777B)};
  ID: --File-- TYPE [2];
  ID: --NSFile-- TYPE [5];
  ID: --NSSegment-- TYPE = CARDINAL;
  ID: --PhysicalVolume-- TYPE = System.PhysicalVolumeID;
```

```
ID: --Volume-- TYPE = System.VolumeID;
IDArrayHandle: --LibrarianUtility-- TYPE = LONG POINTER TO IDArrayObject;
IDArrayObject: --LibrarianUtility-- TYPE = RECORD [
    numberOfIDs: CARDINAL,
    idArray: LONG DESCRIPTOR FOR ARRAY CARDINAL OF Librarian.LibjectID];
IgnoreReadOnlyProc: --FormSW-- ReadOnlyProcType;
illegal: --RS232CCorrespondents-- RS232CEnvironment.AutoRecognitionOutcome;
IllegalEnumerate: --LogFile-- ERROR;
IllegalUserIdentifier: --CH-- ERROR [why: Result];
Inconsistent: --LogFile-- ERROR;
IncrementBand: --LsepFace-- PROCEDURE;
IncrementLine: --LsepFace-- PROCEDURE;
Index: --Log-- TYPE = CARDINAL;
Index: --RavenFace-- TYPE [1];
IndexAttributes: --NSVolumeControl-- TYPE = RECORD [
    size: LONG CARDINAL ← 100,
    pageIncrement: LONG CARDINAL ← 100,
    percentIncrement: Percent ← 20];
IndexFromEnumeratedValue: --FormSW-- PROCEDURE [EnumeratedHandle]
    RETURNS [CARDINAL];
IndexOutOfRange: --MemoryStream-- ERROR;
infiniteWaitTime: --NetworkStream-- READONLY WaitTime;
infinity: --Display-- INTEGER = 77777B;
infinity: --Router-- CARDINAL = 16;
Info: --AsciiSink-- PROCEDURE [sink: TextSink.Handle]
    RETURNS [font: WindowFont.Handle];
Info: --BlockSource-- PROCEDURE [source: Handle] RETURNS [block: Block];
Info: --Cursor-- TYPE = RECORD [type: Type, hotX: [0..15], hotY: [0..15]];
Info: --DiskSource-- PROCEDURE [source: TextSource.Handle]
    RETURNS [name: LONG STRING, s: Stream.Handle, access: TextSource.Access];
Info: --PieceSource-- PROCEDURE [source: TextSource.Handle]
    RETURNS [original: TextSource.Handle, scratch: TextSource.Handle];
Info: --ScratchSource-- PROCEDURE [source: TextSource.Handle]
    RETURNS [
        block: Environment.Block, extraRoom: CARDINAL, access: TextSource.Access,
        expandable: BOOLEAN];
Info: --ScratchSW-- PROCEDURE [sw: Window.Handle]
    RETURNS [
        block: Environment.Block, extraRoom: CARDINAL, expandable: BOOLEAN,
        options: Options];
Info: --Tool-- PROCEDURE [window: Window.Handle]
    RETURNS [
        name: LONG STRING, cmSection: LONG STRING, makeSWsProc:
MakeSWsProc,
        clientTransition: ToolWindow.TransitionProcType,
        movableBoundaries: BOOLEAN];
InfoProc: --FileTransfer-- TYPE = PROCEDURE [Connection]
    RETURNS [source: FileInfo, target: FileInfo];
InHeapFreeHintsProc: --FormSW-- FreeHintsProcType;
Initialize: --MailParse-- PROCEDURE [next: PROCEDURE RETURNS [CHARACTER]]
    RETURNS [Handle];
Initialize: --NSVolumeControl-- PROCEDURE [
    volume: Volume.ID, index: IndexAttributes ← [],
    root: NSFile.AttributeList ← NIL];
Initialize: --WindowFont-- PROCEDURE [font: Handle];
```

```
InitializeBand: --LsepFace-- PROCEDURE [band: Band]
    RETURNS [scanData: LONG POINTER];
InitializeCleanUp: --LsepFace-- PROCEDURE;
InitializeCleanUp: --RavenFace-- PROCEDURE;
InitializeCondition: --Process-- PROCEDURE [
    condition: LONG POINTER TO CONDITION, ticks: Ticks];
InitializeFileSystem: --MFile-- PROCEDURE;
InitializeLine: --LsepFace-- PROCEDURE [band: SingleLineBand]
    RETURNS [scanData: LONG POINTER];
InitializeMonitor: --Process-- PROCEDURE [monitor: LONG POINTER TO MONITORLOCK];
InitializePilotCounter: --PerformancePrograms-- PROCEDURE;
InitializePilotPerfMonitor: --PerformancePrograms-- PROCEDURE;
InitializePool: --ObjAlloc-- PROCEDURE [
    pool: AllocPoolDesc, initialState: AllocFree];
InitializeWindow: --Window-- PROCEDURE [
    window: Handle, display: PROCEDURE [Handle], box: Box,
    parent: Handle ← rootWindow, sibling: Handle ← NIL, child: Handle ← NIL,
    clearingRequired: BOOLEAN ← TRUE, under: BOOLEAN ← FALSE,
    cookieCutter: BOOLEAN ← FALSE];
InitialLength: --MFile-- TYPE = ByteCount;
initialToolStateDefault: --Profile-- READONLY ToolWindow.State;
InitiateBand: --LsepFace-- PROCEDURE [band: Band];
InitiateLine: --LsepFace-- PROCEDURE [band: SingleLineBand];
InitReals: --Real-- PROCEDURE;
Insert: --BTree-- PROCEDURE [tree: Tree, name: LONG STRING, value: Value]
    RETURNS [ok: BOOLEAN, noRoom: BOOLEAN];
InsertIntoTree: --Window-- PROCEDURE [window: Handle];
InsertionProblem: --NSFile-- TYPE = MACHINE DEPENDENT{
    positionUnavailable, fileNotUnique, loopInHierarchy};
InsertRootFile: --Volume-- PROCEDURE [type: File.Type, file: File.File];
Install: --Log-- PROCEDURE [
    file: File.File, firstPageNumber: File.PageNumber ← 1];
InstallBootMicrocode: --FormatPilotDisk-- PROCEDURE [
    h: PhysicalVolume.Handle, getPage: PROCEDURE RETURNS [LONG POINTER]];
Instantiate: --Menu-- PROCEDURE [menu: Handle, window: Window.Handle];
InstWord: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    evenbyte(0:0..7): BYTE, oddbyte(0:8..15): BYTE];
InsufficientSpace: --Volume-- ERROR [currentFreeSpace: PageCount, volume: ID];
Interpretation: --NSFile-- TYPE = MACHINE DEPENDENT{
    none, boolean, cardinal, longCardinal, integer, longInteger, string, time};
InterpretedSelections: --NSFile-- TYPE = PACKED ARRAY AttributeType OF
    BooleanFalseDefault;
Interpreter: --DebugUsefulDefs-- PROCEDURE [
    exp: LONG STRING, results: PROCEDURE [Handle]];
InterpretHandle: --FloppyChannel-- PROCEDURE [handle: Handle]
    RETURNS [drive: Drive];
InterpretHandle: --PhysicalVolume-- PROCEDURE [instance: Handle]
    RETURNS [type: Device.Type, index: CARDINAL];
Interrupt: --Runtime-- PROCEDURE;
interruptWatcher: --PilotSwitches-- PilotDomainA = 70C;
IntersectBoxes: --Window-- PROCEDURE [b1: Box, b2: Box] RETURNS [box: Box];
Interval: --ObjAlloc-- TYPE = RECORD [first: ItemIndex, count: ItemCount];
Interval: --Space-- TYPE = RECORD [
    pointer: LONG POINTER, count: Environment.PageCount];
InvalidAddress: --DebugUsefulDefs-- ERROR [address: LONG POINTER];
InvalidArguments: --Courier-- ERROR;
```

```
InvalidateBox: --Window-- PROCEDURE [
    window: Handle, box: Box, clarity: Clarity ← isDirty];
InvalidBase: --ExtendedString-- ERROR;
InvalidFile: --LogFile-- ERROR;
InvalidFrame: --DebugUsefulDefs-- ERROR [f: POINTER];
InvalidFrame: --Runtime-- ERROR [frame: UNSPECIFIED];
InvalidGlobalFrame: --Runtime-- ERROR [frame: GenericProgram];
InvalidLineNumber: --RS232C-- ERROR;
InvalidLineNumber: --TTYPort-- ERROR;
InvalidNode: --MDSStorage-- ERROR [p: POINTER];
InvalidNumber: --DebugUsefulDefs-- ERROR [p: LONG POINTER];
InvalidNumber: --NSString-- ERROR;
InvalidParameter: --RS232C-- ERROR;
InvalidProcess: --Process-- ERROR [process: UNSPECIFIED];
InvalidString: --NSString-- ERROR;
InvalidSwitches: --HeraldWindow-- SIGNAL;
InvalidType: --CHLookup-- ERROR;
InvalidVersion: --OthelloOps-- ERROR;
Invert: --Cursor-- PROCEDURE RETURNS [BOOLEAN];
Invert: --Display-- PROCEDURE [window: Handle, box: Window.Box];
Invoke: --Menu-- PROCEDURE [window: Window.Handle, place: Window.Place];
IRS: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    address(0:0..95): System.NetworkAddress, location(6:0..63): NSString.String];
IRSDescribe: --CHLookup-- Courier.Description;
IRSPt: --CHLookup-- TYPE = LONG POINTER TO IRS;
IsBandFinished: --LsepFace-- PROCEDURE [band: Band] RETURNS [BOOLEAN];
IsBandImageBegun: --LsepFace-- PROCEDURE [band: Band] RETURNS [BOOLEAN];
IsBitmapUnderVariant: --Window-- PROCEDURE [Handle] RETURNS [BOOLEAN];
IsBound: --Runtime-- PROCEDURE [link: ControlLink] RETURNS [BOOLEAN];
IsCookieVariant: --Window-- PROCEDURE [Handle] RETURNS [BOOLEAN];
IsDescendantOfRoot: --Window-- PROCEDURE [Handle] RETURNS [BOOLEAN];
isDirectory: --NSAssignedTypes-- AttributeType = 5;
IsEditable: --FileSW-- PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
IsIt: --AsciiSink-- PROCEDURE [sink: TextSink.Handle] RETURNS [BOOLEAN];
IsIt: --BlockSource-- PROCEDURE [source: Handle] RETURNS [yes: BOOLEAN];
IsIt: --DiskSource-- PROCEDURE [source: TextSource.Handle] RETURNS [BOOLEAN];
IsIt: --FileSW-- PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
IsIt: --FileWindow-- PROCEDURE [sw: Window.Handle] RETURNS [BOOLEAN];
IsIt: --FormSW-- PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
IsIt: --MsgSW-- PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
IsIt: --MStream-- PROCEDURE [stream: Handle] RETURNS [BOOLEAN];
IsIt: --PieceSource-- PROCEDURE [source: TextSource.Handle]
    RETURNS [yes: BOOLEAN];
IsIt: --ScratchSource-- PROCEDURE [source: TextSource.Handle]
    RETURNS [yes: BOOLEAN];
IsIt: --ScratchSW-- PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
IsIt: --Tool-- PROCEDURE [window: Window.Handle] RETURNS [BOOLEAN];
IsIt: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [yes: BOOLEAN];
IsItemInverted: --FormSW-- PROCEDURE [sw: Window.Handle, index: CARDINAL]
    RETURNS [yes: BOOLEAN];
IsLineFinished: --LsepFace-- PROCEDURE [band: SingleLineBand] RETURNS
[BOOLEAN];
IsLineImageBegun: --LsepFace-- PROCEDURE [band: SingleLineBand]
    RETURNS [BOOLEAN];
IsMember: --CH-- PROCEDURE [
    cred: Authenticator.Credentials, ver: Authenticator.Verifier,
```

```
        element: Element, name: Pattern, pn: PropertyID, distingName: Name]
        RETURNS [rc: ReturnCode, isMember: BOOLEAN];
    IsMemberClosure: --CH-- PROCEDURE [
        cred: Authenticator.Credentials, ver: Authenticator.Verifier,
        element: Element, name: Pattern, pn: PropertyID, distingName: Name,
        pn2: PropertyID ← unspecified] RETURNS [rc: ReturnCode, isMember:
BOOLEAN];
    IsMemberOfDomainAccess: --MoreCH-- PROCEDURE [
        cred: Authenticator.Credentials, ver: Authenticator.Verifier,
        element: CH.Element, domain: CH.Name, acl: ACLFlavor,
        pn2: CH.PropertyID ← unspecified]
        RETURNS [rc: CH.ReturnCode, isMember: BOOLEAN];
    IsMemberOfOrgAccess: --MoreCH-- PROCEDURE [
        cred: Authenticator.Credentials, ver: Authenticator.Verifier,
        element: CH.Element, org: CH.Name, acl: ACLFlavor,
        pn2: CH.PropertyID ← unspecified]
        RETURNS [rc: CH.ReturnCode, isMember: BOOLEAN];
    IsMemberOfPropertyAccess: --MoreCH-- PROCEDURE [
        cred: Authenticator.Credentials, ver: Authenticator.Verifier,
        element: CH.Element, name: CH.Name, pn: CH.PropertyID, acl: ACLFlavor,
        distingName: CH.Name, pn2: CH.PropertyID ← unspecified]
        RETURNS [rc: CH.ReturnCode, isMember: BOOLEAN];
    IsPlaceInBox: --Window-- PROCEDURE [place: Place, box: Box] RETURNS [BOOLEAN];
    IsPlaceInWindow: --ToolWindow-- PROCEDURE [place: Place, window: Handle]
        RETURNS [BOOLEAN];
    IsReady: --PhysicalVolume-- PROCEDURE [instance: Handle]
        RETURNS [ready: BOOLEAN];
    IsSegmentedFileType: --NSVolumeControl-- PROCEDURE [type: NSFile.Type]
        RETURNS [BOOLEAN];
    IsSegmentEmpty: --Zone-- PROCEDURE [zH: Handle, sH: SegmentHandle]
        RETURNS [empty: BOOLEAN];
    isTemporary: --NSAssignedTypes-- AttributeType = 6;
    IsTimeValid: --OthelloOps-- PROCEDURE RETURNS [valid: BOOLEAN];
    IsZoneEmpty: --Zone-- PROCEDURE [zH: Handle] RETURNS [empty: BOOLEAN];
    Item: --BodyDefs-- TYPE = LONG POINTER TO ItemHeader;
    Item: --Token-- PROCEDURE [h: Handle, temporary: BOOLEAN ← TRUE]
        RETURNS [value: LONG STRING];
    ItemCount: --ObjAlloc-- TYPE = LONG CARDINAL;
    ItemDescriptor: --FormSW-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
        ItemHandle;
    ItemError: --FormSW-- SIGNAL [code: ItemErrorCode, index: CARDINAL];
    ItemErrorCode: --FormSW-- TYPE = {
        illegalCoordinate, notStringOtherItem, nilBackingStore, other};
    ItemFlags: --FormSW-- TYPE = RECORD [
        readOnly: BOOLEAN ← FALSE,
        invisible: BOOLEAN ← FALSE,
        drawBox: BOOLEAN ← FALSE,
        hasContext: BOOLEAN ← FALSE,
        clientOwnsItem: BOOLEAN ← FALSE,
        modified: BOOLEAN ← FALSE];
    ItemHandle: --FormSW-- TYPE = LONG POINTER TO ItemObject;
    ItemHandle: --Menu-- TYPE = LONG POINTER TO ItemObject;
    ItemHeader: --BodyDefs-- TYPE = MACHINE DEPENDENT RECORD [
        type(0:0..15): ItemType, length(1:0..31): ItemLength];
    ItemIndex: --ObjAlloc-- TYPE = LONG CARDINAL;
    ItemLength: --BodyDefs-- TYPE = LONG CARDINAL;
```

```
ItemObject: --FormSW-- TYPE = RECORD [
    tag: LONG STRING,
    place: Window.Place,
    flags: ItemFlags,
    body: SELECT type: ItemType FROM
        boolean => [switch: LONG POINTER TO BOOLEAN, proc: NotifyProcType],
        command => [proc: ProcType],
        enumerated => [
            feedback: EnumeratedFeedback,
            copyChoices: BOOLEAN,
            value: LONG POINTER,
            proc: EnumeratedNotifyProcType,
            choices: EnumeratedDescriptor],
        longNumber => [
            signed: BOOLEAN,
            notNegative: BOOLEAN,
            radix: Radix,
            boxWidth: CARDINAL [0..255],
            proc: LongNumberNotifyProcType,
            default: LONG UNSPECIFIED,
            value: LONG POINTER TO LONG UNSPECIFIED,
            string: LONG STRING,
            bias: INTEGER],
        number => [
            signed: BOOLEAN,
            notNegative: BOOLEAN,
            radix: Radix,
            boxWidth: CARDINAL [0..127],
            proc: NumberNotifyProcType,
            default: UNSPECIFIED,
            value: LONG POINTER,
            string: LONG STRING,
            bias: INTEGER],
        source => [
            source: TextSource.Handle,
            boxWidth: CARDINAL,
            filterProc: FilterProcType,
            menuProc: MenuProcType],
        string => [
            feedback: StringFeedback,
            inHeap: BOOLEAN,
            string: LONG POINTER TO LONG STRING,
            boxWidth: CARDINAL,
            filterProc: FilterProcType,
            menuProc: MenuProcType],
        tagOnly => [sw: Window.Handle, otherItem: CARDINAL],
        ENDCASE];
ItemObject: --Menu-- TYPE = RECORD [keyword: LONG STRING, mcrProc: MCRType];
Items: --Menu-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF ItemObject;
ItemType: --BodyDefs-- TYPE = MACHINE DEPENDENT{
    PostMark(8), Sender(16), ReturnTo(24), Recipients(32), Text(520),
    Capability(528), Audio(536), updateItem(1024), reMail(2100B),
    LastItem(177777B)};
ItemType: --FormSW-- TYPE = {
    boolean, command, enumerated, longNumber, number, source, string,
tagOnly};
```

```
ITS: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    address(0:0..95): System.NetworkAddress, location(6:0..63): NSString.String];
ITSDescribe: --CHLookup-- Courier.Description;
ITSPt: --CHLookup-- TYPE = LONG POINTER TO ITS;
k10: --KeyStations-- Bit = 19;
k11: --KeyStations-- Bit = 21;
k12: --KeyStations-- Bit = 17;
k13: --KeyStations-- Bit = 53;
k14: --KeyStations-- Bit = 64;
k15: --KeyStations-- Bit = 51;
k16: --KeyStations-- Bit = 16;
k17: --KeyStations-- Bit = 23;
k18: --KeyStations-- Bit = 65;
k19: --KeyStations-- Bit = 66;
k1: --KeyStations-- Bit = 48;
k20: --KeyStations-- Bit = 18;
k21: --KeyStations-- Bit = 55;
k22: --KeyStations-- Bit = 67;
k23: --KeyStations-- Bit = 68;
k24: --KeyStations-- Bit = 20;
k25: --KeyStations-- Bit = 70;
k26: --KeyStations-- Bit = 22;
k27: --KeyStations-- Bit = 54;
k28: --KeyStations-- Bit = 69;
k29: --KeyStations-- Bit = 71;
k2: --KeyStations-- Bit = 35;
k30: --KeyStations-- Bit = 39;
k31: --KeyStations-- Bit = 25;
k32: --KeyStations-- Bit = 38;
k33: --KeyStations-- Bit = 43;
k34: --KeyStations-- Bit = 41;
k35: --KeyStations-- Bit = 42;
k36: --KeyStations-- Bit = 24;
k37: --KeyStations-- Bit = 58;
k38: --KeyStations-- Bit = 27;
k39: --KeyStations-- Bit = 59;
k3: --KeyStations-- Bit = 37;
k40: --KeyStations-- Bit = 26;
k41: --KeyStations-- Bit = 28;
k42: --KeyStations-- Bit = 74;
k43: --KeyStations-- Bit = 44;
k44: --KeyStations-- Bit = 75;
k45: --KeyStations-- Bit = 45;
k46: --KeyStations-- Bit = 61;
k47: --KeyStations-- Bit = 107;
k48: --KeyStations-- Bit = 49;
k4: --KeyStations-- Bit = 33;
k5: --KeyStations-- Bit = 56;
k6: --KeyStations-- Bit = 34;
k7: --KeyStations-- Bit = 36;
k8: --KeyStations-- Bit = 32;
k9: --KeyStations-- Bit = 40;
Key: --Authenticator-- TYPE [4];
KeyBits: --JLevelIVKeys-- TYPE = PACKED ARRAY KeyName OF DownUp;
KeyBits: --Keys-- TYPE = PACKED ARRAY KeyName OF DownUp;
KeyBits: --KeyStations-- TYPE = PACKED ARRAY KeyStation OF DownUp;
```

```
KeyBits: --LevelIIIKeys-- TYPE = PACKED ARRAY KeyName OF DownUp;
KeyBits: --LevelIVKeys-- TYPE = PACKED ARRAY KeyName OF DownUp;
keyboard: --UserTerminal-- READONLY LONG POINTER TO READONLY ARRAY CARDINAL OF
    WORD;
KeyboardAndMouseTest: --OnlineDiagnostics-- PROCEDURE [
    keyboardType: KeyboardType, screenHeight: CARDINAL [0..77777B],
    screenWidth: CARDINAL [0..77777B],
    SetBackground: PROCEDURE [background: Background],
    SetBorder: PROCEDURE [oddPairs: [0..255], evenPairs: [0..255]],
    GetMousePosition: PROCEDURE RETURNS [Coordinate],
    SetMousePosition: PROCEDURE [newMousePosition: Coordinate],
    SetCursorPattern: PROCEDURE [cursorArray: CursorArray],
    SetCursorPosition: PROCEDURE [newCursorPosition: Coordinate],
    keyboard: LONG POINTER, Beep: PROCEDURE [duration: CARDINAL],
    ClearDisplay: PROCEDURE,
    BlackenScreen: PROCEDURE [
        x: CARDINAL, y: CARDINAL, width: CARDINAL, height: CARDINAL],
    InvertScreen: PROCEDURE [
        x: CARDINAL, y: CARDINAL, width: CARDINAL, height: CARDINAL],
    WaitForKeyTransition: PROCEDURE];
KeyboardType: --OnlineDiagnostics-- TYPE = {american, european, japanese};
KeyName: --JLevelIVKeys-- TYPE = MACHINE DEPENDENT{
    Red(13), Blue, Five(16), Four, Six, E, Seven, D, U, V, Zero, K, Dash, P,
    Slash, Font, SameAs, BS, Three, Two, W, Q, S, A, Nine, I, X, O, L, Comma,
    Quote, RightBracket, Open, Special, One, Tab, ParaTab, F, Props, C, J, B, Z,
    LeftHandakuonShift, Period, SemiColon, NewPara, Para, Delete, Next, R, T,
G,
    Y, H, Eight, N, M, Lock, Hiragana, Half, Equal, RightDakuonShift, Stop, Move,
    Undo, Margins, English(86), Katakana(88), Copy, Find, Again, Help, Expand,
    Center(97), Bold(99), Italic, Underlined, Superscript, Subscript, Smaller,
    LeftDakuonShift(107), Defaults(109), Space, RightHandakuonShift};
KeyName: --Keys-- TYPE = MACHINE DEPENDENT{
    Keyset1(8), Keyset2, Keyset3, Keyset4, Keyset5, Point, Adjust, Menu, Five,
    Four, Six, E, Seven, D, U, V, Zero, K, Dash, P, Slash, BackSlash, PASTE, BS,
    Three, Two, W, Q, S, A, Nine, I, X, O, L, Comma, Quote, RightBracket, STUFF,
    COMMAND, One, COMPLETE, TAB, F, CONTROL, C, J, B, Z, LeftShift, Period,
    SemiColon, Return, Arrow, DELETE, NEXT, R, T, G, Y, H, Eight, N, M, LOCK,
    Space, LeftBracket, Equal, RightShift, USERABORT, MOVE, UNDO, DOIT, R9,
L10,
    L7, L4, L1, A9, R10, A8, COPY, FIND, AGAIN, HELP, EXPAND, R4, D2, D1,
MENU,
    T1, SCROLLBAR, JFIRST, JSELECT, RESERVED, CLIENT1, CLIENT2, T10, R3,
Key47,
    A10, ATTENTION, A11, A12};
KeyName: --LevelIIIKeys-- TYPE = MACHINE DEPENDENT{
    Red(13), Blue, Yellow, Five, Four, Six, E, Seven, D, U, V, Zero, K, Dash, P,
    Slash, Font, GlobIRplce, BS, Three, Two, W, Q, S, A, Nine, I, X, O, L, Comma,
    Quote, RightBracket, Again, Special, One, TAB(50), F, Props, C, J, B, Z,
    LeftShift, Period, SemiColon, Return, Para, Delete, Next, R, T, G, Y, H,
    Eight, N, M, Lock, Space, Half, Equal, RightShift, R12, Move, R6, Carriage,
    R9, L10, L7, L4, L1, A9, A8(88), Copy, Find, Undo, Help, Expand, Indent(97),
    T1, Justify, Center, Bold, Italics, Underline, Subscript, T10, R3,
    Smaller(109)};
KeyName: --LevelIVKeys-- TYPE = MACHINE DEPENDENT{
    Red(13), Blue, Five(16), Four, Six, E, Seven, D, U, V, Zero, K, Dash, P,
    Slash, Font, Same, BS, Three, Two, W, Q, S, A, Nine, I, X, O, L, Comma, Quote,
```

```
        RightBracket, Open, Special, One, Tab, ParaTab, F, Props, C, J, B, Z,
        LeftShift, Period, SemiColon, NewPara, Para, Delete, Next, R, T, G, Y, H,
        Eight, N, M, Lock, Space, Half, Equal, RightShift, Stop, Move, Undo, Margins,
        Copy(89), Find, Again, Help, Expand, Center(97), Bold(99), Italic, Underlined,
        Superscript, Subscript, Smaller, Defaults(109)};
    KeyStation: --KeyStations-- TYPE = [0..111];
    Kill: --MSegment-- PROCEDURE [segment: Handle];
    KS1: --KeyStations-- Bit = 8;
    KS2: --KeyStations-- Bit = 9;
    KS3: --KeyStations-- Bit = 10;
    KS4: --KeyStations-- Bit = 11;
    KS5: --KeyStations-- Bit = 12;
    L10: --KeyStations-- Bit = 82;
    L11: --KeyStations-- Bit = 46;
    L12: --KeyStations-- Bit = 52;
    L1: --KeyStations-- Bit = 85;
    L2: --KeyStations-- Bit = 91;
    L3: --KeyStations-- Bit = 62;
    L4: --KeyStations-- Bit = 84;
    L5: --KeyStations-- Bit = 90;
    L6: --KeyStations-- Bit = 89;
    L7: --KeyStations-- Bit = 83;
    L8: --KeyStations-- Bit = 30;
    L9: --KeyStations-- Bit = 78;
    LabelHandle: --FormSW-- TYPE = TagOnlyHandle;
    LabelItem: --FormSW-- PROCEDURE [
        tag: LONG STRING ← NIL, readOnly: BOOLEAN ← FALSE, invisible: BOOLEAN ← FALSE,
        drawBox: BOOLEAN ← FALSE, hasContext: BOOLEAN ← FALSE,
        place: Window.Place ← nextPlace, z: UNCOUNTED ZONE ← NIL]
        RETURNS [LabelHandle];
    largeAnonymousBackingFile: --PilotSwitches-- AnonymousBackingFileSize =
  175C;
    LargeReturnSlot: --PrincOps-- CARDINAL = 31;
    LargestNumber: --Real-- REAL;
    LastAVHeapSlot: --PrincOps-- CARDINAL = 30;
    LastBand: --RavenFace-- PROCEDURE [Index];
    LastLine: --MsgSW-- PROCEDURE [sw: Window.Handle, ss: String.SubString];
    lastPageCount: --Environment-- PageCount = 77777777B;
    lastPageCount: --File-- PageCount = 37777777B;
    lastPageCount: --Floppy-- PageNumber = 37777777B;
    lastPageCount: --PhysicalVolume-- PageCount = 377777777776B;
    lastPageCount: --Volume-- PageCount = 40000000B;
    lastPageNumber: --Environment-- PageNumber = 77777776B;
    lastPageNumber: --File-- PageNumber = 37777776B;
    lastPageNumber: --PhysicalVolume-- PageNumber = 377777777776B;
    lastPageNumber: --Volume-- PageNumber = 37777777B;
    lastPageOffset: --Environment-- PageOffset = 77777776B;
    lastPosition: --NSFile-- READONLY Position;
    lastPositionRepresentation: --NSFile-- ARRAY [0..0] OF UNSPECIFIED;
    lastServicesAType: --NSAssignedTypes-- AssignedType = 10377B;
    lastServicesBType: --NSAssignedTypes-- AssignedType = 11777B;
    lastStandardType: --NSAssignedTypes-- AssignedType = 7777B;
    lastStarType: --NSAssignedTypes-- AssignedType = 10777B;
    lastWS860Type: --NSAssignedTypes-- AssignedType = 12017B;
    LatchBitClearMask: --RS232C-- TYPE = DeviceStatus;
```

