# MACINTOSH® TOOLBOX INTERFACE

Macintosh Toolbox Interface (TI Part Number 2559092-0001)

Original Issue ............................................December 1988

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013.

ExperTelligence, Inc.
5638 Hollister Avenue
Goleta, California 93117

Texas Instruments Incorporated
Data Systems Group
P.O. Box 2909 - M/S 2151
Austin Texas 78769-2909

# Contents

13 Dialog Manager

14 Desk Manager

15 Scrap Manager

16 Toolbox Utilities

17 Package Manager

18 Memory Manager

# ABOUT THIS MANUAL

## Purpose

This reference manual documents each Toolbox function and provides examples of how to use them.

For readability and ease of use we have kept the descriptions as short and simple as possible. In most cases, these brief descriptions will be more than sufficient. If you desire additional information about a particular routine, please refer to the volume and page number of *Inside Macintosh* listed in brackets after the definition of each routine.

## Who Should Use This Manual

This reference manual is designed for Lisp programmers who need to use the Toolbox routines. All the examples are written in Lisp and the arguments and conventions refer to Lisp data types. Nevertheless, this manual might not meet your casual expectations.

The problem is that these Toolbox functions merely give access to the Macintosh operating system. The Macintosh OS, on the other hand, is a piece of sophisticated software which is something of a milestone in the computer industry. When you decide to use the Macintosh Toolbox, you have committed yourself to learning a new OS which is markedly different from anything which has come before.

You must understand the structure of the Macintosh OS and its unique way of doing business. For example, the Macintosh's highly developed notion of a *resource* is both unique and central to the understanding of how many Macintosh features are implemented. Therefore, knowing how to call the **tb:!OpenRefFile** function is one thing; knowing what to do with an open "Res" file is something else again.

Another difference is that there are many varied and subtle interactions among Toolbox functions. The **tb:!WaitNextEvent** function you see in main event loops is just one example. You are not just calling some mathpack trig utility; you are manipulating the internal data structures of a complex OS in real time. You bear sole responsibility for supporting all Macintosh OS conventions ... whether you are aware of them or not.

## Structure of the Document

This document is composed of 29 chapters organized in the same way as *Inside Macintosh*. Each chapter begins with a short introduction describing the specific Toolbox feature.

## Style and Conventions

The following notational conventions help you recognize Toolbox functions, methods, and arguments. The table below defines the three typefaces used in this manual and their respective meanings.

| Typeface | Meaning |
|---|---|
| **boldface** | Indicates a function, method, or symbol name. |
| *italics* | Indicates an argument to a function or method. Names in italics can be replaced by any appropriate value you choose to substitute. |
| `monowidth` | Indicates sample code or program output. |

The naming convention adopted for the Toolbox Interface says that if *Inside Macintosh* documents a symbol *name*, then that symbol appears in the Toolbox as **tb:!***name*.

Furthermore, you will notice that many of the symbols mentioned in the text appear in upper and lower case. For example, the flavor which implements a Macintosh color graph port appears as **tb:!cGrafPort** rather than all lower case as most Explorer™ symbols are documented. These mixed case symbols are simply following the convention established in the Macintosh's Pascal-oriented documentation. Since the Lisp system uppercases all symbols internally, the symbols **tb:!cGrafPort**, **tb:!cgrafport**, and **TB:!CGRAFPORT** are all the same as far as Lisp is concerned.

# Chapter 1
# EVERYTHING YOU ALWAYS WANTED TO KNOW ABOUT THE TOOLBOX INTERFACE

## Introduction

1.1 The standard Macintosh interface is probably the most important aspect of any Macintosh application. It is this interface that sets those applications apart as the easiest to learn and use.

If you are familiar with one Macintosh application, MacDraw® for example, you should be immediately familiar with navigating any other Macintosh application.

To the user, the screen appears to act like a desktop, and the various windows like sheets of paper. Actually, this appearance is an illusion carefully maintained by the programmer. For every possible action the user can make, including dragging, growing, zooming, closing, scrolling through the contents, or switching to another window, the programmer must call the necessary functions to maintain this illusion.

The Macintosh Toolbox is a collection of more than 700 specialized routines stored in read-only memory (ROM) that can be called to perform such tasks as drawing a rectangle, displaying a window, or pulling down a menu. While the Macintosh Toolbox is very complex and difficult to master using traditional languages, programming is made easier with the Toolbox Interface.

Rather than providing a simplistic, statically typed interface using macros or requiring developers to invoke remote procedure calls to programs written in C, the ToolBox Interface was implemented with the dynamic nature of Lisp in mind. It is implemented with rapid debugging, accurate error checking, and an object-oriented approach to programming in mind. The ToolBox Interface lets you write code calling Macintosh Toolbox routines in the same way you would call a typical Lisp function on any Lisp machine.

The ToolBox Interface consists of two parts: Lisp code on the microExplorer™ and assembly language code in the form of a Toolbox server application on the Macintosh. The microExplorer consists of a Texas Instruments Lisp chip (generally referred to as a Lisp machine) and a Macintosh connected by the NuBus™. The Lisp machine can send calls to and receive calls from the Macintosh via the NuBus.

## The TbServer: How the microExplorer Communicates With the Macintosh

1.2 When a Macintosh function is called from Lisp, the number and type of the arguments are checked. If correct, the trap number and the data from the arguments are then placed into a packet of memory (actually, a piece of memory that can be accessed by both the Macintosh and the microExplorer). If no return value is needed (for example, when calling a line drawing function), the packet is queued and Lisp continues execution without waiting for the trap to be run. Of course, if the trap returns a value, Lisp must wait for the data.

On the Macintosh side of the bus, a Toolbox server is receiving the packets sent to it by Lisp and invoking the appropriate Toolbox functions. This process is complicated by the fact that there are many different Toolbox calling conventions, and data can be passed in byte, word, or long lengths. The server handles this for you by using the information contained in the "TRAP" resources.

There can be up to eight servers running at once. Each application you define that uses the ToolBox Interface must have its own Toolbox server. The first of these servers is built into the microExplorer driver. It communicates on *application-channel* 8. For the most part, this server is hidden from users.

An unobvious feature of these Toolbox servers is that they automatically take care of the standard initialization functions common to all Macintosh applications: **tb:!InitGraf**, **tb:!InitFonts**, **tb:!InitWindows**, **tb:!InitMenus**, **tb:!TEInit**, **tb:!InitDialogs**, and **tb:!InitCursor**. The documentation for each of these traps mentions that there is no need for you to call it directly.

Since the *default* Toolbox server is running in the microExplorer application, a fatal Toolbox error may cause the microExplorer application to terminate, thereby crashing the Lisp machine. Therefore, when making Toolbox calls, you should launch an independent Toolbox server. MultiFinder™ launches the application named "TbServer" (note the lack of a separating space) in the :microExp:MACSYS: folder. A fatal error in one of these stand-alone Toolbox server applications will not harm the Lisp machine. At any time, you can kill the Toolbox server. Of course, you can launch and kill servers as much as you want throughout a session. See the functions defined below to determine how to do this.

**tb:launch-default-tb-server** &optional *kill-and-relaunch-p* Function

**tb:launch-default-tb-server** launches a Toolbox server and initializes it. Normally, this function can be called from your login-init file. This function causes a Toolbox server to be available to run your calls to the Toolbox from any microExplorer process.

**tb:kill-default-tb-server** Function

**tb:kill-default-tb-server** causes the default Toolbox server to shut down and reset itself. If a Toolbox server has an untimely exit (whether by signaling a Macintosh system error, by doing an exit to shell from any debugger, or by explicitly calling the **tb:!ExitToShell** trap on the microExplorer), you still must call **tb:kill-default-tb-server**. Calling this function adjusts the value of certain global variables and allows you to relaunch a new server without causing problems.

Alternately, instead of calling **tb:kill-default-tb-server** before you call **tb:launch-default-tb-server** again, you may simply call **tb:launch-default-tb-server** with an argument of t which is the equivalent of calling **tb:kill-default-tb-server** followed by **tb:launch-default-tb-server**.

To determine which Toolbox servers are running, look at the Apple Menu under the MultiFinder applications list. You can switch to other applications or to the microExplorer by clicking in one of their windows or selecting them from the Apple® Menu.

If a Toolbox server exits unexpectedly, there are several ways to shut it down and bring it back up. If you are running a debugger (such as ICOM's TMON™ debugger), system errors will cause you to enter the debugger automatically. From the debugger, you can exit to shell and that will close the default Toolbox server. Be careful when exiting to shell under MultiFinder because multiple applications can be running at the same time. Make sure you are exiting Toolbox server and not some other application. Remember, when you return to the Lisp environment, be sure to execute the form **(tb:launch-default-tb-server t)** again. The **t** argument makes certain the microExplorer side and the Macintosh side are in sync.

If no main event loop is running in the Toolbox server (this is the default), you may have to click the mouse several times in another application's window in order to switch from the Toolbox server to that application.

## What's in a Name?

1.3 You may notice that some of the names of the traps documented in this manual differ from the names as they appear in *Inside Macintosh*. The most obvious difference is that most of the names in this manual begin with a bang ("!") character. The naming convention followed in most of the traps is a bang followed by the assembly language name of the trap, not the high level, or Pascal, name. Most traps are named the same in Pascal and assembly language, but do not always assume the names will be the same. For example, the Macintosh trap PBOpen exists in the Toolbox Interface as **tb:!Open**.

## What Are VARs and How Did They Get Into Lisp?

1.4 VAR variables are a contrivance invented for Pascal because Pascal functions cannot return multiple values. Pascal treats VARs differently depending on the data length of the variable. Unfortunately, these declarations found their way into the Macintosh Toolbox and consequently were introduced into Lisp in order to avoid changing calling conventions in dozens of traps. For each trap that requires VAR arguments, there is an alternate trap name that performs the same operation but does not return information with VAR arguments. Instead, these traps return the information as function results. These trap names are identical to the VAR trap names except the ! is missing. For example, the VAR-less version of **tb:!PtToAngle** is **tb:PtToAngle**.

### True VAR Variables

1.4.1 The only true VAR variables for the microExplorer are: (VAR integer), (VAR longint), (VAR string), (VAR character), and (VAR restype). When calling traps that use these VAR variables, you must wrap the variable in a VAR form.

*Example:*

```
(tb:!GetResInfo han (VAR id) (VAR type) (VAR name))
```

The variables **id**, **type**, and **name** may be local or global, and may or may not have any value. After invoking the Toolbox function, these variables will be set to their respective values.

The Lisp compiler is very smart. When compiling:

```
(defun foo ()
  (let ((type "")
        (name "")
        (id   nil))
    (tb:!GetResInfo han (VAR id) (VAR type) (VAR name))
    type))
```

the compiler notices that both **name** and **type** are constants and doesn't understand VARs, so it actually sets both variables to the same empty string. Thus, this function will actually return the wrong value because it treats **type** and **name** as the same string. Instead, define **foo** as follows:

*Example:*

```
(defun foo (&aux type name id)
  (let ((type nil)
        (name nil)
        (id   nil))
    (tb:!GetResInfo han (VAR id) (VAR type) (VAR name))
    type))
```

### Pseudo VAR Variables

1.4.2 Since Macintosh data types are really instances, window pointers and pointers may optionally be passed as VARs. For VAR window pointers (e.g., **tb:!FindWindow**) and VAR control records (e.g., **tb:!FindControl**), it is best to call using the VAR form because then the trap will return the *same* instance (as opposed to another instance with the same pointer). See Chapters 9 and 10 on the Window Manager and the Control Manager, respectively.

VAR pointers are often used to pass information into a trap as well as to return information. Therefore, if you pass a VAR argument, make sure that its value is an instance of **tb:mac-pointer.**

### VARs That Aren't

1.4.3 Sometimes traps are declared to take VAR arguments when they really take a data structure that may be modified by the trap. In this situation, just pass the data structure.

Since VAR handles are instances in Lisp, just pass the instance to the trap and the **:handle** instance variable will be modified by the trap.

VAR FontInfo, SFReply, PenState, Points, Rectangles, and EventRecords are in another category. Since these instances are true Lisp objects (not pointers to Macintosh objects), you need only pass the instance. Do not put the (VAR...) form around these arguments. The functions with these argument types do, however, wait for the instances to be modified by the ROM call.

## Who Lives Where?

1.5 Most data structures used by Macintosh Toolbox traps reside in the Macintosh heap. Pointers or handles to these objects are encapsulated by instances of an appropriate flavor. The two most important data types used by the Toolbox are **tb:mac-handle** and **tb:mac-pointer**. For the Toolbox Interface, these and all data types are instances of flavors. This means that the address of a **tb:mac-pointer** is really stored in the instance variable **:pointer** of an instance of the flavor **tb:mac-pointer.**

For example, the result of doing a **(make-instance 'tb:window)** is an instance of a Macintosh window. It contains, however, only a single instance variable **:pointer** which points to where the window is stored in the Macintosh heap. The rest of the information about the window resides entirely in Macintosh memory. The only way to access or change the Macintosh data structure is through so-called instance accessors. Thus, while it may appear to a user that a field like window kind is an instance variable, it is not.

A handful of data structures which are both small and frequently used are actually true instances, and the information is copied back and forth when the trap is invoked. The most common examples of these are rectangles and points. Because these data structures are true instances, it is faster to use the methods provided to do calculations directly on rectangles and points than to send the information across the bus, and then wait for MultiFinder to give the CPU to the server and return the result.

Other data structures that reside as true Lisp objects include: **tb:eventrecord, tb:sfreply,** and **tb:fontinfo.**

## Toolbox Interface Structures in the Load Band

1.6 Each **tb:mac-handle** and **tb:mac-pointer** instance normally holds a dynamic Macintosh-relative address. These addresses are dynamic in the sense that they become invalid when the Macintosh is rebooted. If these instances should be saved in a load band, then on the next reboot they will effectively introduce *random* Macintosh addresses into Toolbox execution. Therefore, pointers and handles may ***not*** be saved in a microExplorer load band.

**CAUTION: Handles, pointers, and any Toolbox Interface structures which include a handle or a pointer *cannot* be saved in a microExplorer load band. This applies to any Toolbox data structure which include handles or pointers directly or indirectly (and that includes almost everything).**

The only exceptions are the trivial case of NIL handles and pointers and the special case of a constant pointer to a Macintosh global variable--the only Macintosh addresses guaranteed to remain constant across boots.

## Procedure Pointers

1.7 Many traps take procedure pointers as arguments. These arguments are pointers to routines which are expected to lie in Macintosh memory. These routines are called during the execution of the traps and are expected to follow Pascal calling conventions. Using the name of a Lisp function as a procedure pointer will not work. For this reason it is best to pass **tb:!nilPtr** in these parameters. However, if you install a routine into Macintosh memory (perhaps written in assembly language or another language on the Macintosh), you may use a pointer to it with these routines. If you do so, be very careful that the routines you write follow the correct calling conventions, as there is no checking done whatsoever on these routines. Read *Inside Macintosh* carefully for the descriptions of the traps which use procedure pointers for details on these calling conventions.

## Heap Management

1.8 All of the Macintosh memory allocated to a particular Macintosh application is located in its heap. This means that if you plan to create large handles or use lots of color pictures, etc., you must allocate enough memory in the "SIZE" -1 resource to hold the data you want to keep in Macintosh memory. In addition, heed the warnings in the *Inside Macintosh* chapter that discusses memory management including: not keeping pointers to unlocked handles, not leaving large locked handles in the middle of the heap, and remembering to dispose of handles no longer in use.

For your safety and convenience, many data structures that are defined by *Inside Macintosh* as pointers to Pascal records are actually allocated as handles. As traps using these data structures are invoked, the handle is first locked, then dereferenced for you automatically. Upon completion of the trap, the locking status is returned to its original condition.

## Flavors and Records

1.9 To allocate new Macintosh data structures, the universal way is make an instance of its flavor. The **:init** method will automatically create the Macintosh object of the correct size in one of two ways. Flavors that mix in **tb:AutoHandleTo** (like **tb:RGBColor**) automatically create a handle of the necessary size, while flavors that mix in **tb:SystemCreatedTo** (like **tb:window**) automatically call the appropriate trap to allocate the object (in this case, **tb:!NewCWindow**).

At times, you may need to get an instance without having it automatically generate a Toolbox data structure. Use **tb:make-instance-no-init** for this purpose.

## Accessors and Fields

**1.10** Since the Macintosh data structures exist in Macintosh memory, not microExplorer memory, a mechanism is needed to access the fields of Toolbox records. Instance accessors perform this operation. Since all Macintosh data types are defined as flavors, the instance accessors are methods. You can access a field in the same way that you get the value of an instance variable. Fields may be set by passing a new value for a field to the method or by using a **setf** form. Since these fields are not in Lisp memory, some overhead does exist in accessing the data. You can look at all of the fields in many records by doing a **describe** on the record. Similarly, a **tb:describe-class** on a flavor name will tell you the instance accessors and their offsets defined for a flavor.