```
  Layout: --PhysicalVolume-- TYPE = {
      partialLogicalVolume, singleLogicalVolume, multipleLogicalVolumes,
empty};
  LDIVMOD: --Inline-- PROCEDURE [numlow: WORD, numhigh: CARDINAL, den:
CARDINAL]
      RETURNS [quotient: CARDINAL, remainder: CARDINAL];
  LeftShift: --JLevelIVKeys-- KeyName = LeftHandakuonShift;
  Lengthen: --DebugUsefulDefs-- PROCEDURE [ClientSource] RETURNS
[LongClientDest];
  LengthRange: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT RECORD [
      low(0:0..15): [0..999], high(1:0..15): [0..999]];
  Level: --Log-- TYPE = State [error..remark];
  LF: --Ascii-- CHARACTER = 12C;
  LFDisplayTest: --OnlineDiagnostics-- PROCEDURE [
      screenHeight: CARDINAL [0..77777B], screenWidth: CARDINAL [0..77777B],
      SetBackground: PROCEDURE [background: Background],
      SetBorder: PROCEDURE [oddPairs: [0..255], evenPairs: [0..255]],
      GetNextAction: PROCEDURE RETURNS [NextAction], ClearDisplay: PROCEDURE,
      BlackenScreen: PROCEDURE [
          x: CARDINAL, y: CARDINAL, width: CARDINAL, height: CARDINAL],
      FillScreenWithObject: PROCEDURE [p: LONG POINTER TO ARRAY [0..15] OF WORD]];
  librarian: --EventTypes-- Supervisor.Event;
  Life: --Space-- TYPE = {alive, dead};
  LimitProcType: --ToolWindow-- TYPE = PROCEDURE [window: Handle, box: Box]
      RETURNS [Box];
  line0: --FormSW-- INTEGER = -3;
  line1: --FormSW-- INTEGER = -4;
  line2: --FormSW-- INTEGER = -5;
  line3: --FormSW-- INTEGER = -6;
  line4: --FormSW-- INTEGER = -7;
  line5: --FormSW-- INTEGER = -8;
  line6: --FormSW-- INTEGER = -9;
  line7: --FormSW-- INTEGER = -10;
  line8: --FormSW-- INTEGER = -11;
  line9: --FormSW-- INTEGER = -12;
  Line: --CmFile-- PROCEDURE [
      fileName: LONG STRING, title: LONG STRING, name: LONG STRING]
      RETURNS [LONG STRING];
  Line: --Display-- PROCEDURE [
      window: Handle, start: Window.Place, stop: Window.Place,
      bounds: Window.BoxHandle ← NIL];
  Line: --Format-- PROCEDURE [
      proc: StringProc, s: LONG STRING, clientData: LONG POINTER ← NIL];
  Line: --Put-- PROCEDURE [h: Window.Handle ← NIL, s: LONG STRING];
  Line: --Token-- FilterProcType;
  LineHeight: --FormSW-- PROCEDURE [sw: Window.Handle ← NIL] RETURNS [CARDINAL];
  LineN: --FormSW-- PROCEDURE [n: CARDINAL] RETURNS [INTEGER];
  LineOverflow: --TTY-- SIGNAL [s: LONG STRING] RETURNS [ns: LONG STRING];
  LineOverflow: --TTYSW-- SIGNAL [s: LONG STRING] RETURNS [ns: LONG STRING];
  LineSpeed: --RS232C-- TYPE = RS232CEnvironment.LineSpeed;
  LineSpeed: --RS232CEnvironment-- TYPE = {
      bps50, bps75, bps110, bps134p5, bps150, bps300, bps600, bps1200, bps2400,
      bps3600, bps4800, bps7200, bps9600, bps19200, bps28800, bps38400,
bps48000,
      bps56000, bps57600};
  LineSpeed: --TTYPort-- TYPE = TTYPortEnvironment.LineSpeed;
```

```
LineSpeed: --TTYPortEnvironment-- TYPE = {
    bps50, bps75, bps110, bps134p5, bps150, bps300, bps600, bps1200, bps1800,
    bps2000, bps2400, bps3600, bps4800, bps7200, bps9600, bps19200};
linesPerBand: --LsepFace-- CARDINAL = 16;
LineType: --RS232C-- TYPE = RS232CEnvironment.LineType;
LineType: --RS232CEnvironment-- TYPE = {
    bitSynchronous, byteSynchronous, asynchronous, autoRecognition};
LinkageFault: --Runtime-- ERROR;
List: --NSFile-- PROCEDURE [
    directory: Handle, proc: AttributesProc, selections: Selections,
    scope: Scope ← [], session: Session ← nullSession];
Listen: --NetworkStream-- PROCEDURE [
    listenerH: ListenerHandle,
    connectData: Environment.Block ← Environment.nullBlock,
    listenTimeout: WaitTime ← infiniteWaitTime]
    RETURNS [remote: System.NetworkAddress, bytes: CARDINAL];
ListenerHandle: --NetworkStream-- TYPE [2];
ListenError: --NetworkStream-- ERROR [reason: ListenErrorReason];
ListenErrorReason: --NetworkStream-- TYPE = {
    illegalAddress, illegalHandle, illegalState, blockTooShort};
ListenTimeout: --NetworkStream-- SIGNAL;
ListProc: --FileTransfer-- TYPE = PROCEDURE [
    conn: Connection, clientData: LONG POINTER, file: LONG STRING,
    post: MessageProc, info: InfoProc] RETURNS [Confirmation];
Ln: --RealFns-- PROCEDURE [REAL] RETURNS [REAL];
load: --EventTypes-- Supervisor.Event;
Load: --Exec-- PROCEDURE [
    write: Format.StringProc, name: LONG STRING, codeLinks: BOOLEAN ← FALSE]
    RETURNS [handle: MLoader.Handle];
Load: --MLoader-- PROCEDURE [
    file: MFile.Handle, options: Options ← defaultOptions] RETURNS [Handle];
LoadConfig: --Runtime-- PROCEDURE [
    file: File.File, offset: File.PageCount, codeLinks: BOOLEAN ← FALSE]
    RETURNS [PROGRAM];
LoadMCR: --FileSW-- Menu.MCRType;
LoadWindow: --FileWindow-- PROCEDURE [
    fileName: LONG STRING, position: LONG CARDINAL ← 0, s: Stream.Handle ← NIL,
    loadIfSame: BOOLEAN ← FALSE, sw: Window.Handle ← NIL];
Local: --NSName-- TYPE = String ← NSString.nullString;
localbase: --PrincOps-- CARDINAL = 0;
LocalDest: --DebugUsefulDefs-- TYPE = LONG POINTER;
LocalFrame: --BackstopNub-- TYPE [1];
LocalFrameBase: --PrincOps-- TYPE = POINTER TO LocalOverhead;
LocalFrameHandle: --PrincOps-- TYPE = POINTER TO LocalVariables;
LocalName: --CH-- TYPE = NSName.Local;
LocalOverhead: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    word(0:0..15): LocalWord,
    returnlink(1:0..15): ShortControlLink,
    globallink(2:0..15): GlobalFrameHandle,
    pc(3:0..15): BytePC,
    local(4): LocalVariables];
LocalSource: --DebugUsefulDefs-- TYPE = LONG POINTER TO READONLY UNSPECIFIED;
LocalSystemElement: --Courier-- PROCEDURE RETURNS [SystemElement];
localSystemElement: --NSFile-- READONLY SystemElement;
LocalVariables: --PrincOps-- TYPE = ARRAY CARDINAL [0..0) OF UNSPECIFIED;
```

```
LocalVFN: --FileTransfer-- PROCEDURE [conn: Connection, vfn: FileName.VFN]
    RETURNS [BOOLEAN];
LocalWord: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    available(0:0..7): BYTE, fsi(0:8..15): FSIndex];
localWordOffset: --PrincOps-- CARDINAL = 4;
Lock: --NSFile-- TYPE = MACHINE DEPENDENT{none, share, exclusive};
Log: --MFile-- PROCEDURE [
    name: LONG STRING, release: ReleaseData,
    initialLength: InitialLength ← dontCare] RETURNS [Handle];
Log: --MStream-- PROCEDURE [name: LONG STRING, release: ReleaseData]
    RETURNS [Handle];
Log: --NSVolumeControl-- TYPE = MACHINE DEPENDENT RECORD [
    header(0:0..159): Header, firstEntry(10:0..143): Entry];
Log: --RealFns-- PROCEDURE [base: REAL, arg: REAL] RETURNS [REAL];
logBitsPerByte: --Environment-- CARDINAL = 3;
logBitsPerChar: --Environment-- CARDINAL = 3;
logBitsPerWord: --Environment-- CARDINAL = 4;
logBytesPerPage: --Environment-- CARDINAL = 9;
logBytesPerWord: --Environment-- CARDINAL = 1;
logCap: --Log-- READONLY File.File;
logCharsPerPage: --Environment-- CARDINAL = 9;
logCharsPerWord: --Environment-- CARDINAL = 1;
LogEntry: --VolumeConversion-- TYPE = MACHINE DEPENDENT RECORD [
    oldFileID(0:0..79): OldFileID, newFileID(5:0..31): File.ID];
LogError: --Backstop-- PROCEDURE;
LogFormat: --Scavenger-- TYPE = MACHINE DEPENDENT RECORD [
    header(0:0..191): Header, files(12): ARRAY [0..0) OF FileEntry];
LogFrame: --Backstop-- PROCEDURE [frame: Frame];
LogicalLength: --NSString-- PROCEDURE [s: String] RETURNS [CARDINAL];
LogicalVolumePageNumber: --OthelloOps-- TYPE = LONG CARDINAL;
Login: --Exec-- PROCEDURE [h: Handle, name: LONG STRING, password: LONG STRING];
Logoff: --NSFile-- PROCEDURE [session: Session ← nullSession];
Logon: --NSFile-- PROCEDURE [
    name: String, password: String,
    systemElement: SystemElement ← nullSystemElement] RETURNS [Session];
LogonPrivileged: --NSSessionControl-- PROCEDURE [
    name: NSString.String, password: NSString.String] RETURNS [NSFile.Session];
LogonWithCredentials: --NSFile-- PROCEDURE [
    credentials: Credentials, verifier: Verifier,
    systemElement: SystemElement ← nullSystemElement] RETURNS [Session];
LogProcess: --Backstop-- PROCEDURE [process: Process];
LogSeal: --Scavenger-- CARDINAL = 130725B;
LogState: --VolumeConversion-- TYPE = {logComplete, mappingsMayBeLost};
logWordsPerPage: --Environment-- CARDINAL = 8;
Long: --Environment-- TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
    lc = > [lc(0:0..31): LONG CARDINAL],
    li = > [li(0:0..31): LONG INTEGER],
    lp = > [lp(0:0..31): LONG POINTER],
    lu = > [lu(0:0..31): LONG UNSPECIFIED],
    num = > [lowbits(0:0..15): CARDINAL, highbits(1:0..15): CARDINAL],
    any = > [low(0:0..15): UNSPECIFIED, high(1:0..15): UNSPECIFIED],
    ENDCASE];
LongCARDINAL: --Inline-- TYPE = LONG CARDINAL;
LongClientDest: --DebugUsefulDefs-- TYPE = LONG POINTER;
```

```
  LongClientSource: --DebugUsefulDefs-- TYPE = LONG POINTER TO READONLY
      UNSPECIFIED;
  LongCOPY: --Inline-- PROCEDURE [
      from: LONG POINTER, nwords: CARDINAL, to: LONG POINTER];
  LongCopyREAD: --DebugUsefulDefs-- PROCEDURE [
      from: LongClientSource, nwords: CARDINAL, to: LocalDest];
  LongCOPYReverse: --Inline-- PROCEDURE [
      from: LONG POINTER, nwords: CARDINAL, to: LONG POINTER];
  LongCopyWRITE: --DebugUsefulDefs-- PROCEDURE [
      from: LocalSource, nwords: CARDINAL, to: LongClientDest];
  LongDecimal: --Format-- PROCEDURE [
      proc: StringProc, n: LONG INTEGER, clientData: LONG POINTER ← NIL];
  LongDecimal: --Put-- PROCEDURE [h: Window.Handle ← NIL, n: LONG INTEGER];
  LongDiv: --Inline-- PROCEDURE [num: LONG CARDINAL, den: CARDINAL]
      RETURNS [CARDINAL];
  LongDivMod: --Inline-- PROCEDURE [num: LONG CARDINAL, den: CARDINAL]
      RETURNS [quotient: CARDINAL, remainder: CARDINAL];
  LongMult: --Inline-- PROCEDURE [CARDINAL, CARDINAL]
      RETURNS [product: LONG CARDINAL];
  LongNumber: --Environment-- TYPE = Long;
  LongNumber: --Format-- PROCEDURE [
      proc: StringProc, n: LONG UNSPECIFIED, format: NumberFormat,
      clientData: LONG POINTER ← NIL];
  LongNumber: --Inline-- TYPE = Environment.LongNumber;
  LongNumber: --Put-- PROCEDURE [
      h: Window.Handle ← NIL, n: LONG UNSPECIFIED, format: Format.NumberFormat];
  LongNumber: --Selection-- PROCEDURE [radix: CARDINAL ← 10]
      RETURNS [LONG CARDINAL];
  LongNumberHandle: --FormSW-- TYPE = LONG POINTER TO longNumber
ItemObject;
  LongNumberItem: --FormSW-- PROCEDURE [
      tag: LONG STRING ← NIL, readOnly: BOOLEAN ← FALSE, invisible: BOOLEAN ← FALSE,
      drawBox: BOOLEAN ← FALSE, hasContext: BOOLEAN ← FALSE,
      place: Window.Place ← nextPlace, signed: BOOLEAN ← TRUE,
      notNegative: BOOLEAN ← FALSE, radix: Radix ← decimal,
      boxWidth: CARDINAL [0..255] ← 64,
      proc: LongNumberNotifyProcType ← NopLongNumberNotifyProc,
      default: LONG UNSPECIFIED ← 17777777777B,
      value: LONG POINTER TO LONG UNSPECIFIED, bias: INTEGER ← 0,
      z: UNCOUNTED ZONE ← NIL] RETURNS [LongNumberHandle];
  LongNumberNotifyProcType: --FormSW-- TYPE = PROCEDURE [
      sw: Window.Handle ← NIL, item: ItemHandle ← NIL, index: CARDINAL ←
nullIndex,
      oldValue: LONG UNSPECIFIED ← 17777777777B];
  LongOctal: --Format-- PROCEDURE [
      proc: StringProc, n: LONG UNSPECIFIED, clientData: LONG POINTER ← NIL];
  LongOctal: --Put-- PROCEDURE [h: Window.Handle ← NIL, n: LONG UNSPECIFIED];
  LongPointerFromPage: --Environment-- PROCEDURE [page: PageNumber]
      RETURNS [LONG POINTER];
  LongPointerFromPage: --Space-- PROCEDURE [page: Environment.PageNumber]
      RETURNS [LONG POINTER];
  LongREAD: --DebugUsefulDefs-- PROCEDURE [loc: LongClientSource]
      RETURNS [val: UNSPECIFIED];
  LongString: --Format-- PROCEDURE [
      proc: StringProc, s: LONG STRING, clientData: LONG POINTER ← NIL];
  LongString: --Put-- PROCEDURE [h: Window.Handle ← NIL, s: LONG STRING];
```

```
  LongSubString: --Put-- PROCEDURE [h: Window.Handle ← NIL, ss: String.SubString];
  LongSubStringItem: --Format-- PROCEDURE [
      proc: StringProc, ss: String.SubString, clientData: LONG POINTER ← NIL];
  LongWRITE: --DebugUsefulDefs-- PROCEDURE [
      loc: LongClientDest, val: UNSPECIFIED];
  LookUp: --FileTransfer-- PROCEDURE [conn: Connection, file: FileName.VFN]
      RETURNS [fileInfo: FileInfo];
  LookupAliasesOfName: --CH-- PROCEDURE [
      cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Pattern,
      eachAlias: NameStreamProc, distingName: Name] RETURNS [rc: ReturnCode];
  LookupCIU: --CHLookup-- PROCEDURE [
      name: NSName.Name, clientProc: PROCEDURE [fullName: NSName.Name, info:
CIUPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupCommand: --Exec-- PROCEDURE [command: LONG STRING]
      RETURNS [
          name: LONG STRING, proc: ExecProc, help: ExecProc, unload: ExecProc,
          didExpand: BOOLEAN, clientData: LONG POINTER ← NIL];
  LookupDistinguishedName: --CH-- PROCEDURE [
      cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Pattern,
      distingName: Name] RETURNS [rc: ReturnCode];
  LookupDomainAccess: --MoreCH-- PROCEDURE [
      cred: Authenticator.Credentials, ver: Authenticator.Verifier, domain: CH.Name,
      acl: ACLFlavor, eachElement: CH.NameStreamProc] RETURNS [rc:
CH.ReturnCode];
  LookupECS: --CHLookup-- PROCEDURE [
      name: NSName.Name, clientProc: PROCEDURE [fullName: NSName.Name, info:
ECSPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupFileserver: --CHLookup-- PROCEDURE [
      name: NSName.Name,
      clientProc: PROCEDURE [fullName: NSName.Name, info: FileserverPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupGroupProperty: --CH-- PROCEDURE [
      cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Pattern,
      pn: PropertyID, eachElement: NameStreamProc, distingName: Name]
      RETURNS [rc: ReturnCode];
  LookupGWS: --CHLookup-- PROCEDURE [
      name: NSName.Name, clientProc: PROCEDURE [fullName: NSName.Name, info:
GWSPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupIBMHost: --CHLookup-- PROCEDURE [
      name: NSName.Name,
      clientProc: PROCEDURE [fullName: NSName.Name, info: IBMHostPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupIRS: --CHLookup-- PROCEDURE [
      name: NSName.Name, clientProc: PROCEDURE [fullName: NSName.Name, info:
IRSPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
```

```
  LookupITS: --CHLookup-- PROCEDURE [
      name: NSName.Name, clientProc: PROCEDURE [fullName: NSName.Name, info:
  ITSPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupMailserver: --CHLookup-- PROCEDURE [
      name: NSName.Name,
      clientProc: PROCEDURE [fullName: NSName.Name, info: MailserverPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupOldIBM3270Host: --CHLookup-- PROCEDURE [
      name: NSName.Name,
      clientProc: PROCEDURE [fullName: NSName.Name, info: OldIBM3270HostPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupOrgAccess: --MoreCH-- PROCEDURE [
      cred: Authenticator.Credentials, ver: Authenticator.Verifier, org: CH.Name,
      acl: ACLFlavor, eachElement: CH.NameStreamProc] RETURNS [rc:
  CH.ReturnCode];
  LookupPrintserver: --CHLookup-- PROCEDURE [
      name: NSName.Name,
      clientProc: PROCEDURE [fullName: NSName.Name, info: PrintserverPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupPropertyAccess: --MoreCH-- PROCEDURE [
      cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: CH.Name,
      pn: CH.PropertyID, acl: ACLFlavor, eachElement: CH.NameStreamProc,
      distingName: CH.Name] RETURNS [rc: CH.ReturnCode];
  LookupRemote: --CHLookup-- PROCEDURE [
      name: NSName.Name,
      clientProc: PROCEDURE [fullName: NSName.Name, info: RemotePt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookUpRootFile: --Volume-- PROCEDURE [type: File.Type, volume: ID ← systemID]
      RETURNS [file: File.File];
  LookupRS232CPort: --CHLookup-- PROCEDURE [
      name: NSName.Name,
      clientProc: PROCEDURE [fullName: NSName.Name, info: RS232CPortPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupUser: --CHLookup-- PROCEDURE [
      name: NSName.Name,
      clientProc: PROCEDURE [fullName: NSName.Name, info: UserPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LookupValueProperty: --CH-- PROCEDURE [
      cred: Authenticator.Credentials, ver: Authenticator.Verifier, name: Pattern,
      pn: PropertyID, buffer: Buffer, distingName: Name] RETURNS [rc:
  ReturnCode];
  LookupWorkstation: --CHLookup-- PROCEDURE [
      name: NSName.Name,
      clientProc: PROCEDURE [fullName: NSName.Name, info: WorkstationPt],
      credentials: Authenticator.Credentials ← Authenticator.nullCredentials,
      verifier: Authenticator.Verifier ← Authenticator.firstVerifier];
  LowByte: --Inline-- PROCEDURE [u: UNSPECIFIED] RETURNS [UNSPECIFIED];
  LowerCase: --NSString-- PROCEDURE [c: Character] RETURNS [Character];
```

```
  lowestVersion: --NSFile-- CARDINAL = 0;
  LowHalf: --Inline-- PROCEDURE [u: LONG UNSPECIFIED] RETURNS [UNSPECIFIED];
  LSAdjust: --OnlineDiagnostics-- PROCEDURE [
      cancelSignal: SIGNAL, GetMesaChar: PROCEDURE RETURNS [CHARACTER],
      PutCR: PROCEDURE,
      PutMessage: PROCEDURE [message: LSMessage, char: CHARACTER ← 0C],
      PutMesaChar: PROCEDURE [char: CHARACTER]];
  LSMessage: --OnlineDiagnostics-- TYPE = {
      kTermAdj, kTypeCharFill, kCTLC, kFillScreen, kTypeXHair, kEndAdj,
kTermTest,
      kTestKey, kCTLStop, kLineFeed, kReturnKey, kLetter, kAndCTL, kEscape,
kSpBar,
      kAndShift, kShColon, kShSemiColon, kTypeComma, kHyphen, kTypePeriod,
kVirgule,
      kNumeral, kKey, kLearColon, kSemiColon, kShComma, kShHyphen,
kShPeriod,
      kShVirgule, kAtSign, kLeftBracket, kBackSlash, kRightBracket, kCaret,
kBreak,
      kShAt, kShLeftBracket, kShBackSlash, kShRightBracket, kShCaret, kShBreak,
      kUnknown};
  LSTest: --OnlineDiagnostics-- PROCEDURE [
      cancelSignal: SIGNAL, GetMesaChar: PROCEDURE RETURNS [CHARACTER],
      PutMessage: PROCEDURE [message: LSMessage, char: CHARACTER ← 0C]];
  M1: --KeyStations-- Bit = 13;
  M2: --KeyStations-- Bit = 15;
  M3: --KeyStations-- Bit = 14;
  MailboxState: --RetrieveDefs-- PROCEDURE [handle: Handle]
      RETURNS [state: MBXState];
  mailCourierSocket: --CHPIDs-- CH.PropertyID = 35;
  mailForwardSocket: --CHPIDs-- CH.PropertyID = 32;
  mailPollSocket: --CHPIDs-- CH.PropertyID = 33;
  mailPrimary: --CHPIDs-- CH.PropertyID = 30;
  mailSecondary: --CHPIDs-- CH.PropertyID = 31;
  Mailserver: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
      address(0:0..95): System.NetworkAddress, location(6:0..63): NSString.String];
  MailserverDescribe: --CHLookup-- Courier.Description;
  MailserverPt: --CHLookup-- TYPE = LONG POINTER TO Mailserver;
  mailTelnetSocket: --CHPIDs-- CH.PropertyID = 34;
  MainBodyIndex: --PrincOps-- CARDINAL = 0;
  mainBodyIndex: --PrincOps-- CARDINAL = 0;
  Make: --BTree-- PROCEDURE [
      file: MFile.Handle ← NIL, usage: Space.Usage ← 0,
      valueSize: ValueSize ← defaultValueSize, reset: BOOLEAN ← FALSE]
      RETURNS [tree: Tree];
  Make: --Menu-- PROCEDURE [
      name: LONG STRING, strings: LONG DESCRIPTOR FOR ARRAY CARDINAL OF LONG STRING,
      mcrProc: MCRType, copyStrings: BOOLEAN ← TRUE, permanent: BOOLEAN ←
FALSE]
      RETURNS [Handle];
  MakeAbortedHeader: --ExpeditedCourier-- PROCEDURE [remoteSignalNumber:
CARDINAL]
      RETURNS [h: Header];
  MakeAtom: --Atom-- PROCEDURE [ref: LONG STRING] RETURNS [ATOM];
  MakeBootable: --OthelloOps-- PROCEDURE [
      file: File.File, type: BootFileType, firstPage:.File.PageNumber];
```

```
  MakeClientSW: --Tool-- PROCEDURE [
      window: Window.Handle,
      clientProc: PROCEDURE [sw: Window.Handle, clientData: LONG POINTER],
      clientData: LONG POINTER, swType: SWType, h: INTEGER ← 0]
      RETURNS [sw: Window.Handle];
  makeCodeOnePageSwapUnits: --PilotSwitchesExtras-- PilotSwitches.PilotDomainC
=
      371C;
  MakeDefaultSWs: --Tool-- PROCEDURE [
      window: Window.Handle, messageLines: CARDINAL ← 0,
      formProc: FormSW.ClientItemsProcType ← NIL,
      formHeight: CARDINAL ← DefaultHeight, logName: LONG STRING ← NIL]
      RETURNS [msgSW: Window.Handle, formSW: Window.Handle, logSW:
Window.Handle];
  MakeEditable: --FileSW-- PROCEDURE [sw: Window.Handle] RETURNS [ok: BOOLEAN];
  MakeFileList: --Scavenger-- PROCEDURE [
      volume: Volume.ID, logDestination: Volume.ID] RETURNS [logFile: File.File];
  MakeFileSW: --Tool-- PROCEDURE [
      window: Window.Handle, name: LONG STRING, access: FileSW.Access ← append,
      h: INTEGER ← DefaultHeight, allowTypeIn: BOOLEAN ← TRUE,
      resetLengthOnNewSession: BOOLEAN ← FALSE,
      resetLengthOnActivate: BOOLEAN ← FALSE] RETURNS [sw: Window.Handle];
  MakeFormSW: --Tool-- PROCEDURE [
      window: Window.Handle, formProc: FormSW.ClientItemsProcType,
      options: FormSW.Options ← [], h: INTEGER ← DefaultHeight,
      zone: UNCOUNTED ZONE ← NIL] RETURNS [sw: Window.Handle];
  MakeHeader: --Answer-- PROCEDURE [
      getChar: PROCEDURE [CARDINAL] RETURNS [CHARACTER], getLength: CARDINAL,
      putBlock: PROCEDURE [Environment.Block],
      getPages: PROCEDURE [CARDINAL] RETURNS [LONG POINTER],
      freePages: PROCEDURE [LONG POINTER], userName: LONG STRING,
      userRegistry: LONG STRING,
      arpaGatewayHostNames: DESCRIPTOR FOR ARRAY CARDINAL OF LONG STRING,
      cForCopies: BOOLEAN ← FALSE]
      RETURNS [
          answerError: BOOLEAN, mpCode: MailParse.ErrorCode, charPosition:
CARDINAL];
  MakeImage: --Floppy-- PROCEDURE [
      floppyDrive: CARDINAL ← 0, imageFile: File.File,
      firstImagePage: File.PageNumber];
  MakeItem: --Menu-- PROCEDURE [keyword: LONG STRING, mcrProc: MCRType]
      RETURNS [ItemObject];
  MakeMDSNode: --Heap-- PROCEDURE [z: MDSZone ← systemMDSZone, n:
NWords]
      RETURNS [p: POINTER];
  MakeMsgSW: --Tool-- PROCEDURE [
      window: Window.Handle, lines: CARDINAL ← 1, h: INTEGER ← DefaultHeight]
      RETURNS [sw: Window.Handle];
  MakeName: --NSName-- PROCEDURE [
      z: UNCOUNTED ZONE, orgSize: CARDINAL ← maxOrgLength,
      domainSize: CARDINAL ← maxDomainLength, localSize: CARDINAL ←
maxLocalLength]
      RETURNS [Name];
  MakeNameFields: --NSName-- PROCEDURE [
      z: UNCOUNTED ZONE, destination: Name, orgSize: CARDINAL ← maxOrgLength,
```

```
    domainSize: CARDINAL ← maxDomainLength, localSize: CARDINAL ←
  maxLocalLength];
  MakeNegative: --Cursor-- PROCEDURE;
  MakeNode: --Heap-- PROCEDURE [z: UNCOUNTED ZONE ← systemZone, n: NWords]
    RETURNS [p: LONG POINTER];
  MakeNode: --Zone-- PROCEDURE [
    zH: Handle, n: BlockSize, alignment: Alignment ← a1]
    RETURNS [node: Base RELATIVE POINTER, s: Status];
  MakePermanent: --File-- PROCEDURE [file: File];
  MakePositive: --Cursor-- PROCEDURE;
  MakeReadOnly: --MSegment-- PROCEDURE [segment: Handle];
  MakeReadOnly: --Space-- PROCEDURE [interval: Interval];
  MakeRhs: --CH-- PROCEDURE [maxlength: CARDINAL, heap: UNCOUNTED ZONE]
    RETURNS [rhs: Buffer];
  MakeSize: --ToolWindow-- PROCEDURE [window: Handle, size: Size];
  MakeString: --NSString-- PROCEDURE [z: UNCOUNTED ZONE, bytes: CARDINAL]
    RETURNS [String];
  MakeStringSW: --Tool-- PROCEDURE [
    window: Window.Handle, s: LONG POINTER TO LONG STRING ← NIL,
    access: TextSW.Access ← append, h: INTEGER ← DefaultHeight,
    expandable: BOOLEAN ← FALSE] RETURNS [sw: Window.Handle];
  MakeSWsProc: --Tool-- TYPE = PROCEDURE [window: Window.Handle];
  MakeTextSW: --Tool-- PROCEDURE [
    window: Window.Handle, source: TextSource.Handle, sink: TextSink.Handle ←
  NIL,
    options: TextSW.Options ← TextSW.defaultOptions,
    position: TextSource.Position ← 0, allowTypeIn: BOOLEAN ← TRUE]
    RETURNS [sw: Window.Handle];
  MakeTTYSW: --Tool-- PROCEDURE [
    window: Window.Handle, name: LONG STRING, h: INTEGER ← DefaultHeight,
    resetLengthOnNewSession: BOOLEAN ← FALSE] RETURNS [sw: Window.Handle];
  MakeUnbootable: --OthelloOps-- PROCEDURE [
    file: File.File, type: BootFileType, firstPage: File.PageNumber];
  MakeWritable: --MSegment-- PROCEDURE [segment: Handle];
  MakeWritable: --NSSegment-- PROCEDURE [
    interval: Space.Interval, file: NSFile.Handle, segment: ID ← defaultID,
    session: Session ← nullSession];
  MakeWritable: --Space-- PROCEDURE [interval: Interval];
  Map: --NSSegment-- PROCEDURE [
    origin: Origin, access: NSFile.Access ← NSFile.readAccess,
    usage: Space.Usage ← 0, life: Space.Life ← alive,
    swapUnits: Space.SwapUnitOption ← Space.defaultSwapUnitOption,
    session: Session ← nullSession] RETURNS [mapUnit: Space.Interval];
  Map: --Space-- PROCEDURE [
    window: Window, usage: Usage ← unknownUsage, class: Class ← file,
    access: Access ← readWrite, life: Life ← alive,
    swapUnits: SwapUnitOption ← defaultSwapUnitOption]
    RETURNS [mapUnit: Interval];
  MapAt: --NSSegment-- PROCEDURE [
    at: Space.Interval, origin: Origin, access: NSFile.Access ← NSFile.readAccess,
    usage: Space.Usage ← 0, life: Space.Life ← alive,
    swapUnits: Space.SwapUnitOption ← Space.defaultSwapUnitOption,
    session: Session ← nullSession] RETURNS [mapUnit: Space.Interval];
  MapAt: --Space-- PROCEDURE [
    at: Interval, window: Window, usage: Usage ← unknownUsage,
    class: Class ← file, access: Access ← readWrite, life: Life ← alive,
```

```
    swapUnits: SwapUnitOption ← defaultSwapUnitOption]
    RETURNS [mapUnit: Interval];
MarkItem: --FormSW-- PROCEDURE [
    sw: Window.Handle, index: CARDINAL, action: TextData.MarkingAction,
    mode: TextData.SelectionMode];
MarkPageBad: --PhysicalVolume-- PROCEDURE [pvID: ID, badPage: PageNumber];
MarkProcType: --Caret-- TYPE = PROCEDURE [data: ClientData, action: Action];
Mask: --Expand-- TYPE = RECORD [
    star: BOOLEAN,
    atSign: BOOLEAN,
    quote: BOOLEAN,
    upArrow: UpArrowAction,
    localDirectory: LONG STRING];
MatchPattern: --Exec-- PROCEDURE [string: LONG STRING, pattern: LONG STRING]
    RETURNS [matched: BOOLEAN];
MaxBands: --RavenFace-- CARDINAL = 8;
maxBlockLength: --PacketExchange-- READONLY CARDINAL;
maxBufferSize: --CH-- CARDINAL = 10000B;
maxCARDINAL: --Environment-- CARDINAL = 177777B;
maxCharactersInLabel: --Floppy-- CARDINAL = 40;
maxConnectLength: --BodyDefs-- CARDINAL = 64;
maxCourierDeserializeBufferLength: --CHLookup-- CARDINAL = 35;
maxData: --CommOnlineDiagnostics-- CARDINAL = 1000;
maxDomainLength: --NSName-- CARDINAL = 20;
maxDomainNameLength: --CH-- CARDINAL = 20;
maxDomainNameLength: --NSName-- CARDINAL = 20;
maxEntriesInRootDirectory: --Volume-- READONLY CARDINAL;
MaxFrameSize: --PrincOps-- CARDINAL = 7774B;
maxFrameSize: --PrincOps-- CARDINAL = 7774B;
maxFullNameLength: --NSName-- CARDINAL = 86;
maxINTEGER: --Environment-- INTEGER = 77777B;
maxlengthComment: --CHLookup-- CARDINAL = 100;
maxlengthDescription: --CHLookup-- CARDINAL = 100;
maxlengthLocation: --CHLookup-- CARDINAL = 100;
maxlengthPassword: --CHLookup-- CARDINAL = 40;
maxlengthProduct: --CHLookup-- CARDINAL = 40;
maxlengthTraining: --CHLookup-- CARDINAL = 40;
maxLocalLength: --NSName-- CARDINAL = 40;
maxLocalNameLength: --CH-- CARDINAL = 40;
maxLocalNameLength: --NSName-- CARDINAL = 40;
maxLONGCARDINAL: --Environment-- LONG CARDINAL = 37777777777B;
maxLONGINTEGER: --Environment-- LONG INTEGER = 17777777777B;
maxNameLength: --BTree-- CARDINAL = 100;
maxNameLength: --MFile-- CARDINAL = 100;
maxNameLength: --PhysicalVolume-- CARDINAL = 40;
maxNameLength: --Volume-- CARDINAL = 40;
MaxNLinks: --PrincOps-- CARDINAL = 255;
maxNLinks: --PrincOps-- CARDINAL = 255;
maxNumberOfSegments: --NSSegment-- READONLY CARDINAL;
maxOrgLength: --NSName-- CARDINAL = 20;
maxOrgNameLength: --CH-- CARDINAL = 20;
maxOrgNameLength: --NSName-- CARDINAL = 20;
maxPagesInMDS: --Environment-- CARDINAL = 256;
maxPagesInVM: --Environment-- PageCount = 77777777B;
maxPagesPerFile: --File-- LONG CARDINAL = 37777777B;
maxPagesPerVolume: --Volume-- LONG CARDINAL = 40000000B;
```

```
  maxParamsInStack: --PrincOps-- CARDINAL = 12;
  MaxParamsInStack: --PrincOps-- CARDINAL = 12;
  maxPkt: --ProtocolCertification-- CARDINAL = 576;
  maxRemarkLength: --BodyDefs-- CARDINAL = 64;
  maxRNameLength: --BodyDefs-- CARDINAL = 64;
  MaxSinglePrecision: --Real-- CARDINAL = 9;
  MaxSmallFrameIndex: --PrincOps-- CARDINAL = 17;
  maxSmallFrameIndex: --PrincOps-- CARDINAL = 17;
  maxStringLength: --NSFile-- CARDINAL = 100;
  maxStringLength: --Selection-- CARDINAL = 200;
  maxSubvolumesOnPhysicalVolume: --PhysicalVolume-- READONLY CARDINAL;
  maxWellKnownSocket: --NSConstants-- System.SocketNumber;
  MBXState: --RetrieveDefs-- TYPE = {
      unknown, badName, badPwd, cantAuth, userOK, allDown, someEmpty,
allEmpty,
      notEmpty};
  MCRForKeyword: --Menu-- PROCEDURE [
      sw: Window.Handle, menuName: LONG STRING, keyword: LONG STRING]
      RETURNS [mcr: MCRType, menu: Handle, index: CARDINAL];
  MCRType: --Menu-- TYPE = PROCEDURE [
      window: Window.Handle ← NIL, menu: Handle ← NIL, index: CARDINAL ←
177777B];
  MDS: --Space-- PROCEDURE RETURNS [Interval];
  MDSHandle: --Heap-- TYPE = MDSZone;
  MeasureBlock: --Display-- PROCEDURE [
      window: Handle, block: Environment.Block, lineLength: INTEGER ← infinity,
      place: Window.Place, font: WindowFont.Handle ← NIL]
      RETURNS [newPlace: Window.Place, positions: CARDINAL, why: BreakReason];
  Media: --NSPrint-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF Medium;
  Medium: --NSPrint-- TYPE = MACHINE DEPENDENT RECORD [
      var(0:0..63): SELECT type(0:0..15): MediumType FROM
          paper = > [paper(1:0..47): Paper], ENDCASE];
  mediumAnonymousBackingFile: --PilotSwitches-- AnonymousBackingFileSize =
174C;
  MediumIndex: --NSPrint-- TYPE = CARDINAL [0..0];
  MediumType: --NSPrint-- TYPE = MACHINE DEPENDENT{paper};
  MediumType: --ProtocolCertification-- TYPE = MACHINE DEPENDENT{
      ether, unspecified(15)};
  members: --CHPIDs-- CH.PropertyID = 3;
  MembershipProc: --NSSessionControl-- TYPE = PROCEDURE [
      key: NSString.String, type: NSFile.AccessEntryType, session: NSFile.Session]
      RETURNS [status: MembershipStatus];
  MembershipStatus: --NSSessionControl-- TYPE = {
      member, notAMember, cannotDetermine};
  MenuProcType: --FormSW-- TYPE = PROCEDURE [sw: Window.Handle, index:
CARDINAL]
      RETURNS [hints: Hints, freeHintsProc: FreeHintsProcType, replace: BOOLEAN];
  MergeAttributeLists: --NSFile-- PROCEDURE [
      listA: AttributeList, listB: AttributeList,
      suppressDuplicates: BOOLEAN ← FALSE] RETURNS [mergedList: AttributeList];
  MesaDEFileType: --FileTypes-- TYPE = CARDINAL [22100B..22177B];
  MesaFileType: --FileTypes-- TYPE = CARDINAL [256..511];
  MesaString: --NSString-- TYPE = LONG STRING;
  MesaUsage: --SpaceUsage-- TYPE = Space .Usage[128..255];
```

```
MessageProc: --FileTransfer-- TYPE = PROCEDURE [
    clientData: LONG POINTER, level: Severity, s1: LONG STRING ← NIL,
    s2: LONG STRING ← NIL, s3: LONG STRING ← NIL, s4: LONG STRING ← NIL];
MicrocodeInstallFailure: --FormatPilotDisk-- SIGNAL [m: FailureType];
Milliseconds: --Process-- TYPE = CARDINAL;
MinHeight: --FormSW-- PROCEDURE [items: ItemDescriptor, type: Type]
    RETURNS [CARDINAL];
minimumNodeSize: --Heap-- READONLY NWords;
minimumNodeSize: --Zone-- READONLY BlockSize;
minINTEGER: --Environment-- INTEGER = -32768;
minLength: --MailParse-- CARDINAL = 40;
minLONGINTEGER: --Environment-- LONG INTEGER = -2147483648;
minPagesPerVolume: --Volume-- READONLY PageCount;
minPkt: --ProtocolCertification-- CARDINAL = 30;
MinusInfinity: --Real-- REAL;
MinusLandBitmapUnder: --Window-- TYPE [6];
MinusLandCookieCutter: --Window-- TYPE [2];
MinusZero: --Real-- REAL;
MissingPages: --File-- ERROR [
    file: File, firstMissing: PageNumber, countMissing: PageCount];
ModemChange: --CommOnlineDiagnostics-- TYPE = PROCEDURE [
    modemSignal: ModemSignal, state: BOOLEAN];
ModemSignal: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT{
    dataSetReady, clearToSend, carrierDetect, ringIndicator, ringHeard};
modifiedBy: --NSAssignedTypes-- AttributeType = 7;
modifiedOn: --NSAssignedTypes-- AttributeType = 8;
ModifyBoolean: --FormSW-- PROCEDURE [
    sw: Window.Handle, index: CARDINAL, mark: BOOLEAN, notify: BOOLEAN];
ModifyCommand: --FormSW-- PROCEDURE [
    sw: Window.Handle, index: CARDINAL, mark: BOOLEAN, notify: BOOLEAN];
ModifyEditable: --FormSW-- PROCEDURE [
    sw: Window.Handle, index: CARDINAL, position: CARDINAL, length: CARDINAL,
    new: LONG STRING ← NIL, keepTrash: BOOLEAN ← FALSE];
ModifyEnumerated: --FormSW-- PROCEDURE [
    sw: Window.Handle, index: CARDINAL, mark: BOOLEAN, notify: BOOLEAN,
    newValue: UNSPECIFIED];
mouse: --UserTerminal-- READONLY LONG POINTER TO READONLY Coordinate;
MouseTransformerProc: --Window-- TYPE = PROCEDURE [Handle, Place]
    RETURNS [Handle, Place];
Move: --NSFile-- PROCEDURE [
    file: Handle, destination: Handle,
    attributes: AttributeList ← nullAttributeList,
    session: Session ← nullSession];
Move: --NSSegment-- PROCEDURE [
    file: NSFile.Handle, oldSegment: ID, newSegment: ID,
    session: Session ← nullSession];
MoveByName: --NSFile-- PROCEDURE [
    directory: Handle, path: String, destination: Handle,
    attributes: AttributeList ← nullAttributeList,
    session: Session ← nullSession];
MoveChild: --NSFile-- PROCEDURE [
    directory: Handle, id: ID, destination: Handle,
    attributes: AttributeList ← nullAttributeList,
    session: Session ← nullSession];
MoveIntoWindow: --Cursor-- PROCEDURE [
    window: Window.Handle, place: Window.Place];
```

```
MsecToTicks: --Process-- PROCEDURE [msec: Milliseconds] RETURNS [ticks: Ticks];
multiL1: --ProtocolCertification-- Stage;
MultipleFrames: --DebugUsefulDefs-- ERROR [list: FrameDesc];
MultiplyInfinityNaN: --Real-- LONG CARDINAL = 4;
myAddress: --NSAddr-- READONLY Address;
myAddressBuffer: --NSAddr-- READONLY CH.Buffer;
myHost: --NSAddr-- READONLY System.HostNumber;
myNSAddr: --NSAddr-- READONLY NSAddr;
myNSAddrBuffer: --NSAddr-- READONLY CH.Buffer;
Name: --Authenticator-- TYPE = NSName.Name;
Name: --CH-- TYPE = NSName.Name;
Name: --DebugUsefulDefs-- PROCEDURE [name: LONG STRING, gf: GFHandle];
name: --NSAssignedTypes-- AttributeType = 9;
Name: --NSName-- TYPE = LONG POINTER TO NameRecord;
NameFieldsFromString: --NSName-- PROCEDURE [
    z: UNCOUNTED ZONE, s: String, destination: Name, clientDefaults: Name ← NIL];
NameForError: --MFile-- SIGNAL RETURNS [errorName: LONG STRING];
NameFromString: --NSName-- PROCEDURE [
    z: UNCOUNTED ZONE, s: String, clientDefaults: Name ← NIL] RETURNS [Name];
NameInfo: --MailParse-- TYPE = RECORD [nesting: BracketType, type: NameType];
NameList: --MailParse-- PROCEDURE [
    h: Handle, process: ProcessProc, write: WriteProc ← NIL];
NamePattern: --CH-- TYPE = ThreePartName;
NameRecord: --CH-- TYPE = NSName.NameRecord;
NameRecord: --NSName-- TYPE = RECORD [
    org: Organization, domain: Domain, local: Local];
NameStreamProc: --CH-- TYPE = PROCEDURE [currentName: Element];
NameTooSmall: --NSName-- SIGNAL [
    oldName: Name, orgLenNeeded: CARDINAL, domainLenNeeded: CARDINAL,
    localLenNeeded: CARDINAL] RETURNS [newName: Name];
NameType: --CH-- TYPE = MACHINE DEPENDENT{notFound, found, dead, (177777B)};
NameType: --MailParse-- TYPE = {normal, file, publicDL};
NarrowFault: --Runtime-- ERROR;
NeededHeight: --FormSW-- PROCEDURE [Window.Handle]
    RETURNS [min: CARDINAL, current: CARDINAL];
NeedsScavenging: --PhysicalVolume-- ERROR;
NeedsScavenging: --Volume-- ERROR [volume: ID];
NetAccess: --RS232C-- TYPE = RS232CEnvironment.NetAccess;
NetAccess: --RS232CEnvironment-- TYPE = {directConn, dialConn};
NetFormat: --Format-- TYPE = {octal, hex, productSoftware};
NetFormat: --Put-- TYPE = Format.NetFormat;
netManagementSocket: --NSConstants-- System.SocketNumber;
network: --CHPIDs-- CH.PropertyID = 52;
NetworkAddress: --AddressTranslation-- TYPE = System.NetworkAddress;
NetworkAddress: --Format-- PROCEDURE [
    proc: StringProc, networkAddress: System.NetworkAddress, format: NetFormat,
    clientData: LONG POINTER ← NIL];
NetworkAddress: --Put-- PROCEDURE [
    h: Window.Handle ← NIL, address: System.NetworkAddress,
    format: NetFormat ← octal];
NetworkNonExistent: --Router-- ERROR;
NetworkNumber: --Format-- PROCEDURE [
    proc: StringProc, networkNumber: System.NetworkNumber, format:
```

```
NetFormat,
    clientData: LONG POINTER ← NIL];
 NetworkNumber: --Put-- PROCEDURE [
    h: Window.Handle ← NIL, networkNumber: System.NetworkNumber,
    format: NetFormat];
 networkServers: --CHPIDs-- CH.PropertyID = 53;
 newClearinghouseSocket: --NSConstants-- System.SocketNumber;
 NewConfig: --Runtime-- PROCEDURE [
    file: File.File, offset: File.PageCount, codeLinks: BOOLEAN ← FALSE];
 newLine: --FormSW-- Window.Place;
 NewLine: --TTY-- PROCEDURE [h: Handle] RETURNS [yes: BOOLEAN];
 NewLine: --TTYSW-- PROCEDURE [sw: Window.Handle] RETURNS [BOOLEAN];
 NewRadiusNotifyProc: --ExpeditedCourier-- TYPE = PROCEDURE [newRingRadius:
Hop]
    RETURNS [continue: BOOLEAN];
 newSearchPath: --EventTypes-- Supervisor.Event;
 newSession: --EventTypes-- Supervisor.Event;
 NewUser: --RetrieveDefs-- PROCEDURE [
    handle: Handle, user: BodyDefs.RName, password: LONG STRING];
 NextAction: --OnlineDiagnostics-- TYPE = {nextPattern, invertPattern, quit};
 NextItem: --CmFile-- PROCEDURE [h: Handle]
    RETURNS [name: LONG STRING, value: LONG STRING];
 nextLine: --FormSW-- INTEGER = -2;
 nextPlace: --FormSW-- Window.Place;
 NextServer: --RetrieveDefs-- PROCEDURE [handle: Handle]
    RETURNS [noMore: BOOLEAN, state: ServerState, procs: AccessProcs];
 NextValue: --CmFile-- PROCEDURE [h: Handle, table: StringLookUp.TableDesc]
    RETURNS [index: CARDINAL];
 nil: --Zone-- Base RELATIVE POINTER;
 noAccess: --NSFile-- Access;
 NoBackingFile: --TTY-- ERROR;
 noChange: --Profile-- LONG STRING;
 noControlSelections: --NSFile-- ControlSelections;
 Node: --MDSStorage-- PROCEDURE [nwords: CARDINAL] RETURNS [p: POINTER];
 Node: --Storage-- PROCEDURE [nwords: CARDINAL] RETURNS [p: LONG POINTER];
 NoDefaultInstance: --TTY-- ERROR;
 NodeSize: --Zone-- PROCEDURE [p: LONG POINTER] RETURNS [n: BlockSize];
 noEthernet: --PilotSwitches-- PilotDomainA = 76C;
 noEthernetOne: --PilotSwitches-- PilotDomainA = 74C;
 NoExceptions: --Real-- ExceptionFlags;
 noExtendedSelections: --NSFile-- ExtendedSelections;
 noInterpretedSelections: --NSFile-- InterpretedSelections;
 noMatch: --CmFile-- CARDINAL = 177777B;
 noneDeleted: --Scavenger-- BootFileArray;
 NonTrappingNaN: --Real-- REAL;
 Nop: --FloppyChannel-- PROCEDURE [handle: Handle] RETURNS [status: Status];
 NopCaretProc: --UserInput-- CaretProcType;
 nopCmd1: --BandBLT-- CARDINAL = 14;
 nopCmd2: --BandBLT-- CARDINAL = 15;
 NopDestroyProc: --Context-- DestroyProcType;
 NopEnumeratedNotifyProc: --FormSW-- EnumeratedNotifyProcType;
 NopFreeHintsProc: --FormSW-- FreeHintsProcType;
 NopLongNumberNotifyProc: --FormSW-- LongNumberNotifyProcType;
 NopMarkerProc: --Caret-- MarkProcType;
 NopNotifyProc: --FormSW-- NotifyProcType;
 NopNumberNotifyProc: --FormSW-- NumberNotifyProcType;
```

```
NopReadOnlyProc: --FormSW-- ReadOnlyProcType;
noProblems: --PhysicalVolume-- ScavengerStatus;
NopStringProc: --UserInput-- StringProcType;
noRetries: --FormatPilotDisk-- RetryLimit = 0;
NormalizeVFN: --FileName-- PROCEDURE [vfn: VFN];
normalReturnHeader: --ExpeditedCourier-- Header;
NoRS232CHardware: --RS232C-- ERROR;
noSelections: --NSFile-- READONLY Selections;
noStartCommunication: --PilotSwitches-- PilotDomainA = 75C;
noSuchCharacter: --UserInput-- CHARACTER = 377C;
NoSuchProcedureNumber: --Courier-- ERROR;
NoTableEntryForNet: --Router-- ERROR;
NotAFault: --Backstop-- ERROR;
NotAPilotDisk: --FormatPilotDisk-- ERROR;
NoteArrayDescriptor: --Courier-- TYPE = PROCEDURE [
    site: LONG POINTER, elementSize: CARDINAL, upperBound: CARDINAL];
NoteBlock: --Courier-- TYPE = PROCEDURE [block: Environment.Block];
NoteChoice: --Courier-- TYPE = PROCEDURE [
    site: LONG POINTER, size: CARDINAL,
    variant: LONG DESCRIPTOR FOR ARRAY CARDINAL OF CARDINAL,
    tag: LONG POINTER ← NIL];
NoteData: --ToolDriver-- NoteDataProcType;
NoteDataProcType: --ToolDriver-- TYPE = PROCEDURE [
    toolID: ToolID, data: LONG POINTER];
NoteDeadSpace: --Courier-- TYPE = PROCEDURE [
    site: LONG POINTER, size: CARDINAL];
NoteDisjointData: --Courier-- TYPE = PROCEDURE [
    site: LONG POINTER TO LONG POINTER, description: Description];
NoteLongCardinal: --Courier-- TYPE = PROCEDURE [
    site: LONG POINTER TO LONG CARDINAL];
NoteLongInteger: --Courier-- TYPE = PROCEDURE [
    site: LONG POINTER TO LONG INTEGER];
NoteParameters: --Courier-- TYPE = PROCEDURE [
    site: LONG POINTER, description: Description];
NotErrorEntry: --BackstopNub-- ERROR;
Notes: --Courier-- TYPE = POINTER TO READONLY NotesObject;
Notes: --Date-- TYPE = {
    normal, noZone, zoneGuessed, noTime, timeAndZoneGuessed};
NoteSize: --Courier-- TYPE = PROCEDURE [size: CARDINAL]
    RETURNS [location: LONG POINTER];
NotesObject: --Courier-- TYPE = RECORD [
    zone: UNCOUNTED ZONE,
    operation: {fetch, store, free},
    noteSize: NoteSize,
    noteLongCardinal: NoteLongCardinal,
    noteLongInteger: NoteLongInteger,
    noteParameters: NoteParameters,
    noteChoice: NoteChoice,
    noteDeadSpace: NoteDeadSpace,
    noteString: NoteString,
    noteSpace: NoteSpace,
    noteArrayDescriptor: NoteArrayDescriptor,
    noteDisjointData: NoteDisjointData,
    noteBlock: NoteBlock];
NoteSpace: --Courier-- TYPE = PROCEDURE [site: LONG POINTER, size: CARDINAL];
NoteString: --Courier-- TYPE = PROCEDURE [site: LONG POINTER TO LONG STRING];
```

```
NoteSWs: --ToolDriver-- NoteSWsProcType;
NoteSWsProcType: --ToolDriver-- TYPE = PROCEDURE [
    tool: LONG STRING, subwindows: AddressDescriptor];
NotFound: --DebugUsefulDefs-- ERROR [s: LONG STRING];
NotifyProc: --MFile-- TYPE = PROCEDURE [
    name: LONG STRING, file: Handle, clientInstanceData: LONG POINTER]
    RETURNS [removeNotifyProc: BOOLEAN ← FALSE];
NotifyProcType: --FormSW-- TYPE = ProcType;
NotLoggingError: --Backstop-- ERROR;
NotOnline: --Volume-- ERROR [volume: ID];
NotOpen: --Volume-- ERROR [volume: ID];
NoTTYPortHardware: --TTYPort-- ERROR;
notUsable: --CH-- PropertyID = 37777777777B;
notUsable: --CHPIDs-- CH.PropertyID = 37777777777B;
NSAddr: --NSAddr-- TYPE = LONG POINTER TO NSAddrObject;
nsAddress: --CHPIDs-- CH.PropertyID = 4;
NSAddrObject: --NSAddr-- TYPE = MACHINE DEPENDENT RECORD [
    host(0:0..47): System.HostNumber,
    socket(3:0..15): System.SocketNumber,
    nets(4:0..47): LONG DESCRIPTOR FOR ARRAY CARDINAL OF System.NetworkNumber];
NSAddrToRhs: --NSAddr-- PROCEDURE [nsAddr: NSAddr] RETURNS [rhs: CH.Buffer];
nsProtocol: --RS232CCorrespondents--
RS232CEnvironment.AutoRecognitionOutcome;
nsSystemElement: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
nsSystemElementBSC: --RS232CCorrespondents--
RS232CEnvironment.Correspondent;
NUL: --Ascii-- CHARACTER = 0C;
null: --DeviceTypes-- Device.Type;
nullAttributeList: --NSFile-- AttributeList;
nullBadPage: --PhysicalVolume-- PageNumber = 37777777777B;
nullBlock: --Environment-- Block;
nullBootFilePointer: --Floppy-- BootFilePointer;
nullBox: --ToolWindow-- Box;
nullBox: --Window-- Box;
nullChannelHandle: --TTYPort-- ChannelHandle;
nullChecksum: --Checksum-- CARDINAL = 177777B;
nullCredentials: --Authenticator-- Credentials;
nullCredentials: --NSName-- Credentials;
nullDeviceIndex: --PhysicalVolume-- CARDINAL = 177777B;
nullDrive: --FloppyChannel-- Drive = 177777B;
nullEnumeratedValue: --FormSW-- UNSPECIFIED = 177777B;
nullExchangeHandle: --PacketExchange-- READONLY ExchangeHandle;
nullFile: --File-- File;
nullFileID: --Floppy-- FileID;
nullFilter: --NSFile-- Filter;
nullFrame: --Backstop-- READONLY Frame;
nullGlobalFrame: --PrincOps-- GlobalFrameHandle;
NullGlobalFrame: --PrincOps-- GlobalFrameHandle;
nullHandle: --NSFile-- Handle;
nullHandle: --TTY-- Handle;
nullHandle: --Zone-- Handle;
nullHashedPassword: --NSName-- HashedPassword = 0;
nullID: --File-- ID;
nullID: --NSFile-- ID;
nullID: --NSSegment-- ID = 177777B;
nullID: --PhysicalVolume-- ID;
```

```
nullID: --Volume-- ID;
nullIDRepresentation: --NSFile-- ARRAY [0..4] OF UNSPECIFIED;
nullIndex: --Floppy-- CARDINAL = 177777B;
nullIndex: --FormSW-- CARDINAL = 177777B;
nullIndex: --Log-- Index = 0;
nullInterval: --Space-- Interval;
nullItems: --FormSW-- ItemDescriptor;
nullLineNumber: --RS232C-- CARDINAL = 177777B;
nullLineNumber: --RS232CEnvironment-- CARDINAL = 177777B;
nullLink: --PrincOps-- ControlLink;
NullLink: --PrincOps-- ControlLink;
NullLocalFrame: --PrincOps-- LocalFrameHandle;
nullLocalFrame: --PrincOps-- LocalFrameHandle;
nullOldFileID: --VolumeConversion-- OldFileID;
nullOrdering: --NSFile-- extended Ordering;
nullParameters: --Courier-- Parameters;
nullPeriodicNotify: --UserInput-- PeriodicNotifyHandle;
nullProcess: --Backstop-- READONLY Process;
nullProgram: --Runtime-- PROGRAM;
NullProgram: --Runtime-- PROGRAM;
nullSegment: --Zone-- SegmentHandle;
nullSession: --NSFile-- Session;
nullSession: --NSSegment-- Session;
nullString: --NSFile-- String;
nullString: --NSString-- String;
nullSubtreeSizeLimit: --NSFile-- LONG CARDINAL = 377777777777B;
nullSubVolume: --OthelloOps-- SubVolume;
nullSystemElement: --NSFile-- SystemElement;
nullSystemElementRepresentation: --NSFile-- ARRAY [0..5] OF UNSPECIFIED;
nullTime: --NSFile-- Time;
nullType: --Device-- Type;
nullVerifier: --NSName-- Verifier;
nullVolume: --NSFile-- Volume;
nullVolumeHandle: --Floppy-- READONLY VolumeHandle;
Number: --Dialup-- TYPE = RECORD [
    number: PACKED SEQUENCE n: CARDINAL OF Environment.Byte];
Number: --Format-- PROCEDURE [
    proc: StringProc, n: UNSPECIFIED, format: NumberFormat,
    clientData: LONG POINTER ← NIL];
Number: --Put-- PROCEDURE [
    h: Window.Handle ← NIL, n: UNSPECIFIED, format: Format.NumberFormat];
Number: --Selection-- PROCEDURE [radix: CARDINAL ← 10] RETURNS [CARDINAL];
NumberFormat: --Format-- TYPE = RECORD [
    base: [2..36] ← 12,
    zerofill: BOOLEAN ← FALSE,
    unsigned: BOOLEAN ← TRUE,
    columns: [0..255] ← 0];
NumberFormat: --TTY-- TYPE = Format.NumberFormat;
NumberHandle: --FormSW-- TYPE = LONG POINTER TO number ItemObject;
NumberItem: --FormSW-- PROCEDURE [
    tag: LONG STRING ← NIL, readOnly: BOOLEAN ← FALSE, invisible: BOOLEAN ← FALSE,
    drawBox: BOOLEAN ← FALSE, hasContext: BOOLEAN ← FALSE,
    place: Window.Place ← nextPlace, signed: BOOLEAN ← TRUE,
    notNegative: BOOLEAN ← FALSE, radix: Radix ← decimal,
    boxWidth: CARDINAL [0..127] ← 64,
    proc: NumberNotifyProcType ← NopNumberNotifyProc,
```