MultiFinder considers Window Manager data structures as being owned by the Window Manager. Therefore, these data structures should never be modified directly by an application. This includes all of the fields in a grafPort and the window record itself. Instead use the traps provided by the Toolbox to modify these fields.

## Not in ROM

**1.11** A handful of Toolbox routines are not in the Macintosh ROM. Most of these perform very simple operations like fetching a value from low memory. Many of these routines have Lisp equivalents.

In managers like the File Manager where both high and low-level function calls are provided, only those in ROM are supported. These traps are more complete and provide more control than the so-called high-level Pascal functions.

## That is Illogical

**1.12** Given the limited development environment of the Macintosh, the authors of the Toolbox provided a number of low-level arithmetic traps that make no sense to a Lisp programmer. Logical operators, bignums, and trig functions are more complete in the Lisp environment and much faster than sending the data across the bus for processing. For these reasons, some traps have not been implemented. Others were implemented for completeness, but should not be used.

## MultiFinder

1.13 Since all microExplorer applications must run under MultiFinder, it is important to emphasize a few points contained in the *MultiFinder Development Package*. Most importantly, observe the limitations mentioned above regarding the Window Manager and understand that the Event Manager has been modified extensively. You should use the new Event Manager call !WaitNextEvent. Call this function often enough so the system does not lock up without giving the user a chance to do something. (Remember, MultiFinder does not do preemptive scheduling).

It is very important to tell MultiFinder how much memory your application requires using the "SIZE" -1 resource.

Any Macintosh application (e.g., MacWrite®) may be launched with the Lisp function `launch` (e.g., **(tb:launch "hd:MacWrite")**).

## User Interface Guidelines

1.14 Macintosh programmers have gone to a great deal of trouble to make their programs operate in a consistent manner, yet it is possible to use the traps to create applications that are inconsistent with the established guidelines. Before you create a new Macintosh application, carefully read Chapter 2 of *Inside Macintosh*, The Macintosh User Interface Guidelines. Pay particular attention to the discussion supporting the Edit menu (cut, copy, paste, etc.) so that data can be transferred between your application and other Macintosh applications.

Adapting an existing application to run on the Macintosh can be particularly difficult, but no application has been commercially successful unless those adaptations have been made.

For design tools that make writing consistent Macintosh applications easier and faster or for assistance in converting an existing application, contact ExperTelligence.

## Debugging

1.15 While the dynamic runtime error checks prevent you from making many errors, the low-level non-typed nature of the Macintosh ROM tends to produce errors that are sometimes difficult to debug. You can use debuggers like TMON to view your heap in hexadecimal notation. To enter the debugger, simply type **(tb:!Debugger)**.

**CAUTION: Many traps can write anywhere in memory (even outside your application's heap). There is no hardware protection, so it is possible to crash the Macintosh system. If you make catastrophic errors within your *own* heap, however, you can often exit-to-shell, restart your Toolbox server, and continue without rebooting.**

When you enter the debugger, make sure that you are in your application's heap. Look at the Macintosh global variable in location #x910 to check the name and avoid confusion.

## Putting It All Together: Making a Macintosh Application

1.16 All microExplorer applications that use the Toolbox Interface are invoked the same way. Each application is represented by a double-clickable icon that exists somewhere on the Macintosh desktop. This icon is linked via a "NAME" resource to Lisp code on the microExplorer side of the machine.

**tb:define-mac-application** is the macro that links the Macintosh icon to the Lisp code and the Lisp code to the Macintosh icon, allowing you to launch your application from either the Macintosh or the Lisp machine side of the microExplorer.

Every application has an entry point, a single function that launches and starts running the entire application. It is this function that we use to **tb:define-mac-application**.

The TbServer application is the Toolbox server. It provides access to the various Toolbox routines. During development of your application, you'll want to use the default Toolbox server provided. It's very handy for testing and debugging. You can add any necessary resources to the TbServer application (pictures, icons, cursors, etc.).

Once your application is debugged, however, you need to create your own copy of the TbServer application to run your application as a stand-alone, double-clickable Macintosh application. To do that, perform the following operations:

On the Macintosh side:

- Make a copy of the TbServer application and give it your application's name.
- Copy any pictures, icons, or other resources required by your application into your copy of the TbServer application using ResEdit or some other resource editor.

On the Lisp machine side:

- Link your application to the Macintosh icon described above using the Lisp macro **tb:define-mac-application**. This creates a resource of type "NAME" that contains the flavor to instantiate during the boot process (that is, when your application icon is double-clicked).

**tb:define-mac-application** *name* &optional *args* &key **:directory :lisp-function :server-name** Macro

*Name* is the name of your application. When launching from the Lisp side, *name* is the symbol that you pass to **tb:launch-mac-application**. *Args* are the list of arguments to the Lisp function, if any, that are passed to **:lisp-function**'s Lisp function. The **:lisp-function** value is a symbol that is the name of the entry point to the Lisp application that you want to run when you **tb:launch-mac-**

**application** or double-click on the corresponding Macintosh icon. The **:server-name** argument is a string that contains the current name of your application icon. **:server-name** is very important when launching your application from the Lisp side of the microExplorer because without this information the system is unable to locate the corresponding icon.

When doing a **tb:launch-mac-application** the microExp folders on all mounted volumes are searched for the application named by the keyword **:server-name**. If **:directory** is supplied (or a list of directories), a search of all mounted volumes for **:directory** (a folder name) is done. This search is only one deep, that is, **:directory** should be on a mounted volume's desktop. When the directory is found it is searched for the application named by **:server-name.** When the file is found a "NAME" resource is added to its resource file. This "NAME" resource contains the *name* given above.

*Example:*

```
(tb:define-mac-application color-qix (&optional length)
  (:directory     "microexp:toolbox-examples:"
   :lisp-function 'tb:tb-qix
   :server-name   "color-qix"))
```

**tb:launch-mac-application** *name* Function

*Name* is the symbol that you used in **tb:define-mac-application.** Note that the search path is defined as described above when doing your **tb:define-mac-application.**

*Example:*

```
(tb:launch-mac-application 'color-qix)
```

**tb:mac-application-cleanup** &optional *reinitialize-p* Function

Shuts down all Macintosh applications launched from the microExplorer, including the default Toolbox server. If *reinitialize-p* is true, **tb:mac-application-cleanup** will reinitialize the application channels.

**tb:select-application** &optional (*application* **tb:CurApName**) Function

This function causes the MultiFinder to select *application* as the current application. That is, this is the programmatic version of clicking on a different window or clicking on an application name in the Apple Menu. *application* may be an instance of **tb:mac-pointer** or **tb:mac-handle** or it may be a string matching the name of the application as it would appear in the Apple Menu.

# Chapter 2
# RESOURCE MANAGER

## Introduction

2.1 The Resource Manager is a collection of routines used to manage resources. Resources are data structures that define various objects used by the Macintosh: menus, windows, dialog boxes, and so on. Resources are kept in a resource file. At the beginning of every resource file is a resource map that contains information about all the resources in the resource file. When the resource file is opened, its resource map is read into memory. The resource map tells the Resource Manager how many resources are in the file, their types, their IDs, and their names. The individual resources in the resource file are loaded into memory as needed.

The resources in memory can be made purgeable, meaning they can be thrown out when the Memory Manager needs more memory. When the Memory Manager purges a resource, it is removed from memory such that it can be reloaded when it is later needed. Making non-vital resources purgeable gives the Memory Manager greater flexibility and generally improves the performance of the machine.

Resources are distinguished by two properties: their resource type and their resource ID. There are about fifty Apple-defined resources types, such as "MENU", "WIND", and "DITL". Resource types are listed in Appendix A. The resource type is a string of four characters where case and blanks are significant. The resource ID is a 16-bit integer. Up to 65,536 different resources of the same type can exist, but many of these resource IDs are reserved and are not available for your use. A resource name is a string of up to 255 characters.

## Creating, Opening and Closing Resource Files

2.2 The following traps are used to create, open, and close resource files. Macintosh files are divided into two *forks*: the *data* fork and the *resource* fork. The data fork is always empty. The resource fork contains all the individual resource's data and a resource map, which includes a list of the resources in the file.

**tb:!CreateResFile** *fileName* [I-114] Function

Creates a file with the name *fileName* on the current volume or in the current working directory in HFS (Hierarchical File System), and puts a default resource map in the resource fork.

*Example:*

```
(tb:!CreateResFile "mySampleResFile")
```

Before you can work with a newly created file, you must open it with the trap **tb:!OpenResFile**.

**tb:!OpenResFile** *fileName* [I-115] Function

Opens the resource fork of the file *fileName*, loads in it's resource map, and returns a refNum (reference number) which is used when you need to specify the file.

*Example:*

```
(setf resFileRefNum (tb:!OpenResFile "mySampleResFile"))
=> 378
```

**tb:!OpenRFPerm** *fileName VRefNum permission* [IV-17] Function

Similar to **tb:!OpenResFile** except this trap allows you to define a *permission* and a *VRefNum* for the file. See Chapter 21 on the File Manager for information on the *VRefNum* argument. For available permissions, see **tb:!fsCurPerm** *et al.*

**tb:!CloseResFile** *refNum* [I-115] Function

Closes the file which has a reference number *refNum* and removes that file's resource map from memory.

## Checking for Errors

2.3 This routine checks for errors.

**tb:!ResError** [I-116] Function

Checks to see if the last Resource Manager trap used was successful and returns an error code if it was not.

This trap is normally needed because Resource Manager traps do not individually return result codes. Instead, you typically call the trap and then you call **tb:!ResError** to see if the trap worked.

However, a feature of the Toolbox Interface is that it will automatically signal non-zero result codes for you if the global variable **tb:*signal-mac-oserr*** is true. Therefore, you will need **tb:!ResError** only if you set this variable to false.

## Setting the Current Resource File

2.4 The following traps modify the order in which the resource maps in memory are searched.

These traps manipulate only the current resource file, the first resource file in the open resource file list. That is, they only search "one deep" into the list. When you read or get information about a resource of a particular resource type or resource ID, all the open resource files are searched for that resource, not just the current file. To force the Resource Manager to search only the current resource file, use the "one-deep" traps.

**tb:!CurResFile** [I-116] Function

Returns the *refNum* (reference number) of the current resource file.

*Example:*

```
(setf theCurrentResFile (tb:!CurResFile)) => 284
```

**tb:!HomeResFile** *theResource* [I-117] Function

Searches through the resource maps of all open resource files for a resource with the handle *theResource*. If found, it returns a reference number to the resource file.

**tb:!UseResFile** *refNum* [I-117] Function

Makes the resource file with the reference number of *refNum* the current resource file, the first to be searched by the Resource Manager.

## Getting Resource Types

2.5 The following traps return resource types or the number of resource types.

**tb:!CountTypes** [I-117] Function
**tb:!Count1Types** [IV-15] Function

**tb:!CountTypes** returns the number of resource types in all open resource files.

**tb:!Count!Types** is similar except that it searches only "one deep" in the current resource file.

*Example:*

```
(tb:!CountTypes) => 38
```

**tb:GetIndType** *index* [I-117] Function
**tb:!GetIndType** VAR *theType index* [I-117] Function
**tb:Get1IndType** *index* [IV-15] Function
**tb:!Get1IndType** VAR *theType index* [IV-15] Function

**tb:GetIndType** returns the *index*'th resource type in all open resource files. The maximum value for *index* is the value returned by **tb:!CountTypes**.

**tb:!GetIndType** is similar except that it modifies *theType* to be the resource type.

**tb:Get1IndType** and **tb:!Get1IndType** are similar to **tb:GetIndType** and **tb:!GetIndType** respectively except that they search only "one deep" in the current resource file.

*Example:*

```
(tb:getIndType 6) => "FONT"
(tb:!GetIndType (VAR theType) 6)
theType => "FONT"
```

## Getting, Counting, and Disposing of Resources

2.6 These routines get, count, and load resources.

**tb:!SetResLoad** *load* [I-118] Function

Normally, when you call a resource that is not in memory, it is loaded into memory from the file. However, if you set resLoad to NIL by:

*Example:*

```
(tb:!SetResLoad nil)
```

the resource is not automatically loaded from the file if not already in memory.

**CAUTION: Do not use this trap unless you fully understand the Resource Manager.**

**tb:!CountResources** *theType* [I-118] Function
**tb:!Count1Resources** *theType* [IV-15] Function

**tb:!CountResources** returns the number of resources of type *theType* in all open resource files.

**tb:!Count1Resources** is similar except that it searches only "one deep" in the current resource file.

*Example:*

```
(tb:!CountResources "FONT") => 95
```

**tb:!GetIndResource** *theType index* [I-118] Function
**tb:!Get1IndResource** *theType index* [IV-15] Function

**tb:!GetIndResource** indexes into the resources of type *theType*. This trap returns the handle to the *index*'th resource of *theType*. There is no relationship between a resource's ID and its index.

**tb:!Get1IndResource** is similar except that it searches only "one deep" in the current resource file.

NOTE: The trap **tb:!UseResFile**, which changes the first resource file to be searched, does not affect the order of the resources in the resource map.

To get the handle to the first "FONT" resource in the open resource files, do the following:

*Example:*

```
(setf theRes (tb:!GetIndResource "FONT" 2))
```

**tb:!GetResource** *theType theID* [I-119] Function
**tb:GetResource** *theType theID* [I-119] Function
**tb:!Get1Resource** *theType theID* [IV-16] Function
**tb:Get1Resource** *theType theID* [IV-16] Function

**tb:!GetResource** returns a handle to the resource with a resource type *theType* (a string of four characters) and a resource ID number *theID*. If the resource is not found, **tb:!GetResource** returns a NIL handle (i.e., a handle of zero) and **tb:!ResError** returns noErr. You must either check the handle returned by **tb:!GetResource** or use the alternate function **tb:GetResource**.

**tb:GetResource** is similar except that it signals **tb:!resNotFound** if the resource does not exist.

**tb:!Get1Resource** and **tb:Get1Resource** are similar to **tb:!GetResource** and **tb:GetResource**, respectively, except that they search only "one deep" in the current resource file.

If a resource does exist, the following example will return the handle to the resource of "MENU" resource type with a resource ID of 1.

*Example:*

```
(setf theRes (tb:!GetResource "MENU" 128))
```

**tb:!RGetResource** *theType theID* [V-30] Function
**tb:RGetResource** *theType theID* [V-30] Function

**tb:!RGetResource** is similar to **tb:!GetResource** except that if the resource is not found in the system file, ROM is searched. If the resource is not found, **tb:!RGetResource** returns a NIL handle (i.e., a handle of zero) and **tb:!ResError** returns noErr. You must either check the handle returned by **tb:!RGetResource** or use the alternate function **tb:RGetResource**.

**tb:RGetResource** is similar except that it signals **tb:!resNotFound** if the resource is not found.

**tb:!GetNamedResource** *theType name* [I-119] Function
**tb:!Get1NamedResource** *theType name* [IV-16] Function

**tb:!GetNamedResource** is similar to **tb:!GetResource** except that you must specify the resource you want by its resource type and its name.

**tb:!Get1NamedResource** is similar to **tb:!GetNamedResource** except that it searches only "one deep" into the current resource file.

If you want to get the handle to the Scrapbook desk accessory, do the following:

*Example:*

```
(setf theRes (tb:!GetNamedResource "MENU" "Apple"))
```

**tb:!LoadResource** *theResource* [I-119] Function

Ensures that a resource with the handle *theResource* exists in memory, and reloads it from its resource file if not already in memory.

**tb:!ReleaseResource** *theResource* [I-120] Function

Given a handle to the resource *theResource*, this trap sets the resource handle in the resource map to NIL and then releases the handle data. This means that the resource data contained in the handle is lost. Refer to *Inside Macintosh* for more information about using this trap.

**tb:!DetachResource** *theResource* [I-120] Function

Sets the resource handle *theResource* in the resource map to NIL, but does not release the handle of data so the resource data contained in the handle is not lost. Refer to *Inside Macintosh* for more information about using this trap.

## Getting Resource Information

2.7 All resources can be identified by the following three parameters: a resource type, a resource ID, and a resource name. The resource type (a four character string) and resource ID (a 16-bit integer) are required. Specification of a resource name (a string of up to 255 characters) is optional.

**tb:!UniqueID** *theType* [I-121] Function
**tb:!Unique1ID** *theType* [IV-16] Function

**tb:!UniqueID** returns a resource ID which has not been used by any other resource of the resource type *theType* in any of the currently opened resource files.