```
        default: UNSPECIFIED ← 77777B, value: LONG POINTER, bias: INTEGER ← 0,
        z: UNCOUNTED ZONE ← NIL] RETURNS [NumberHandle];
    NumberNotifyProcType: --FormSW-- TYPE = PROCEDURE [
        sw: Window.Handle ← NIL, item: ItemHandle ← NIL, index: CARDINAL ←
    nullIndex,
        oldValue: UNSPECIFIED ← 77777B];
    numberOfChildren: --NSAssignedTypes-- AttributeType = 10;
    NumberOfSegments: --NSSegment-- PROCEDURE [
        file: NSFile.Handle, session: Session ← nullSession] RETURNS [CARDINAL];
    NumberType: --Real-- TYPE = MACHINE DEPENDENT{normal, zero, infinity, nan};
    NWords: --Heap-- TYPE = [0..77775B];
    Object: --Courier-- TYPE = RECORD [
        remote: SystemElement,
        programNumber: LONG CARDINAL,
        versionNumber: CARDINAL,
        zone: UNCOUNTED ZONE,
        sH: Stream.Handle,
        classOfService: NetworkStream.ClassOfService];
    Object: --Cursor-- TYPE = RECORD [info: Info, array: UserTerminal.CursorArray];
    Object: --DebugUsefulDefs-- TYPE;
    Object: --Event-- TYPE;
    Object: --Exec-- TYPE;
    Object: --MailParse-- TYPE;
    Object: --Menu-- TYPE = RECORD [
        permanent: BOOLEAN,
        nInstances: CARDINAL [0..77777B],
        name: LONG STRING,
        items: Items];
    Object: --MFile-- TYPE;
    Object: --MLoader-- TYPE;
    Object: --MSegment-- TYPE;
    Object: --Window-- TYPE [19];
    Object: --WindowFont-- TYPE = RECORD [
        height: [0..7777B] ← NULL,
        kerned: BOOLEAN ← FALSE,
        width: PACKED ARRAY CHARACTER [0C..377C] OF [0..255] ← ALL[0],
        raster: CARDINAL ← NULL,
        maxWidth: CARDINAL ← NULL,
        min: CHARACTER ← NULL,
        max: CHARACTER ← NULL,
        address: LONG POINTER,
        bitmap: LONG POINTER TO ARRAY [0..0) OF WORD ← NULL,
        xInSegment: LONG POINTER TO ARRAY CHARACTER [0C..0C) OF CARDINAL ← NULL];
    ObscuredBySibling: --Window-- PROCEDURE [Handle] RETURNS [BOOLEAN];
    Octal: --Format-- PROCEDURE [
        proc: StringProc, n: UNSPECIFIED, clientData: LONG POINTER ← NIL];
    Octal: --Put-- PROCEDURE [h: Window.Handle ← NIL, n: UNSPECIFIED];
    OctalFormat: --Format-- NumberFormat;
    Offline: --LsepFace-- PROCEDURE;
```

```
Offline: --PhysicalVolume-- PROCEDURE [pvID: ID];
  OldControllerRecord: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
      controllerAddress(0:0..15): CARDINAL,
      portsOnController(1:0..15): CARDINAL,
      linkType(2:0..15): ControllerLinkType,
      path(3:0..63): NSString.String];
  oldestTime: --BodyDefs-- Timestamp;
  OldFileID: --VolumeConversion-- TYPE = System.UniversalID;
  OldIBM3270Host: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
      description(0:0..63): NSString.String,
      controllers(4:0..47): LONG DESCRIPTOR FOR ARRAY CARDINAL OF
          OldControllerRecord];
  OldIBM3270HostDescribe: --CHLookup-- Courier.Description;
  OldIBM3270HostPt: --CHLookup-- TYPE = LONG POINTER TO OldIBM3270Host;
  oneHour: --Authenticator-- Seconds = 7020B;
  Online: --LsepFace-- PROCEDURE;
  onlyEnumerateCurrentType: --Volume-- TypeSet;
  OnOff: --ToolWindow-- TYPE = {on, off};
  Open: --CmFile-- PROCEDURE [fileName: LONG STRING] RETURNS [h: Handle];
  Open: --Floppy-- PROCEDURE [drive: CARDINAL ← 0] RETURNS [volume: VolumeHandle];
  Open: --Log-- PROCEDURE [file: File.File, firstPageNumber: File.PageNumber ← 1];
  Open: --NSFile-- PROCEDURE [
      attributes: AttributeList, directory: Handle ← nullHandle,
      controls: Controls ← [], session: Session ← nullSession]
      RETURNS [file: Handle];
  Open: --NSVolumeControl-- PROCEDURE [volume: Volume.ID];
  Open: --Volume-- PROCEDURE [volume: ID];
  OpenByName: --NSFile-- PROCEDURE [
      directory: Handle, path: String, controls: Controls ← [],
      session: Session ← nullSession] RETURNS [Handle];
  OpenByReference: --NSFile-- PROCEDURE [
      reference: Reference, controls: Controls ← [], session: Session ← nullSession]
      RETURNS [file: Handle];
  OpenChild: --NSFile-- PROCEDURE [
      directory: Handle, id: ID, controls: Controls ← [],
      session: Session ← nullSession] RETURNS [Handle];
  OpenSink: --NSDataStream-- PROCEDURE [ticket: Ticket, cH: Courier.Handle]
      RETURNS [SinkStream];
  OpenSource: --NSDataStream-- PROCEDURE [ticket: Ticket, cH: Courier.Handle]
      RETURNS [SourceStream];
  OperateOnSink: --NSDataStream-- PROCEDURE [
      sink: Sink, operation: PROCEDURE [SinkStream]];
  OperateOnSource: --NSDataStream-- PROCEDURE [
      source: Source, operation: PROCEDURE [SourceStream]];
  Operation: --CHLookup-- TYPE = {get, put, free};
  OperationClass: --RS232C-- TYPE = {input, output, other, all};
  OperationType: --ProtocolCertification-- TYPE = {request, reply, reject, end};
  Options: --FileSW-- TYPE = TextSW.Options;
  Options: --FormSW-- TYPE = RECORD [
      type: Type ← fixed,
      boldTags: BOOLEAN ← TRUE,
      autoScroll: BOOLEAN ← TRUE,
      scrollVertical: BOOLEAN ← TRUE];
  Options: --MLoader-- TYPE = RECORD [codeLinks: BOOLEAN];
  Options: --ScratchSW-- TYPE = TextSW.Options;
  OpTrapTable: --PrincOps-- TYPE = POINTER TO ARRAY BYTE OF ControlLink;
```

```
ordering: --NSAssignedTypes-- AttributeType = 11;
Ordering: --NSFile-- TYPE = MACHINE DEPENDENT RECORD [
    var(0:0..79): SELECT type(0:0..15): OrderingType FROM
        key = > [
            key(1:0..15): AttributeType,
            ascending(3:0..15): BOOLEAN ← TRUE,
            dummy1(2:0..15): CARDINAL ← 0,
            dummy2(4:0..15): CARDINAL ← 0],
        extended = > [
            key(1:0..31): ExtendedAttributeType,
            ascending(3:0..15): BOOLEAN ← TRUE,
            interpretation(4:0..15): Interpretation ← none],
        ENDCASE];
OrderingType: --NSFile-- TYPE = MACHINE DEPENDENT{key, extended};
organization: --EventTypes-- Supervisor.Event;
Organization: --NSName-- TYPE = String ← NSString.nullString;
OrgName: --CH-- TYPE = NSName.Organization;
Origin: --NSSegment-- TYPE = RECORD [
    file: NSFile.Handle,
    base: PageNumber,
    count: PageCount,
    segment: ID ← defaultID];
Original: --DebugUsefulDefs-- PROCEDURE [new: GFHandle] RETURNS [old: GFHandle];
OrphanHandle: --Scavenger-- TYPE [2];
OtherCHProblem: --AddressTranslation-- ERROR [reason: Reason];
OtherEvents: --EventTypes-- TYPE = [700..799];
Outcome: --Dialup-- TYPE = {
    success, failure, aborted, formatError, transmissionError, dataLineOccupied,
    dialerNotPresent, dialingTimeout, transferTimeout};
Outcome: --Exec-- TYPE = MACHINE DEPENDENT{
    normal, warning, error, abort, spare1, spare2, spare3, last(177777B)};
OutOfInstances: --TTY-- ERROR;
OutputProc: --Exec-- PROCEDURE [h: Handle] RETURNS [proc: Format.StringProc];
outsideXeroxFirstSocket: --NSConstants-- System.SocketNumber;
outsideXeroxLastSocket: --NSConstants-- System.SocketNumber;
Overflow: --Log-- TYPE = MACHINE DEPENDENT{reset, disable, wrap};
OverLapOption: --ByteBlt-- TYPE = {ripple, move};
owner: --NSAssignedTypes-- AttributeType = 10377B;
OwnerChecking: --Heap-- PROCEDURE [z: UNCOUNTED ZONE] RETURNS [BOOLEAN];
OwnerCheckingMDS: --Heap-- PROCEDURE [z: MDSZone] RETURNS [BOOLEAN];
Packed: --Date-- TYPE = Time.Packed;
PackedTime: --BodyDefs-- TYPE = LONG CARDINAL;
PackedToString: --Date- PROCEDURE [Packed] RETURNS [LONG STRING];
PackFilename: --FileName-- PROCEDURE [
    vfn: VFN, h: BOOLEAN ← FALSE, d: BOOLEAN ← FALSE, n: BOOLEAN ← FALSE,
    v: BOOLEAN ← FALSE] RETURNS [s: LONG STRING];
PageCount: --Environment-- TYPE = LONG CARDINAL;
PageCount: --File-- TYPE = LONG CARDINAL;
PageCount: --Floppy-- TYPE = PageNumber;
PageCount: --NSSegment-- TYPE = LONG CARDINAL;
PageCount: --PhysicalVolume-- TYPE = LONG CARDINAL;
PageCount: --Space-- TYPE = Environment.PageCount;
PageCount: --Volume-- TYPE = LONG CARDINAL;
PageFromLongPointer: --Environment-- PROCEDURE [pointer: LONG POINTER]
    RETURNS [PageNumber];
```

```
  PageFromLongPointer: --Space-- PROCEDURE [pointer: LONG POINTER]
    RETURNS [Environment.PageNumber];
  PageNumber: --BandBLT-- TYPE = CARDINAL;
  PageNumber: --Environment-- TYPE = LONG CARDINAL;
  PageNumber: --File-- TYPE = LONG CARDINAL;
  PageNumber: --Floppy-- TYPE = LONG CARDINAL;
  PageNumber: --NSSegment-- TYPE = LONG CARDINAL;
  PageNumber: --PageScavenger-- TYPE = LONG CARDINAL;
  PageNumber: --PhysicalVolume-- TYPE = LONG CARDINAL;
  PageNumber: --Space-- TYPE = Environment.PageNumber;
  PageNumber: --Volume-- TYPE = LONG CARDINAL;
  PageOffset: --Environment-- TYPE = PageNumber;
  PageOffset: --Space-- TYPE = Environment.PageOffset;
  Pages: --MDSStorage-- PROCEDURE [npages: CARDINAL] RETURNS [base: POINTER];
  Pages: --MSegment-- PROCEDURE [segment: Handle] RETURNS [Environment.PageCount];
  Pages: --Storage-- PROCEDURE [npages: CARDINAL] RETURNS [base: LONG POINTER];
  PagesForImage: --Floppy-- PROCEDURE [floppyDrive: CARDINAL ← 0]
    RETURNS [File.PageCount];
  PagesForWords: --MDSStorage-- PROCEDURE [nWords: CARDINAL] RETURNS [CARDINAL];
  PagesForWords: --MSegment-- PROCEDURE [nWords: CARDINAL] RETURNS [CARDINAL];
  PagesForWords: --Storage-- PROCEDURE [nWords: CARDINAL] RETURNS [CARDINAL];
  PagesFromWords: --Space-- PROCEDURE [wordCount: LONG CARDINAL]
    RETURNS [pageCount: Environment.PageCount];
  PagesToPrint: --NSPrint-- TYPE = MACHINE DEPENDENT RECORD [
    beginningPageNumber(0:0..15): CARDINAL, endingPageNumber(1:0..15):
CARDINAL];
  paintFlags: --Display-- BitBltFlags;
  paintGrayFlags: --Display-- BitBltFlags;
  PairToReal: --Real-- PROCEDURE [fr: LONG INTEGER, exp10: INTEGER]
    RETURNS [REAL];
  Paper: --NSPrint-- TYPE = MACHINE DEPENDENT RECORD [
    var(0:0..47): SELECT type(0:0..15): PaperType FROM
      unknown = > NULL,
      knownSize = > [knownSize(1:0..15): PaperSize],
      otherSize = > [otherSize(1:0..31): PaperDimensions],
      ENDCASE];
  PaperDimensions: --NSPrint-- TYPE = MACHINE DEPENDENT RECORD [
    length(0:0..15): CARDINAL, width(1:0..15): CARDINAL];
  PaperIndex: --NSPrint-- TYPE = CARDINAL [0..2];
  PaperSize: --NSPrint-- TYPE = MACHINE DEPENDENT{
    dontUse, usLetter, usLegal, a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, isoB0,
    isoB1, isoB2, isoB3, isoB4, isoB5, isoB6, isoB7, isoB8, isoB9, isoB10, jisB0,
    jisB1, jisB2, jisB3, jisB4, jisB5, jisB6, jisB7, jisB8, jisB9, jisB10};
  PaperSource: --LsepFace-- TYPE = MACHINE DEPENDENT{auto, manual};
  PaperSource: --RavenFace-- TYPE = MACHINE DEPENDENT{bottom, top};
  PaperStacking: --RavenFace-- TYPE = MACHINE DEPENDENT{aligned, offset};
  PaperType: --NSPrint-- TYPE = MACHINE DEPENDENT{unknown, knownSize, otherSize};
  Parameter: --RS232C-- TYPE = RECORD [
    SELECT type: ParameterType FROM
    charLength = > [charLength: CharLength],
    correspondent = > [correspondent: Correspondent],
    dataTerminalReady = > [dataTerminalReady: BOOLEAN],
    echoing = > [echoing: BOOLEAN],
    flowControl = > [flowControl: FlowControl],
    frameTimeout = > [frameTimeout: CARDINAL],
    latchBitClear = > [latchBitClearMask: LatchBitClearMask],
```

```
        lineSpeed => [lineSpeed: LineSpeed],
        parity => [parity: Parity],
        requestToSend => [requestToSend: BOOLEAN],
        stopBits => [stopBits: StopBits],
        syncChar => [syncChar: SyncChar],
        syncCount => [syncCount: SyncCount],
        ENDCASE];
Parameter: --TTYPort-- TYPE = RECORD [
    SELECT parameter: * FROM
        breakDetectedClear => [breakDetectedClear: BOOLEAN],
        characterLength => [characterLength: CharacterLength],
        clearToSend => [clearToSend: BOOLEAN],
        dataSetReady => [dataSetReady: BOOLEAN],
        lineSpeed => [lineSpeed: LineSpeed],
        parity => [parity: Parity],
        stopBits => [stopBits: StopBits],
        ENDCASE];
ParameterGrouping: --CH-- TYPE = MACHINE DEPENDENT{first(1), second, (177777B)};
Parameters: --Courier-- TYPE = RECORD [
    location: LONG POINTER, description: Description];
ParameterType: --RS232C-- TYPE = {
    charLength, correspondent, dataTerminalReady, echoing, flowControl,
    frameTimeout, latchBitClear, lineSpeed, parity, requestToSend, stopBits,
    syncChar, syncCount};
parentID: --NSAssignedTypes-- AttributeType = 12;
Parity: --RS232C-- TYPE = RS232CEnvironment.Parity;
Parity: --RS232CEnvironment-- TYPE = {none, odd, even, one, zero};
Parity: --TTYPort-- TYPE = TTYPortEnvironment Parity;
Parity: --TTYPortEnvironment-- TYPE = {none, odd, even};
Password: --BodyDefs-- TYPE = ARRAY [0..3] OF CARDINAL;
password: --CHPIDs-- CH.PropertyID = 6;
PasswordStringToKey: --Authenticator-- PROCEDURE [
    flavor: Flavor ← superWeak, password: NSString.String]
    RETURNS [passwordKey: Key];
pathname: --NSAssignedTypes-- AttributeType = 21;
Pattern: --CH-- TYPE = LONG POINTER TO NamePattern;
PatternType: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT{
    zero, ones, oneZeroes, constant, byteIncr};
pause: --Dialup-- Environment.Byte = 255;
Pause: --LsepFace-- PROCEDURE;
Pause: --Process-- PROCEDURE [ticks: Ticks];
pauseStage: --ProtocolCertification-- Stage;
PC: --BackstopNub-- TYPE [1];
pcClientType: --ProtocolCertification-- PacketExchange.ExchangeClientType =
    protocolCertification;
pcControlSocket: --ProtocolCertification-- System.SocketNumber;
pcControlType: --ProtocolCertification-- NSTypes.PacketType = pccPacket;
pcOffset: --PrincOps-- CARDINAL = 1;
pcRoutingSocket: --ProtocolCertification-- System.SocketNumber;
pcTestSocket: --ProtocolCertification-- System.SocketNumber;
Percent: --NSVolumeControl-- TYPE = [0..100];
Percent: --Scrollbar-- TYPE = [0..100];
PerformanceToolFileType: --FileTypes-- TYPE = CARDINAL [22200B..22277B];
PerformanceToolFileType: --PerformanceToolFileTypes-- TYPE =
    FileTypes.PerformanceToolFileType;
PeriodicNotifyHandle: --UserInput-- TYPE [1];
```

```
PeriodicProcType: --UserInput-- TYPE = PROCEDURE [
    window: Window.Handle, place: Window.Place];
pexReplier: --ProtocolCertification-- Stage;
pexReplierThruput: --ProtocolCertification-- Stage;
pexRequestor: --ProtocolCertification-- Stage;
pexRequestorThruput: --ProtocolCertification-- Stage;
PhysicalMedium: --Router-- TYPE = {ethernet, ethernetOne, phonenet, clusternet};
PhysicalRecord: --RS232C-- TYPE = RS232CEnvironment.PhysicalRecord;
PhysicalRecord: --RS232CEnvironment-- TYPE = RECORD [
    header: Environment.Block,
    body: Environment.Block,
    trailer: Environment.Block];
PhysicalRecordHandle: --RS232C-- TYPE = RS232CEnvironment.PhysicalRecordHandle;
PhysicalRecordHandle: --RS232CEnvironment-- TYPE = POINTER TO PhysicalRecord;
PilotDisk: --Device-- TYPE = CARDINAL [64..1023];
PilotDomainA: --PilotSwitches-- TYPE = SwitchName [0C..100C];
PilotDomainB: --PilotSwitches-- TYPE = SwitchName [133C..140C];
PilotDomainC: --PilotSwitches-- TYPE = SwitchName [173C..377C];
PilotFileType: --FileTypes-- TYPE = CARDINAL [0..255];
PilotKernelUsage: --SpaceUsage-- TYPE = Space.Usage[0..63];
pixelsPerInch: --UserTerminal-- READONLY CARDINAL;
place: --Profile-- READONLY Place;
Place: --Profile-- TYPE = MACHINE DEPENDENT{
    unknown, tajo, copilot, last(177777B)};
Place: --Window-- TYPE = UserTerminal.Coordinate;
PleaseReleaseProc: --MFile-- TYPE = PROCEDURE [
    file: Handle, instanceData: LONG POINTER] RETURNS [ReleaseChoice];
PleaseReleaseProc: --MSegment-- TYPE = PROCEDURE [
    segment: Handle, instanceData: LONG POINTER] RETURNS [MFile.ReleaseChoice];
PleaseReleaseProc: --MStream-- TYPE = PROCEDURE [
    stream: Handle, instanceData: LONG POINTER] RETURNS [MFile.ReleaseChoice];
PlusInfinity: --Real-- REAL;
PlusZero: --Real-- REAL;
Point: --Display-- PROCEDURE [window: Handle, point: Window.Place];
Pointer: --Space-- PROCEDURE [pointer: LONG POINTER] RETURNS [POINTER];
PointerFault: --Runtime-- SIGNAL;
PointerFromPage: --Space-- PROCEDURE [page: Environment.PageNumber]
    RETURNS [POINTER];
PopAlternateInputStreams: --TTY-- PROCEDURE [h: Handle, howMany: CARDINAL ← 1];
PopAlternateInputStreams: --TTYSW-- PROCEDURE [
    sw: Window.Handle, howMany: CARDINAL ← 1];
Port: --GSort-- TYPE = MACHINE DEPENDENT RECORD [
    in(0:0..31): LONG UNSPECIFIED,
    out(2:0..31): PROCEDURE [
        GetProcType, PutProcType, CompareProcType, CARDINAL, CARDINAL, CARDINAL]];
Port: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
    representation = > [
        in(0:0..31): LONG UNSPECIFIED, out(2:0..31): LONG UNSPECIFIED],
    links = > [
        frame(0:0..15): LocalFrameHandle,
        fill(1:0..15): WORD,
        dest(2:0..31): ControlLink],
    ENDCASE];
PortClientType: --CHLookup-- TYPE = MACHINE DEPENDENT{
    unassigned, outOfService, its, irs, gws, ibm3270Host, ttyEmulation};
```

```
    PortDialerType: --CHLookup-- TYPE = MACHINE DEPENDENT{
        none, vadic, hayes, ventel};
    PortEchoingLocation: --CHLookup-- TYPE = MACHINE DEPENDENT{
        application, ciu, terminal};
    PortFault: --Runtime-- ERROR;
    PortHandle: --PrincOps-- TYPE = POINTER TO Port;
    PortRange: --CHLookup-- TYPE = CARDINAL [0..7];
    PortSyncType: --CHLookup-- TYPE = MACHINE DEPENDENT{
        asynchronous, synchronous, bitSynchronous, byteSynchronous, any};
    Position: --FileWindow-- PROCEDURE [sw: Window.Handle, position: LONG CARDINAL];
    position: --NSAssignedTypes-- AttributeType = 13;
    Position: --NSFile-- TYPE = Words;
    Position: --Stream-- TYPE = LONG CARDINAL;
    Post: --MsgSW-- PROCEDURE [
        sw: Window.Handle, string: LONG STRING, severity: Severity ← info,
        prefix: BOOLEAN ← TRUE, endOfMsg: BOOLEAN ← TRUE];
    PostAndLog: --MsgSW-- PROCEDURE [
        sw: Window.Handle, string: LONG STRING, severity: Severity ← info,
        prefix: BOOLEAN ← TRUE, endOfMsg: BOOLEAN ← TRUE, logSW: Window.Handle ←
    NIL];
    Power: --RealFns-- PROCEDURE [base: REAL, exponent: REAL] RETURNS [REAL];
    powerOff: --Event-- READONLY Supervisor.SubsystemHandle;
    powerOff: --EventTypes-- Supervisor.Event;
    PrefixHandle: --PrincOps-- TYPE = LONG BASE POINTER TO CodeSegment;
    PrefixHeader: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
        globalFsi(0:0..7): BYTE,
        nlinks(0:8..15): [0..255],
        stops(1:0..0): BOOLEAN,
        available(1:1..15): NAT,
        mainBodyPC(2:0..15): BytePC,
        catchCode(3:0..15): BytePC];
    PrependCommands: --Exec-- PROCEDURE [h: Handle, command: LONG STRING];
    primaryCredentials: --Event-- READONLY Supervisor.SubsystemHandle;
    primaryCredentials: --EventTypes-- Supervisor.Event;
    Print: --NSPrint-- PROCEDURE [
        master: NSDataStream.Source, printAttributes: PrintAttributes,
        printOptions: PrintOptions, systemElement: SystemElement]
        RETURNS [printRequestID: RequestID];
    PrintAttribute: --NSPrint-- TYPE = MACHINE DEPENDENT RECORD [
        var(0:0..79): SELECT type(0:0..15): PrintAttributeType FROM
            printObjectName = > [printObjectName(1:0..63): String ← [NIL, 0, 0]],
            printObjectCreateDate = > [printObjectCreateDate(1:0..31): Time ← 0],
            senderName = > [senderName(1:0..63): String ← [NIL, 0, 0]],
            ENDCASE];
    PrintAttributes: --NSPrint-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
        PrintAttribute;
    PrintAttributesIndex: --NSPrint-- TYPE = CARDINAL [0..2];
    PrintAttributeType: --NSPrint-- TYPE = MACHINE DEPENDENT{
        printObjectName, printObjectCreateDate, senderName};
    PrintCHReturnCode: --AddressTranslation-- PROCEDURE [
        rc: CH.ReturnCode, proc: Format.StringProc];
    Printer: --DebugUsefulDefs-- TYPE = PROCEDURE [Handle] RETURNS [BOOLEAN];
    Printer: --NSPrint-- TYPE = MACHINE DEPENDENT{
        available, busy, disabled, needsAttention, needsKeyOperator};
    PrinterProperties: --NSPrint-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
        PrinterProperty;
```

```
  PrinterPropertiesIndex: --NSPrint-- TYPE = CARDINAL [0..2];
  PrinterProperty: --NSPrint-- TYPE = MACHINE DEPENDENT RECORD [
      var(0:0..63): SELECT type(0:0..15): PrinterPropertyType FROM
          media = > [media(1:0..47): Media],
          staple = > [staple(1:0..15): BOOLEAN],
          twoSided = > [twoSided(1:0..15): BOOLEAN],
          ENDCASE];
  PrinterPropertyType: --NSPrint-- TYPE = MACHINE DEPENDENT{
      media, staple, twoSided};
  PrinterStatus: --LsepFace-- TYPE = MACHINE DEPENDENT{
      noStatus, oneMegaHz(16), halfMegaHz, (32), (33), (34), (35), (36), (37), (38),
      (39), (40), (41), (42), (43), (44), (45), (46), (47), (48), (49), (50), (51),
      (52), (53), (54), (55), (56), (57), (58), (59), (60), (61), keyPause,
      keyHomeFeed, warming, standBy, feederFault, noInkDonor, registrationJam,
(69),
      (70), (71), interlockOpen, (73), feeding, readyToFeed, (76), parityError,
      illegalCharacter, illegalSequence, (80), noPaper, pageSync, pageTailSync,
      (84), goingOffLine, offLine, onLine, (88), feedingOut, (90), pause(95), (96),
      paperA4, paperB4, paperB5, paperUnknown(103), (104), (124), statusError(126),
      statusOverRun};
  PrinterStatus: --NSPrint-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
      PrinterStatusComponent;
  PrinterStatus: --RavenFace-- TYPE = MACHINE DEPENDENT{
      noStatus, key0(48), key1, key2, key3, key4, key5, key6, key7, key8, key9,
      keyClear, keyTest, keyOnLine, keyOffLine, (62), (63), warming, standBy,
      feederFault, registrationJam(68), fuserJam, noExit, interlockOpen(72),
      fuserCold, feeding, readyToFeed, displayAcknowledge, parityError,
      illegalCharacter, illegalSequence, (80), noPaper, pageSync, pageAtOutputTray,
      tonerLow, goingOffLine, offLine, onLine, outputTrayFull, aboutToDozeOff,
      (124), statusError(126), statusOverRun};
  PrinterStatusComponent: --NSPrint-- TYPE = MACHINE DEPENDENT RECORD [
      var(0:0..63): SELECT type(0:0..15): PrinterStatusType FROM
          spooler = > [spooler(1:0..15): Spooler],
          formatter = > [formatter(1:0..15): Formatter],
          printer = > [printer(1:0..15): Printer],
          media = > [media(1:0..47): Media],
          ENDCASE];
  PrinterStatusIndex: --NSPrint-- TYPE = CARDINAL [0..3];
  PrinterStatusType: --NSPrint-- TYPE = MACHINE DEPENDENT{
      spooler, formatter, printer, media};
  PrintLexicon: --LexiconDefs-- PROCEDURE [TTY.Handle];
  PrintOption: --NSPrint-- TYPE = MACHINE DEPENDENT RECORD [
      var(0:0..79): SELECT type(0:0..15): PrintOptionType FROM
          printObjectSize = > [printObjectSize(1:0..31): LONG CARDINAL ← 0],
          recipientName = > [recipientName(1:0..63): String ← [NIL, 0, 0]],
          message = > [message(1:0..63): String ← [NIL, 0, 0]],
          copyCount = > [copyCount(1:0..15): CARDINAL ← 1],
          pagesToPrint = > [pagesToPrint(1:0..31): PagesToPrint ← [1, LAST[CARDINAL]]],
          mediumHint = > [
              mediumHint(1:0..63): Medium ← [paper[[knownSize[usLetter]]]]],
          priorityHint = > [priorityHint(1:0..15): PriorityHint ← normal],
          releaseKey = > [releaseKey(1:0..15): CARDINAL ← 177777B],
          staple = > [staple(1:0..15): BOOLEAN ← FALSE],
          twoSided = > [twoSided(1:0..15): BOOLEAN ← FALSE],
          ENDCASE];
```

```
PrintOptions: --NSPrint-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
    PrintOption;
PrintOptionsIndex: --NSPrint-- TYPE = CARDINAL [0..9];
PrintOptionType: --NSPrint-- TYPE = MACHINE DEPENDENT{
    printObjectSize, recipientName, message, copyCount, pagesToPrint,
mediumHint,
    priorityHint, releaseKey, staple, twoSided};
Printserver: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    address(0:0..95): System.NetworkAddress, location(6:0..63): NSString.String];
PrintserverDescribe: --CHLookup-- Courier.Description;
PrintserverPt: --CHLookup-- TYPE = LONG POINTER TO Printserver;
Priority: --Process-- TYPE = [0..7];
priorityBackground: --Process-- READONLY Priority;
priorityForeground: --Process-- READONLY Priority;
PriorityHint: --NSPrint-- TYPE = MACHINE DEPENDENT{low, normal, high};
priorityNormal: --Process-- READONLY Priority;
Probe: --NSFile-- PROCEDURE [session: Session] RETURNS [probeWithin: CARDINAL];
Problem: --NSVolumeControl-- TYPE = MACHINE DEPENDENT RECORD [
    trouble(0:0..127): SELECT problemType(0:0..15): ProblemType FROM
        changedToDirectory = > NULL,
        duplicatePage = > NULL,
        fileDeleted = > NULL,
        leaderExtensionDeleted = > NULL,
        leaderExtensionMissing = > NULL,
        leaderExtensionReinserted = > NULL,
        leaderExtensionWrongType = > NULL,
        leaderExtensionZeroLength = > NULL,
        newRootCreated = > NULL,
        orphanDirectoryCreated = > NULL,
        orphanPage = > NULL,
        variableAttributesBad = > NULL,
        zeroLength = > NULL,
        duplicateSegmentID = > [
            old(1:0..15): NSSegment.ID, changedTo(2:0..15): NSSegment.ID],
        illegalSegmentID = > [
            old(1:0..15): NSSegment.ID, changedTo(2:0..15): NSSegment.ID],
        illegalAttributeValue = > [old(1:0..111): NSFile.Attribute],
        illegalAttributeValueForNonDirectory = > [old(1:0..111): NSFile.Attribute],
        invalidAttributeValue = > [type(1:0..15): NSFile.AttributeType],
        stringTooLong = > [type(1:0..15): NSFile.AttributeType],
        loopInHierarchy = > [oldParent(1:0..79): NSFile.ID],
        orphanFile = > [oldParent(1:0..79): NSFile.ID],
        missingPages = > [
            first(1:0..31): File.PageNumber, count(3:0..31): File.PageCount],
        unreadablePages = > [
            first(1:0..31): File.PageNumber, count(3:0..31): File.PageCount],
        orphanLeaderExtension = > [id(1:0..79): NSFile.ID],
        orphanSegment = > [id(1:0..79): NSFile.ID, segment(6:0..15): NSSegment.ID],
        segmentDeleted = > [segment(1:0..15): NSSegment.ID],
        segmentMissing = > [segment(1:0..15): NSSegment.ID],
        segmentReinserted = > [segment(1:0..15): NSSegment.ID],
        segmentWrongType = > [segment(1:0..15): NSSegment.ID],
        segmentZeroLength = > [segment(1:0..15): NSSegment.ID],
        tooManySegments = > [oldCount(1:0..15): CARDINAL],
        wrongNumberOfChildren = > [
            old(1:0..15): CARDINAL, changedTo(2:0..15): CARDINAL],
```

```
            wrongSegmentID = > [
                inEntry(1:0..15): NSSegment.ID, inFile(2:0..15): NSSegment.ID],
            wrongSizeInBytes = > [
                old(1:0..31): LONG CARDINAL, changedTo(3:0..31): LONG CARDINAL],
            wrongSizeInPages = > [
                old(1:0..31): LONG CARDINAL, changedTo(3:0..31): LONG CARDINAL],
            ENDCASE];
    Problem: --Scavenger-- TYPE = MACHINE DEPENDENT RECORD [
        trouble(0:0..79): SELECT entryType(0:0..15): EntryType FROM
            unreadable = > [
                first(1:0..31): File.PageNumber, count(3:0..31): File.PageCount],
            missing = > [
                first(1:0..31): File.PageNumber, count(3:0..31): File.PageCount],
            duplicate = > [id(1:0..31): OrphanHandle],
            orphan = > [id(1:0..31): OrphanHandle],
            ENDCASE];
    ProblemArray: --NSVolumeControl-- TYPE = ARRAY [0..0) OF Problem;
    ProblemPointer: --NSVolumeControl-- TYPE = LONG POINTER TO Problem;
    ProblemType: --NSVolumeControl-- TYPE = MACHINE DEPENDENT{
        changedToDirectory, duplicatePage, duplicateSegmentID, fileDeleted,
        illegalAttributeValue, illegalAttributeValueForNonDirectory, illegalSegmentID,
        invalidAttributeValue, leaderExtensionDeleted, leaderExtensionMissing,
        leaderExtensionReinserted, leaderExtensionWrongType,
        leaderExtensionZeroLength, loopInHierarchy, missingPages, orphanFile,
        orphanLeaderExtension, orphanPage, orphanSegment, segmentDeleted,
        segmentMissing, segmentReinserted, segmentWrongType, segmentZeroLength,
        stringTooLong, tooManySegments, unreadablePages, variableAttributesBad,
        wrongNumberOfChildren, wrongSegmentID, wrongSizeInBytes,
    wrongSizeInPages,
        zeroLength, newRootCreated, orphanDirectoryCreated, (256)};
    ProcDesc: --PrincOps-- TYPE = procedure ControlLink;
    Proceed: --Backstop-- PROCEDURE [boot: Volume.ID];
    Process: --Backstop-- TYPE [1];
    ProcessCommandLine: --Exec-- PROCEDURE [
        cmd: LONG STRING, write: Format.StringProc, checkAbort: CheckAbortProc]
        RETURNS [outcome: Outcome];
    ProcessProc: --MailParse-- TYPE = PROCEDURE [
        h: Handle, local: LONG STRING, registry: LONG STRING, domain: LONG STRING,
        info: NameInfo] RETURNS [write: BOOLEAN ← TRUE];
    ProcType: --FormSW-- TYPE = PROCEDURE [
        sw: Window.Handle ← NIL, item: ItemHandle ← NIL, index: CARDINAL ← nullIndex];
    ProductDomain: --PilotSwitches-- TYPE = SwitchName [141C..172C];
    PropagationDate: --MFileProperty-- MFile.Property;
    Properties: --CH-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF PropertyID;
    PropertiesAllocator: --CH-- TYPE = PROCEDURE [count: CARDINAL]
        RETURNS [Properties];
    Property: --MFile-- TYPE = RECORD [property: CARDINAL];
    PropertyArray: --LibrarianUtility-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
        PropertyDescription;
    PropertyDescription: --LibrarianUtility-- TYPE = RECORD [
        pn: Librarian.PropertyNumber, tag: LONG STRING, use: BOOLEAN];
    PropertyError: --MFile-- ERROR [code: PropertyErrorCode];
    PropertyErrorCode: --MFile-- TYPE = {
        noSuchProperty, noRoomInPropertyList, insufficientSpaceForProperty,
        wrongSize};
    PropertyID: --CH-- TYPE = LONG CARDINAL;
```

```
PropertyID: --CHPIDs-- TYPE = CH.PropertyID;
protocolCertificationControl: --NSConstants-- System.SocketNumber;
protocolCertificationTest: --NSConstants-- System.SocketNumber;
ProtocolLevel: --ProtocolCertification-- TYPE = CARDINAL [0..15];
ProtocolName: --ProtocolCertification-- TYPE = MACHINE DEPENDENT{
    echo, routing, error, spp, pex, unspecified(15)};
Prune: --Heap-- PROCEDURE [z: UNCOUNTED ZONE];
Prune: --MDSStorage-- PROCEDURE RETURNS [BOOLEAN];
Prune: --Storage-- PROCEDURE RETURNS [BOOLEAN];
PruneMDS: --Heap-- PROCEDURE [z: MDSZone];
PSBIndex: --BackstopNub-- TYPE [1];
pupAddressTranslation: --NSConstants-- System.SocketNumber;
PushAlternateInputStream: --TTY-- PROCEDURE [h: Handle, stream: Stream.Handle];
PushAlternateInputStreams: --TTYSW-- PROCEDURE [
    sw: Window.Handle, stream: Stream.Handle];
Put: --PieceSource-- PROCEDURE [source: TextSource.Handle, name: LONG STRING]
    RETURNS [new: TextSource.Handle];
Put: --RS232C-- PROCEDURE [channel: ChannelHandle, rec: PhysicalRecordHandle]
    RETURNS [CompletionHandle];
Put: --TTYPort-- PROCEDURE [channel: ChannelHandle, data: CHARACTER]
    RETURNS [status: TransferStatus];
PutBackChar: --TTY-- PROCEDURE [h: Handle, c: CHARACTER];
PutBackChar: --TTYSW-- PROCEDURE [sw: Window.Handle, char: CHARACTER];
PutBlank: --TTY-- PROCEDURE [h: Handle, n: CARDINAL ← 1];
PutBlanks: --TTY-- PROCEDURE [h: Handle, n: CARDINAL ← 1];
PutBlock: --Log-- PROCEDURE [
    level: Level, pointer: LONG POINTER, size: CARDINAL,
    forceOut: BOOLEAN ← FALSE];
PutBlock: --Stream-- PROCEDURE [
    sH: Handle, block: Block, endRecord: BOOLEAN ← FALSE];
PutBlock: --TTY-- PROCEDURE [h: Handle, block: Environment.Block];
PutByte: --Stream-- PROCEDURE [sH: Handle, byte: Byte];
PutByteProcedure: --Stream-- TYPE = PROCEDURE [sH: Handle, byte: Byte];
PutChar: --Exec-- PROCEDURE [h: Handle, c: CHARACTER];
PutChar: --Stream-- PROCEDURE [sH: Handle, char: CHARACTER];
PutChar: --TTY-- PROCEDURE [h: Handle, c: CHARACTER];
PutCommand: --RavenFace-- PROCEDURE [CARDINAL [0..127]];
PutCR: --TTY-- PROCEDURE [h: Handle];
PutDate: --TTY-- PROCEDURE [
    h: Handle, gmt: Time.Packed, format: DateFormat ← noSeconds,
    zone: Time.TimeZoneStandard ← ANSI];
PutDecimal: --TTY-- PROCEDURE [h: Handle, n: INTEGER];
PutEditableFile: --FileSW-- PROCEDURE [sw: Window.Handle, name: LONG STRING]
    RETURNS [ok: BOOLEAN];
PutLine: --TTY-- PROCEDURE [h: Handle, s: LONG STRING];
PutLongDecimal: --TTY-- PROCEDURE [h: Handle, n: LONG INTEGER];
PutLongNumber: --TTY-- PROCEDURE [
    h: Handle, n: LONG UNSPECIFIED, format: NumberFormat];
PutLongOctal: --TTY-- PROCEDURE [h: Handle, n: LONG UNSPECIFIED];
PutLongString: --TTY-- PROCEDURE [h: Handle, s: LONG STRING];
PutLongSubString: --TTY-- PROCEDURE [h: Handle, ss: String.SubString];
PutMessageProc: --OnlineDiagnostics-- TYPE = PROCEDURE [msg: FloppyMessage];
PutNumber: --TTY-- PROCEDURE [h: Handle, n: UNSPECIFIED, format: NumberFormat];
PutOctal: --TTY-- PROCEDURE [h: Handle, n: UNSPECIFIED];
PutProcedure: --Stream-- TYPE = PROCEDURE [
    sH: Handle, block: Block, endRecord: BOOLEAN];
```

```
PutProcType: --GSort-- TYPE = PROCEDURE [p: LONG POINTER, len: CARDINAL];
PutSnapShotToFile: --LibrarianUtility-- PROCEDURE [
    fileName: LONG STRING, snap: Librarian.SnapShotHandle];
PutString: --Log-- PROCEDURE [
    level: Level, string: LONG STRING, forceOut: BOOLEAN ← FALSE];
PutString: --Stream-- PROCEDURE [
    sH: Handle, string: LONG STRING, endRecord: BOOLEAN ← FALSE];
PutString: --TTY-- PROCEDURE [h: Handle, s: LONG STRING];
PutSubString: --TTY-- PROCEDURE [h: Handle, ss: String.SubString];
PutText: --TTY-- PROCEDURE [h: Handle, s: LONG STRING];
PutWord: --Log-- PROCEDURE [
    level: Level, data: UNSPECIFIED, forceOut: BOOLEAN ← FALSE];
PutWord: --Stream-- PROCEDURE [sH: Handle, word: Word];
PutWordProcedure: --Stream-- TYPE = PROCEDURE [sH: Handle, word: Word];
q2000: --DeviceTypes-- Device.Type;
q2010: --DeviceTypes-- Device.Type;
Q2010pagesPerCylinder: --FormatPilotDisk-- CARDINAL = 32;
q2020: --DeviceTypes-- Device.Type;
Q2020pagesPerCylinder: --FormatPilotDisk-- CARDINAL = 64;
q2030: --DeviceTypes-- Device.Type;
Q2030pagesPerCylinder: --FormatPilotDisk-- CARDINAL = 96;
q2040: --DeviceTypes-- Device.Type;
Q2040pagesPerCylinder: --FormatPilotDisk-- CARDINAL = 128;
q2080: --DeviceTypes-- Device.Type;
Q2080pagesPerCylinder: --FormatPilotDiskExtras-- CARDINAL = 112;
Qualification: --Profile-- TYPE = {registry, clearinghouse, none};
Qualify: --Profile-- PROCEDURE [
    token: String, newToken: String, qualification: Qualification];
Quiesce: --TTYPort-- PROCEDURE [channel: ChannelHandle];
R10: --KeyStations-- Bit = 87;
R11: --KeyStations-- Bit = 47;
R12: --KeyStations-- Bit = 77;
R1: --KeyStations-- Bit = 63;
R2: --KeyStations-- Bit = 92;
R3: --KeyStations-- Bit = 106;
R4: --KeyStations-- Bit = 94;
R5: --KeyStations-- Bit = 80;
R6: --KeyStations-- Bit = 79;
R7: --KeyStations-- Bit = 93;
R8: --KeyStations-- Bit = 29;
R9: --KeyStations-- Bit = 81;
Radix: --FormSW-- TYPE = {decimal, octal};
Random: --SpyClient-- PROCEDURE RETURNS [CARDINAL];
RandomDelay: --SpyClient-- PROCEDURE;
rcvL0: --ProtocolCertification-- Stage;
rcvL1: --ProtocolCertification-- Stage;
Read: --Floppy-- PROCEDURE [
    file: FileHandle, first: PageNumber, count: PageCount, vm: LONG POINTER];
readAccess: --NSFile-- Access;
ReadBadPage: --Scavenger-- PROCEDURE [
    file: File.File, page: File.PageNumber, destination: Space.PageNumber]
    RETURNS [readErrors: BOOLEAN];
readBy: --NSAssignedTypes-- AttributeType = 14;
ReadID: --FloppyChannel-- PROCEDURE [
    handle: Handle, address: DiskAddress, buffer: LONG POINTER]
    RETURNS [status: Status];
```

```
ReadLineOrToken: --CmFile-- PROCEDURE [
    h: Token.Handle, buffer: LONG STRING, terminator: CHARACTER];
ReadNextStream: --FileTransfer-- PROCEDURE [Stream.Handle]
    RETURNS [Stream.Handle];
readOn: --NSAssignedTypes-- AttributeType = 15;
ReadOnly: --BTree-- ERROR [tree: Tree];
ReadOnly: --MFile-- PROCEDURE [
    name: LONG STRING, release: ReleaseData, mightWrite: BOOLEAN ← FALSE]
    RETURNS [Handle];
ReadOnly: --MStream-- PROCEDURE [name: LONG STRING, release: ReleaseData]
    RETURNS [Handle];
ReadOnly: --Volume-- ERROR [volume: ID];
ReadOnlyProcType: --FormSW-- TYPE = ProcType;
ReadOrphanPage: --Scavenger-- PROCEDURE [
    volume: Volume.ID, id: OrphanHandle, destination: Space.PageNumber]
    RETURNS [
        file: File.File, type: File.Type, pageNumber: File.PageNumber,
        readErrors: BOOLEAN];
ReadReal: --Real-- PROCEDURE [
    get: PROCEDURE RETURNS [CHARACTER],
    putback: PROCEDURE [CHARACTER] ← DefaultPutback] RETURNS [REAL];
ReadSectors: --FloppyChannel-- PROCEDURE [
    handle: Handle, address: DiskAddress, buffer: LONG POINTER,
    count: CARDINAL ← 1, incrementDataPtr: BOOLEAN ← TRUE]
    RETURNS [status: Status, countDone: CARDINAL];
ReadStream: --FileTransfer-- PROCEDURE [
    conn: Connection, files: FileName.VFN, veto: VetoProc ← NIL,
    showDates: BOOLEAN ← FALSE, type: StreamType ← remote]
    RETURNS [Stream.Handle];
ReadValue: --DebugUsefulDefs-- PROCEDURE [Handle];
ReadWrite: --MFile-- PROCEDURE [
    name: LONG STRING, release: ReleaseData, type: Type,
    initialLength: InitialLength ← dontCare] RETURNS [Handle];
ReadWrite: --MStream-- PROCEDURE [
    name: LONG STRING, release: ReleaseData, type: MFile.Type] RETURNS [Handle];
RealControl: --Real-- PROGRAM;
RealError: --Real-- ERROR;
RealException: --Real-- SIGNAL [
    flags: ExceptionFlags, vp: LONG POINTER TO Extended]
    RETURNS [LONG POINTER TO Extended];
RealToPair: --Real-- PROCEDURE [
    r: REAL, precision: CARDINAL ← DefaultSinglePrecision]
    RETURNS [type: NumberType, fr: LONG INTEGER, exp10: INTEGER];
Reason: --AddressTranslation-- TYPE = {
    noUsefulProperties, ambiguousSeparators, tooManySeparators};
Reason: --NSSessionControl-- TYPE = {logoff, timeout, abort};
Recalibrate: --FloppyChannel-- PROCEDURE [handle: Handle]
    RETURNS [status: Status];
RecordTooLong: --GSort-- ERROR;
Recreate: --Zone-- PROCEDURE [storage: LONG POINTER, zoneBase: Base]
    RETURNS [zH: Handle, rootNode: Base RELATIVE POINTER, s: Status];
rectangleCmd: --BandBLT-- CARDINAL = 10;
RedisplayItem: --FormSW-- PROCEDURE [
    sw: Window.Handle, index: CARDINAL, sameSize: BOOLEAN];
Reference: --NSFile-- TYPE = LONG POINTER TO ReferenceRecord;
```

```
ReferenceRecord: --NSFile-- TYPE = RECORD [
    fileID: ID,
    systemElement: SystemElement ← nullSystemElement,
    volumeID: Volume ← nullVolume];
Register: --NSDataStream-- PROCEDURE [
    stream: Handle, forUseAt: Courier.SystemElement, cH: Courier.Handle,
    useImmediateTicket: BOOLEAN ← TRUE] RETURNS [Ticket];
RegisterBaseDirectoryProc: --NSVolumeControl-- PROCEDURE [
    baseDirectoryProc: BaseDirectoryProc];
RegisterCheckCredentialsProc: --NSSessionControl-- PROCEDURE [
    checkCredentialsProc: CheckCredentialsProc];
RegisterGetCredentialsProc: --NSSessionControl-- PROCEDURE [
    getCredentialsProc: GetCredentialsProc];
RegisterMembershipProc: --NSSessionControl-- PROCEDURE [
    membershipProc: MembershipProc];
RegisterPage: --LsepFace-- PROCEDURE [paperSource: PaperSource ← manual];
registry: --EventTypes-- Supervisor.Event;
RejectRequest: --PacketExchange-- PROCEDURE [
    h: ExchangeHandle, rH: RequestHandle];
Relation: --NSString-- TYPE = {less, equal, greater};
Release: --Context-- PROCEDURE [type: Type, window: Window.Handle];
Release: --MFile-- PROCEDURE [file: Handle];
ReleaseChoice: --MFile-- TYPE = {later, no, goAhead, allowRename};
ReleaseData: --MFile-- TYPE = RECORD [
    proc: PleaseReleaseProc ← NIL, clientInstanceData: LONG POINTER ← NIL];
ReleaseData: --MSegment-- TYPE = RECORD [
    proc: PleaseReleaseProc ← NIL, clientInstanceData: LONG POINTER ← NIL];
ReleaseData: --MStream-- TYPE = RECORD [
    proc: PleaseReleaseProc ← NIL, clientInstanceData: LONG POINTER ← NIL];
ReleaseDataStream: --Courier-- PROCEDURE [cH: Handle];
ReleaseTTY: --Exec-- PROCEDURE [tty: TTY.Handle];
Relock: --NSSessionControl-- PROCEDURE [session: NSFile.Session, id: ServiceID];
Remark: --BodyDefs-- TYPE = LONG STRING;
Remote: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    region(0:0..63): NSName.Organization, domain(4:0..63): NSName.Domain];
remoteDebug: --PilotSwitches-- PilotDomainA = 65C;
RemoteDirectoryDescribe: --CHLookup-- Courier.Description;
RemoteErrorSignalled: --Courier-- ERROR [
    errorNumber: CARDINAL, arguments: Arguments];
RemoteName: --MFileProperty-- MFile.Property;
RemotePt: --CHLookup-- TYPE = LONG POINTER TO Remote;
Remove: --BTree-- PROCEDURE [tree: Tree, name: LONG STRING, value: Value]
    RETURNS [ok: BOOLEAN];
RemoveCharacter: --TTY-- PROCEDURE [h: Handle, n: CARDINAL ← 1];
RemoveCharacter: --TTYSW-- PROCEDURE [sw: Window.Handle, n: CARDINAL ← 1];
RemoveCharacters: --TTY-- PROCEDURE [h: Handle, n: CARDINAL ← 1];
RemoveCharacters: --TTYSW-- PROCEDURE [sw: Window.Handle, n: CARDINAL ← 1];
RemoveCommand: --Exec-- PROCEDURE [h: Handle, name: LONG STRING];
RemovedStatus: --Exec-- TYPE = {ok, noCommand, noProgram};
RemoveFromTree: --Window-- PROCEDURE [Handle];
RemoveNotifyProc: --MFile-- PROCEDURE [
    proc: NotifyProc, filter: Filter, clientInstanceData: LONG POINTER];
RemovePrinter: --DebugUsefulDefs-- PROCEDURE [type: LONG STRING, proc: Printer];
RemoveProperties: --MFile-- PROCEDURE [file: Handle];
RemoveProperty: --MFile-- PROCEDURE [file: Handle, property: Property];
RemoveRootFile: --Volume-- PROCEDURE [type: File.Type, volume: ID ← systemID];
```

```
RemoveSegment: --Zone-- PROCEDURE [zH: Handle, sH: SegmentHandle]
    RETURNS [storage: LONG POINTER, s: Status];
Rename: --DiskSource-- PROCEDURE [
    source: TextSource.Handle, newName: LONG STRING, access: TextSource.Access]
    RETURNS [TextSource.Handle];
Rename: --FileTransfer-- PROCEDURE [
    conn: Connection, old: FileName.VFN, new: FileName.VFN];
Rename: --MFile-- PROCEDURE [file: Handle, newName: LONG STRING];
RenameCommand: --Exec-- PROCEDURE [old: LONG STRING, new: LONG STRING]
    RETURNS [ok: BOOLEAN];
RepairStatus: --PhysicalVolume-- TYPE = {okay, damaged, repaired};
RepairType: --PhysicalVolume-- TYPE = {checkOnly, safeRepair, riskyRepair};
RepairType: --Scavenger-- TYPE = MACHINE DEPENDENT{
    checkOnly, safeRepair, riskyRepair};
Repeat: --LsepFace-- PROCEDURE;
Replace: --MDSStorage-- PROCEDURE [to: POINTER TO STRING, from: LONG STRING];
Replace: --NSFile-- PROCEDURE [
    file: Handle, source: Source, attributes: AttributeList ← nullAttributeList,
    session: Session ← nullSession];
Replace: --Storage-- PROCEDURE [
    to: LONG POINTER TO LONG STRING, from: LONG STRING];
ReplaceBadPage: --Scavenger-- PROCEDURE [
    file: File.File, page: File.PageNumber, source: Space.PageNumber]
    RETURNS [writeErrors: BOOLEAN];
ReplaceBadSector: --Floppy-- PROCEDURE [file: FileHandle, page: PageNumber]
    RETURNS [readError: BOOLEAN];
ReplaceByName: --NSFile-- PROCEDURE [
    directory: Handle, path: String, source: Source,
    attributes: AttributeList ← nullAttributeList,
    session: Session ← nullSession];
ReplaceChild: --NSFile-- PROCEDURE [
    directory: Handle, id: ID, source: Source,
    attributes: AttributeList ← nullAttributeList,
    session: Session ← nullSession];
replaceFlags: --Display-- BitBltFlags;
replaceGrayFlags: --Display-- BitBltFlags;
ReplacementIDFollows: --LibrarianUtility-- INTEGER = -3;
RequestHandle: --PacketExchange-- TYPE = LONG POINTER TO READONLY RequestObject;
RequestID: --NSPrint-- TYPE = System.UniversalID;
RequestObject: --PacketExchange-- TYPE = RECORD [
    nBytes: CARDINAL,
    requestType: ExchangeClientType,
    requestorsExchangeID: ExchangeID,
    requestorsAddress: System.NetworkAddress];
RequestStatus: --NSPrint-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
    RequestStatusComponent;
RequestStatusComponent: --NSPrint-- TYPE = MACHINE DEPENDENT RECORD [
    var(0:0..79): SELECT type(0:0..15): RequestStatusType FROM
        status = > [status(1:0..15): Status],
        statusMessage = > [statusMessage(1:0..63): String],
        ENDCASE];
RequestStatusIndex: --NSPrint-- TYPE = CARDINAL [0..1];
RequestStatusType: --NSPrint-- TYPE = MACHINE DEPENDENT{status, statusMessage};
Reserved: --VolumeConversion-- TYPE [249];
reservedA: --PilotSwitches-- PilotDomainA = 40C;
reservedB: --PilotSwitches-- PilotDomainA = 42C;
```

```
reservedC: --PilotSwitches-- PilotDomainA = 47C;
reservedD: --PilotSwitches-- PilotDomainA = 53C;
reservedE: --PilotSwitches-- PilotDomainA = 55C;
reservedF: --PilotSwitches-- PilotDomainA = 57C;
reservedG: --PilotSwitches-- PilotDomainB = 134C;
reservedH: --PilotSwitches-- PilotDomainC = 176C;
reservedI: --PilotSwitches-- PilotDomainC = 177C;
ReserveType: --RS232C-- TYPE = RS232CEnvironment.ReserveType;
ReserveType: --RS232CEnvironment-- TYPE = {
    preemptNever, preemptAlways, preemptInactive};
reset: --EventTypes-- Supervisor.Event;
Reset: --Log-- PROCEDURE;
Reset: --LogFile-- PROCEDURE [
    file: File.File, firstPageNumber: File.PageNumber ← 1];
Reset: --MSegment-- PROCEDURE [
    segment: Handle, file: MFile.Handle ← dontChangeFile,
    release: ReleaseData ← dontChangeReleaseData,
    fileBase: File.PageNumber ← dontChangeFileBase,
    pages: Environment.PageCount ← dontChangePages,
    swapInfo: SwapUnitOption ← defaultSwapUnitOption,
    usage: Space.Usage ← dontChangeUsage];
Reset: --PieceSource-- PROCEDURE [source: TextSource.Handle]
    RETURNS [original: TextSource.Handle];
ResetBands: --RavenFace-- PROCEDURE
    RETURNS [firstBand: Index, firstBandAddress: BandPointer];
ResetEditableFile: --FileSW-- PROCEDURE [sw: Window.Handle];
ResetOnMatch: --Caret-- PROCEDURE [data: ClientData];
resetStage: --ProtocolCertification-- Stage;
ResetUserAbort: --TTY-- PROCEDURE [h: Handle];
ResetUserAbort: --UserInput-- PROCEDURE [Window.Handle];
ResetVFN: --FileName-- PROCEDURE [
    vfn: VFN, h: BOOLEAN ← FALSE, d: BOOLEAN ← FALSE, n: BOOLEAN ← FALSE,
    v: BOOLEAN ← FALSE];
resolution: --LsepFace-- CARDINAL;
resolution: --RavenFace-- READONLY resolutionPair;
resolutionPair: --RavenFace-- TYPE = ARRAY {fast, slow} OF CARDINAL;
ResolveBlock: --Display-- PROCEDURE [
    window: Handle, block: Environment.Block,
    offsets: LONG POINTER TO ARRAY CARDINAL [0..0) OF CARDINAL,
    font: WindowFont.Handle ← NIL]
    RETURNS [positions: CARDINAL, why: BreakReason];
ResponseProc: --ExpeditedCourier-- TYPE = PROCEDURE [
    hopsToResponder: Hop, elapseTime: ElapseTime, header: Header,
    serializedResponse: Environment.Block] RETURNS [continue: BOOLEAN];
Restart: --LogFile-- TYPE = MACHINE DEPENDENT RECORD [
    message(0:0..15): UNSPECIFIED, time(1:0..31): System.GreenwichMeanTime];
Restart: --RS232C-- PROCEDURE [channel: ChannelHandle, class: OperationClass];
Result: --CH-- TYPE = MACHINE DEPENDENT RECORD [
    flavor(0:0..15): Authenticator.Flavor, status(1:0..15): Authenticator.Status];
Results: --Courier-- TYPE = PROCEDURE [
    resultsRecord: Parameters ← nullParameters,
    requestDataStream: BOOLEAN ← FALSE] RETURNS [sH: Stream.Handle];
Resume: --LsepFace-- PROCEDURE;
resumeDebuggee: --EventTypes-- Supervisor.Event;
resumeSession: --EventTypes-- Supervisor.Event;
```

```
Retrieve: --NSFile-- PROCEDURE [
    file: Handle, sink: Sink, session: Session ← nullSession];
RetrieveByName: --NSFile-- PROCEDURE [
    directory: Handle, path: String, sink: Sink, session: Session ← nullSession];
RetrieveChild: --NSFile-- PROCEDURE [
    directory: Handle, id: ID, sink: Sink, session: Session ← nullSession];
RetryCount: --Dialup-- TYPE = RS232CEnvironment.RetryCount;
RetryCount: --RS232CEnvironment-- TYPE = [0..7];
retryLimit: --FormatPilotDisk-- RetryLimit = 253;
RetryLimit: --FormatPilotDisk-- TYPE = [0..253];
ReturnCode: --CH-- TYPE = MACHINE DEPENDENT RECORD [
    code(0:0..15): Code,
    type(1:0..15): NameType,
    which(2:0..15): ParameterGrouping];
returnOffset: --PrincOps-- CARDINAL = 3;
ReturnToNotifier: --UserInput-- ERROR [string: LONG STRING];
ReturnWait: --Space-- TYPE = {return, wait};
RewritePage: --Scavenger-- PROCEDURE [
    file: File.File, page: File.PageNumber, source: Space.PageNumber]
    RETURNS [writeErrors: BOOLEAN];
RgflagsPtr: --Fonts-- TYPE = LONG POINTER TO PACKED ARRAY CHARACTER OF Flags;
RhsToAddress: --NSAddr-- PROCEDURE [rhs: CH.Buffer]
    RETURNS [succeeded: BOOLEAN, address: Address];
RhsToNSAddr: --NSAddr-- PROCEDURE [rhs: CH.Buffer, nsAddr: NSAddr]
    RETURNS [succeeded: BOOLEAN];
RightShift: --JLevelIVKeys-- KeyName = RightDakuonShift;
RingBound: --ExpeditedCourier-- TYPE = RECORD [low: Hop, high: Hop] ← [
    FIRST[Hop], LAST[Hop]];
RName: --BodyDefs-- TYPE = LONG STRING;
RNameSize: --BodyDefs-- PROCEDURE [name: RName] RETURNS [CARDINAL];
Root: --RealFns-- PROCEDURE [index: REAL, arg: REAL] RETURNS [REAL];
Root: --Window-- PROCEDURE RETURNS [Handle];
RootDirectoryError: --Volume-- ERROR [type: RootDirectoryErrorType];
RootDirectoryErrorType: --Volume-- TYPE = {
    directoryFull, duplicateRootFile, invalidRootFileType, rootFileUnknown};
rootWindow: --Window-- READONLY Handle;
RoundC: --Real-- PROCEDURE [REAL] RETURNS [CARDINAL];
RoundI: --Real-- PROCEDURE [REAL] RETURNS [INTEGER];
RoundLI: --Real-- PROCEDURE [REAL] RETURNS [LONG INTEGER];
RoutersFunction: --Router-- TYPE = {vanillaRouting, interNetworkRouting};
routingInformationSocket: --NSConstants-- System.SocketNumber;
routingServer: --ProtocolCertification-- Stage;
routingUser: --ProtocolCertification-- Stage;
RS232CDiagError: --CommOnlineDiagnostics-- ERROR [reason: RS232CErrorReason];
RS232CDiagStopping: --RemoteCommDiags-- ERROR [
    reason: CommOnlineDiagnostics.StopReason];
RS232CErrorReason: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT{
    aborted, noHardware, noSuchLine, channelInUse, unimplementedFeature,
    invalidParameter, otherError};
RS232CLoopback: --CommOnlineDiagnostics-- PROCEDURE [
    rs232cParams: RS232CParams, setDiagnosticLine: SetDiagnosticLine ← NIL,
    writeMsg: WriteMsg ← NIL, modemChange: ModemChange ← NIL,
    host: System.NetworkAddress ← System.nullNetworkAddress];
RS232CLoopback: --RemoteCommDiags-- PROCEDURE [
    host: System.NetworkAddress, testCount: CARDINAL, lineSpeed: RS232C.LineSpeed,
    correspondent: RS232C.Correspondent, lineNumber: CARDINAL,
```