**tb:!Unique1ID** is similar except that it searches only "one deep" in the current resource file.

Appendix A contains a list of reserved resource types that should not be used when creating application defined resources. Resource ID's less than 128 are reserved for the system and also should not be used.

To get an unused "DITL" (dialog item list) resource ID, do the following:

*Example:*

```
(setf newResID (tb:!UniqueID "DITL")) => 7823
```

**tb:GetResInfo** *theResource* [I-121] Function
**tb:!GetResInfo** *theResource* VAR *theID* VAR *theType* VAR *name* [I-121] Function

**tb:GetResInfo** returns information about a resource with the handle *theResource*. The trap returns three values: the resource's resource ID (a 16-bit integer), the resource's resource type (a four character string), and the resource's name (a string of up to 255 characters).

**tb:!GetResInfo** is similar except that it modifies *theID*, *theType*, and *name* to be the resource ID, type, and name, respectively.

Suppose you have a resource handle *h* and wish to determine its resource type. You would do the following:

*Example:*

```
(setf h (tb:!GetResource "MENU" 128))
(multiple-value-bind (theID theType name)
    (tb:getResInfo h)
  ...body within which...
  theID    => 128
  theType  => "MENU"
  name     => "Apple"
  )
```

**tb:!GetResAttrs** *theResource* [I-121] Function

Returns the resource attributes (resAttributes) of the resource *theResource*. The resource attributes are a group of flags that tell the Resource Manager the status and properties of a resource. The following constant masks may be used to examine the resource attributes returned by this function.

| | |
|---|---|
| **tb:!resSysHeap** | [I-111] Constant |
| **tb:!resPurgable** | [I-111] Constant |
| **tb:!resLocked** | [I-111] Constant |
| **tb:!resProtected** | [I-111] Constant |
| **tb:!resPreload** | [I-111] Constant |
| **tb:!resChanged** | [I-111] Constant |

These are constant masks for the resource attributes indicating this resource is to be read into system heap, is purgeable, is locked, is protected, is to be preloaded, or has been changed (and therefore needs to be written), respectively.

**tb:!MaxSizeRsrc** *theResource* [IV-16] Function

Returns the resource size by looking at the resource map. The trap **tb:!SizeResource** also returns the resource size but is much slower as it must read the information from the disk.

**tb:!SizeResource** *theResource* [I-121] Function

Returns the size, in bytes, of the resource *theHandle*.

**tb:!RsrcMapEntry** *theResource* [IV-16] Function

Returns an offset into the resource map of the entry for the resource *theResource*.

## Modifying Resources

2.8 Except for **tb:!UpdateResFile** and **tb:!WriteResource**, the following routines described in this section change the resource map in memory and not the map in the resource file itself.

**tb:!SetResInfo** *theResource theID name* [I-122] Function

Changes the resource information of the resource specified in *theResource*. The resource ID is changed to *theID* and the resource name is changed to *name*. Do not change a resource's ID unless you know exactly what you are doing.

**tb:!SetResAttrs** *theResource attrs* [I-122] Function

Sets the resource attributes of *theResource* to *attrs*. See the trap **tb:!GetResAttrs** for the attributes table.

**tb:!ChangedResource** *theResource* [I-123] Function

Used after the resource information, resource attributes, or resource data of *theResource* has been changed. This trap sets the *resÇhanged* resource attribute of the resource. When the resource file is updated, or when the **tb:!WriteResource** trap is called with the resource *theResource*, the Resource Manager writes any changes to the resource file.

**tb:!ChangedResource** verifies that there is sufficient disk space to write out the modified file. The **tb:!ResError** trap returns an error if there is not enough disk space to save the changed resource. Check the error code returned by that trap before proceeding with the **tb:!WriteResource** trap.

**tb:!AddResource** *theData theType theID name* [I-124] Function

Add resources to a resource file. Given a handle *theData*, this trap adds *theData* to the resource map of the current resource file giving it a resource type of *theType*, a resource ID of *theID*, and a resource name of *name*.

*Example:*

```
(setf resHandle (tb:!NewHandle 30))
(tb:!AddResource resHandle "TEST" 1 "testResource")
```

**tb:!RmveResource** *theResource* [I-124] Function

Removes *theResource* from the resource map. This differs from the **tb:!DetachResource** and **tb:!ReleaseResource** traps which set the resource handle to NIL, but leave the resource in the resource map. Refer to *Inside Macintosh* before using this trap.

**tb:!UpdateResFile** *refNum* [I-125] Function

Does the required housekeeping necessary to keep the resource file consistent with the resource map. This trap updates all of the resources which have their **tb:!resChanged** attributes set to the resource file.

**tb:!WriteResource** *theResource* [I-125] Function

Checks the resChanged resource attribute of *theResource* (see **tb:!resChanged**). If resChanged is set, the trap writes the resource out to the resource file and clears the resChanged attribute of *theResource*. Unlike the **tb:!ChangedResource** trap, this trap does not check for sufficient disk space.

The following example creates a new handle, makes a resource of type "TEST" with a resource ID of 1 and then writes it to the current resource file.

*Example:*

```
(setf current (tb:!CurResFile))
(tb:!CreateResFile "ResFile")
(setf refnum (tb:!OpenResFile "ResFile"))
(setf resHandle (tb:!NewHandle 30))
(tb:!AddResource resHandle "TEST" 1 "test resource")
(tb:!WriteResource reshandle)
(tb:!CloseResFile refnum)
(tb:!UseResFile current)
```

**tb:!SetResPurge** *install* [I-126] Function

Calling (**tb:!SetResPurge t**) tells the Memory Manager to call the Resource Manager when it attempts to purge any purgeable blocks in memory. The Resource Manager then verifies that the handle is a resource, and if so, calls the **tb:!WriteResource** trap if the resource's resChanged resource attribute is set (see **tb:!resChanged**).

**tb:!GetResFileAttrs** *refNum* [I-127] Function

Returns the file attributes of the resource file with a file reference number *refNum*. The file attributes tell the Resource Manager the status and properties of the resource file. The following mask constants may be used to examine the resource file attributes returned by this function.

**tb:!mapChanged** [I-126] Constant
**tb:!mapCompact** [I-126] Constant
**tb:!mapReadOnly** [I-126] Constant

These constants are masks for the resource file attribute indicating that the resource map has been changed and therefore needs to be written, should be compacted when written, or is read-only respectively.

**tb:!SetResFileAttrs** *refNum attrs* [I-127] Function

Sets the file attributes of the resource file with a reference number *refNum*.

# Chapter 3
# QUICKDRAW

## Introduction

3.1 QuickDraw is the name given to the group of over one hundred Macintosh Toolbox traps that draw and manipulate graphic objects. There are traps for drawing and manipulating simple graphic objects such as:

- Lines
- Rectangles
- Round-cornered rectangles
- Ovals
- Arcs
- Text

and more complex graphic objects, including:

- Polygons - A group of connected straight lines.
- Pictures - A list of QuickDraw drawing commands which can be played back.
- Regions - A rectangle which contains a group of graphic objects.

In most cases, a new grafPort is automatically set up when you create a window. Just call the trap **tb:!SetPort** to make the new window's grafPort the current grafPort.

Methods are provided for most of the functions that draw or perform calculations on graphic objects. By calling the method instead of the function, portability between systems is greatly simplified. In addition, it is often faster to invoke the method than the function. If possible, use the method given rather than the function.

## GrafPorts

3.2 The most frequently used grafPort traps are **tb:!GetPort** and **tb:!SetPort**.

### GrafPort

**3.2.1** To create a new grafPort object, make an instance of the **tb:grafPort** flavor.

### tb:grafPort

[I-148] Flavor

This flavor defines a black-and-white grafPort. It is unlikely that you will ever have to explicitly create a **tb:grafPort** instance. Normally, you will use a flavor which has **tb:grafPort** flavor as a mixin. **tb:grafPort** instances have the following instance accessor methods:

| | | |
|---|---|---|
| • :DEVICE | ;0 | [ integer ] |
| • :PORTBITSBASEADDR | ;2 | [ pointer ] |
| • :PORTBITSROWBYTES | ;6 | [ integer ] |
| • :PORTBITSBOUNDSTOP | ;8 | [ integer ] |
| • :PORTBITSBOUNDSLEFT | ;10 | [ integer ] |
| • :PORTBITSBOUNDSBOTTOM | ;12 | [ integer ] |
| • :PORTBITSBOUNDSRIGHT | ;14 | [ integer ] |

| | | |
|---|---|---|
| • :PORTRECTTOP | ;16 | [ integer ] |
| • :PORTRECTLEFT | ;18 | [ integer ] |
| • :PORTRECTBOTTOM | ;20 | [ integer ] |
| • :PORTRECTRIGHT | ;22 | [ integer ] |
| • :VISRGN | ;24 | [ rgnhandle ] |
| • :CLIPRGN | ;28 | [ rgnhandle ] |
| • :BKPATONE | ;32 | [ unsigned-integer ] |
| • :BKPATTWO | ;34 | [ unsigned-integer ] |
| • :BKPATTHREE | ;36 | [ unsigned-integer ] |
| • :BKPATFOUR | ;38 | [ unsigned-integer ] |
| • :FILLPATONE | ;40 | [ unsigned-integer ] |
| • :FILLPATTWO | ;42 | [ unsigned-integer ] |
| • :FILLPATTHREE | ;44 | [ unsigned-integer ] |
| • :FILLPATFOUR | ;46 | [ unsigned-integer ] |
| • :PNLOCV | ;48 | [ integer ] |
| • :PNLOCH | ;50 | [ integer ] |
| • :PNSIZEV | ;52 | [ integer ] |
| • :PNSIZEH | ;54 | [ integer ] |
| • :PNMODE | ;56 | [ integer ] |
| • :PNPATONE | ;58 | [ unsigned-integer ] |
| • :PNPATTWO | ;60 | [ unsigned-integer ] |
| • :PNPATTHREE | ;62 | [ unsigned-integer ] |
| • :PNPATFOUR | ;64 | [ unsigned-integer ] |
| • :PNVIS | ;66 | [ integer ] |
| • :TXFONT | ;68 | [ integer ] |
| • :TXFACE | ;70 | [ style ] |
| • :TXMODE | ;72 | [ integer ] |
| • :TXSIZE | ;74 | [ integer ] |
| • :SPEXTRA | ;76 | [ fixed ] |
| • :FGCOLOR | ;80 | [ longint ] |
| • :BKCOLOR | ;84 | [ longint ] |
| • :COLRBIT | ;88 | [ integer ] |
| • :PICSAVE | ;92 | [ handle ] |
| • :RGNSAVE | ;96 | [ handle ] |
| • :POLYSAVE | ;100 | [ handle ] |
| • :GRAFPROCS | ;104 | [ qdprocsptr ] |

The only instance variables of a **tb:grafPort** instance you are likely to be interested in are the **tb:PortRect** ones which define the boundary rectangle of the **tb:grafPort** instance.

*Example:*

```
(setf gp (make-instance 'tb:grafport))
(tb:!GetPort gp) =>  T  gp  #<GRAFPORT Pointer 010E6C>
```

**CGrafPort (color grafPort)**

3.2.2 To create a new color grafport, make an instance of the **tb:cGrafPort** flavor.

**tb:cGrafPort** [V-50] Flavor

This flavor defines a color grafPort data structure. It is unlikely that you will ever have to explicitly create an instance of this flavor. Normally, you will use a flavor which mixes in the **tb:cGrafPort** flavor such as

**tb:Window** or **tb:DialogRecord.** **tb:cGrafPort** instances have the following instance accessor methods:

| | | |
|---|---|---|
| • :DEVICE | ;0 | [ integer ] |
| • :PORTPIXMAP | ;2 | [ pixmaphandle ] |
| • :PORTVERSION | ;6 | [ integer ] |
| • :GRAFVARS | ;8 | [ handle ] |
| • :CHEXTRA | ;12 | [ integer ] |
| • :PENLOCHFRAC | ;14 | [ integer ] |
| • :PORTRECTTOP | ;16 | [ integer ] |
| • :PORTRECTLEFT | ;18 | [ integer ] |
| • :PORTRECTBOTTOM | ;20 | [ integer ] |
| • :PORTRECTRIGHT | ;22 | [ integer ] |
| • :VISRGN | ;24 | [ rgnhandle ] |
| • :CLIPRGN | ;28 | [ rgnhandle ] |
| • :BKPIXPAT | ;32 | [ pixpathandle] |
| • :RGBFGCOLORRED | ;36 | [ unsigned-integer ] |
| • :RGBFGCOLORGREEN | ;38 | [ unsigned-integer ] |
| • :RGBFGCOLORBLUE | ;40 | [ unsigned-integer ] |
| • :RGBBKCOLORRED | ;42 | [ unsigned-integer ] |
| • :RGBBKCOLORGREEN | ;44 | [ unsigned-integer ] |
| • :RGBBKCOLORBLUE | ;46 | [ unsigned-integer ] |
| • :PNLOCV | ;48 | [ integer ] |
| • :PNLOCH | ;50 | [ integer ] |
| • :PNSIZEV | ;52 | [ integer ] |
| • :PNSIZEH | ;54 | [ integer ] |
| • :PNMODE | ;56 | [ integer ] |
| • :PNPIXPAT | ;58 | [ pixpathandle ] |
| • :FILLPIXPAT | ;62 | [ pixpathandle ] |
| • :PNVIS | ;66 | [ integer ] |
| • :TXFONT | ;68 | [ integer ] |
| • :TXFACE | ;70 | [ style ] |
| • :TXMODE | ;72 | [ integer ] |
| • :TXSIZE | ;74 | [ integer ] |
| • :SPEXTRA | ;76 | [ fixed ] |
| • :FGCOLOR | ;80 | [ longint ] |
| • :BKCOLOR | ;84 | [ longint ] |
| • :COLRBIT | ;88 | [ integer ] |
| • :PICSAVE | ;92 | [ handle ] |
| • :RGNSAVE | ;96 | [ handle ] |
| • :POLYSAVE | ;100 | [ handle ] |
| • :GRAFPROCS | ;104 | [ cqdprocsptr ] |

*Example:*

```
(setf cgp (make-instance 'tb:cgrafport))
(tb:!GetPort cgp) => T  cgp  #<CGRAFPORT Pointer 010E6C>
```

### GrafPort and CGrafPort Routines

3.2.3 These traps and methods are used to create, modify, and dispose of grafPorts and cGrafPorts. Most of the time you will not use grafPorts directly; instead you will use windows and dialogrecords which are extended types of grafPorts.

**tb:!InitGraf** *pointer* [I-162] Function

Initializes the QuickDraw global variables. Since QuickDraw has already been initialized by the TbServer, do not use this function.

**:open** — Method of **tb:grafPort**
**:open** — Method of **tb:cGrafPort**
**tb:!OpenPort** *grafPort* — [I-163] Function
**tb:!OpenCPort** *cGrafPort* — [V-67] Function

The two methods above allocate space in the Macintosh heap via a **tb:!NewPtr** and then call **tb:!OpenPort** or **tb:!OpenCPort**, respectively. The two functions set up the various fields of the given *grafPort* (or *cGrafPort*). You will rarely need to call either of these functions because they are called by the Window Manager when a new window is created.

**CAUTION: Never use tb:!OpenPort or tb:!OpenCPort without allocating space via tb:!NewPtr. It is not sufficient to merely make an instance of tb:grafport.**

You will normally want to save the current port before using the **:open** methods or traps because they will cause the new port to be the current grafPort. Calling the methods or traps from the top level will cause the new grafPort to become the current grafPort.

**tb:!InitPort** *grafPort* — [I-164] Function
**tb:!InitCPort** *cGrafPort* — [V-67] Function

These are internally called traps and should not be used.

**:dispose** — Method of **tb:grafPort**
**:dispose** — Method of **tb:cGrafPort**
**tb:!ClosePort** *grafPort* — [V-164] Function
**tb:!CloseCPort** *cGrafPort* — [V-68] Function

These functions and methods close a grafPort (or cGrafPort), disposing of any data objects that may have been created. You should never need to call these, except if you are working with off-screen bitmaps, as they are called internally by **tb:!CloseWindow**.

*Example:*

```
;;;Example of creating, opening, getting, setting, and closing a new grafPort
(defun ALLOC-NEW-GRAFPORT ()
  (let ((temp-port (make-instance 'tb:cgrafport))
        (cgp       nil))
    (tb:!GetPort temp-port)                         ;save current grafPort
    (setf cgp (make-instance 'tb:cgrafport))        ;make new cgrafPort
    (send cgp :open)                                ;allocate & init structs
    (tb:!SetPort temp-port)                         ;restore original port
    cgp))                                           ;return a new color gp

(setf cgp (alloc-new-grafport))
... work with off screen bitmap...
(send cgp :dispose)                                 ;deallocate memory
```

**tb:!SetPort** *grafPort* — [I-165] Function

Makes *grafPort* the current grafPort. This means that all further QuickDraw traps will refer to and act upon *grafPort*. You usually call this trap after receiving an activate event or an update event for a window. To call this trap, do the following:

*Example:*

```
(tb:!SetPort myWindow)
```

All further QuickDraw commands will be drawn into the window *myWindow*.