```
    parity: RS232C.Parity, charLength: RS232C.CharLength,
    pattern: CommOnlineDiagnostics.PatternType, constant: CARDINAL ← 0,
    counters: LONG POINTER TO CommOnlineDiagnostics.CountType,
    dataLengths: CommOnlineDiagnostics.LengthRange,
    setDiagnosticLine: PROCEDURE [lineNumber: CARDINAL] RETURNS [BOOLEAN],
    writeMsg: PROCEDURE [msg: CommOnlineDiagnostics.RS232CTestMessage] ← NIL,
    ModemChange: PROCEDURE [
        modemSignal: CommOnlineDiagnostics.ModemSignal, state: BOOLEAN] ← NIL];
RS232CParams: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT RECORD [
    testCount(0:0..15): CARDINAL ← 177777B,
    safetyTOInMsecs(1:0..31): LONG CARDINAL ← 165140B,
    lineSpeed(3:0..15): RS232C.LineSpeed,
    correspondent(4:0..15): RS232C.Correspondent,
    lineType(5:0..15): RS232C.LineType,
    lineNumber(6:0..15): CARDINAL,
    parity(7:0..15): RS232C.Parity,
    charLength(8:0..15): RS232C.CharLength,
    pattern(9:0..15): PatternType,
    constant(10:0..15): CARDINAL ← 0,
    dataLengths(11:0..31): LengthRange];
RS232CPort: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    description(0:0..63): NSString.String,
    owningCIU(4:0..63): NSString.String,
    owningECS(8:0..63): NSString.String,
    owningClient(12:0..63): NSString.String,
    owningClientType(16:0..15): PortClientType,
    preemptionAllowed(17:0..15): BOOLEAN,
    lineNumber(18:0..15): CARDINAL,
    dialerNumber(19:0..15): CARDINAL,
    portNumber(20:0..15): CARDINAL,
    syncType(21:0..15): PortSyncType,
    duplexity(22:0..15): RS232CEnvironment.Duplexity,
    dialingHardware(23:0..15): PortDialerType,
    charLength(24:0..15): RS232CEnvironment.CharLength,
    echoing(25:0..15): PortEchoingLocation,
    flowControl(26:0..47): RS232CEnvironment.FlowControl,
    lineSpeed(29:0..15): RS232CEnvironment.LineSpeed,
    parity(30:0..15): RS232CEnvironment.Parity,
    stopBits(31:0..15): RS232CEnvironment.StopBits,
    portActsAsDCE(32:0..15): BOOLEAN,
    timeStamp(33:0..31): System.GreenwichMeanTime];
RS232CPortDescribe: --CHLookup-- Courier.Description;
RS232CPortPt: --CHLookup-- TYPE = LONG POINTER TO RS232CPort;
RS232CTestMessage: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT{
    recvOk, recvErrors, deviceError, dataLost, xmitErrors, badSeq, missing,
    sendOk, sendErrors};
Rubout: --TTY-- SIGNAL;
Rubout: --TTYSW-- SIGNAL;
ruletteCmd: --BandBLT-- CARDINAL = 13;
Run: --Exec-- PROCEDURE [
    h: Token.Handle, write: Format.StringProc,
    checkAbort: PROCEDURE RETURNS [abort: BOOLEAN], codeLinks: BOOLEAN ← FALSE];
Run: --MLoader-- PROCEDURE [
    file: MFile.Handle, options: Options ← defaultOptions] RETURNS [Handle];
Run: --PilotClient-- PROCEDURE;
```

```
RunConfig: --Runtime-- PROCEDURE [
    file: File.File, offset: File.PageCount, codeLinks: BOOLEAN ← FALSE];
sa1000: --DeviceTypes-- Device.Type;
sa1004: --DeviceTypes-- Device.Type;
SA1004pagesPerCylinder: --FormatPilotDisk-- CARDINAL = 64;
sa4000: --DeviceTypes-- Device.Type;
SA4000FirstPageForPilot: --FormatPilotDisk-- PROCEDURE [c: SA4000Model44Count]
    RETURNS [DiskPageNumber];
SA4000Model44Count: --FormatPilotDisk-- TYPE = [0..4];
SA4000startOfModel44s: --FormatPilotDisk-- DiskPageNumber = 224;
sa4008: --DeviceTypes-- Device.Type;
SA4008pagesPerCylinder: --FormatPilotDisk-- CARDINAL = 224;
sa800: --DeviceTypes-- Device.Type;
SameFile: --MFile-- PROCEDURE [file1: Handle, file2: Handle] RETURNS [BOOLEAN];
sameLine: --FormSW-- INTEGER = -1;
saveDisplayPagesIndexA: --PilotSwitchesExtraExtras--
    PilotSwitches.PilotDomainC = 366C;
saveDisplayPagesIndexB: --PilotSwitchesExtraExtras--
    PilotSwitches.PilotDomainC = 367C;
SBSOFileType: --FileTypes-- TYPE = CARDINAL [896..959];
Scan: --FormatPilotDisk-- PROCEDURE [
    h: PhysicalVolume.Handle, firstPage: DiskPageNumber, count: LONG CARDINAL,
    retries: RetryLimit ← 10];
ScanError: --AddressTranslation-- ERROR [position: CARDINAL];
ScanForCharacter: --NSString-- PROCEDURE [
    c: Character, s: String, start: CARDINAL ← 0] RETURNS [CARDINAL];
ScanSwitches: --HeraldWindow-- PROCEDURE [
    s: LONG STRING, defaultSwitches: System.Switches ← System.defaultSwitches]
    RETURNS [switches: System.Switches];
ScanWordsPerLine: --LsepFace-- TYPE = [1..253];
Scavenge: --Floppy-- PROCEDURE [volume: VolumeHandle]
    RETURNS [numberOfBadSectors: PageCount];
Scavenge: --NSVolumeControl-- PROCEDURE [
    volume: Volume.ID, options: ScavengerOptions, logVolume: Volume.ID]
    RETURNS [logFile: File.ID];
Scavenge: --PageScavenger-- PROCEDURE [
    device: DeviceIndex, diskPage: PageNumber, overwrite: BOOLEAN]
    RETURNS [
        action: Action, contentsReliable: BOOLEAN, diskStatus: DiskStatus,
        file: File.ID, filePage: File.PageNumber, type: File.Type];
Scavenge: --PhysicalVolume-- PROCEDURE [
    instance: Handle, repair: RepairType, okayToConvert: BOOLEAN]
    RETURNS [status: ScavengerStatus];
Scavenge: --Scavenger-- PROCEDURE [
    volume: Volume.ID, logDestination: Volume.ID, repair: RepairType,
    okayToConvert: BOOLEAN] RETURNS [logFile: File.File];
ScavengerOptions: --NSVolumeControl-- TYPE = RECORD [
    rootType: NSFile.Type,
    index: IndexAttributes,
    orphanDirectoryName: NSString.String,
    orphanDirectoryType: NSFile.Type];
ScavengerStatus: --PhysicalVolume-- TYPE = RECORD [
    badPageList: DamageStatus,
    bootFile: DamageStatus,
    germ: DamageStatus,
    softMicrocode: DamageStatus,
```

```
    hardMicrocode: DamageStatus,
    internalStructures: RepairStatus];
Scope: --NSFile-- TYPE = RECORD [
    count: CARDINAL ← 177777B,
    direction: Direction ← forward,
    filter: Filter ← nullFilter,
    ordering: Ordering ← nullOrdering];
ScopedSerializeIntoRhs: --CH-- PROCEDURE [
    parms: Courier.Parameters, callback: PROCEDURE [Buffer]];
ScopeType: --NSFile-- TYPE = MACHINE DEPENDENT{
    count, direction, filter, ordering};
ScratchMap: --Space-- PROCEDURE [
    count: Environment.PageCount, usage: Usage ← unknownUsage]
    RETURNS [pointer: LONG POINTER];
screenHeight: --UserTerminal-- READONLY CARDINAL [0..77777B];
screenWidth: --UserTerminal-- READONLY CARDINAL [0..77777B];
Scroll: --UserTerminalExtras-- PROCEDURE [
    line: Environment.BitAddress, lineCount: CARDINAL, increment: INTEGER];
ScrollbarProcType: --Scrollbar-- TYPE = PROCEDURE [window: Window.Handle]
    RETURNS [box: Window.Box, offset: Percent, portion: Percent];
scrollingInhibitsCursor: --UserTerminalExtras-- READONLY BOOLEAN;
ScrollProcType: --Scrollbar-- TYPE = PROCEDURE [
    window: Window.Handle, direction: Direction, percent: Percent];
scrollXQuantum: --UserTerminalExtras-- READONLY CARDINAL;
scrollYQuantum: --UserTerminalExtras-- READONLY CARDINAL;
SDDivMod: --Inline-- PROCEDURE [num: LONG INTEGER, den: LONG INTEGER]
    RETURNS [quotient: LONG INTEGER, remainder: LONG INTEGER];
Seal: --VolumeConversion-- CARDINAL = 27272B;
SearchPath: --MFile-- TYPE = LONG POINTER TO SearchPathObject;
searchPathNotUsed: --MFile-- CARDINAL = 177777B;
SearchPathObject: --MFile-- TYPE = RECORD [
    length: CARDINAL, directories: SEQUENCE l: CARDINAL OF LONG STRING];
Seconds: --Authenticator-- TYPE = LONG CARDINAL;
Seconds: --Process-- TYPE = CARDINAL;
SecondsToTicks: --Process-- PROCEDURE [seconds: Seconds] RETURNS [ticks: Ticks];
SectorLength: --OnlineDiagnostics-- TYPE = {one28, two56, five12, one024};
SegmentHandle: --Zone-- TYPE [1];
Selections: --NSFile-- TYPE = RECORD [
    interpreted: InterpretedSelections ← noInterpretedSelections,
    extended: ExtendedSelections ← noExtendedSelections];
SelectNearestAddr: --NSAddr-- PROCEDURE [nsAddr: NSAddr]
    RETURNS [na: System.NetworkAddress];
SelectNearestAddress: --NSAddr-- PROCEDURE [address: Address]
    RETURNS [na: System.NetworkAddress];
SelfDestruct: --Runtime-- PROCEDURE;
Send: --SendDefs-- PROCEDURE [handle: Handle];
SendAttention: --Stream-- PROCEDURE [sH: Handle, byte: Byte];
SendAttentionProcedure: --Stream-- TYPE = PROCEDURE [sH: Handle, byte: Byte];
SendBreak: --RS232C-- PROCEDURE [channel: ChannelHandle];
SendBreak: --TTYPort-- PROCEDURE [channel: ChannelHandle];
SendBreakIllegal: --RS232C-- ERROR;
SendFailed: --SendDefs-- ERROR [notDelivered: BOOLEAN];
SendFromClient: --SendDefs-- PROCEDURE [
    handle: Handle, fromNet: [0..255], fromHost: [0..255],
    senderKey: BodyDefs.Password, sender: BodyDefs.RName,
    returnTo: BodyDefs.RName, validate: BOOLEAN] RETURNS [StartSendInfo];
```

```
SendNow: --Stream-- PROCEDURE [sH: Handle, endRecord: BOOLEAN ← TRUE];
SendNowProcedure: --Stream-- TYPE = PROCEDURE [sH: Handle, endRecord: BOOLEAN];
SendReply: --PacketExchange-- PROCEDURE [
    h: ExchangeHandle, rH: RequestHandle, replyBlk: Environment.Block,
    replyType: ExchangeClientType ← unspecified];
SendRequest: --PacketExchange-- PROCEDURE [
    h: ExchangeHandle, remote: System.NetworkAddress,
    requestBlk: Environment.Block, replyBlk: Environment.Block,
    requestType: ExchangeClientType ← unspecified]
    RETURNS [nBytes: CARDINAL, replyType: ExchangeClientType];
separator: --CH-- CHARACTER = 72C;
separator: --NSName-- CHARACTER = 72C;
separatorCharacter: --NSName-- NSString.Character;
Serialize: --NSFile-- PROCEDURE [
    file: Handle, sink: Sink, session: Session ← nullSession];
SerializeHeader: --ExpeditedCourier-- PROCEDURE [
    rmsH: Stream.Handle, header: Header];
SerializeIntoRhs: --CH-- PROCEDURE [
    parms: Courier.Parameters, heap: UNCOUNTED ZONE] RETURNS [rhs: Buffer];
SerializeParameters: --Courier-- PROCEDURE [
    parameters: Parameters, sH: Stream.Handle];
ServerName: --RetrieveDefs-- PROCEDURE [
    handle: Handle, serverName: BodyDefs.RName];
ServerOff: --CommOnlineDiagnostics-- PROCEDURE;
ServerOn: --CommOnlineDiagnostics-- PROCEDURE;
ServerState: --RetrieveDefs-- TYPE = {unknown, empty, notEmpty};
ServerType: --FileTransfer-- TYPE = MACHINE DEPENDENT{
    unknown, local, ifs, tenex, ns, null(7)};
ServerType: --RetrieveDefs-- TYPE = {MTP, GV};
Service: --ExpeditedCourier-- TYPE = RECORD [
    programNumber: LONG CARDINAL,
    versionRange: Courier.VersionRange,
    bindRequestProcedure: CARDINAL,
    dispatcher: DispatcherProc];
ServiceData: --NSSessionControl-- TYPE = LONG UNSPECIFIED;
ServiceID: --NSSessionControl-- TYPE [1];
ServiceProblem: --NSFile-- TYPE = MACHINE DEPENDENT{
    cannotAuthenticate, serviceFull, serviceUnavailable, sessionInUse};
services: --CHPIDs-- CH.PropertyID = 51;
Services: --ExpeditedCourier-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF
    Service;
ServicesFileType: --FileTypes-- TYPE = CARDINAL [22000B..22077B];
ServicesUsage: --SpaceUsage-- TYPE = Space .Usage[256..383];
Session: --NSFile-- TYPE [2];
Session: --NSSegment-- TYPE = NSFile.Session;
SessionAttributes: --NSSessionControl-- TYPE = RECORD [
    name: NSString.String,
    password: NSString.String,
    systemElement: NSFile.SystemElement,
    createTime: System.GreenwichMeanTime,
    lastActiveTime: System.GreenwichMeanTime,
    privileged: BOOLEAN];
SessionProblem: --NSFile-- TYPE = MACHINE DEPENDENT{sessionInvalid};
SessionRestrictions: --NSSessionControl-- TYPE = RECORD [
    sessionsAllowed: BOOLEAN,
```

```
    maxSessionsAllowed: CARDINAL,
    inactivityTimeout: CARDINAL];
SessionRestrictionSelections: --NSSessionControl-- TYPE = PACKED ARRAY
    SessionRestrictionType OF BooleanFalseDefault;
SessionRestrictionType: --NSSessionControl-- TYPE = {
    sessionsAllowed, maxSessionsAllowed, inactivityTimeout};
Set: --BlockSource-- PROCEDURE [source: Handle, block: Block];
Set: --Caret-- PROCEDURE [data: ClientData, marker: MarkProcType];
Set: --Context-- PROCEDURE [type: Type, data: Data, window: Window.Handle];
Set: --Cursor-- PROCEDURE [Defined];
Set: --Selection-- PROCEDURE [
    pointer: LONG POINTER, conversion: ConvertProcType, actOn: ActOnProcType];
SetAccess: --MFile-- PROCEDURE [file: Handle, access: Access];
SetAccess: --MStream-- PROCEDURE [stream: Handle, access: MFile.Access];
SetAccess: --Space-- PROCEDURE [interval: Interval, access: Access];
SetAttention: --UserInput-- PROCEDURE [
    window: Window.Handle, attention: AttentionProcType];
SetBackground: --UserTerminal-- PROCEDURE [new: Background]
    RETURNS [old: Background];
SetBackingSize: --TTY-- PROCEDURE [h: Handle, size: LONG CARDINAL];
SetBackingSize: --TTYSW-- PROCEDURE [sw: Window.Handle, size: LONG CARDINAL];
SetBalanceBeamChoice: --Profile-- PROCEDURE [BalanceBeamChoice];
SetBitmapUnder: --Window-- PROCEDURE [
    window: Handle, pointer: LONG POINTER ← NIL,
    underChanged: UnderChangedProc ← NIL,
    mouseTransformer: MouseTransformerProc ← NIL] RETURNS [LONG POINTER];
SetBootFiles: --Floppy-- PROCEDURE [
    volume: VolumeHandle, pilotMicrocode: BootFilePointer ← nullBootFilePointer,
    diagnosticMicrocode: BootFilePointer ← nullBootFilePointer,
    germ: BootFilePointer ← nullBootFilePointer,
    pilotBootFile: BootFilePointer ← nullBootFilePointer];
SetBorder: --UserTerminal-- PROCEDURE [oddPairs: [0..255], evenPairs: [0..255]];
SetChecking: --Heap-- PROCEDURE [z: UNCOUNTED ZONE, checking: BOOLEAN];
SetChecking: --Zone-- PROCEDURE [zH: Handle, checking: BOOLEAN]
    RETURNS [s: Status];
SetCheckingMDS: --Heap-- PROCEDURE [z: MDSZone, checking: BOOLEAN];
SetChild: --Window-- PROCEDURE [window: Handle, newChild: Handle]
    RETURNS [oldChild: Handle];
SetClearingRequired: --Window-- PROCEDURE [window: Handle, required: BOOLEAN]
    RETURNS [old: BOOLEAN];
SetClientSystemElement: --NSSessionControl-- PROCEDURE [
    session: NSFile.Session, systemElement: NSFile.SystemElement];
SetClockRate: --LsepFace-- PROCEDURE [rate: VideoClockRate];
SetContext: --FloppyChannel-- PROCEDURE [handle: Handle, context: Context]
    RETURNS [ok: BOOLEAN];
SetCurrent: --FormSW-- PROCEDURE [sw: Window.Handle, index: CARDINAL];
SetCursor: --HeraldWindow-- PROCEDURE [slot: Slot, cursor: Cursor.Defined];
SetCursorPattern: --UserTerminal-- PROCEDURE [cursorPattern: CursorArray];
SetCursorPosition: --UserTerminal-- PROCEDURE [newCursorPosition: Coordinate];
SetCursorState: --HeraldWindow-- PROCEDURE [slot: Slot, state: CursorState];
SetDebugger: --OthelloOps-- PROCEDURE [
    debuggeeFile: File.File, debuggeeFirstPage: File.PageNumber,
    debugger: Volume.ID, debuggerType: Device.Type, debuggerOrdinal: CARDINAL]
    RETURNS [SetDebuggerSuccess];
```

```
SetDebuggerSuccess: --OthelloOps-- TYPE = {
    success, nullBootFile, cantWriteBootFile, notInitialBootFile,
    cantFindStartListHeader, startListHeaderHasBadVersion, other, noDebugger};
SetDebugging: --Profile-- PROCEDURE [BOOLEAN];
SetDefault: --WindowFont-- PROCEDURE [font: Handle];
SetDefaultDomain: --Profile-- PROCEDURE [domain: String];
SetDefaultName: --NSVolumeControl-- PROCEDURE [name: NSString.String];
SetDefaultOrganization: --Profile-- PROCEDURE [organization: String];
SetDefaultOutputSink: --Format-- PROCEDURE [
    new: StringProc, clientData: LONG POINTER ← NIL]
    RETURNS [old: StringProc, oldClientData: LONG POINTER];
SetDefaultRegistry: --Profile-- PROCEDURE [registry: String];
SetDefaultServerType: --FileTransfer-- PROCEDURE [
    conn: Connection, type: ServerType];
SetDefaultSession: --NSFile-- PROCEDURE [session: Session];
SetDefaultTimeout: --NSVolumeControl-- PROCEDURE [timeout: NSFile.Timeout];
SetDefaultVolume: --NSVolumeControl-- PROCEDURE [volume: Volume.ID];
SetDeleteProtect: --MFile-- PROCEDURE [file: Handle, deleteProtected: BOOLEAN];
SetDesiredProperties: --FileTransfer-- PROCEDURE [
    conn: Connection, props: DesiredProperties];
SetDiagnosticLine: --CommOnlineDiagnostics-- TYPE = PROCEDURE [
    lineNumber: CARDINAL] RETURNS [lineSet: BOOLEAN];
SetDisplayProc: --Window-- PROCEDURE [Handle, PROCEDURE [Handle]]
    RETURNS [PROCEDURE [Handle]];
SetEcho: --TTY-- PROCEDURE [h: Handle, new: EchoClass] RETURNS [old: EchoClass];
SetEcho: --TTYSW-- PROCEDURE [sw: Window.Handle, new: TTY.EchoClass]
    RETURNS [old: TTY.EchoClass];
SetExpirationDate: --OthelloOps-- PROCEDURE [
    file: File.File, firstPage: File.PageNumber,
    expirationDate: System.GreenwichMeanTime] RETURNS [SetExpirationDateSuccess];
SetExpirationDateSuccess: --OthelloOps-- TYPE = SetDebuggerSuccess
    [success..other];
SetExtension: --FileWindow-- PROCEDURE [ext: LONG STRING];
SetFile: --FileSW-- PROCEDURE [
    sw: Window.Handle, name: LONG STRING, s: Stream.Handle ← NIL,
    position: TextSource.Position ← 0];
SetFileServerProtocol: --Profile-- PROCEDURE [FileServerProtocol];
SetFont: --Menu-- PROCEDURE [font: WindowFont.Handle];
SetGetSwitchesSuccess: --OthelloOps-- TYPE = SetDebuggerSuccess
    [success..other];
SetIndex: --MemoryStream-- PROCEDURE [
    sH: Stream.Handle, position: Stream.Position];
setInkCmd: --BandBLT-- CARDINAL = 12;
SetInputFocus: --UserInput-- PROCEDURE [
    w: Window.Handle, notify: PROCEDURE [Window.Handle, LONG POINTER],
    takesInput: BOOLEAN, data: LONG POINTER ← NIL];
SetInputOptions: --Stream-- PROCEDURE [sH: Handle, options: InputOptions];
SetInsertion: --Selection-- PROCEDURE [
    pointer: LONG POINTER, conversion: ConvertProcType,
    clear: ClearTrashBinProcType];
SetInterruptMasks: --LsepFace-- PROCEDURE [
    control: WORD, status: WORD, data: WORD];
SetInterruptMasks: --RavenFace-- PROCEDURE [
    control: WORD, status: WORD, data: WORD];
SetLength: --MFile-- PROCEDURE [file: Handle, length: ByteCount];
SetLength: --MStream-- PROCEDURE [stream: Handle, fileLength: MFile.ByteCount];
```

```
setLevelCmd: --BandBLT-- CARDINAL = 11;
SetLibrarian: --Profile-- PROCEDURE [
    name: String ← noChange, prefix: String ← noChange,
    suffix: String ← noChange];
SetLineType: --RS232C-- PROCEDURE [channel: ChannelHandle, lineType: LineType];
SetLogReadLength: --MStream-- PROCEDURE [
    stream: Handle, position: MFile.ByteCount];
SetMaxDiskLength: --DiskSource-- PROCEDURE [
    source: TextSource.Handle, maxLength: LONG CARDINAL];
SetMinimumWindows: --FileWindow-- PROCEDURE [keep: CARDINAL];
SetModifyNotificationProc: --FormSW-- PROCEDURE [
    sw: Window.Handle, proc: ProcType ← NIL];
SetMousePosition: --UserTerminal-- PROCEDURE [newMousePosition: Coordinate];
SetMTPRetrieveDefault: --RetrieveDefs-- PROCEDURE [
    host: LONG STRING, reg: LONG STRING];
SetNetworkID: --Router-- PROCEDURE [
    physicalOrder: CARDINAL, medium: PhysicalMedium,
    newNetID: System.NetworkNumber] RETURNS [oldNetID: System.NetworkNumber];
SetNotifier: --Scrollbar-- PROCEDURE [
    window: Window.Handle, type: Type, notify: ScrollProcType]
    RETURNS [ScrollProcType];
SetOptions: --FormSW-- PROCEDURE [sw: Window.Handle, options: Options];
SetOverflow: --Log-- PROCEDURE [option: Overflow];
SetPageOffsets: --RavenFace-- PROCEDURE [
    linesFromLeft: CARDINAL, wordTabFromBottom: CARDINAL];
SetParameter: --RS232C-- PROCEDURE [
    channel: ChannelHandle, parameter: Parameter];
SetParameter: --TTYPort-- PROCEDURE [
    channel: ChannelHandle, parameter: Parameter];
SetParent: --Window-- PROCEDURE [window: Handle, newParent: Handle]
    RETURNS [oldParent: Handle];
SetPhysicalVolumeBootFile: --OthelloOps-- PROCEDURE [
    file: File.File, type: BootFileType, firstPage: File.PageNumber];
SetPNR: --Menu-- PROCEDURE [Window.Handle];
SetPosition: --Stream-- PROCEDURE [sH: Handle, position: Position];
SetPositionProcedure: --Stream-- TYPE = PROCEDURE [
    sH: Handle, position: Position];
SetPrimaryCredentials: --FileTransfer-- PROCEDURE [
    conn: Connection, user: LONG STRING, password: LONG STRING];
SetPriority: --Process-- PROCEDURE [priority: Priority];
SetProcessorTime: --OthelloOps-- PROCEDURE [time: System.GreenwichMeanTime];
SetProcs: --FileTransfer-- PROCEDURE [
    conn: Connection, clientData: LONG POINTER, messages: MessageProc ← NIL,
    login: ClientProc ← NIL, noteProgress: ClientProc ← NIL,
    checkAbort: CheckAbortProc ← NIL];
SetProperties: --MFile-- PROCEDURE [
    file: Handle, create: Time.Packed ← System.gmtEpoch,
    write: Time.Packed ← System.gmtEpoch, read: Time.Packed ← System.gmtEpoch,
    length: ByteCount, type: Type, deleteProtected: BOOLEAN ← FALSE,
    writeProtected: BOOLEAN ← FALSE, readProtected: BOOLEAN ← FALSE];
SetProperty: --MFile-- PROCEDURE [
    file: Handle, property: Property, block: Environment.Block];
SetProtection: --MFile-- PROCEDURE [
    file: Handle, deleteProtected: BOOLEAN ← FALSE,
    writeProtected: BOOLEAN ← FALSE, readProtected: BOOLEAN ← FALSE];
SetReadProtect: --MFile-- PROCEDURE [file: Handle, readProtected: BOOLEAN];
```