It is a good idea to save the current grafPort using the trap **tb:!GetPort** before using **tb:!SetPort**. You can then restore the original grafPort when you are finished.

**tb:GetPort** [I-165] Function
**tb:!GetPort** *grafPort* [I-165] Function

**tb:GetPort** returns the current grafPort. The current grafPort is the grafPort in which all QuickDraw traps are drawn. If the current grafPort is a window, GetPort will return a window instance instead of a grafPort. Notice in the example below how the current grafPort is saved before operations on a different port are done, and then how the original grafPort is restored.

**tb:!GetPort** is similar except that it updates *grafPort* with the new grafPort information.

*Example:*

```
(defun FOO ()
  (let ((temp-port (make-instance 'tb:cGrafPort)))
      (setf temp-port (tb:GetPort)) ;save current port in it
    (tb:!SetPort myWindow)
    ...some operations on a different grafport...
      (tb:!SetPort temp-port)))       ;set back to original port
```

**tb:!GrafDevice** *device* [I-165] Function

Sets the current grafPort field device to the integer value specified in *device*. You will never call this trap.

**tb:!SetPortBits** *bitMap* [I-165] Function
**tb:!SetCPortPix** *pixMap* [V-76] Function

These traps set the bitMap (or the pixMap) of the current grafPort to a previously defined *bitmap* (or *pixMap*). They are useful for graphic animation where you create an off-screen grafPort, draw into it, and then copy it into the on-screen grafPort using **tb:!CopyBits**.

**tb:!PortSize** *width height* [I-165] Function

Sets the width and height of the grafPort's portRect. This does not affect the screen; it merely changes the size of the "active area" of the grafPort. You will never call this trap. It is normally called by the Window Manager.

**tb:!MovePortTo** *leftGlobal topGlobal* [I-166] Function

This trap is only called internally. You will never call this trap. It is normally called by the Window Manager.

**tb:!SetOrigin** *h v* [I-166] Function

Sets the local coordinate system of the grafPort. The integers *h* and *v* are the new coordinates of the grafPort portRect's top and left coordinates. See the discussion in *Inside Macintosh.*

**:clip** *rect-or-region* Method of **tb:grafPort**
**tb:!SetClip** *region* [I-166] Function
**tb:!ClipRect** *rect* [I-167] Function

Sets the grafPort's clipRegion equivalent to *region* or *rect.*

*Example:*

```
(defun FOO ()
  (let ((w (make-instance 'tb:window :title
                          "Half of a Color Circle"))
        (r (make-instance 'rect :left 10   :top 10
                                :right 110 :bottom 110)))
    (tb:!SetPort w)
    (send r :FrameOval)
    (send w :clip                     ; clip to left half
          (make-instance 'rect :left 10  :top 10
                               :right 60 :bottom 110))
    (send r :FillCOval)               ; fill half a circle
    (sleep 5)
    (send w :dispose)))
```

**tb:!GetClip** *region* [I-167] Function

Changes *region* to be equivalent to the current grafPort's clipRegion. This is the opposite of **tb:!SetClip**.

**tb:!BackPixPat** *pixPat* [V-74] Function
**tb:!BackPat** *pattern* [I-167] Function

Set the background pattern of the current grafPort to the given *pixPat* (or *pattern*). To set the background pattern of the current grafPort to light gray, do the following:

*Example:* `(tb:!BackPat  tb:!ltgray)`

**tb:!OpColor** *RGBColor* [V-77] Function

Sets the operand red, blue, and green colors used by **tb:!AddPin**, **tb:!SubPin**, and **tb:!Blend** drawing modes if the current grafPort is a color grafPort.

**tb:!HiliteColor** *RGBColor* [V-77] Function

Overides the system color and allows you to change the highlighting color used by the current port if the current grafPort is a color grafPort.

**tb:!CharExtra** *fixed* [V-77] Function

Specifies the number of pixels to widen every character, excluding the space character, in a line of text.

**tb:!blend** [V-59] Constant
**tb:!addPin** [V-59] Constant
**tb:!addOver** [V-59] Constant
**tb:!subPin** [V-59] Constant
**tb:!adMax** [V-59] Constant
**tb:!subOver** [V-59] Constant
**tb:!asMin** [V-59] Constant
**tb:!transparent** [V-59] Constant

These constants represent the color QuickDraw arithmetic transfer modes.

## Cursor Handling

3.3 A cursor is the small image that appears on the screen and is controlled by the mouse. The mouse position is always linked to the cursor position. You can't reposition the cursor through software; the only control you have is whether or not it is visible and what shape it will assume.

### Cursor

3.3.1 Normally you can get the cursor you need from a resource using **tb:!GetCursor**. If, however, you were writing a cursor editor and needed a blank cursor object, you could make an instance of the **tb:cursor** flavor. The system would then automatically give you a handle 68 bytes long.

### Color Cursor

3.3.2 A color cursor is a handle 96 bytes long. Color cursors are much more complicated than regular cursors. Normally, you will use **tb:!GetCCursor** to get a color cursor. To get a blank cursor to use in a cursor editor, for example, make an instance of the **tb:cCursor** flavor.

### Cursor Handling Routines

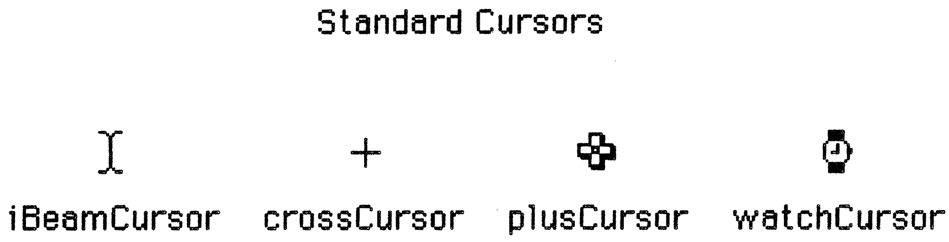
3.3.3 Cursor handling routines are the functions that control the appearance and visibility of the cursor.

**tb:!InitCursor** [I-167] Function

Sets the cursor to the arrow cursor and makes it visible. This trap is called for you initially when you launch the TbServer.

**tb:!GetCursor** *cursorID* [I-474] Function
**tb:!GetCCursor** *cursorID* [V-75] Function

Return a handle to a cursor with the given resource ID of *cursorID* in the "CURS" resource. The Toolbox Utility trap **tb:!GetCursor** can be used to select any cursor. There are four predefined cursors shown below and defined by the following constants:



**tb:!IBeamCursor** [I-474] Constant
**tb:!PlusCursor** [I-474] Constant
**tb:!WatchCursor** [I-474] Constant
**tb:!CrossCursor** [I-474] Constant
**tb:!ArrowCursor** [I-474] Constant

These are the "CURS" resource IDs for standard cursors.

- iBeam selects text
- thin cross draws graphics
- thick plus selects cells in structured documents
- watch indicates a long wait
- arrow points

*Example:*

```
(tb:!SetCursor (tb:!GetCursor tb:!WatchCursor))

;;;Another example that changes cursors
(defun FOO ()
  (dotimes (i 16)
    (tb:!SetCursor (tb:!GetCursor (1+ (mod i 4))))
    (sleep 1))
  (tb:!InitCursor))
```

**tb:!SetCursor** *cursor* [I-167] Function
**tb:!SetCCursor** *cursor* [V-75] Function

Set the current cursor to the one specified in *cursor*.

**tb:!ShowCursor** [I-168] Function
**tb:!HideCursor** [I-168] Function

Makes the cursor visible or invisible.

**tb:!ObscureCursor** [I-168] Function

Hides the cursor until the next time the mouse is moved.

**tb:!DisposCCursor** *cursor* [V-75] Function

Disposes of the memory associated with a color cursor.

**tb:!AllocCursor** [V-75] Function

Reallocates color cursor memory. See *Inside Macintosh* before using.

**tb:!ShieldCursor** *shieldRect point* [I-474] Function

Removes the cursor from the screen if the cursor and the rectangle *shieldRect* intersect.

## Icon Handling

3.4 These traps are used to create and dispose of icons.

**tb:!GetIcon** *iconID* [I-473] Function
**tb:!GetCIcon** *iconID* [V-76] Function
**tb:GetIcon** *iconID* [I-473] Function

Get an icon (or a color icon) from a resource with an ID *iconID*. **tb:GetIcon** signals an OSErr is the icon is not found

**tb:!PlotIcon** *rect Icon* [I-473] Function
**tb:!PlotCIcon** *rect cIcon* [V-76] Function

Draw the icon whose handle is *icon* (or *cIcon*) in *rect*.

**tb:!DisposCIcon** *cIcon* [V-76] Function

Disposes of the color icon.

## Pen and Line Drawing

3.5 Two data structures are used when drawing with the pen: **tb:PenState** and **tb:Pattern**. To get a new pen state or pattern , make in instance of the **tb:PenState** or **tb:Pattern** flavor, respectively.

**tb:PenState** [I-169] Flavor

This flavor defines an "empty" pen state data structure. An instance of this flavor is passed to the trap **tb:!GetPenState**. The trap updates various information about the pen in the current grafPort.

**:PnLocV** Method of **tb:PenState**
**:set-PnLocV** *integer* Method of **tb:PenState**
**:PnLocH** Method of **tb:PenState**
**:set-PnLocH** *integer* Method of **tb:PenState**

Pen location as point coordinates.

**:PnSizeV** Method of **tb:PenState**
**:set-PnSizeV** *integer* Method of **tb:PenState**
**:PnSizeH** Method of **tb:PenState**
**:set-PnSizeH** *integer* Method of **tb:PenState**

Pen size as height and width

**:PnMode** Method of **tb:PenState**
**:set-PnMode** *integer* Method of **tb:PenState**

Pen drawing mode (e.g., **tb:!patCopy**).

| | |
|---|---|
| **:PnPat1** | Method of **tb:PenState** |
| **:set-PnPat1** *16b-integer* | Method of **tb:PenState** |
| **:PnPat2** | Method of **tb:PenState** |
| **:set-PnPat2** *16b-integer* | Method of **tb:PenState** |
| **:PnPat3** | Method of **tb:PenState** |
| **:set-PnPat3** *16b-integer* | Method of **tb:PenState** |
| **:PnPat4** | Method of **tb:PenState** |
| **:set-PnPat4** *16b-integer* | Method of **tb:PenState** |

The 8-byte pen pattern expressed as four 16-bit integers.

**tb:Pattern** [I-146] Flavor

A **tb:Pattern** instance consists of 8 bytes of data organized into four 16-bit unsigned integer instance variables.

| | |
|---|---|
| **:one** | Method of **tb:Pattern** |
| **:set-one** *16b-unsigned-integer* | Method of **tb:Pattern** |
| **:two** | Method of **tb:Pattern** |
| **:set-two** *16b-unsigned-integer* | Method of **tb:Pattern** |
| **:three** | Method of **tb:Pattern** |
| **:set-three** *16b-unsigned-integer* | Method of **tb:Pattern** |
| **:four** | Method of **tb:Pattern** |
| **:set-four** *16b-unsigned-integer* | Method of **tb:Pattern** |

| | |
|---|---|
| **tb:!Black** | Variable |
| **tb:!dkGray** | Variable |
| **tb:!ltGray** | Variable |
| **tb:!White** | Variable |

These are the predefined patterns for solid black, dark gray, light gray, and solid white. They are effectively constants since the pattern they represent never changes. However, they are classed as variables rather than constants because they reside on the Macintosh side and must be reestablished each time the Toolbox server is launched.

**tb:!HidePen** [I-168] Function

Makes the pen in the current grafPort invisible. All further effects of QuickDraw traps which use the pen will be invisible. Actually, **tb:!HidePen** decrements the pnVis counter. See *Inside Macintosh* for details.

**tb:!ShowPen** [I-168] Function

Makes the pen in the current grafPort visible. All further effects of QuickDraw traps which use the pen will be visible. Actually, **tb:!ShowPen** increments the pnVis counter. See *Inside Macintosh* for details.

**tb:!GetPen** *point* [I-169] Function

Returns the current location of the grafPort pen in *point*.

*Example:*

```
(setf pt (make-instance 'tb:point)) ;get a point instance
(tb:!GetPen pt)                     ;set it to current pos
pt => #<POINT x=0 y=0>              ;examine it
```

The point now has the coordinates of the current grafPort's pen.

**tb:!GetPenState** *penState* [I-169] Function

Returns the current grafPort's pen status in *penState*. A PenState instance is passed to the trap and the PenState is returned in the instance.

*Example:*

```
(setf pnstate (make-instance 'tb:PenState))
(tb:!GetPenState pnstate)
(send pnState :pnMode) => 8
```

In this example, the current pen mode is **tb:!patCopy.**

**tb:!SetPenState** *penState* [I-169] Function

Sets the current grafPort's pen status to the values of the *penState* instance.

**tb:!PenSize** *width height* [I-169] Function

Sets the current grafPort's pen width (in pixels) to *width*, and its height to *height*.

**tb:!PenMode** *mode* [I-169] Function

Sets the transfer mode which QuickDraw uses to draw onto the grafPort's bitmap. The constants defining the available transfer modes follow.

| | | |
|---|---|---|
| **tb:!patCopy** | [I-157] | Constant |
| **tb:!patOr** | [I-157] | Constant |
| **tb:!patXOr** | [I-157] | Constant |
| **tb:!patBic** | [I-157] | Constant |
| **tb:!notPatCopy** | [I-157] | Constant |
| **tb:!notPatOr** | [I-157] | Constant |
| **tb:!notPatXOr** | [I-157] | Constant |
| **tb:!notPatBic** | [I-157] | Constant |

These are QuickDraw transfer modes.

**tb:!PenPixPat** *pixPat* [V-74] Function
**tb:!PenPat** *pattern* [I-170] Function

Set the pen pattern of the current grafPort to the *pixPat* or *pattern* specified.

**tb:!PenNormal** [I-170] Function

Restores the current grafPort's pen status to the default value. The default pen's width is one pixel, it's height one pixel. The pen mode is **tb:!patCopy** and the pen pattern is black.

**:moveTo** Method of **tb:Point**

Moves the pen to the location specified by *point.*

**tb:!MoveTo** *h v* [I-170] Function

Moves the pen to the horizontal position *h* and the vertical position *v* in the current grafPort's local coordinate system.

**tb:!Move** *dh dv* [I-170] Function

Moves the pen *dh* horizontally, *dv* vertically, from its present position.

**:lineTo** Method of **tb:Point**

Draws a line from the pen's present position to *point* and leaves the pen there.

**tb:!LineTo** *h v* [I-170] Function

Draws a line from the pen's present position to a point with local coordinates (*h*,*v*) and leaves the pen at (*h*,*v*).

**tb:!Line** *dh dv* [I-171] Function

Draws a line from the pen's present position to a point which is located at a distance *dh* horizontally and *dv* vertically away, and leaves the pen there.

## Text Drawing

3.6 These routines control the characteristics of text elements: assigning type styles, setting pen modes, etc.

**tb:!TextFont** *font* [I-171] Function

Sets the current grafPort's font to the *font* indicated. To determine the font number of a desired font, use the Font Manager trap **tb:!GetFNum**.

**tb:!TextFace** *face* [I-171] Function

Sets the current grafPort's character style. The presently defined character styles are:

| | |
|---|---|
| **tb:!Bold** | [I-152] Constant |
| **tb:!Italic** | [I-152] Constant |
| **tb:!Underline** | [I-152] Constant |
| **tb:!Outline** | [I-152] Constant |
| **tb:!Shadow** | [I-152] Constant |
| **tb:!Condense** | [I-152] Constant |
| **tb:!Extend** | [I-152] Constant |

Additive masked used to defined text styles. To get any combination of character styles you must add the masks together. For example, to set the current grafPort's text character style to Bold and Underline do the following:

*Example:* `(tb:!TextFace (+ tb:!bold tb:!Underline))`

**tb:!TextMode** *mode* [I-171] Function

Sets the current grafPort's text transfer mode as indicated by the integer in *mode*. See **tb:!PenMode** for the various pen transfer modes.

**tb:!TextSize** *size* [I-171] Function

Sets the current grafPort's font size as indicated in *size*. To determine if a font of the desired size exists, call the Font Manager trap **tb:!RealFont.**

**tb:!SpaceExtra** *integer* [I-172] Function

Sets the average number of pixels to pad out the spaces in a line of text.

**tb:!DrawChar** *character* [I-172] Function

Draws *character* at the present pen position and advances the pen the character's width.

**tb:!DrawString** *string* [I-172] Function

Draws the given string at the present pen position and advances the pen the width of the string.

**tb:!DrawText** *textBuf offset byteCount* [I-172] Function

Draws *byteCount* number of characters, starting at *offset* (an integer), into a text buffer pointed to by *textBuf* and advances the pen the width of the text.

**tb:!CharWidth** *character* [I-173] Function

Returns the width, in pixels, of the character indicated in *character*.

**tb:!StringWidth** *string* [I-173] Function

Returns the width of *string* in pixels, i.e, the sum of all the component character widths.

**tb:!Textwidth** *textBuf offset byteCount* [I-173] Function

Returns the width, in pixels, of *byteCount* number of characters in a text buffer pointed to by *textBuf*, starting at *offset*.

**tb:!MeasureText** *byteCount textAddr charLocs* [IV-25] Functions

This is an array-based version of the trap **tb:!TextWidth**. It returns an array of the character widths in *charLocs* of the *byteCount* number of characters starting at *textAddr*. The object pointed to by *charLocs* should be at least (*byteCount* * 2) bytes in size.

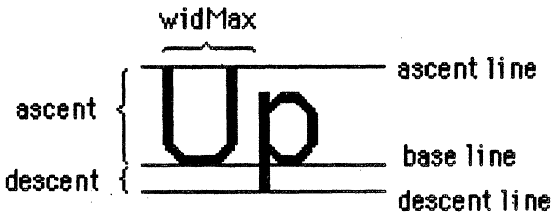
**tb:!GetFontInfo** *FontInfo* [I-173] Function

Returns information (ascent, descent, etc.) about the current grafPort's font in the data structure *FontInfo*. To create a new object suitable for use as this trap's argument, make an instance of the **tb:FontInfo** flavor.

**tb:FontInfo** [I-173] Flavor

This flavor defines a FontInfo data structure. The **:init** method for this flavor automatically calls **tb:!GetFontInfo** to initialize the new instance. Therefore, it is generally not necessary for you to call **tb:!GetFontInfo** yourself.

**:Ascent** Init Option of **tb:FontInfo**
**:Descent** Init Option of **tb:FontInfo**
**:WidMax** Init Option of **tb:FontInfo**
**:Leading** Init Option of **tb:FontInfo**



The above diagram explains the **tb:FontInfo** instance variables.

*Example:*

```
(setf info (make-instance 'tb:fontinfo))
 => #<FontInfo ascent:12 descent:3 widmax:15 leading:1>
(send info :ascent) => 12
```

For reasons of efficiency, **tb:FontInfo** instances reside on the microExplorer unlike most of the Toolbox objects.

## Drawing in Color

3.7 These routines will enable applications to do color drawing. All nonwhite colors will appear as black on black-and-white output devices. Colors in cGrafPorts are represented by RGBColor objects. To create a new RGBColor object, make an instance of the **tb:RGBColor** flavor.

**tb:RGBColor** [V-48] Flavor

This flavor represents a color as three 16-bit unsigned integers corresponding to the saturation levels for red, green, and blue.

| | |
|---|---|
| **:red** | Method of **tb:RGBColor** |
| **:set-red** *16b-unsigned-integer* | Method of **tb:RGBColor** |
| **:green** | Method of **tb:RGBColor** |
| **:set-green** *16b-unsigned-integer* | Method of **tb:RGBColor** |
| **:blue** | Method of **tb:RGBColor** |
| **:set-blue** *16b-unsigned-integer* | Method of **tb:RGBColor** |

These methods read and write the color state of the flavor.

*Example:* `(make-instance 'tb:RGBColor)`

:= &optional *red green blue* — Method of **tb:RGBColor**

Sets the RGBColor to the given *red*, *green*, and *blue* values.

**tb:!RGBForeColor** *RGBcolor* — [V-68] Function
**tb:!ForeColor** *color* — [I-173] Function

Set the foreground color of the current grafPort.

**tb:!RGBBackColor** *RGBcolor* — [V-68] Function
**tb:!BackColor** *color* — [I-174] Function

Set the background color of the current grafPort.

**tb:!ColorBit** *whichBit* — [I-174] Function

Tells QuickDraw into which color plane to draw (0-31).

**tb:!GetForeColor** *RGBcolor* — [V-69] Function
**tb:!GetBackColor** *RGBcolor* — [V-69] Function

Returns the RGB components of the foreground (or background) colors set in the current port. This call works for both grafPorts and cGrafPorts.

## Operations on Color Tables

3.8 These procedures create and dispose of color tables.

**tb:!GetCTable** *integer* — [V-77] Function

Allocates and returns a handle to a new color table data structure and initializes it using the information in the "clut" resource whose resource ID is *integer*.

**tb:!DisposCTable** *colorTable* — [V-77] Function

Disposes of the colorTable.

## Operations on Pixel Patterns

3.9 These routines create, modify, and dispose of pixel patterns. To create a new pixel pattern, make an instance of the **tb:pixPat** flavor.

**tb:pixPat** [V-55] Flavor

This flavor defines a pixel pattern. **tb:pixPat** instances have the following instance accessor methods:

| | | |
|---|---|---|
| • :PATTYPE | ;0 | [ integer ] |
| • :PATMAP | ;2 | [ pixmaphandle ] |
| • :PATDATA | ;6 | [ handle ] |
| • :PATXDATA | ;10 | [ handle ] |
| • :PATXVALID | ;14 | [ integer ] |
| • :PATXMAP | ;16 | [ handle ] |
| • :PAT1DATAONE | ;20 | [ integer ] |
| • :PAT1DATATWO | ;22 | [ integer ] |
| • :PAT1DATATHREE | ;24 | [ integer ] |
| • :PAT1DATAFOUR | ;26 | [ integer ] |

**tb:!NewPixPat** [V-72] Function

Creates a new pixel pattern data structure and all its associated data structures, and returns a handle to it. The preferred method of creating a pixPat is to make in instance of the **tb:pixPat** flavor as shown above.

**:dispose** Method of **tb:pixPat**
**tb:!DispospixPat** *pixPat* [V-73] Function

Dispose of a pixel pattern data structure and all its associated data structures.

**tb:!CopyPixPat** *srcPixPat dstPixPat* [V-73] Function

Copies the pixel pattern in the source pixPat to the pixel pattern in the destination pixPat.

**tb:!GetPixPat** *integer* [V-73] Function

Creates a new pixel pattern using the information stored in the "ppat" resource whose resource ID is *integer*.

**tb:!MakeRGBPat** *pixPat RGBColor* [V-73] Function

Creates a new pattern that approximates *RGBColor* and returns it in the pixel pattern *pixPat*.

## Calculations With Rectangles

**3.10** Calculation routines are independent of the current coordinate system. A calculation will operate the same regardless of which grafPort is active. To create a new rectangle, make an instance of the **tb:Rect** flavor.

Some of the following traps which have equivalent flavor methods also carry the comment that the method version is faster. In these particular cases, the trap functionality does not require the use of Macintosh system data structures or of Macintosh hardware. Therefore, the methods simply perform the trap's function in ordinary Lisp code using flavor data structures on the microExplorer side. If you choose to use the trap version, however, the trap must be sent to the Macintosh for execution and results from the Macintosh-side must be returned to those same flavor data structures back on the microExplorer side. Therefore, the results are the same, but using a method to get them is significantly faster.

**tb:Rect** [I-141] Flavor

This flavor defines a rectangle. All of the information related to this rectangle is maintained in instances of this flavor on the microExplorer side.

| | |
|---|---|
| **:top** | Init Option of **tb:Rect** |
| **:top** | Method of **tb:Rect** |
| **:set-top** *integer* | Method of **tb:Rect** |
| **:left** | Init Option of **tb:Rect** |
| **:left** | Method of **tb:Rect** |
| **:set-left** *integer* | Method of **tb:Rect** |
| **:bottom** | Init Option of **tb:Rect** |
| **:bottom** | Method of **tb:Rect** |
| **:set-bottom** *integer* | Method of **tb:Rect** |
| **:right** | Init Option of **tb:Rect** |
| **:right** | Method of **tb:Rect** |
| **:set-right** *integer* | Method of **tb:Rect** |

These values define the sides of the rectangle.

**:=** *args...* Method of **tb:Rect**

This method is a general purpose "rectangle definition" operator whose action depends upon the number and type of its arguments. In each case, the argument(s) define the new top, left, bottom, and right coordinates of the modified rectangle.

- One argument is a **tb:Rect** instance (i.e., simple assignment).
- Two arguments are two **tb:Point** instances similar to **tb:!Pt2Rect**,
- Four arguments are top, left, bottom, and right specifications similar to **tb:!SetRect**.

*Example:*

```
(setf r  (make-instance 'tb:rect))
 => #<RECT 50,50 100,100>

;;;sets x1,y1 x2,y2 (left,top right,bottom)
(send r := 1 2 5 6) => #<RECT 1,2 5,6>
(setf p1 (make-instance 'tb:point :h 3 :v 4))
=> #<POINT x=3 y=4>
(setf p2 (make-instance 'tb:point :h 7 :v 8))
=> #<POINT x=7 y=8>

;;;sets to rect enclosed by two points
(send r := p1 p2) => #<RECT 3,4 7,8>
(setf r2 (make-instance 'tb:rect :left 0 :top 0
                                 :right 5 :bottom 5))
=> #<RECT 0,0 5,5>

;;;sets to values from another rect
(send r := r2) =>.#<RECT 0,0 5,5>
```

**tb:!SetRect** *rect left top right bottom* [I-174] Function

Sets the rectangle's coordinates. The methods are significantly faster than the trap (see explanation under Calculations With Rectangles). See also the := method of **tb:Rect**.

**:width** Method of **tb:Rect**
**:height** Method of **tb:Rect**

Return the rectangle's width and height, respectively.

**:center-x** Method of **tb:Rect**
**:center-y** Method of **tb:Rect**

Return the rectangle's center coordinate on the x and y axes, respectively.

**:center &optional** *point* Method of **tb:Rect**

Returns the rectangle's center coordinates as a point. If the optional point is supplied, it moves the rectangle to be centered around the given point.

**:offset** *dh dv* Method of **tb:Rect**
**tb:!OffsetRect** *rect dh dv* [I-174] Function

Offset the rectangle by the horizontal value *dh* and the vertical value *dv*. The method is significantly faster than the trap (see explanation under Calculations With Rectangles).

*Example:*

```
(setf r (make-instance 'tb:rect))
 => #<RECT 50,50 100,100>
(send r :offset 10 20)
 => #<RECT 60,70 110,120>
```

**:insert** *dh dv* — Method of **tb:Rect**
**tb:!InsetRect** *rect dh dv* — [I-175] Function

Enlarge or shrink the rectangle *rect* by amounts *dh* and *dv*. The value *dh* is added to the rectangle's left coordinate and subtracted from the right coordinate. The value *dv* is subtracted from the rectangle's top coordinate and added to the rectangle's bottom coordinate. The method is significantly faster than the trap (see explanation under Calculations With Rectangles).

*Example:*

```
(setf r (make-instance 'tb:rect))
 => #<RECT 50,50 100,100>
(send r :inset 10 20)
 => #<RECT 60,70 90,80>
```

**:intersection** *rectB* — Method of **tb:Rect**
**:intersection-p** *rectB* — Method of **tb:Rect**
**tb:!SectRect** *rectA rectB dstRect* — [I-175] Function

Calculate *dstRect*, the intersection of the two rectangles *rectA* and *rectB*. Note that the method **:intersection** destructively modifies *rectA*. If you only want to test whether two rectangles intersect, use the method **:intersection-p**. All of the above return true if the rectangles intersect and false if they do not. The methods are significantly faster than the trap (see explanation under Calculations With Rectangles above).

*Example:*

```
(setf r  (make-instance 'tb:rect))
 => #<RECT 50,50 100,100>
(setf r1 (make-instance 'tb:rect :left 0 :top 0
                                 :right 50 :bottom 50))
 => #<RECT 0,0 50,50>
;;;After computing the intersection, it returns true if they intersect
(send r :intersection r1) => NIL
r => #<RECT 50,50 50,50>
```

**:union** *rectB* — Method of **tb:Rect**
**tb:!UnionRect** *rectA rectB dstRect* — [I-175] Function

Return a rectangle *dstRect* which is the smallest rectangle enclosing the two rectangles *rectA* and *rectB*. Note that the method **:union** destructively modifies *rectA*.

**:inside-p** *point-or-rect* — Method of **tb:Rect**
**tb:!PtInRect** *point rect* — [I-175] Function

Return true if the point is in the rectangle and false if it is not. The method is significantly faster than the trap (see explanation under Calculations With Rectangles).

*Example:*

```
(setf pt  (make-instance 'tb:point :h 5 :v 5))
(setf r1  (make-instance 'tb:rect :left 0 :top 0 :right 10
                         :bottom 10))
(setf r2  (make-instance 'tb:rect :left 1 :top 1 :right 6
                         :bottom 6))
(send r1 :inside-p pt)      ;check to see if pt is in r1
(send r1 :inside-p r2)      ;check to see if r2 is inside r1
```

**tb:!Pt2Rect** *ptA ptB rect* [I-175] Function

Modifies *rect* to be the smallest rectangle that encloses the two points *ptA* and *ptB*. Returns true if the point is in the rectangle and false if it is not. The method is significantly faster than the trap (see explanation under Calculations With Rectangles). See also the := method of **tb:Rect**.

**tb:PtToAngle** *rect point* [I-175] Function
**tb:!PtToAngle** *rect point* VAR *angle* [I-175] Function

**tb:PtToAngle** returns the angle calculated from the center of the rectangle *rect* to the *point* indicated. **tb:!PtToAngle** is similar except it modifies *angle* to be the calculated result.

NOTE: These traps are slow and are not accurate unless *rect* is a square.

**:equal** *rectB* Method of **tb:Rect**
**tb:!EqualRect** *rectA rectB* [I-176] Function

Return true if the rectangles *rectA* and *rectB* are equal. The method is significantly faster than the trap (see explanation under Calculations With Rectangles above).

*Example:*

```
(setf r1 (make-instance 'tb:rect))
 => #<RECT 50,50 100,100>
(setf r2 (make-instance 'tb:rect))
 => #<RECT 50,50 100,100>
(send r1 :equal r2) => T
```

**:empty-p** Method of **tb:Rect**
**tb:!EmptyRect** *rect* [I-176] Function

Return true if the rectangle is empty, false if it is not. The method is significantly faster than the trap (see explanation under Calculations With Rectangles above).

*Example:*

```
(setf r1 (make-instance 'tb:rect :left 50 :top 50
                                 :right 25 :bottom 25))
 => #<RECT 50,50 25,25>
(send r1 :empty-p) => T
```

## Graphic Operations on Rectangles

3.11 These procedures perform graphic operations on rectangles. These traps do not move the pen.

**:frame** — Method of **tb:Rect**
**tb:!FrameRect** *rect* — [I-176] Function

Draw an outline just inside the rectangle, using the current grafPort's pen size, pen mode, and pen pattern. If there is a region open, the rectangle is added to this region.

**:paint** — Method of **tb:Rect**
**tb:!PaintRect** *rect* — [I-177] Function

Fill the rectangle *rect* with the current grafPort's pen pattern and transfer mode.

**:erase** — Method of **tb:Rect**
**tb:!EraseRect** *rect* — [I-177] Function

Fill the rectangle *rect* with the current grafPort's background pattern (bkPat) using the transfer mode **tb:!patCopy**.

**:invert** — Method of **tb:Rect**
**tb:!InvertRect** *rect* — [I-177] Function

Invert every pixel inside the rectangle; every white pixel becomes black, every black one becomes white.

**:fill** *pattern* — Method of **tb:Rect**
**:fillC** *pixPatHandle* — Method of **tb:Rect**
**tb:!FillRect** *rect pattern* — [I-177] Function
**tb:!FillCRect** *rect pixPatHandle* — [V-69] Function

Fill the given rectangle *rect* with the pattern specified by *pattern* (or *pixPatHandle*) using the **tb:!patCopy** transfer mode.

## Graphic Operations on Ovals

3.12 An oval is defined by the smallest rectangle in which it will fit. If the rectangle you specify is a square, QuickDraw draws a circle. These traps do not move the pen.