```
SetReleaseData: --MFile-- PROCEDURE [file: Handle, release: ReleaseData];
SetReleaseData: --MSegment-- PROCEDURE [segment: Handle, release: ReleaseData];
SetReleaseData: --MStream-- PROCEDURE [stream: Handle, release: ReleaseData];
SetRemoteName: --FileName-- PROCEDURE [
    file: MFile.Handle, remoteName: LONG STRING];
SetRestart: --Log-- PROCEDURE [message: UNSPECIFIED];
SetRootFile: --Floppy-- PROCEDURE [file: FileHandle];
SetRootNode: --Zone-- PROCEDURE [zH: Handle, node: Base RELATIVE POINTER];
SetScanLineLength: --LsepFace-- PROCEDURE [scanLineWords: ScanWordsPerLine];
SetScanLineLength: --RavenFace-- PROCEDURE [activeWordsEachScanline: [1..256]];
SetSearchPath: --MFile-- PROCEDURE [SearchPath]
    RETURNS [succeeded: BOOLEAN ← TRUE];
SetSecondaryCredentials: --FileTransfer-- PROCEDURE [
    conn: Connection, connectName: LONG STRING, connectPassword: LONG STRING];
SetSelection: --FormSW-- PROCEDURE [
    sw: Window.Handle, index: CARDINAL, first: CARDINAL, last: CARDINAL];
SetServiceData: --NSSessionControl-- PROCEDURE [
    session: NSFile.Session, id: ServiceID, data: ServiceData,
    handler: TerminationHandler];
SetSeverity: --MsgSW-- PROCEDURE [sw: Window.Handle, severity: Severity];
SetSibling: --Window-- PROCEDURE [window: Handle, newSibling: Handle]
    RETURNS [oldSibling: Handle];
SetSize: --File-- PROCEDURE [file: File, size: PageCount];
SetSize: --FileWindow-- PROCEDURE [sw: Window.Handle, box: Window.Box];
SetSizeInBytes: --NSSegment-- PROCEDURE [
    file: NSFile.Handle, bytes: ByteCount, segment: ID ← defaultID,
    session: Session ← nullSession];
SetSizeInPages: --NSSegment-- PROCEDURE [
    file: NSFile.Handle, pages: PageCount, segment: ID ← defaultID,
    session: Session ← nullSession];
SetSourceMenu: --FileWindow-- PROCEDURE [menu: Menu.Handle];
SetSST: --Stream-- PROCEDURE [sH: Handle, sst: SubSequenceType];
SetSSTProcedure: --Stream-- TYPE = PROCEDURE [sH: Handle, sst: SubSequenceType];
SetState: --Log-- PROCEDURE [state: State];
SetState: --UserTerminal-- PROCEDURE [new: State] RETURNS [old: State];
SetStickyFlags: --Real-- PROCEDURE [new: ExceptionFlags ← NoExceptions]
    RETURNS [old: ExceptionFlags];
SetStreamTimeout: --NSDataStream-- PROCEDURE [
    stream: Handle, waitTimeInSeconds: LONG CARDINAL];
SetStringIn: --UserInput-- PROCEDURE [
    window: Window.Handle, proc: StringProcType] RETURNS [old: StringProcType];
SetStringOut: --UserInput-- PROCEDURE [
    window: Window.Handle, proc: StringProcType] RETURNS [old: StringProcType];
SetSwapCtrlAndCommand: --Profile-- PROCEDURE [BOOLEAN];
SetSwitches: --HeraldWindow-- PROCEDURE [new: System.Switches];
SetSwitches: --OthelloOps-- PROCEDURE [
    file: File.File, firstPage: File.PageNumber, switches: System.Switches]
    RETURNS [SetGetSwitchesSuccess];
SetTabs: --AsciiSink-- PROCEDURE [
    sink: TextSink.Handle, tabStops: TabStops ← NIL];
SetTagPlaces: --FormSW-- PROCEDURE [
    items: ItemDescriptor,
    tabStops: LONG DESCRIPTOR FOR ARRAY CARDINAL OF CARDINAL, bitTabs: BOOLEAN];
SetTimeout: --Process-- PROCEDURE [
    condition: LONG POINTER TO CONDITION, ticks: Ticks];
```

```
SetTimeoutProcedure: --Stream-- TYPE = PROCEDURE [
    sH: Handle, waitTime: Milliseconds];
SetTimes: --MFile-- PROCEDURE [
    file: Handle, create: Time.Packed ← System.gmtEpoch,
    read: Time.Packed ← System.gmtEpoch, write: Time.Packed ← System.gmtEpoch];
SetTrashBin: --Selection-- PROCEDURE [
    pointer: LONG POINTER, conversion: ConvertProcType,
    clear: ClearTrashBinProcType];
SetType: --MFile-- PROCEDURE [file: Handle, type: Type];
SetTypeIn: --FormSW-- PROCEDURE [
    sw: Window.Handle, index: CARDINAL, position: CARDINAL];
SetTypescriptSize: --FileSW-- PROCEDURE [
    sw: Window.Handle, size: LONG CARDINAL];
SetUser: --Profile-- PROCEDURE [
    name: String ← noChange, password: String ← noChange];
SetUserAbort: --UserInput-- PROCEDURE [Window.Handle];
SetVolumeBootFile: --OthelloOps-- PROCEDURE [
    file: File.File, type: BootFileType, firstPage: File.PageNumber];
SetWaitTime: --NetworkStream-- PROCEDURE [sH: Stream.Handle, time: WaitTime];
SetWaitTimes: --PacketExchange-- PROCEDURE [
    h: ExchangeHandle, waitTime: WaitTime, retransmissionInterval: WaitTime];
SetWriteProtect: --MFile-- PROCEDURE [file: Handle, writeProtected: BOOLEAN];
Severity: --FileTransfer-- TYPE = {verbose, terse, warning, fatal};
Severity: --MsgSW-- TYPE = {info, warning, fatal};
ShareBlock: --MStream-- PROCEDURE [
    stream: Handle, start: MFile.ByteCount, length: CARDINAL]
    RETURNS [block: Environment.Block];
Shift: --Display-- PROCEDURE [
    window: Handle, box: Window.Box, newPlace: Window.Place];
ShortBlock: --Stream-- ERROR;
ShortControlLink: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID ControlLinkTag FROM
    frame => [frame(0:0..15): LocalFrameHandle],
    procedure => NULL,
    indirect => [
        SELECT OVERLAID * FROM
        port => [port(0:0..15): PortHandle],
        link => [link(0:0..15): POINTER TO ControlLink],
        ENDCASE],
    rep => [
        fill0(0:0..13): [0..37777B],
        indirect(0:14..14): BOOLEAN,
        proc(0:15..15): BOOLEAN],
    ENDCASE];
ShortCopyREAD: --DebugUsefulDefs-- PROCEDURE [
    from: ClientSource, nwords: CARDINAL, to: LocalDest];
ShortCopyWRITE: --DebugUsefulDefs-- PROCEDURE [
    from: LocalSource, nwords: CARDINAL, to: ClientDest];
ShortREAD: --DebugUsefulDefs-- PROCEDURE [loc: ClientSource]
    RETURNS [val: UNSPECIFIED];
ShortWRITE: --DebugUsefulDefs-- PROCEDURE [loc: ClientDest, val: UNSPECIFIED];
Sides: --Floppy-- TYPE = {one, two, default};
siemens9750: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
Signal: --BackstopNub-- TYPE [2];
SignalMsg: --BackstopNub-- TYPE [1];
```

```
SignalRemoteError: --Courier-- ERROR [
    errorNumber: CARDINAL, arguments: Parameters ← nullParameters];
SimpleCredentials: --NSName-- TYPE = NameRecord;
SimpleDestroyProc: --Context-- DestroyProcType;
SimpleVerifier: --NSName-- TYPE = HashedPassword;
Sin: --RealFns-- PROCEDURE [radians: REAL] RETURNS [sin: REAL];
SinDeg: --RealFns-- PROCEDURE [degrees: REAL] RETURNS [sin: REAL];
SingleDouble: --OnlineDiagnostics-- TYPE = {single, double};
SingleLineBand: --LsepFace-- TYPE = LONG POINTER;
Sink: --NSDataStream-- TYPE = RECORD [
    SELECT type: * FROM
    proc = > [proc: PROCEDURE [SourceStream]],
    stream = > [stream: SinkStream],
    none = > NULL,
    ENDCASE];
Sink: --NSFile-- TYPE = NSDataStream.Sink;
SinkStream: --NSDataStream-- TYPE = RECORD [Handle];
sizeInBytes: --NSAssignedTypes-- AttributeType = 16;
sizeInPages: --NSAssignedTypes-- AttributeType = 26;
SizeOfSerializedData: --NSName-- PROCEDURE [parameters: Courier.Parameters]
    RETURNS [sizeInWords: CARDINAL];
SkipBand: --LsepFace-- PROCEDURE;
SkipToNext: --FormSW-- PROCEDURE [sw: Window.Handle];
Sleep: --FormSW-- PROCEDURE [Window.Handle];
Slide: --Window-- PROCEDURE [window: Handle, newPlace: Place];
SlideAndSize: --Window-- PROCEDURE [
    window: Handle, newBox: Box, gravity: Gravity ← nw];
SlideAndSizeAndStack: --Window-- PROCEDURE [
    window: Handle, newBox: Box, newSibling: Handle, newParent: Handle ← NIL,
    gravity: Gravity ← nw];
SlideAndStack: --Window-- PROCEDURE [
    window: Handle, newPlace: Place, newSibling: Handle, newParent: Handle ←
NIL];
SlideIconically: --Window-- PROCEDURE [window: Handle, newPlace: Place];
Slot: --HeraldWindow-- TYPE = LONG POINTER TO SlotObject;
SlotObject: --HeraldWindow-- TYPE;
smallAnonymousBackingFile: --PilotSwitches-- AnonymousBackingFileSize = 173C;
SmallestNormalizedNumber: --Real-- REAL;
SocketNumber: --Format-- PROCEDURE [
    proc: StringProc, socketNumber: System.SocketNumber, format: NetFormat,
    clientData: LONG POINTER ← NIL];
SocketNumber: --Put-- PROCEDURE [
    h: Window.Handle ← NIL, socketNumber: System.SocketNumber, format:
NetFormat];
SolicitClock: --LsepFace-- PROCEDURE;
SolicitPaperSource: --LsepFace-- PROCEDURE [paperSource: PaperSource];
SolicitStatus: --LsepFace-- PROCEDURE;
SolicitStatus: --RavenFace-- PROCEDURE;
Sort: --GSort-- PROCEDURE [
    get: GetProcType, put: PutProcType, compare: CompareProcType,
    expectedItemSize: CARDINAL ← 30, maxItemSize: CARDINAL ← 1000,
    pagesInHeap: CARDINAL ← 100];
SortItemPort: --GSort-- TYPE = PORT [len: CARDINAL] RETURNS [p: LONG POINTER];
SortStarter: --GSort-- TYPE = PORT [
    nextItem: LONG POINTER TO SortItemPort, put: PutProcType,
    compare: CompareProcType, expectedItemSize: CARDINAL ← 30,
```

```
    maxItemSize: CARDINAL ← 1000, pagesInHeap: CARDINAL ← 100]
    RETURNS [p: LONG POINTER];
SortStopper: --GSort-- TYPE = PORT [len: CARDINAL ← 0];
Source: --NSDataStream-- TYPE = RECORD [
    SELECT type: * FROM
    proc = > [proc: PROCEDURE [SinkStream]],
    stream = > [stream: SourceStream],
    none = > NULL,
    ENDCASE];
Source: --NSFile-- TYPE = NSDataStream.Source;
Source: --Selection-- TYPE = LONG POINTER TO SourceObject;
SourceEditProc: --FormSW-- FilterProcType;
SourceHandle: --FormSW-- TYPE = LONG POINTER TO source ItemObject;
SourceItem: --FormSW-- PROCEDURE [
    tag: LONG STRING ← NIL, readOnly: BOOLEAN ← FALSE, invisible: BOOLEAN ← FALSE,
    drawBox: BOOLEAN ← FALSE, hasContext: BOOLEAN ← FALSE,
    inHeap: BOOLEAN ← FALSE, place: Window.Place ← nextPlace,
    boxWidth: CARDINAL ← defaultBoxWidth,
    filterProc: FilterProcType ← SourceEditProc,
    menuProc: MenuProcType ← VanillaMenuProc, source: TextSource.Handle,
    z: UNCOUNTED ZONE ← NIL] RETURNS [SourceHandle];
SourceObject: --Selection-- TYPE = RECORD [
    data: LONG POINTER, proc: SourceProc, destroy: DestroyProc];
SourceProc: --Selection-- TYPE = PROCEDURE [
    data: ClientData, string: LONG STRING];
SourceStream: --NSDataStream-- TYPE = RECORD [Handle];
SP: --Ascii-- CHARACTER = 40C;
SpaceProblem: --NSFile-- TYPE = MACHINE DEPENDENT{
    allocationExceeded, attributeAreaFull, mediumFull};
spare1: --Event-- READONLY Supervisor.SubsystemHandle;
spare1: --EventTypes-- Supervisor.Event;
spare2: --Event-- READONLY Supervisor.SubsystemHandle;
spare2: --EventTypes-- Supervisor.Event;
spare3: --Event-- READONLY Supervisor.SubsystemHandle;
spare3: --EventTypes-- Supervisor.Event;
spare4: --Event-- READONLY Supervisor.SubsystemHandle;
spare4: --EventTypes-- Supervisor.Event;
spare5: --Event-- READONLY Supervisor.SubsystemHandle;
spare5: --EventTypes-- Supervisor.Event;
SpareEvents: --EventTypes-- TYPE = [1000..177776B];
SplitNode: --Zone-- PROCEDURE [zH: Handle, p: LONG POINTER, n: BlockSize]
    RETURNS [s: Status];
Spooler: --NSPrint-- TYPE = MACHINE DEPENDENT{available, busy, disabled, full};
sppAttn: --ProtocolCertification-- Stage;
sppConnect: --ProtocolCertification-- Stage;
sppDuplex: --ProtocolCertification-- Stage;
sppListen: --ProtocolCertification-- Stage;
sppMulti: --ProtocolCertification-- Stage;
sppOutOfSeq: --ProtocolCertification-- Stage;
sppProbing: --ProtocolCertification-- Stage;
sppRetrans: --ProtocolCertification-- Stage;
sppSink: --ProtocolCertification-- Stage;
sppSource: --ProtocolCertification-- Stage;
sppSst: --ProtocolCertification-- Stage;
sppThruput: --ProtocolCertification-- Stage;
SqRt: --RealFns-- PROCEDURE [REAL] RETURNS [REAL];
```

```
SrcDesc: --BitBlt-- TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
    gray => [gray(0:0..15): GrayParm],
    srcBpl => [srcBpl(0:0..15): INTEGER],
    ENDCASE];
SrcFunc: --BitBlt-- TYPE = {null, complement};
SSTChange: --Stream-- SIGNAL [sst: SubSequenceType, nextIndex: CARDINAL];
Stack: --Window-- PROCEDURE [
    window: Handle, newSibling: Handle, newParent: Handle ← NIL];
stackDepth: --PrincOps-- CARDINAL = 14;
stackSize: --BackstopNub-- CARDINAL = 14;
Stage: --ProtocolCertification-- TYPE = MACHINE DEPENDENT RECORD [
    mediumType(0:0..7): MediumType,
    protocolLevel(0:8..15): ProtocolLevel,
    protocolName(1:0..7): ProtocolName,
    stageNumber(1:8..15): StageNumber];

END.
StageNumber: --ProtocolCertification-- TYPE = CARDINAL [0..15];
Start: --Exec-- PROCEDURE [handle: MLoader.Handle];
Start: --MLoader-- PROCEDURE [Handle];
Start: --NSSessionControl-- PROCEDURE;
Start: --NSVolumeControl-- PROCEDURE;
Start: --RS232CControl-- PROCEDURE;
StartCounting: --SpyClient-- PROCEDURE;
StartEchoUser: --CommOnlineDiagnostics-- PROCEDURE [
    targetSystemElement: System.NetworkAddress, echoParams: EchoParams,
    eventReporter: EventReporter ← NIL,
    host: System.NetworkAddress ← System.nullNetworkAddress];
StartEchoUser: --RemoteCommDiags-- PROCEDURE [
    host: System.NetworkAddress, targetSystemElement: System.NetworkAddress,
    echoParams: CommOnlineDiagnostics.EchoParams,
    eventReporter: CommOnlineDiagnostics.EventReporter ← NIL]
    RETURNS [echoUser: CommOnlineDiagnostics.EchoUserHandle];
Started: --DebugUsefulDefs-- PROCEDURE [GFHandle] RETURNS [BOOLEAN];
startEnumeration: --Router-- READONLY System.NetworkNumber;
StartFault: --Runtime-- ERROR [dest: PROGRAM];
StartImage: --LsepFace-- PROCEDURE;
StartImage: --RavenFace-- PROCEDURE [firstBand: Index];
StartIndexGreaterThanStopIndexPlusOne: --ByteBlt-- ERROR;
StartingProcess: --Event-- PROCEDURE [id: LONG STRING] RETURNS [Handle];
StartItem: --SendDefs-- PROCEDURE [handle: Handle, type: BodyDefs.ItemType];
StartSend: --SendDefs-- PROCEDURE [
    handle: Handle, senderPwd: LONG STRING, sender: BodyDefs.RName,
    returnTo: BodyDefs.RName ← NIL, validate: BOOLEAN] RETURNS [StartSendInfo];
StartSendInfo: --SendDefs-- TYPE = {
    ok, badPwd, badSender, badReturnTo, allDown};
StartStop: --UserInput-- TYPE = {start, stop};
StartText: --SendDefs-- PROCEDURE [handle: Handle];
StarUsage: --SpaceUsage-- TYPE = Space .Usage[384..511];
State: --Log-- TYPE = MACHINE DEPENDENT{off, error, warning, remark};
State: --UserTerminal-- TYPE = {on, off, disconnected};
StateVector: --PrincOps-- TYPE = MACHINE DEPENDENT RECORD [
    stk(0:0..223): ARRAY [0..13] OF UNSPECIFIED,
    instbyte(14:0..7): BYTE,
    stkptr(14:8..15): BYTE,
```

```
        data(15:0..47): SELECT OVERLAID * FROM
            dst = > NULL,
            fault = > [
                frame(15:0..15): LocalFrameHandle,
                faultData(16:0..31): SELECT OVERLAID * FROM
                    allocFault = > [fsi(16:0..15): FrameSizeIndex],
                    memFault = > [memPointer(16:0..31): LONG POINTER],
                    otherFault = > [dataArray(16): ARRAY [0..0) OF UNSPECIFIED],
                    ENDCASE],
            ENDCASE];
    StatsIndices: --CommOnlineDiagnostics-- TYPE = {
        echoServerPkts, EchoServerBytes, packetsRecv, wordsRecv, packetsMissed,
        badRecvStatus, okButDribble, badCrc, badAlignmentButOkCrc,
    crcAndBadAlignment,
        packetTooLong, overrun, idleInput, packetsSent, wordsSent, badSendStatus,
        tooManyCollisions, lateCollisions, underrun, stuckOutput, coll0, coll1, coll2,
        coll3, coll4, coll5, coll6, coll7, coll8, coll9, coll10, coll11, coll12,
        coll13, coll14, coll15, spare};
    Status: --Authenticator-- TYPE = MACHINE DEPENDENT{
        OK, invalidVerifier, expiredVerifier, reusedVerifier, invalidCredentials,
        expiredCredentials, (177777B)};
    Status: --FloppyChannel-- TYPE = MACHINE DEPENDENT RECORD [
        diskChanged(0:0..0): BOOLEAN,
        tbd1(0:1..1): BOOLEAN,
        twoSided(0:2..2): BOOLEAN,
        tbd2(0:3..3): BOOLEAN,
        error(0:4..4): BOOLEAN,
        inProgress(0:5..5): BOOLEAN,
        recalibrateError(0:6..6): BOOLEAN,
        sectorTooLarge(0:7..7): BOOLEAN,
        notReady(0:8..8): BOOLEAN,
        writeProtect(0:9..9): BOOLEAN,
        deletedData(0:10..10): BOOLEAN,
        recordNotFound(0:11..11): BOOLEAN,
        crcError(0:12..12): BOOLEAN,
        track00(0:13..13): BOOLEAN,
        hardwareError(0:14..14): BOOLEAN,
        goodCompletion(0:15..15): BOOLEAN];
    Status: --NSPrint-- TYPE = MACHINE DEPENDENT{
        pending, inProgress, completed, completedWithWarnings, unknown, rejected,
        aborted, canceled, held};
    Status: --Volume-- TYPE = {
        unknown, partiallyOnLine, closedAndInconsistent, closedAndConsistent,
        openRead, openReadWrite};
    Status: --Zone-- TYPE = {
        okay, noRoomInZone, nonEmptySegment, storageOutOfRange, zoneTooSmall,
        segmentTooSmall, invalidNode, invalidZone, invalidSegment, nodeLoop,
        wrongSeal, wrongVersion};
    StatusWait: --RS232C-- PROCEDURE [channel: ChannelHandle, stat: DeviceStatus]
        RETURNS [newstat: DeviceStatus];
    StatusWait: --TTYPort-- PROCEDURE [channel: ChannelHandle, stat: DeviceStatus]
        RETURNS [newstat: DeviceStatus];
    stdDandelionMemorySize: --PilotSwitches-- PilotDomainA = 71C;
    Stop: --RS232CControl-- PROCEDURE [suspendActiveChannels: BOOLEAN];
    StopBits: --RS232C-- TYPE = RS232CEnvironment.StopBits;
    StopBits: --RS232CEnvironment-- TYPE = [1..2];
```

```
StopBits: --TTYPort-- TYPE = TTYPortEnvironment.StopBits;
StopBits: --TTYPortEnvironment-- TYPE = {none, one, oneAndHalf, two};
StopCounting: --SpyClient-- PROCEDURE;
StopImage: --LsepFace-- PROCEDURE;
Store: --Cursor-- PROCEDURE [Handle];
store: --EventTypes-- Supervisor.Event;
Store: --NSFile-- PROCEDURE [
    directory: Handle, source: Source,
    attributes: AttributeList ← nullAttributeList, controls: Controls ← [],
    session: Session ← nullSession] RETURNS [file: Handle];
StoreCursor: --HeraldWindow-- PROCEDURE [
    slot: Slot, cursor: LONG POINTER TO UserTerminal.CursorArray];
StoreStream: --FileTransfer-- PROCEDURE [
    conn: Connection, remote: FileName.VFN, veto: VetoProc ← NIL,
    showDates: BOOLEAN ← FALSE, stream: Stream.Handle, creation: Time.Packed,
    bytes: LONG CARDINAL, fileType: FileType];
StreamType: --FileTransfer-- TYPE = {remote, local, temporary};
String: --MDSStorage-- PROCEDURE [nchars: CARDINAL] RETURNS [s: STRING];
String: --NSFile-- TYPE = NSString.String;
String: --NSName-- TYPE = NSString.String;
String: --NSPrint-- TYPE = NSString.String;
String: --NSString-- TYPE = RECORD [
    bytes: LONG POINTER TO PACKED ARRAY CARDINAL OF Environment.Byte,
    length: CARDINAL ← 0,
    maxlength: CARDINAL ← 0];
String: --Profile-- TYPE = LONG STRING;
String: --Storage-- PROCEDURE [nchars: CARDINAL] RETURNS [s: LONG STRING];
StringBody: --NSVolumeControl-- TYPE = MACHINE DEPENDENT RECORD [
    length(0:0..15): CARDINAL, bytes(1): PACKED ARRAY [0..0) OF Environment.Byte];
StringBoundsFault: --NSString-- SIGNAL [old: String, increaseBy: CARDINAL]
    RETURNS [new: String];
StringBoundsFault: --String-- SIGNAL [s: LONG STRING] RETURNS [ns: LONG STRING];
StringEditProc: --FormSW-- FilterProcType;
StringExpToDecimal: --DebugUsefulDefs-- PROCEDURE [LONG STRING]
    RETURNS [INTEGER];
StringExpToLDecimal: --DebugUsefulDefs-- PROCEDURE [LONG STRING]
    RETURNS [LONG INTEGER];
StringExpToLNum: --DebugUsefulDefs-- PROCEDURE [
    exp: LONG STRING, radix: CARDINAL] RETURNS [LONG UNSPECIFIED];
StringExpToLOctal: --DebugUsefulDefs-- PROCEDURE [LONG STRING]
    RETURNS [LONG CARDINAL];
StringExpToNum: --DebugUsefulDefs-- PROCEDURE [
    exp: LONG STRING, radix: CARDINAL] RETURNS [UNSPECIFIED];
StringExpToOctal: --DebugUsefulDefs-- PROCEDURE [LONG STRING]
    RETURNS [CARDINAL];
StringFeedback: --FormSW-- TYPE = {normal, password};
StringForErrorCode: --MailParse-- PROCEDURE [code: ErrorCode, s: LONG STRING];
StringFromMesaString: --NSString-- PROCEDURE [s: MesaString] RETURNS [String];
StringHandle: --FormSW-- TYPE = LONG POINTER TO string ItemObject;
StringItem: --FormSW-- PROCEDURE [
    tag: LONG STRING ← NIL, readOnly: BOOLEAN ← FALSE, invisible: BOOLEAN ← FALSE,
    drawBox: BOOLEAN ← FALSE, hasContext: BOOLEAN ← FALSE,
    inHeap: BOOLEAN ← FALSE, place: Window.Place ← nextPlace,
    feedback: StringFeedback ← normal, boxWidth: CARDINAL ← defaultBoxWidth,
    filterProc: FilterProcType ← StringEditProc,
    menuProc: MenuProcType ← VanillaMenuProc, string: LONG POINTER TO LONG
```

```
STRING,
    z: UNCOUNTED ZONE ← NIL] RETURNS [StringHandle];
StringLength: --MDSStorage-- PROCEDURE [s: LONG STRING] RETURNS [CARDINAL];
StringLength: --Storage-- PROCEDURE [s: LONG STRING] RETURNS [CARDINAL];
StringLength: --String-- PROCEDURE [s: LONG STRING] RETURNS [CARDINAL];
StringOut: --UserInput-- PROCEDURE [window: Window.Handle, string: LONG STRING];
StringProc: --Format-- TYPE = PROCEDURE [
    s: LONG STRING, clientData: LONG POINTER ← NIL];
StringProcType: --UserInput-- TYPE = PROCEDURE [
    window: Window.Handle, string: LONG STRING];
StringToDecimal: --ExtendedString-- PROCEDURE [
    field: LONG POINTER, size: CARDINAL, string: LONG STRING];
StringToDecimal: --NSString-- PROCEDURE [s: String] RETURNS [INTEGER];
StringToDecimal: --String-- PROCEDURE [s: LONG STRING] RETURNS [INTEGER];
StringToHostNumber: --AddressTranslation-- PROCEDURE [LONG STRING]
    RETURNS [System.HostNumber];
StringToLongNumber: --NSString-- PROCEDURE [s: String, radix: CARDINAL ← 10]
    RETURNS [LONG UNSPECIFIED];
StringToLongNumber: --String-- PROCEDURE [s: LONG STRING, radix: CARDINAL ← 10]
    RETURNS [LONG UNSPECIFIED];
StringToNetworkAddress: --AddressTranslation-- PROCEDURE [
    s: LONG STRING, defaultCHPID: CH.PropertyID ← 0,
    distingName: NSName.Name ← NIL]
    RETURNS [addr: NetworkAddress, chUsed: BOOLEAN, usedCHPID: CH.PropertyID];
StringToNetworkNumber: --AddressTranslation-- PROCEDURE [LONG STRING]
    RETURNS [System.NetworkNumber];
StringToNumber: --ExtendedString-- PROCEDURE [
    field: LONG POINTER, size: CARDINAL, base: CARDINAL, string: LONG STRING];
StringToNumber: --NSString-- PROCEDURE [s: String, radix: CARDINAL ← 10]
    RETURNS [UNSPECIFIED];
StringToNumber: --String-- PROCEDURE [s: LONG STRING, radix: CARDINAL ← 10]
    RETURNS [UNSPECIFIED];
StringToOctal: --ExtendedString-- PROCEDURE [
    field: LONG POINTER, size: CARDINAL, string: LONG STRING];
StringToOctal: --NSString-- PROCEDURE [s: String] RETURNS [UNSPECIFIED];
StringToOctal: --String-- PROCEDURE [s: LONG STRING] RETURNS [UNSPECIFIED];
StringToPacked: --Date-- PROCEDURE [
    s: LONG STRING, zoneFormat: Time.TimeZoneStandard ← ANSI]
    RETURNS [dt: Packed, notes: Notes, length: NATURAL];
StringToReal: --Real-- PROCEDURE [LONG STRING] RETURNS [REAL];
StuffCharacter: --UserInput-- PROCEDURE [window: Window.Handle, char: CHARACTER]
    RETURNS [BOOLEAN];
StuffCurrentSelection: --UserInput-- PROCEDURE [window: Window.Handle]
    RETURNS [BOOLEAN];
StuffString: --UserInput-- PROCEDURE [
    window: Window.Handle, string: LONG STRING] RETURNS [BOOLEAN];
StuffTrashBin: --UserInput-- PROCEDURE [window: Window.Handle]
    RETURNS [BOOLEAN];
SubdivideName: --NSName-- PROCEDURE [
    s: String, callBack: PROCEDURE [Name], clientDefaults: Name ← NIL];
SubSequenceType: --Stream-- TYPE = [0..255];
SubString: --Format-- PROCEDURE [
    proc: StringProc, ss: String.SubString, clientData: LONG POINTER ← NIL];
SubString: --NSString-- TYPE = LONG POINTER TO SubStringDescriptor;
SubString: --Put-- PROCEDURE [h: Window.Handle ← NIL, ss: String.SubString];
SubString: --String-- TYPE = LONG POINTER TO SubStringDescriptor;
```

```
SubStringDescriptor: --NSString-- TYPE = RECORD [
    base: String, offset: CARDINAL, length: CARDINAL];
SubStringDescriptor: --String-- TYPE = RECORD [
    base: LONG STRING, offset: CARDINAL, length: CARDINAL];
SubsystemHandle: --Supervisor-- TYPE [1];
subtreeSize: --NSAssignedTypes-- AttributeType = 27;
subtreeSizeLimit: --NSAssignedTypes-- AttributeType = 28;
SubVolume: --OthelloOps-- TYPE = RECORD [
    lvID: Volume.ID,
    subVolumeSize: Volume.PageCount,
    firstLVPageNumber: LogicalVolumePageNumber,
    firstPVPageNumber: PhysicalVolume.PageNumber];
SubVolumeUnknown: --OthelloOps-- ERROR [sv: SubVolume];
Suspend: --RS232C-- PROCEDURE [channel: ChannelHandle, class: OperationClass];
SuspendReason: --NetworkStream-- TYPE = {
    notSuspended, transmissionTimeout, noRouteToDestination,
    remoteServiceDisappeared};
SVPointer: --PrincOps-- TYPE = POINTER TO StateVector;
Swap: --Cursor-- PROCEDURE [old: Handle, new: Handle];
swapCancelled: --EventTypes-- Supervisor.Event;
swapCtrlAndCommand: --Profile-- READONLY BOOLEAN;
SwapNames: --MFile-- PROCEDURE [f1: Handle, f2: Handle];
swapping: --Event-- READONLY Supervisor.SubsystemHandle;
SwapReason: --BackstopNub-- TYPE [1];
SwapUnitOption: --MSegment-- TYPE = Space.SwapUnitOption;
SwapUnitOption: --Space-- TYPE = RECORD [
    body: SELECT swapUnitType: SwapUnitType FROM
        unitary => NULL,
        uniform => [size: SwapUnitSize ← defaultSwapUnitSize],
        irregular => [sizes: LONG DESCRIPTOR FOR ARRAY [0..0) OF SwapUnitSize],
        ENDCASE];
SwapUnitSize: --MSegment-- TYPE = Space.SwapUnitSize;
SwapUnitSize: --Space-- TYPE = CARDINAL;
SwapUnitType: --MSegment-- TYPE = Space.SwapUnitType;
SwapUnitType: --Space-- TYPE = {unitary, uniform, irregular};
SwapValue: --BTree-- PROCEDURE [
    tree: Tree, name: LONG STRING, oldValue: Value, newValue: Value]
    RETURNS [ok: BOOLEAN];
switches: --HeraldWindow-- READONLY System.Switches;
SwitchName: --PilotSwitches-- TYPE = CHARACTER;
SyncChar: --RS232C-- TYPE = RS232CEnvironment.SyncChar;
SyncChar: --RS232CEnvironment-- TYPE = Environment.Byte;
SyncCount: --RS232C-- TYPE = RS232CEnvironment.SyncCount;
SyncCount: --RS232CEnvironment-- TYPE = [0..7];
system6: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
SystemElement: --Courier-- TYPE = System.NetworkAddress;
systemElement: --NSAssignedTypes-- AttributeType = 29;
SystemElement: --NSFile-- TYPE = System.NetworkAddress;
SystemElement: --NSPrint-- TYPE = System.NetworkAddress;
SystemElementIsLocal: --NSSessionControl-- PROCEDURE [
    systemElement: NSFile.SystemElement] RETURNS [BOOLEAN];
systemFont: --EventTypes-- Supervisor.Event;
SystemID: --Volume-- PROCEDURE RETURNS [ID];
systemID: --Volume-- READONLY ID;
systemMDSZone: --Heap-- READONLY MDSZone;
systemVolume: --NSVolumeControl-- READONLY Volume.ID;
```