**tb:Oval** — Flavor

This flavor defines an oval. This flavor mixes in the **tb:Rect** flavor so that **tb:Oval** has all the initialization options and instance variables of **tb:Rect**.

| | |
|---|---|
| **:frame** | Method of **tb:Oval** |
| **:frameOval** | Method of **tb:Rect** |
| **tb:!FrameOval** *rect* | [I-177] Function |

Draws an oval that fits just inside the rectangle, using the current grafPort's pen mode, pen size, and pen pattern. If there is a region open, the rectangle is added to this region.

| | |
|---|---|
| **:paint** | Method of **tb:Oval** |
| **:paintOval** | Method of **tb:Rect** |
| **tb:!PaintOval** *rect* | [I-178] Function |

Fill the oval that fits inside the rectangle with the current grafPort's transfer mode and pen pattern.

| | |
|---|---|
| **:erase** | Method of **tb:Oval** |
| **:eraseOval** | Method of **tb:Rect** |
| **tb:!EraseOval** *rect* | [I-178] Function |

Fill the oval that fits inside the rectangle with the current grafPort's background pattern (*bkPattern*) using the transfer mode **tb:!patCopy**.

| | |
|---|---|
| **:invert** | Method of **tb:Oval** |
| **:invertOval** | Method of **tb:Rect** |
| **tb:!InvertOval** *rect* | [I-178] Function |

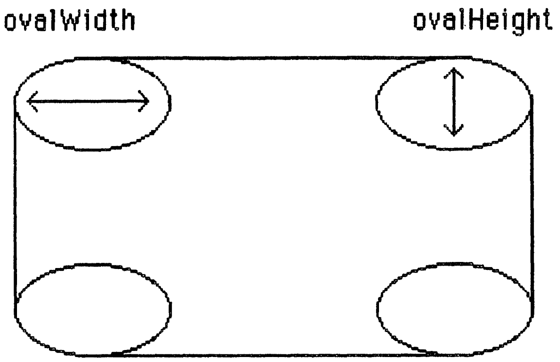
Invert every pixel inside the oval that fits inside the rectangle; every white pixel becomes black, every black one becomes white.

| | |
|---|---|
| **:fill** *pattern* | Method of **tb:Oval** |
| **:fillOval** *pattern* | Method of **tb:Rect** |
| **:fillCOval** *pixPatHandle* | Method of **tb:Rect** |
| **tb:!FillOval** *rect pattern* | [I-178] Function |
| **tb:!FillCOval** *rect pixPatHandle* | [V-68] Function |

Fill the oval that fits inside the rectangle with the pattern *pattern* (or *pixPatHandle*) using the **tb:!patCopy** transfer mode.

## Graphic Operations on Round-Cornered Rectangles

3.13 Round cornered rectangles are rectangles whose corners are defined by ovals. The oval is defined by two arguments: *ovalWidth* and *ovalHeight*. The same oval is used for all four corners of the round cornered rectangle. These traps do not move the pen.



**tb:RoundRect** Flavor

This flavor defines a rectangle in which the corners are asymmetrically rounded as if each corner contained an oval rather than a circle. This flavor mixes in **tb:Rect** so it shares all initialization options and instance variables with **tb:Rect**.

**:OvalWidth** *integer* Init Option of **tb:RoundRect**
**:OvalWidth** Method of **tb:RoundRect**
**:set-OvalWidth** *integer* Method of **tb:RoundRect**
**:OvalHeight** Init Option of **tb:RoundRect**
**:OvalHeight** Method of **tb:RoundRect**
**:set-OvalHeight** *integer* Method of **tb:RoundRect**

These values control the degree and the orientation of the asymmetrical rounding of the rectangle corners.

**:frame** Method of **tb:RoundRect**
**tb:!FrameRoundRect** *rect ovalWidth ovalHeight* [I-178] Function

Draw an outline just inside the round-cornered rectangle, with the diameter of curvature *ovalWidth* and *ovalHeight* (two integers) on a rectangle, using the current grafPort's pen mode, pen size, and pen pattern. If there is a region open, the rounded rectangle is added to this region.

**:paint** Method of **tb:RoundRect**
**tb:!PaintRoundRect** *rect ovalWidth ovalHeight* [I-179] Function

Fill the round-cornered rectangle, with the diameter of curvature *ovalWidth* and *ovalHeight* (two integers) on a rectangle, with the current grafPort's pen pattern and transfer mode.

**:erase** Method of **tb:RoundRect**
**tb:!EraseRoundRect** *rect ovalWidth ovalHeight* [I-179] Function

Fill the round-cornered rectangle, with the diameter of curvature *ovalWidth* and *ovalHeight* (two integers) on a rectangle, with the current grafPort's background pattern (*bkPattern*) using the transfer mode **tb:!patCopy**.

**:invert** Method of **tb:RoundRect**
**tb:!InvertRoundRect** *rect ovalWidth ovalHeight* [I-179] Function

Invert every pixel inside the round-cornered rectangle, with the diameter of curvature *ovalWidth* and *ovalHeight* (two integers) on a rectangle. Every white pixel becomes black, every black one white.

**:fill** *pattern* Method of **tb:RoundRect**
**tb:!FillRoundRect** *rect ovalWidth ovalHeight pattern* [I-179] Function
**tb:!FillCRoundRect** *rect ovalWidth ovalHeight pixPatHandle* [V-69] Function

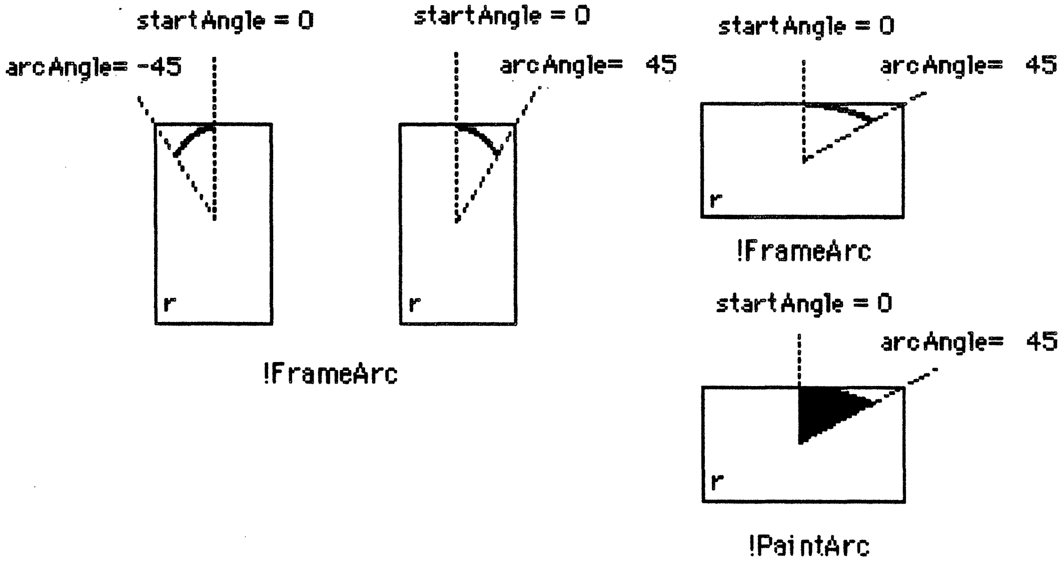
Fill the round-cornered rectangle, with the diameter of curvature *ovalWidth* and *ovalHeight* (two integers) on a rectangle, with the pattern using the **tb:!patCopy** transfer mode.

## Graphic Operations on Arcs and Wedges

**3.14** These procedures perform graphic operations on arcs and wedge-shaped sections of ovals. These traps do not move the pen.

Arcs and wedges are defined by three parameters:

- A startAngle, which is where the orientation the arc starts.
- An arcAngle, which is the compass of the arc (in degrees).
- A bounding rectangle.



**tb:!FrameArc** *rect startAngle arcAngle* [I-180] Function

Draws an arc of the oval that fits inside the rectangle *rect*, using the current grafPort's pen size, pen mode, and pen pattern. *StartAngle* and *arcAngle* are integers. If there is a region open, the arc is not added to the region.

**tb:!PaintArc** *rect startAngle arcAngle* [I-180] Function

Fills the wedge of the oval that fits inside the rectangle *rect* with the current grafPort's pen pattern and transfer mode. *StartAngle* and *arcAngle* are integers.

**tb:!EraseArc** *rect startAngle arcAngle* [I-180] Function

Fills the wedge of the oval that fits inside the rectangle *rect*, with the current grafPort's background pattern (*bkPattern*) using the transfer mode **tb:!patCopy**. *StartAngle* and *arcAngle* are integers.

**tb:!InvertArc** *rect startAngle arcAngle* [I-181] Function

Inverts every pixel inside the wedge of the oval that fits inside the rectangle; every white pixel becomes black, every black one becomes white. *StartAngle* and *arcAngle* are integers.

**tb:!FillArc** *rect startAngle arcAngle pattern* [I-181] Function
**tb:!FillCArc** *rect startAngle arcAngle pixPatHandle* [V-69] Function

Fills the wedge of the oval that fits inside the rectangle with the pattern specified in *pattern* (or *pixPatHandle*) using the **tb:!patCopy** transfer mode.

## Calculations With Regions

3.15 Regions are complex graphic objects that are defined by the boundary of the saved graphic object framing traps. Regions are created by calling the trap **tb:!NewRgn**. The Region is defined by calling the trap **tb:!OpenRgn** which saves all the relevant QuickDraw traps until **tb:!CloseRgn** is called. When **tb:!CloseRgn** is called, the region definition is put in the new region and it can then be manipulated and drawn. To create a new region, make an instance of the **tb:Region** flavor.

**tb:Region** [I-142] Flavor

This flavor defines a QuickDraw region. Upon instantiation, it defines an empty region.

**:rgnSize** *16b-unsigned-integer* Method of **tb:Region**

This is the size of the region in bytes.

| | |
|---|---|
| **:rgnBBoxTop** *16b-integer* | Method of **tb:Region** |
| **:rgnBBoxTop** | Method of **tb:Region** |
| **:set-rgnBBoxTop** *16b-integer* | Method of **tb:Region** |
| **:rgnBBoxLeft** *16b-integer* | Method of **tb:Region** |
| **:rgnBBoxLeft** | Method of **tb:Region** |
| **:set-rgnBBoxLeft** *16b-integer* | Method of **tb:Region** |
| **:rgnBBoxBottom** *16b-integer* | Method of **tb:Region** |
| **:rgnBBoxBottom** | Method of **tb:Region** |
| **:set-rgnBBoxBottom** *16b-integer* | Method of **tb:Region** |
| **:rgnBBoxRight** *16b-integer* | Method of **tb:Region** |
| **:rgnBBoxRight** | Method of **tb:Region** |
| **:set-rgnBBoxRight** *16b-integer* | Method of **tb:Region** |

These are the boundaries of the region expressed in the top, left, bottom, and right attributes of the bounding rectangle.

**tb:!NewRgn** [I-181] Function

Allocates a relocatable block for a new empty region and returns a handle to the region. The preferred method for creating a new region is to make an instance of the **tb:region** flavor.

**tb:!nilRgn** Constant

This constant is a **tb:Region** instance with coordinates of (0 0 0 0). This constant is used in Lisp for those situation where the Macintosh's documentation says to pass a (Pascal) NIL as a region.

**:open** Method of **tb:Region**
**tb:!OpenRgn** [I-181] Function

Make QuickDraw save all further line drawing calls for incorporation into the region. The QuickDraw traps that are included in the definition of the region include **tb:!Line**, **tb:!LineTo** and all the **tb:!Frame** traps (except **tb:!FrameArc**). The methods related to these traps (**:frame**, for example) will also be saved in the region.

**CAUTION: You can only have one region and one polygon open at the same time. If you have more than one open at a time, strange things will happen to the saved data structures.**

**:close** Method of **tb:Region**
**tb:!CloseRgn** *region* [I-182] Function

Terminate the recording of the line drawing traps by QuickDraw. All the saved drawing commands are used to build up a region structure and the resulting structure is saved in *region.* Regions have a maximum size of 32K bytes. You can determine the size of a region by calling the Memory Manager trap **tb:!GetHandleSize**.

**:dispose** Method of **tb:Region**
**tb:!DisposeRgn** *region* [I-182] Function

Dispose of a region, de-allocating the relocatable block in memory.

:= *args...* Method of **tb:Region**

This is a general purpose region modification operator whose exact operation depends upon the number and type of its arguments. In each case, the arguments imply a new set of region coordinates, such as:

- One argument which is an instance of **tb:Region** similar to **tb:!CopyRgn**.
- One argument which is **nil**, implying an empty region similar to **tb:!SetEmptyRgn**.
- Tne argument which is an instance of **tb:Rect** similar to **!RectRgn**.
- Two arguments which are instances of **tb:Point** (the corners of a rectangle).
- Four arguments which are the top, left, bottom, and right coordinates of a rectangle similar to **tb:!SetRectRgn.**

**tb:!CopyRgn** *srcRegion dstRegion* [I-183] Function

Creates a copy of the source region *srcRegion* in the destination region *dstRegion*.

**tb:!SetEmptyRgn** *region* [I-183] Function

Destroys the previous structure and sets *region* back to a null (empty) region.

**tb:!SetRectRgn** *region left top right bottom* [I-183] Function

Destroys the previous structure and sets *region* to the rectangle defined by the coordinates *left*, *top*, *right*, and *bottom* (all of which are integers).

**tb:!RectRgn** *region rect* [I-183] Function

Destroys the previous structure and sets *region* to the rectangle *rect*. **tb:!RectRgn** is the same as **tb:!SetRectRgn** except the rectangle is specified by a rectangle rather than its coordinate points.

**:offset** *dh hv* Method of **tb:Region**
**tb:!OffsetRgn** *region dh dv* [I-183] Function

Moves the region a distance of *dh* horizontally and *dv* vertically.

NOTE: The following traps use a lot of Macintosh stack space, at least twice the size of the total region.

**:inset** *dh hv* Method of **tb:Region**
**tb:!InsetRgn** *region dh dv* [I-184] Function

Enlarges or shrinks *region* by a horizontal amount *dh* and a vertical amount *dv*. If the value of *dh* or *dv* is positive, the region is shrunk in that coordinate's direction; if the value is negative, the region is grown in the coordinate's direction.

**:intersection** *srcRegion* Method of **tb:Region**
**tb:!SectRgn** *srcRegionA srcRegionB dstRegion* [I-184] Function

Calculate the intersection of the two regions *srcRegionA* and *srcRegionB* and place the result in the destination region *dstRegion*. Note that the method destructively modifies the instance to which it is sent.

**:union** *srcRegion* Method of **tb:Region**
**tb:!UnionRgn** *srcRegionA srcRegionB dstRegion* [I-184] Function

Calculate the union of the two regions *srcRegionA* and *srcRegionB* and place the result in the destination region *dstRegion*. Note that the method destructively modifies the instance to which it is sent.

**tb:!DiffRgn** *srcRgnA srcRgnB dstRgn* [I-184] Function

Calculates the difference of the two regions *srcRgnA* and *srcRgnB* and places the result in the destination region *dstRgn*.

**tb:!XorRgn** *srcRgnA srcRgnB dstRgn* [I-185] Function

Calculates the difference between the union and the intersection of the two regions *srcRgnA* and *srcRgnB* and places the result in the destination region *dstRgn*.

**:inside-p** *point-or-rect* Method of **tb:Region**
**tb:!PtInRgn** *point region* [I-185] Function
**tb:!RectInRgn** *rect region* [I-185] Function

**tb:!PtInRgn** returns true if the point is in the region specified. **tb:!RectInRgn** returns true if any part of the rectangle is in the region. **:inside-p** performs either functions depending upon the type of its arguments.

**:equal** *regionB* Method of **tb:Region**
**tb:!EqualRgn** *regionA regionB* [I-185] Function

Return true if the two regions *regionA* and *regionB* are absolutely identical in size, shape, and location.

**:empty-p** Method of **tb:Region**
**tb:!EmptyRgn** *region* [I-185] Function

Return true if *region* is an empty region.

## Graphic Operations on Regions

3.16 These routines all depend on the coordinate system of the current grafPort. If a region is drawn in a different grafPort than the one in which it was defined, it may not appear in the proper position inside the port. These traps do not move the pen.

**:frame** Method of **tb:Region**
**tb:!FrameRgn** *region* [I-186] Function

Draw an outline just inside the region using the current grafPort's pen size, pen mode, and pen pattern. If the region is open, the outside outline of the region being framed is added to the open region's boundary. Under no circumstances will the frame go outside the region boundary.

**CAUTION: If there are more than 25 intersections of a line with the outline of a region, strange things start happening and may eventually cause the Macintosh to die.**

**:paint** Method of **tb:Region**
**tb:!PaintRgn** *region* [I-186] Function

Paint the region with the current grafPort's pen pattern and transfer mode.

**:erase** Method of **tb:Region**
**tb:!EraseRgn** *region* [I-186] Function

Fill the region with the current grafPort's background pattern (*bkPattern*) using the transfer mode **tb:!patCopy**.

**:invert** Method of **tb:Region**
**tb:!InvertRgn** *region* [I-186] Function

Invert every pixel inside the region; every white pixel becomes black, every black one becomes white.

**:fill** &optional *pattern* Method of **tb:Region**
**tb:!FillRgn** *region pattern* [I-187] Function
**tb:!FillCRgn** *region pixPatHandle* [V-69] Function

Fill the region with the pattern using the **tb:!patCopy** transfer mode.

## Creating Pixel Maps

3.17 These procedures create, modify, and dispose of pixel maps. To create a new pixel map, make an instance of the **tb:PixMap** flavor.

**tb:PixMap** [V-52] Flavor

This flavor describes a pixel map. **tb:PixMap** instances have the following instance accessor methods:

| | | |
|---|---|---|
| • :BASEADDR | ;0 | [ pointer ] |
| • :ROWBYTES | ;4 | [ integer ] |
| • :BOUNDSTOP | ;6 | [ integer ] |
| • :BOUNDSLEFT | ;8 | [ integer ] |
| • :BOUNDSBOTTOM | ;10 | [ integer ] |
| • :BOUNDSRIGHT | ;12 | [ integer ] |
| • :PMVERSION | ;14 | [ integer ] |
| • :PACKTYPE | ;16 | [ integer ] |
| • :PACKSIZE | ;18 | [ longint ] |
| • :HRES | ;22 | [ fixed ] |
| • :VRES | ;26 | [ fixed ] |
| • :PIXELTYPE | ;30 | [ integer ] |
| • :PIXELSIZE | ;32 | [ integer ] |
| • :CMPCOUNT | ;34 | [ integer ] |
| • :CMPSIZE | ;36 | [ integer ] |
| • :PLANEBYTES | ;38 | [ longint ] |
| • :PMTABLE | ;42 | [ ctabhandle ] |
| • :PMRESERVED | ;46 | [ longint ] |

**tb:!NewPixMap** [V-70] Function

Creates a new pixMap data structure and returns a handle to it. The preferred method of creating a pixMap is to make an instance of the **tb:PixMap** flavor.

**:dispose** Method of **tb:PixMap**
**tb:!DisposPixMap** *pixMap* [V-70] Function

Dispose of the pixel map and all its associated data structures.

**tb:!CopyPixMap** *srcPixMap dstPixMap* [V-70] Function

Copies a pixel map from the source pixMap to the destination pixMap.

## Bit Transfer Operations

**3.18** These procedures perform bit transfer operations on either bitMaps or pixMaps. When using these routines with pixMaps, be sure to set the type bits in the **:rowBytes** field correctly or you may crash the system. (See figure 3 on page 52 of *Inside Macintosh* Volume V.) To create a new bitmap, make an instance of the **tb:BitMap** flavor.

**tb:BitMap** [I-144] Flavor

This flavor defines a new bitmap.

**:baseAddr** Method of **tb:BitMap**
**:set-baseAddr** *pointer* Method of **tb:BitMap**

This is the pointer to the bitmap array.

**:rowBytes** Method of **tb:BitMap**
**:set-rowBytes** *pointer* Method of **tb:BitMap**

This is the width of a row in the bitmap measured in bytes.

**:boundTop** Method of **tb:BitMap**
**:set-boundTop** *16b-integer* Method of **tb:BitMap**
**:boundLeft** Method of **tb:BitMap**
**:set-boundLeft** *16b-integer* Method of **tb:BitMap**
**:boundBottom** Method of **tb:BitMap**
**:set-boundBottom** *16b-integer* Method of **tb:BitMap**
**:boundRight** Method of **tb:BitMap**
**:set-boundRight** *16b-integer* Method of **tb:BitMap**

These values define the bounding rectangle of the bitmap.

**:scroll** *dh dv updateRegion* Method of **tb:Rect**
**tb:!ScrollRect** *rect dh dv updateRegion* [I-187] Function

Scroll the bits (pixels) that are inside the rectangle that is the intersection of *rect* and the grafPort's visRgn, clipRgn, portRect, and portBits' boundaries. This intersecting rectangle is scrolled by a distance of *dh* horizontally and *dv* vertically. The bits scrolled off the screen are lost and the space created by the scroll is filled with the current grafPort's background pattern (*bkPattern*). This newly created area is added to the update region *updateRegion*.

**tb:!CopyBits** *srcbits dstBits srcRect dstRect mode region* [I-188] Function

Transfers the part of the source bitMap (or source pixMap) defined by the rectangle *srcRect*, to the part of the destination bitMap (or destination pixMap) defined by the rectangle *dstRect*, using a transfer mode *mode* (an integer) and a mask *region*. If you don't want to clip to the masked region, pass **tb:!nilRgn**.

**tb:!SeedFill** *srcPointer dstPointer srcRow dstRow height words seedH seedV* [IV-24] Function

From a source bitMap, calculates a destination bitMap which has the bits set only where the paint can leak from a starting seed point. This is like the MacPaint® bucket tool.

**tb:!SeedCFill** *srcBitMap dstBitMap srcRect dstRect seedH seedV procpointer matchData* [V-71] Function

From a source bitMap (or a source pixMap, calculates a destination bitMap (or a destination pixMap) which has the bits set only where the paint can leak from a starting seed point. This is like the MacPaint bucket tool. Usually, **tb:!nilPtr** will be passed as the *procPointer*. See *Inside Macintosh* Volume V for more details.

**tb:!CalcMask** *srcPointer dstPointer srcRow dstRow height words* [IV-24] Function

From a source bitMap, calculates a destination bitMap which has the bits set only where the paint could not leak from any of the outer edges. This is like the MacPaint lasso tool.

**tb:!CalcCMask** *srcBits dstBits srcRect dstRect RGBColor procPointer longInt* [V-72] Function

From a source pixMap, calculates a destination pixMap which has the bits set only where the paint could not leak from any of the outer edges. This is like the MacPaint lasso tool. Usually **tb:!nilPtr** will be passed as the *procPointer*.

**tb:!CopyMask** *srcBits maskBits dstBits srcRect maskRect dstRect* IV-24] Function

This trap is like **tb:!CopyBits** except it copies from the bitMap *srcBits* to *dstBits* using *maskBits* as the mask.

## Pictures

3.19 These procedures open, close, modify, and dispose of pictures. To make a new picture, make an instance of **tb:Picture** flavor

**tb:Picture** [I-159] Flavor

This flavor defines a QuickDraw picture.

**CAUTION: Do not call tb:!OpenPicture or create a new Picture object if another picture is already open. Always resize the clipRgn to a suitably sized rectangle (using the trap tb:!ClipRect) before calling tb:!OpenPicture.**

**:PicFrame** Init Option of **tb:Picture**

This flavor defines a QuickDraw data structure. Creating an instance of this flavor has the side effect of opening the picture so that QuickDraw begins recording all the calls to drawing routines and picture comments.

**:picsize** Method of **tb:Picture**

This is the picture size in bytes.

**:picframeTop** Method of **tb:Picture**
**:picframeLeft** Method of **tb:Picture**
**:picframeBottom** Method of **tb:Picture**
**:picframeRight** Method of **tb:Picture**

These values describe the enclosing rectangle of the picFrame.

*Example:*

```
(setf myRect (make-instance 'tb:rect))
(make-instance 'tb:Picture  :picFrame  myRect)
```

**tb:!OpenPicture** *rect* [I-189] Function

Makes QuickDraw begin recording all the calls to drawing routines and picture comments. It returns a handle to the picture that has a picture frame defined by *rect*. The preferred method for creating a picture is to make an instance of the **tb:Picture** flavor.

**CAUTION: Do not call tb:!OpenPicture or create a new Picture object if another picture is already open. Always resize the clipRgn to a suitably sized rectangle (using the trap tb:!ClipRect) before calling tb:!OpenPicture.**

**:close** Method of **tb:Picture**
**tb:!ClosePicture** [I-189] Function

Stop the recording of QuickDraw calls for the currently open picture.

**tb:!PicComment** *kind dataSize dataHandle* [I-189] Function

Inserts a picture comment of type *kind* into the currently open picture. Any additional information is passed in *dataHandle*, its size in *dataSize*.

**:draw** Method of **tb:Picture**
**tb:!DrawPicture** *picture rect* [I-190] Function

Draw all of the picture inside its picture frame into *rect*.

**:dispose** Method of **tb:Picture**
**tb:!KillPicture** *picture* [I-190] Function

Dispose of the picture, releasing any memory it uses.

## Calculations with Polygons

3.20 These procedures create, modify, and dispose of polygons. To make a new polygon, make an instance of the **tb:Polygon** flavor.

**tb:Polygon** [I-159] Flavor

This flavor defines a data structure for a QuickDraw polygon data structure. Making an instance of this flavor has the effect of opening the new polygon so that QuickDraw starts saving all line-drawing calls.

**CAUTION: Do not instantiate tb:!OpenPoly or create another polygon object while another region or polygon is still open.**

**:polysize** Method of **tb:Polygon**

This is the polygon size in bytes.

**:polyframeTop** Method of **tb:Polygon**
**:polyframeLeft** Method of **tb:Polygon**
**:polyframeBottom** Method of **tb:Polygon**
**:polyframeRight** Method of **tb:Polygon**

These values describe the enclosing rectangle of the polygon.

**tb:!OpenPoly** [I-190] Function

Tells QuickDraw to start saving all line-drawing calls, returning a new polygon. The preferred method for creating a polygon is to make an instance of the **tb:Polygon** flavor.

**CAUTION: Do not instantiate tb:!OpenPoly or create another polygon object while another region or polygon is still open.**

**:close** Method of **tb:Polygon**
**tb:!ClosePoly** [I-190] Function

Stop the saving of the QuickDraw calls. The maximum size of a polygon is 32K bytes. If you need to know the size of a polygon, use the Memory Manager trap **tb:!GetHandleSize**.

**:dispose** Method of **tb:Polygon**
**tb:!KillPoly** *polygon* [I-191] Function

Dispose of the polygon.

**:offset** Method of **tb:Polygon**
**tb:!OffsetPoly** *polygon dh dv* [I-191] Function

Move *polygon* a horizontal distance of *dh* and a vertical distance *dv*.

## Graphic Operations on Polygons

3.21 These routines perform graphic operations on polygons. They do not move the pen.

**CAUTION: If any line intersects the outline of a polygon more than 50 times, strange things will happen.**

**:frame** Method of **tb:Polygon**
**tb:!FramePoly** *polygon* [I-192] Function

Play back the QuickDraw calls that define the polygon using the current grafPort's pen size, pen mode, and pen pattern.

**:paint** Method of **tb:Polygon**
**tb:!PaintPoly** *polygon* [I-192] Function

Paint the polygon with the current grafPort's pen pattern and transfer mode.

**:erase** Method of **tb:Polygon**
**tb:!ErasePoly** *polygon* [I-192] Function

Fill the polygon with the current grafPort's background pattern (*bkPattern*) using the transfer mode **tb:!patCopy**.

| | |
|---|---|
| **:invert** | Method of **tb:Polygon** |
| **tb:!InvertPoly** *polygon* | [I-192] Function |

Invert every pixel inside the polygon; every white pixel becomes black, every black one becomes white.

| | |
|---|---|
| **:fill** &optional *pattern* | Method of **tb:Polygon** |
| **tb:!FillPoly** *polygon pattern* | [I-192] Function |
| **tb:!FillCPoly** *polygon pixPatHandle* | [V-69] Function |

Fill the the polygon with the pattern using the **tb:!patCopy** transfer mode.

## Calculations with Points

3.22 These routines perform calculations using points. Notice that some of these traps are matched with equivalent methods. Of these trap/method sets, some carry the comment that the method is faster. See the previous paragraph Calculations With Rectangles for an explanation of the speed difference. To create a new point, make an instance of the **tb:Point** flavor.

| | |
|---|---|
| **tb:Point** | [I-139] Flavor |

This flavor defines a QuickDraw pointer. All data associated with a point is stored in one of these instances on the microExplorer side.

| | |
|---|---|
| **:h** *16b-integer* | Init Option of **tb:Point** |
| **:h** | Method of **tb:Point** |
| **:set-h** *16b-integer* | Method of **tb:Point** |
| **:v** *16b-integer* | Init Option of **tb:Point** |
| **:v** | Method of **tb:Point** |
| **:set-v** *16b-integer* | Method of **tb:Point** |

These values define the horizontal and vertical coordinates of the point.

NOTE: For your convenience, the **tb:EventRecord** flavor uses **tb:Point** as a mixin. Therefore, if you have an event record which contains point information, then you can use that **tb:EventRecord** instance anywhere a **tb:Point** instance is needed.

| | |
|---|---|
| **:add** *dh-or-srcPoint* &optional *hv* | Method of **tb:Point** |
| **tb:!AddPt** *srcPoint dstPoint* | [I-193] Function |

Add the coordinates of point *srcPoint* and *dstPoint* and return the resulting point in *dstPoint*. The methods are significantly faster than the trap (see explanation under Calculations With Rectangles). **:add** accepts one **tb:Point** instance or two positions as arguments.

| | |
|---|---|
| **:sub** *dh-or-srcPoint* &optional *hv* | Method of **tb:Point** |
| **tb:!SubPt** *srcPoint dstPoint* | [I-193] Function |

Subtract the coordinates of points *srcPoint* and *dstPoint* and return the resulting point in *dstPoint*. The methods are significantly faster than the trap (see explanation under Calculations With Rectangles). **:sub** accepts one **tb:Point** instance or two positions as arguments.

:= *h-or-srcPoint* &optional *v* — Method of **tb:Point**
**tb:!SetPt** *point h v* — [I-193] Function

Set the horizontal coordinate of the point to *h*, and its vertical coordinate to *v*. The methods are significantly faster than the trap (see explanation under Calculations With Rectangles). := accepts one **tb:Point** instance or two positions as arguments.

**:equal** *ptB* — Method of **tb:Point**
**tb:!EqualPt** *ptA ptB* — [I-193] Function

Return true if *ptA* is equal to *ptB*. The method is significantly faster than the trap (see explanation under Calculations With Rectangles above).

**tb:!LocalToGlobal** *point* — [I-193] Function

Converts the point from the grafPort's local coordinate system to a global coordinate system with the origin at the top left coordinate of the grafPort's bitMap.

**tb:!GlobalToLocal** *point* — [I-193] Function

Converts the point from global coordinates to the grafPort's local coordinate system . This trap is most often used to convert a point that contains the mouse position, which is in global coordinates, into the local coordinates of the current grafPort.

## Miscellaneous Routines

3.23 These routines perform miscellaneous utility functions.

**tb:!Random** — [I-194] Function

Returns a pseudo random 16-bit integer (± 32,767).

**tb:!GetPixel** *h v* — [I-195] Function

Returns true if the pixel at horizontal coordinate *h* and vertical coordinate *v* is black, false if it is white.

**tb:!GetCPixel** *h v RGBColor* — [V-69] Function

Sets *RGBColor* to be the RGB value of the pixel at horizontal coordinate *h* and vertical coordinate *v*.

*Example:*

```
(setf myRGB (make-instance 'tb:RGBColor))
(tb:!GetCPixel 25 44 myRGB) => T
```

**tb:!SetCPixel** *h v RGBColor* [V-69] Function

Sets the color of the pixel (designated by *h* and *v*) to *RGBColor*.

**tb:!StuffHex** *pointer string* [I-195] Function

Stuffs the hexadecimal value in *string* into memory starting at the location *pointer*.

**CAUTION: This is a potentially dangerous trap as no range checking is done. You could easily overwrite vital application or system information unless you know exactly what you are doing.**

**:scale** *srcRect dstRect* Method of **tb:Point**
**tb:!ScalePt** *point srcRect dstRect* [I-195] Function

Multiply the point's horizontal coordinate by the ratio of the destination rectangle's width to the source rectangle's width, and multiply the point's vertical coordinate by the ratio of the destination rectangle's height to the source rectangle's height. The result is returned in *point*.

**:map** *srcRect dstRect* Method of **tb:Point**
**tb:!MapPt** *point srcRect dstRect* [I-196] Function

Map the point in the rectangle *srcRect* to an equivalent position in the rectangle *dstRect*. The result is returned in *point*.

**:map** *srcRect dstRect* Method of **tb:Rect**
**tb:!MapRect** *resultRect srcRect dstRect* [I-196] Function

Map the rectangle *resultRect* within the source rectangle *srcRect* to an equivalently positioned rectangle in the destination rectangle *dstRect*. The result is returned in *resultRect*.

**:map** *srcRect dstRect* Method of **tb:Region**
**tb:!MapRgn** *region srcRect dstRect* [I-196] Function

Map the region *region* in the rectangle *srcRect* to an equivalently positioned region in the rectangle *dstRect*.