```
systemZone: --Heap-- READONLY UNCOUNTED ZONE;
T10: --KeyStations-- Bit = 105;
T1: --KeyStations-- Bit = 98;
T2: --KeyStations-- Bit = 97;
t300: --DeviceTypes-- Device.Type;
t300pagesPerCylinder: --FormatPilotDisk-- CARDINAL = 570;
T3: --KeyStations-- Bit = 99;
T4: --KeyStations-- Bit = 100;
T5: --KeyStations-- Bit = 101;
T6: --KeyStations-- Bit = 102;
T7: --KeyStations-- Bit = 103;
t80: --DeviceTypes-- Device.Type;
t80pagesPerCylinder: --FormatPilotDisk-- CARDINAL = 150;
T8: --KeyStations-- Bit = 104;
T9: --KeyStations-- Bit = 109;
TAB: --Ascii-- CHARACTER = 11C;
Table: --StringLookUp-- TYPE = ARRAY CARDINAL OF LONG STRING;
TableDesc: --StringLookUp-- TYPE = LONG DESCRIPTOR FOR Table;
TableError: --CmFile-- SIGNAL [h: Handle, name: LONG STRING];
TabStops: --AsciiSink-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF CARDINAL;
TagOnlyHandle: --FormSW-- TYPE = LONG POINTER TO tagOnly ItemObject;
TagOnlyItem: --FormSW-- PROCEDURE [
    tag: LONG STRING ← NIL, readOnly: BOOLEAN ← FALSE, invisible: BOOLEAN ← FALSE,
    drawBox: BOOLEAN ← FALSE, hasContext: BOOLEAN ← FALSE,
    place: Window.Place ← nextPlace, otherItem: CARDINAL ← nullIndex,
    z: UNCOUNTED ZONE ← NIL] RETURNS [TagOnlyHandle];
TajoDefaultEvents: --EventTypes-- TYPE = [300..399];
tajoDefaults: --Event-- READONLY Supervisor.SubsystemHandle;
Tan: --RealFns-- PROCEDURE [radians: REAL] RETURNS [tan: REAL];
TanDeg: --RealFns-- PROCEDURE [degrees: REAL] RETURNS [tan: REAL];
Target: --Selection-- TYPE = MACHINE DEPENDENT{
    window, subwindow, string, source, length, position, pieceList, longInteger,
    interpressMaster, potentialInterpressMaster, token, firstFree, last(255)};
tBackstopDebuggee: --CommonSoftwareFileTypes-- File.Type;
tBackstopDebugger: --CommonSoftwareFileTypes-- File.Type;
tBackstopLog: --CommonSoftwareFileTypes-- File.Type;
tBen: --PerformanceToolFileTypes-- File.Type;
tCarryVolumeDirectory: --CommonSoftwareFileTypes-- File.Type;
tClearingHouseBackupFile: --CommonSoftwareFileTypes-- File.Type;
tDirectory: --CommonSoftwareFileTypes-- File.Type;
tDirectory: --NSAssignedTypes-- FileType = 1;
teleDebugSocket: --NSConstants-- System.SocketNumber;
tellFileSystemSwappingIn: --EventTypes-- Supervisor.Event;
tellFileSystemSwappingOut: --EventTypes-- Supervisor.Event;
tEmpty: --NSAssignedTypes-- FileType = 4;
TerminationHandler: --NSSessionControl-- TYPE = PROCEDURE [
    session: NSFile.Session, id: ServiceID, data: ServiceData, reason: Reason];
TestFileType: --FileTypes-- TYPE = CARDINAL [768..895];
Text: --DebugUsefulDefs-- Format.StringProc;
Text: --Display-- PROCEDURE [
    window: Handle, string: LONG STRING, place: Window.Place,
    font: WindowFont.Handle ← NIL, lineLength: INTEGER ← infinity,
    flags: BitBltFlags ← textFlags, bounds: Window.BoxHandle ← NIL]
    RETURNS [newPlace: Window.Place];
Text: --Format-- PROCEDURE [
    proc: StringProc, s: LONG STRING, clientData: LONG POINTER ← NIL];
```

```
Text: --Put-- PROCEDURE [h: Window.Handle ← NIL, s: LONG STRING];
textFlags: --Display-- BitBltFlags;
TextInline: --Display-- PROCEDURE [
    window: Handle, string: LONG STRING, place: Window.Place,
    font: WindowFont.Handle ← NIL, lineLength: INTEGER ← infinity,
    flags: BitBltFlags ← textFlags, bounds: Window.BoxHandle ← NIL]
    RETURNS [Window.Place];
tFileList: --CommonSoftwareFileTypes-- File.Type;
ThreePartName: --CH-- TYPE = NSName.NameRecord;
Ticket: --NSDataStream-- TYPE [11];
Ticks: --Process-- TYPE = CARDINAL;
TicksToMsec: --Process-- PROCEDURE [ticks: Ticks] RETURNS [msec: Milliseconds];
Time: --NSFile-- TYPE = System.GreenwichMeanTime;
Time: --NSPrint-- TYPE = LONG CARDINAL;
Timeout: --NSFile-- TYPE = Process.Seconds;
Timeout: --PacketExchange-- SIGNAL;
TimeOut: --Stream-- SIGNAL [nextIndex: CARDINAL];
TimeServerError: --OthelloOps-- ERROR [error: TimeServerErrorType];
TimeServerErrorType: --OthelloOps-- TYPE = {
    noCommunicationFacilities, noResponse};
timeServerSocket: --NSConstants-- System.SocketNumber;
Timestamp: --BodyDefs-- TYPE = MACHINE DEPENDENT RECORD [
    net(0:0..7): [0..255], host(0:8..15): [0..255], time(1:0..31): PackedTime];
tinyDandelionMemorySize: --PilotSwitches-- PilotDomainA = 63C;
TIPCSourceDate: --MFileProperty-- MFile.Property;
TitleMatch: --CmFile-- PROCEDURE [buffer: LONG STRING, title: LONG STRING]
    RETURNS [matches: BOOLEAN];
ToggleFlag: --FormSW-- PROCEDURE [
    sw: Window.Handle, index: CARDINAL, flag: Flag];
ToggleVisibility: --FormSW-- PROCEDURE [sw: Window.Handle, index: CARDINAL];
toolWindow: --Event-- READONLY Supervisor.SubsystemHandle;
ToolWindowEvents: --EventTypes-- TYPE = [600..699];
TooManyProcesses: --Process-- ERROR;
Trajectory: --Display-- PROCEDURE [
    window: Handle, box: Window.Box ← Window.nullBox, proc: TrajectoryProc,
    source: LONG POINTER ← NIL, bpl: CARDINAL ← 16, height: CARDINAL ← 16,
    flags: BitBltFlags ← bitFlags, missesChildren: BOOLEAN ← FALSE,
    brick: Brick ← NIL];
TrajectoryProc: --Display-- TYPE = PROCEDURE [Handle]
    RETURNS [Window.Box, INTEGER];
TransferProblem: --NSFile-- TYPE = MACHINE DEPENDENT{
    aborted, checksumIncorrect, formatIncorrect, noRendezvous, wrongDirection};
TransferProblem: --NSPrint-- TYPE = MACHINE DEPENDENT{
    aborted, formatIncorrect(2), noRendezvous, wrongDirection};
TransferStatus: --RS232C-- TYPE = {
    success, dataLost, deviceError, frameTimeout, checksumError, parityError,
    asynchFramingError, invalidChar, invalidFrame, aborted, disaster};
TransferWait: --RS232C-- PROCEDURE [
    channel: ChannelHandle, event: CompletionHandle]
    RETURNS [byteCount: CARDINAL, status: TransferStatus];
TransmitNow: --RS232C-- PROCEDURE [
    channel: ChannelHandle, event: CompletionHandle]
    RETURNS [byteCount: CARDINAL, status: TransferStatus];
trapLink: --PrincOps-- ControlLink;
TrapLink: --PrincOps-- ControlLink;
TrapNonTrappingNaN: --Real-- LONG CARDINAL = 1;
```

```
TrappingNaN: --Real-- REAL;
TrapTrappingNaN: --Real-- LONG CARDINAL = 2;
Tree: --BTree-- TYPE = LONG POINTER TO TreeObject;
TreeObject: --BTree-- TYPE;
TrimBoxStickouts: --Window-- PROCEDURE [window: Handle, box: Box] RETURNS [Box];
TrinityFileEntry: --ScavengerExtras-- TYPE = MACHINE DEPENDENT RECORD [
    file(0:0..79): System.UniversalID,
    numberOfProblems(5:0..15): CARDINAL,
    problems(6): ARRAY [0..0) OF Scavenger.Problem];
TrinityHeader: --ScavengerExtras-- TYPE = MACHINE DEPENDENT RECORD [
    volume(0:0..79): Volume.ID,
    date(5:0..31): System.GreenwichMeanTime,
    incomplete(7:0..14): BOOLEAN,
    repaired(7:15..15): BOOLEAN,
    numberOfFiles(8:0..31): LONG CARDINAL];
TruncateString: --NSString-- PROCEDURE [s: String, bytes: CARDINAL]
    RETURNS [String];
tScavengerLog: --Scavenger-- READONLY File.Type;
tScavengerLogOtherVolume: --Scavenger-- READONLY File.Type;
tSerialized: --NSAssignedTypes-- FileType = 3;
tText: --NSAssignedTypes-- FileType = 2;
ttyHost: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
tUnassigned: --CommonSoftwareFileTypes-- File.Type;
tUnassigned: --FileTypes-- File.Type;
tUnspecified: --NSAssignedTypes-- FileType = 0;
tUntypedFile: --FileTypes-- File.Type;
tVolumeConversionLog: --VolumeConversion-- READONLY File.Type;
tWillard: --PerformanceToolFileTypes-- File.Type;
Type: --Context-- TYPE = MACHINE DEPENDENT{
    all, first, lastAllocated(37737B), last(37777B)};
Type: --Cursor-- TYPE = MACHINE DEPENDENT{
    activate, blank, bullseye, confirm, crossHairsCircle, ftp, ftpBoxes,
    hourGlass, lib, menu, mouseRed, mouseYellow, mouseBlue, mtp, pointDown,
    pointLeft, pointRight, pointUp, questionMark, retry, scrollDown, scrollLeft,
    scrollLeftRight, scrollRight, scrollUp, scrollUpDown, textPointer, typeKey,
    groundedText, last(255)};
Type: --Device-- TYPE = PRIVATE RECORD [CARDINAL];
Type: --File-- TYPE = RECORD [CARDINAL];
Type: --FormSW-- TYPE = {fixed, relative};
Type: --Heap-- TYPE = {normal, uniform, mds};
Type: --LogFile-- TYPE = MACHINE DEPENDENT{null, block, string, (63)};
Type: --MFile-- TYPE = MACHINE DEPENDENT{
    unknown, text, binary, directory, null(255)};
type: --NSAssignedTypes-- AttributeType = 17;
Type: --NSFile-- TYPE = LONG CARDINAL;
Type: --Scrollbar-- TYPE = {horizontal, vertical};
Type: --Volume-- TYPE = MACHINE DEPENDENT{
    normal, debugger, debuggerDebugger, nonPilot};
TypeSet: --Volume-- TYPE = PACKED ARRAY Type OF BooleanDefaultFalse;
ubBootServeeSocket: --NSConstants-- System.SocketNumber;
ubBootServerSocket: --NSConstants-- System.SocketNumber;
ubIPCSocket: --NSConstants-- System.SocketNumber;
UDDivMod: --Inline-- PROCEDURE [num: LONG CARDINAL, den: LONG CARDINAL]
    RETURNS [quotient: LONG CARDINAL, remainder: LONG CARDINAL];
UnboundLink: --PrincOps-- ControlLink;
unboundLink: --PrincOps-- ControlLink;
```

```
UnboundProcedure: --Runtime-- SIGNAL [dest: ControlLink];
UndefinedProblem: --NSFile-- TYPE = CARDINAL;
UndefinedProblem: --NSPrint-- TYPE = CARDINAL;
UnderChangedProc: --Window-- TYPE = PROCEDURE [Handle, Box];
UnexportExpeditedPrograms: --ExpeditedCourier-- PROCEDURE [
    h: ExpeditedServiceHandle];
UnexportRemoteProgram: --Courier-- PROCEDURE [
    programNumber: LONG CARDINAL, versionRange: VersionRange];
UnifyAccessLists: --NSFile-- PROCEDURE [
    directory: Handle, session: Session ← nullSession];
UnimplementedFeature: --RS232C-- ERROR;
Uninstantiate: --Menu-- PROCEDURE [menu: Handle, window: Window.Handle];
Unintelligible: --Date-- ERROR [vicinity: NATURAL];
UniqueAction: --Caret-- PROCEDURE RETURNS [Action];
UniqueAction: --Selection-- PROCEDURE RETURNS [Action];
uniqueConnID: --NetworkStream-- READONLY ConnectionID;
uniqueNetworkAddr: --NetworkStream-- READONLY System.NetworkAddress;
uniqueSocketID: --NSConstants-- System.SocketNumber;
UniqueTarget: --Selection-- PROCEDURE RETURNS [Target];
UniqueType: --Context-- PROCEDURE RETURNS [type: Type];
UniqueType: --Cursor-- PROCEDURE RETURNS [Type];
UniversalID: --System-- TYPE [5];
Unknown: --File-- ERROR [file: File];
Unknown: --Volume-- ERROR [volume: ID];
unknownChecksum: --NSFile-- CARDINAL = 177777B;
UnknownCommandFile: --Expand-- SIGNAL [name: LONG STRING] RETURNS [LONG STRING];
unknownConnID: --NetworkStream-- READONLY ConnectionID;
unknownSocketID: --NSConstants-- System.SocketNumber;
unknownUsage: --Space-- Usage = 0;
unlimitedSize: --Heap-- Environment.PageCount = 77777777B;
Unload: --Exec-- PROCEDURE [handle: MLoader.Handle];
Unload: --MLoader-- PROCEDURE [Handle];
UnloadCommand: --Exec-- PROCEDURE [h: Handle, name: LONG STRING]
    RETURNS [RemovedStatus];
Unlock: --NSSessionControl-- PROCEDURE [session: NSFile.Session, id: ServiceID];
Unmap: --Space-- PROCEDURE [
    pointer: LONG POINTER, returnWait: ReturnWait ← wait]
    RETURNS [nil: LONG POINTER];
UnmapAt: --Space-- PROCEDURE [
    pointer: LONG POINTER, returnWait: ReturnWait ← wait]
    RETURNS [interval: Interval];
UnNew: --Runtime-- PROCEDURE [frame: PROGRAM];
UnNewConfig: --Runtime-- PROCEDURE [link: ControlLink];
UnpackFilename: --FileName-- PROCEDURE [s: LONG STRING, vfn: VFN];
unspecified: --CH-- PropertyID = 0;
UpArrowAction: --Expand-- TYPE = {skip, remove, none};
UpperCase: --NSString-- PROCEDURE [c: Character] RETURNS [Character];
UpperCase: --String-- PROCEDURE [c: CHARACTER] RETURNS [CHARACTER];
Usage: --Space-- TYPE = [0..3777B];
useLargeHeap: --PilotSwitchesExtras-- PilotSwitches.PilotDomainB = 135C;
User: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    lastNameIndex(0:0..15): CARDINAL,
    password(1:0..63): NSString.String,
    systemAdministrator(5:0..15): BOOLEAN,
    fileserver(6:0..31): NSName.Name,
    mailserver(8:0..31): NSName.Name,
```

```
    description(10:0..63): NSString.String,
    product(14:0..63): NSString.String,
    training(18:0..63): NSString.String,
    help(22:0..15): BOOLEAN];
UserAbort: --UserInput-- PROCEDURE [Window.Handle] RETURNS [BOOLEAN];
UserAborted: --DebugUsefulDefs-- SIGNAL;
UserDescribe: --CHLookup-- Courier.Description;
UserDotCmLine: --CmFile-- PROCEDURE [title: LONG STRING, name: LONG STRING]
    RETURNS [s: LONG STRING];
UserDotCmOpen: --CmFile-- PROCEDURE RETURNS [h: Handle];
UserIsMember: --NSSessionControl-- PROCEDURE [
    key: NSString.String, type: NSFile.AccessEntryType, session: NSFile.Session,
    tryHard: BOOLEAN] RETURNS [status: MembershipStatus];
UserPt: --CHLookup-- TYPE = LONG POINTER TO User;
useSpecialMemory: --PilotSwitchesExtras-- PilotSwitches.PilotDomainC = 372C;
useSpecialMemoryIfNoDisplay: --PilotSwitchesExtras--
    PilotSwitches.PilotDomainC = 373C;
useStdHeap: --PilotSwitchesExtras-- PilotSwitches.PilotDomainA = 45C;
useTinyHeap: --PilotSwitchesExtras-- PilotSwitches.PilotDomainB = 133C;
UsingGenerator: --StringLookUp-- PROCEDURE [
    key: LONG STRING, generator: GeneratorProcType, caseFold: BOOLEAN ← TRUE,
    noAbbreviation: BOOLEAN ← FALSE, bufferBytes: CARDINAL ← 500]
    RETURNS [index: CARDINAL];
UsingGeneratorWithBuffer: --StringLookUp-- PROCEDURE [
    key: LONG STRING, generator: GeneratorProcType, caseFold: BOOLEAN ← TRUE,
    noAbbreviation: BOOLEAN ← FALSE, buffer: LONG STRING]
    RETURNS [index: CARDINAL];
UsualExceptions: --Real-- ExceptionFlags;
Valid: --DebugUsefulDefs-- PROCEDURE [GFHandle] RETURNS [BOOLEAN];
ValidAsMesaString: --NSString-- PROCEDURE [s: String] RETURNS [BOOLEAN];
Validate: --Window-- PROCEDURE [window: Handle];
ValidateFrame: --Runtime-- PROCEDURE [frame: UNSPECIFIED];
ValidateGlobalFrame: --Runtime-- PROCEDURE [frame: GenericProgram];
ValidateProcess: --Process-- PROCEDURE [process: UNSPECIFIED];
ValidateTree: --Window-- PROCEDURE [window: Handle ← rootWindow];
ValidFilename: --MFile-- PROCEDURE [name: LONG STRING] RETURNS [ok: BOOLEAN];
ValidProperties: --FileTransfer-- TYPE = {
    host, directory, body, version, author, size, type, oldFile, readProtect};
Value: --BTree-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF CARDINAL;
ValueSize: --BTree-- TYPE = [1..31];
ValueTooSmall: --BTree-- ERROR [tree: Tree];
VanillaMenuProc: --FormSW-- MenuProcType;
Verifier: --Authenticator-- TYPE = NSName.Verifier;
Verifier: --NSFile-- TYPE = NSName.Verifier;
Verifier: --NSName-- TYPE [2];
VersatecFileType: --FileTypes-- TYPE = CARDINAL [23400B..23477B];
VersatecUsage: --SpaceUsageExtras-- TYPE = Space .Usage[512..639];
version: --NSAssignedTypes-- AttributeType = 18;
VersionMismatch: --Backstop-- SIGNAL;
VersionMismatch: --Courier-- ERROR [versionRange: VersionRange];
VersionMismatch: --MLoader-- SIGNAL [module: LONG STRING];
VersionMismatch: --Runtime-- SIGNAL [module: LONG STRING];
VersionRange: --Courier-- TYPE = RECORD [low: CARDINAL, high: CARDINAL];
VetoEvents: --EventTypes-- TYPE = [100..199];
```

```
VetoProc: --FileTransfer-- TYPE = PROCEDURE [
    conn: Connection, clientData: LONG POINTER, post: MessageProc, info: InfoProc,
    showingDates: BOOLEAN] RETURNS [confirm: Confirmation, showDates: BOOLEAN];
VFN: --FileName-- TYPE = VirtualFilename;
VideoClockRate: --LsepFace-- TYPE = MACHINE DEPENDENT{clock1MHz, clock500KHz};
VirtualFilename: --FileName-- TYPE = LONG POINTER TO VirtualFilenameObject;
VirtualFilenameObject: --FileName-- TYPE = RECORD [
    host: LONG STRING,
    directory: LONG STRING,
    name: LONG STRING,
    version: LONG STRING];
virtualMemory: --Space-- READONLY Interval;
VoidPhysicalVolumeBootFile: --OthelloOps-- PROCEDURE [
    pvID: PhysicalVolume.ID, type: BootFileType];
VoidVolumeBootFile: --OthelloOps-- PROCEDURE [
    lvID: Volume.ID, type: BootFileType];
Volume: --NSFile-- TYPE = System.VolumeID;
volumeClosed: --EventTypes-- Supervisor.Event;
VolumeHandle: --Floppy-- TYPE [2];
volumeID: --NSAssignedTypes-- AttributeType = 22;
VolumeID: --System-- TYPE = RECORD [UniversalID];
VolumeNotClosed: --OthelloOps-- ERROR;
VolumeNotOpen: --BTree-- ERROR [volume: Volume.ID];
volumeOpened: --EventTypes-- Supervisor.Event;
VolumeType: --PhysicalVolume-- TYPE = {
    notPilot, probablyNotPilot, probablyPilot, isPilot};
voyeurSocket: --NSConstants-- System.SocketNumber;
WaitAttentionProcedure: --Stream-- TYPE = PROCEDURE [sH: Handle] RETURNS [Byte];
WaitForAttention: --Stream-- PROCEDURE [sH: Handle] RETURNS [Byte];
WaitForConfirmation: --UserInput-- PROCEDURE
    RETURNS [place: Window.Place, okay: BOOLEAN];
WaitForMail: --RetrieveDefs-- PROCEDURE [handle: Handle];
WaitForRequest: --PacketExchange-- PROCEDURE [
    h: ExchangeHandle, requestBlk: Environment.Block,
    requiredRequestType: ExchangeClientType ← unspecified]
    RETURNS [rH: RequestHandle];
WaitForScanLine: --UserTerminal-- PROCEDURE [scanLine: INTEGER];
WaitNoButtons: --UserInput-- PROCEDURE;
WaitTime: --NetworkStream-- TYPE = LONG CARDINAL;
WaitTime: --PacketExchange-- TYPE = LONG CARDINAL;
Wakeup: --FormSW-- PROCEDURE [Window.Handle];
WakeUp: --RavenFace-- PROCEDURE;
WellFormed: --NSString-- PROCEDURE [s: String] RETURNS [BOOLEAN];
WestEast: --System-- TYPE = MACHINE DEPENDENT{west, east};
White: --Display-- PROCEDURE [window: Handle, box: Window.Box];
wildCard: --CH-- CHARACTER = 52C;
wildCard: --NSName-- CHARACTER = 52C;
wildCardCharacter: --NSName-- NSString.Character;
window: --DebugUsefulDefs-- READONLY Window.Handle;
window: --HeraldWindow-- READONLY Window.Handle;
Window: --Space-- TYPE = RECORD [
    file: File.File, base: File.PageNumber, count: Environment.PageCount];
WindowForFile: --FileWindow-- PROCEDURE [fileName: LONG STRING]
    RETURNS [Window.Handle];
WindowNowDelinked: --Scrollbar-- PROCEDURE [window: Window.Handle];
WindowNowEnlinked: --Scrollbar-- PROCEDURE [window: Window.Handle];
```

```
Word: --Environment-- TYPE = [0..177777B];
Word: --Stream-- TYPE = Environment.Word;
WordBoolean: --FormSW-- TYPE = RECORD [
    SELECT OVERLAID * FROM f1 => [b: BOOLEAN], f2 => [w: WORD], ENDCASE];
Words: --MDSStorage-- PROCEDURE [nwords: CARDINAL] RETURNS [base: POINTER];
Words: --NSFile-- TYPE = LONG DESCRIPTOR FOR ARRAY CARDINAL OF UNSPECIFIED;
Words: --Storage-- PROCEDURE [nwords: CARDINAL] RETURNS [base: LONG POINTER];
WordsForBitmapUnder: --Window-- PROCEDURE [window: Handle] RETURNS [CARDINAL];
WordsForString: --NSString-- PROCEDURE [bytes: CARDINAL] RETURNS [CARDINAL];
WordsForString: --String-- PROCEDURE [nchars: CARDINAL] RETURNS [CARDINAL];
WordsInPacket: --CommOnlineDiagnostics-- TYPE = MACHINE DEPENDENT{
    all0s, all1s, incrWords, allConstant, dontCare};
wordsPerPage: --Environment-- CARDINAL = 256;
wordsPerPage: --Space-- CARDINAL = 256;
Workstation: --CHLookup-- TYPE = MACHINE DEPENDENT RECORD [
    address(0:0..95): System.NetworkAddress, location(6:0..63): NSString.String];
WorkstationDescribe: --CHLookup-- Courier.Description;
WorkstationPt: --CHLookup-- TYPE = LONG POINTER TO Workstation;
wpp: --MSegment-- CARDINAL = 256;
Write: --Floppy-- PROCEDURE [
    file: FileHandle, first: PageNumber, count: PageCount, vm: LONG POINTER];
WriteDeletedSectors: --FloppyChannel-- PROCEDURE [
    handle: Handle, address: DiskAddress, buffer: LONG POINTER,
    count: CARDINAL ← 1, incrementDataPtr: BOOLEAN ← TRUE]
    RETURNS [status: Status, countDone: CARDINAL];
WriteMsg: --CommOnlineDiagnostics-- TYPE = PROCEDURE [msg: RS232CTestMessage];
WriteOnly: --MFile-- PROCEDURE [
    name: LONG STRING, release: ReleaseData, type: Type,
    initialLength: InitialLength ← dontCare] RETURNS [Handle];
WriteOnly: --MStream-- PROCEDURE [
    name: LONG STRING, release: ReleaseData, type: MFile.Type] RETURNS [Handle];
WriteProc: --MailParse-- TYPE = PROCEDURE [string: LONG STRING];
WriteProtected: --DebugUsefulDefs-- ERROR [page: Environment.PageNumber];
WriteReal: --Real-- PROCEDURE [
    cp: PROCEDURE [CHARACTER], r: REAL,
    precision: CARDINAL ← DefaultSinglePrecision, forceE: BOOLEAN ← FALSE];
WriteSectors: --FloppyChannel-- PROCEDURE [
    handle: Handle, address: DiskAddress, buffer: LONG POINTER,
    count: CARDINAL ← 1, incrementDataPtr: BOOLEAN ← TRUE]
    RETURNS [status: Status, countDone: CARDINAL];
WriteStream: --FileTransfer-- PROCEDURE [
    conn: Connection, file: FileName.VFN, veto: VetoProc ← NIL,
    showDates: BOOLEAN ← FALSE, creation: Time.Packed, type: StreamType ← remote]
    RETURNS [Stream.Handle];
x860ToFileServer: --NSConstants-- System.SocketNumber;
xerox800: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
xerox850: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
xerox860: --RS232CCorrespondents-- RS232CEnvironment.Correspondent;
xmtL0: --ProtocolCertification-- Stage;
xmtL1: --ProtocolCertification-- Stage;
xorBoxFlags: --Display-- BitBltFlags;
xorFlags: --Display-- BitBltFlags;
xorGrayFlags: --Display-- BitBltFlags;
YesOrNo: --OnlineDiagnostics-- TYPE = {yes, no};
Yield: --Process-- PROCEDURE;
ZeroDivisor: --Runtime-- SIGNAL;
```

```
zeroMaxLengthNames: --CH-- READONLY Name;
zeroScratchMem: --PilotSwitches-- PilotDomainA = 64C;
ZoneTooSmall: --MDSStorage-- ERROR [p: POINTER];
ZoneTooSmall: --Storage-- ERROR [p: LONG POINTER];
```

Index

OFFICE SYSTEMS DIVISION

Reader's Feedback

Xerox's Technical Publications Departments want to provide documents that meet the needs of all our product users. Your comments help us correct and improve our publications. Please take a few minutes to respond. If you have comments on the product this document describes, contact your Xerox representative.

1. Did you find any errors in this publication? What were they? On which pages?

_____

_____

2. Were there any areas that were hard to understand because of descriptions or wording? What were they? Where?

_____

_____

3. Did this publication give you all the information you needed? If not, what was missing?

_____

_____

4. Was this manual at the right level for your needs? If not, what other types of publications do you need?

_____

_____

5. What *one thing* could we do to improve this manual for you?

_____

NAME_____DATE_____

TITLE_____ COMPANY_____

ADDRESS _____

CITY_____STATE_____ZIP _____

XDE3.0-4001