**:map** *srcRect dstRect* Method of **tb:Polygon**
**tb:!MapPoly** *polygon srcRect dstRect* [I-197] Function

Map the polygon *polygon* in the rectangle *srcRect* to an equivalently positioned polygon in the rectangle *dstRect*.

**tb:!GetMaskTable** [IV 25] Function

Returns a pointer to a ROM table containing some useful bit masks. See *Inside Macintosh*.

## Customizing QuickDraw Operations

3.24 These are low-level QuickDraw traps, the bottleneck routines. See *Inside Macintosh* pages I-198 through I-200 for more details if you want to use them.

**tb:!SetStdProcs**
**tb:!SetStdCProcs**
**tb:!StdText**
**tb:!StdLine**
**tb:!StdRect**
**tb:!StdRRect**
**tb:!StdOval**
**tb:!StdArc**
**tb:!StdPoly**
**tb:!StdRgn**
**tb:!StdBits**
**tb:!StdComment**
**tb:!StdTxMeas**
**tb:!StdGetPic**
**tb:!StdPutPic**

# Chapter 4
# COLOR MANAGER

## Introduction

4.1 The Color Manager acts as the interface between Color QuickDraw and the display device. It provides a consistent way of displaying color independently of the display device. However, for most applications you will not want to use the Color Manager. Instead, use the Palette Manager.

### Graphic Devices

4.1.1 Every graphic device is characterized by a data structure *gDevice* which contains information about that particular graphic device.

### Color Tables

4.1.2 The complete set of colors in use at any given time for a particular gDevice is kept in a color table record. This table contains a list of all the colors, their concrete values, and their RGB values.

### Inverse Tables

4.1.3 The inverse tables are used to map an RGB value into the nearest equivalent concrete color available for that device.

### Using the Color Manager

4.1.4 Normally, you will not use the Color Manager directly; it is called indirectly when you use Color QuickDraw.

## Color Conversion Traps

4.2 These routines are used for color conversion.

**tb:!Color2Index** *myColor* [V-141] Function

Returns the index of an available color that most closely resembles the absolute color specified by *myColor*, an instance of **tb:RGBColor**.

*Example:*

```
(setf myColor (make-instance 'tb:RGBColor))
(send myColor := 0 0 65535)              ;Set myColor to blue.
(setf blue-index (tb:!Color2Index myColor)) => 6
```

**tb:!Index2Color** *index aColor* [V-141] Function

Sets *aColor*, an instance of **tb:RGBColor**, to the absolute color that corresponds to the color table index *index*.

*Example:*

```
(setf myColor (make-instance 'tb:RGBColor))
(tb:!Index2Color 3 myColor)     ;Get the third color in the table
(send myColor :red)    => 56683
(send myColor :green)  => 2242
(send myColor :blue)   => 1698
```

**tb:!InvertColor** *myColor* [V-141] Function

Sets *myColor*, an instance of **tb:RGBColor**, to the complement of the color *myColor*.

*Example:*

```
(setf myColor (make-instance 'tb:RGBColor))
(send myColor := 0 0 65535)         ; Set myColor to blue.
(tb:!InvertColor myColor)           ; Get complement of blue.
(send myColor :red)    => 65535
(send myColor :green)  => 65535
(send myColor :blue)   => 0
```

**tb:!RealColor** *color* [V-141] Function

Returns true if the color in *color*, an instance of **tb:RGBColor**, exists in the current device's color table.

*Example:*

```
(setf myColor (make-instance 'tb:RGBColor))
(send myColor := 65535 65535 65535) ; Set myColor to white
(tb:!RealColor myColor) => T        ; Is it real?
```

**tb:!GetSubTable** *myColors iTabRes targetTbl* [V-142] Function

Maps the absolute colors in the color table *myColors* onto the nearest available colors and then stores them in the colorSpec value fields of *myColors*.

**tb:!MakeITable** *cTabH iTabH res* [V-142] Function

Generates an inverse color table for the color table *cTabH* with a resolution of *res* bits per channel.

## Color Table Management

4.3 These routines control color table management.

**tb:!GetCTSeed** [V-143] Function

Generates a unique seed value that can be placed in the CTSeed field of a color table created by an application to uniquely distinguish it .

**tb:!ProtectEntry** *index protect* [V-143] Function

Protects or unprotects the entry index in the current grafDevice's color table. If protect is true, the entry is protected; if false, it is unprotected.

**tb:!ReserveEntry** *index reserve* [V-143] Function

Reserves or unreserves the entry *index* in the current grafDevice's color table. If reserve is true, the entry is reserved; if false, the entry is unreserved.

**tb:!SetEntries** *start count aTable* [V-143] Function

Sets the values of *count* number of entries, starting at *start*, in the current grafDevice's color table, using the ColorSpecs pointed to by *aTable*.

**tb:!RestoreEntries** *srcTable dstTable selection* [V-145] Function

Sets a selection of entries from the color table *srcTable* into the color table *dstTable*. *Selection* points at a ReqListRec data structure. See *Inside Macintosh* for details.

**tb:!SaveEntries** *srcTable resultTable selection* [V-144] Function

Sets a selection of entries from the color table *srcTable* into the color table *resultTable*. *Selection* points at a ReqListRec data structure. See *Inside Macintosh* for details.

## Error Handling

4.4 This trap is used to determine the last QuickDraw or Color Manager error that occurred.

**tb:!QDError** [V-145] Function

Returns the error code of the last QuickDraw or Color Manager trap.

## Search and Complement Procedures

4.5 These routines allow an application to override the inverse table matching code.

**tb:!AddSearch** *searchProc* [V-147] Function

Prepends a procedure to the current device record's procedure search list. *searchProc* is a pointer to a procedure in Macintosh memory.

**tb:!AddComp** *compProc* [V-147] Function

Adds a procedure to the head of the current device record's list of complement procedures. *compProc* is a pointer to a procedure in Macintosh memory.

**tb:!DelSearch** *searchProc* [V-147] Function

Removes a custom search procedure from the current device record's list of search procedures. *searchProc* is a pointer to a procedure in Macintosh memory.

**tb:!DelComp** *compProc* [V-147] Function

Removes a custom complement procedure from the current device record's list of complement procedures. *compProc* is a pointer to a procedure in Macintosh memory.

**tb:!SetClientID** *id* [V-147] Function

Sets the gdID field in the current device record to id to identify this client program to its search and complement procedures.

# Chapter 5
# PALETTE MANAGER

## Introduction

5.1 The Palette Manager is used to manage shared color resources, provide exact colors for imaging, and initiate color table animation.

## Color Palette Manager Routines

5.2 These routines initialize the Palette Manager and create, modify, and dispose of palettes.

**tb:!NewPalette** *entries srcColors srcUsage srcTolerance* [V-161] Function

Creates a new palette with *entries* colors from the color table *srcColors*, and returns the new palette as the result. **tb:!NewPalette** sets the usage and tolerance fields of the new palette to *srcUsage* and *srcTolerance*, respectively.

*Example:*

```
(setf myColors (tb:!GetCTable 127))
(setf myPalette (tb:!NewPalette 20 myColors 0 0))
```

**tb:!GetNewPalette** *paletteID* [V-162] Function

Gets a palette object from the Resource Manager and initializes it.

NOTE: A palette ID of 0 is reserved for the system palette resource which is used as the default palette for non-color windows and color windows without assigned palettes.

*Example:*

```
(setf myPalette (tb:!GetNewPalette 128))
```

**tb:!DisposePalette** *myPalette* [V-162] Function

Disposes of the palette *myPalette* and its associated data structures.

*Example:*

```
(tb:!DisposePalette myPalette)
```

**tb:!ActivatePalette** *srcWindow* [V-162] Function

Attempts to provide the color environment described in *srcWindow*'s palette.

**tb:!SetPalette** *dstWindow srcPalette cUpdates* [V-162] Function

Changes *dstWindow's* palette to *srcPalette*. If you want the window to be updated whenever its color environment changes, pass T in *cUpdates*; otherwise, pass NIL.

*Example:*

```
(setf myWindow (make-instance 'tb:window))
(setf myPalette (tb:!GetNewPalette 128))
(tb:!SetPalette myWindow myPalette t)
```

**tb:!GetPalette** *srcWindow* [V-163] Function

Returns the palette associated with *srcWindow*.

**tb:!PmForeColor** *dstEntry* [V-163] Function

Sets the foreground color of the current cGrafPort to the color in palette entry *dstEntry* in the current palette.

**tb:!PmBackColor** *dstEntry* [V-163] Function

Sets the background color of the current cGrafPort to the color in palette entry *dstEntry* in the current palette.

**tb:!AnimateEntry** *dstWindow dstEntry srcRGB* [V-164] Function

Changes the RGB value of *dstEntry* in the palette associated with *dstWindow* to *srcRGB*.

**tb:!AnimatePalette** *dstWindow srcCTab srcIndex* [V-164] Function
*dstEntry dstlLength*

Starting at *srcIndex*, the next *dstLength* entries are copied from *srcCTab* to *dstWindow*'s palette beginning at *dstEntry*.

**tb:!GetEntryColor** *srcPalette srcEntry dstRGB* [V-164] Function

Sets *dstRGB* to the color in the entry *srcEntry* in *srcPalette*.

**tb:!SetEntryColor** *dstPalette dstEntry srcRGB* [V-165] Function

Sets the color in the entry *srcEntry* in *srcPalette* to *srcRGB*.

**tb:GetEntryUsage** *srcPalette srcEntry* [V-165] Function
**tb:!GetEntryUsage** *srcPalette srcEntry* VAR *dstUsage* [V-165] Function
VAR *dstTolerance*

**tb:GetEntryUsage** returns two values: the usage and the tolerance values of entry number *srcEntry* in the palette *srcPalette*. **tb:!GetEntryUsage** is similar except it modifies *dstUsage* and *dstTolerance* to be the usage and tolerance values.

**tb:!SetEntryUsage** *dstPalette dstEntry srcUsage srcTolerance* [V-165] Function

Modifies the usage and tolerance values of *srcEntry* in the palette *srcPalette* to *srcUsage* and *srcTolerance*, respectively.

**tb:!CTab2Palette** *myCTab myPalette srcUsage srcTolerance* [V-165] Function

Copies the color table *myCTab* into the palette *myPalette*. If the *myPalette* is not the same size as the color table, *myPalette* is resized. The usage and tolerance fields of the new entries are set to *srcUsage* and *srcTolerance*, respectively.

*Example:*

```
(setf myColors (tb:!GetCTable 127))
(tb:!CTab2Palette myColors myPalette 0 0)
```

**tb:!Palette2CTab** *myPalette myCTab* [V-166] Function

Copies the palette *myPalette* into the color table *myCTab*. If the color table is not the same size as *myPalette*, the color table is resized.

# Chapter 6
# COLOR PICKER

## Introduction

6.1 The Color Picker is a package that enables an application to ask you to select colors. The package also contains utilities to convert colors between the different color representational schemes.

## Color Picker Package Routines

6.2 This routine displays the Color Picker dialog box.

**tb:!GetColor** *where prompt inColor outColor* [V-174] Function

Displays the Color Picker dialog box at a point *where* with a prompt string *prompt*. The color displayed is *inColor* and the selected color is returned in *outColor* only if you click the OK button. Both *inColor* and *outColor* are instances of **tb:RGBColor**. If you click OK, T is returned. If you cancel, NIL is returned.

*Example:*

```
(setf where    (make-instance 'tb:point))
(setf inColor  (make-instance 'tb:RGBColor))
(setf outColor (make-instance 'tb:RGBColor))
(tb:!GetColor where "Pick a color" inColor outColor)
(send outColor :red)   => 65535
(send outColor :green) => 17508
(send outColor :blue)  => 15005
```

## Color Picker Conversion Routines

6.3 The Color Picker provides routines for converting between the RGBcolor data structures and three other color data structures: CMYColor, HSLColor, and HSVColor. These data structures enable you to use alternate color models. The smallFract data type mentioned in the accessor methods below is a floating point number between zero and one.

**tb:CMYColor** [V-176] Flavor

This flavor defines a CMY color. A new instance of this flavor defaults to black.

| | |
|---|---|
| **:cyan** | Method of **tb:CMYColor** |
| **:magenta** | Method of **tb:CMYColor** |
| **:yellow** | Method of **tb:CMYColor** |
| **:set-cyan** *smallFract* | Method of **tb:CMYColor** |
| **:set-magenta** *smallFract* | Method of **tb:CMYColor** |
| **:set-yellow** *smallFract* | Method of **tb:CMYColor** |

These are the three component values of a CMY color expressed as a SmallFract numbers.

:= &optional *cyan magenta yellow* Method of **tb:CMYColor**

Sets the CMYColor to the given *cyan*, *magenta* ,and *yellow* values. The arguments are smallFract numbers.

**tb:HSLColor** [V-176] Flavor

This flavor defines an HSL color. A new instance of this flavor defaults to black.

**:hue** Method of **tb:HSLColor**
**:saturation** Method of **tb:HSLColor**
**:lightness** Method of **tb:HSLColor**
**:set-hue** *smallFract* Method of **tb:HSLColor**
**:set-saturation** *smallFract* Method of **tb:HSLColor**
**:set-lightness** *smallFract* Method of **tb:HSLColor**

These are the three component values of an HSL color expressed as SmallFract numbers.

:= &optional *hue saturation lightness* Method of **tb:HSLColor**

Sets the HSLColor to the given *hue*, *saturation*, and *lightness* values. The arguments are smallFract numbers.

**tb:HSVColor** [V-176] Flavor

This flavor defines a HSV color. A new instance of this flavor defaults to black.

**:hue** Method of **tb:HSVColor**
**:saturation** Method of **tb:HSVColor**
**:value** Method of **tb:HSVColor**
**:set-hue** *smallFract* Method of **tb:HSVColor**
**:set-saturation** *smallFract* Method of **tb:HSVColor**
**:set-value** *smallFract* Method of **tb:HSVColor**

These are the three component values of an HSV color expressed as SmallFract numbers.

:= &optional *hue saturation value* Method of **tb:HSVColor**

Sets the HSVColor to the given *hue*, *saturation*, and *value* values. The argument values are smallFract numbers.

**tb:!CMY2RGB** *cColor rColor* [V-175] Function
**tb:!RGB2CMY** *rColor cColor* [V-175] Function

This pair of functions converts between a CMY color to an RGB color.

**tb:!HSL2RGB** *hColor rColor* [V-175] Function
**tb:!RGB2HSL** *rColor hColor* [V-175] Function

This pair of functions converts between a HSL color to an RGB color.

**tb:!HSV2RGB** *hColor rColor* [V-175] Function
**tb:!RGB2HSV** *rColor hColor* [V-175] Function

This pair of functions converts between a HSV color to an RGB color.

**tb:!Fix2SmallFract** *f* [V-175] Function
**tb:!SmallFract2Fix** *s* [V-175] Function

This pair of function converts between a fixed-point number *and* a smallFract number and returns the converted value.

# Chapter 7
# FONT MANAGER

## Introduction

7.1 The Font Manager is used by QuickDraw to generate and display various character fonts. The only time you will use these traps is when your application includes a Font or a Style menu. Use the Menu Manager trap **tb:!AddResMenu**, with a resource type "FONT", to generate the Font menu. This trap adds the names of all the fonts in the currently opened resource files to the menu. The Style menu is built up like a normal menu by appending each item to the menu. Use the trap **tb:!RealFont** to see if a font of a particular size is available. If it is, use the Menu Manager trap **tb:!SetItemStyle** to outline the font size item.

With the introduction of the Macintosh Plus two new traps and a new data structure, the **tb:FontMetric** record, were added to the Font Manager. These traps are used for supporting fractional character widths and are of interest only if you are printing directly to a laser printer or making some other use of PostScript®.

The Font Manager has also been changed to handle color fonts. These changes are transparent to the user.

## Initializing the Font Manager

7.2 The routine which initializes the Font Manager should be called once before calling any other Font Manager routine or any Toolbox routine that will call the Font Manager.

**tb:!InitFonts** [I-222] Function

Initializes the Font Manager. You do not need to call this function as it is called for you when you launch a TbServer.

## Getting Font Information

7.3 These routines identify font names and numbers and determine whether a font of the desired size exists.

**tb:GetFontName** *fontNum* [I-223] Function

Returns the font name of the font number *fontNum*. Use this trap instead of **tb:!GetFontName**. To get the name of the font that has a font number of 4, do:

*Example:*

```
(GetFontName 4) => "Monaco"
```

**tb:!GetFontName** *fontNum* VAR *theName* [I-223] Function

Modifies *theName* to be the font name of the font number *fontNum*. To get the name of the font that has a font number of 4, do:

*Example:*

```
(tb:!GetFontName 4 (VAR theName))
theName => "Monaco"
```

**tb:GetFNum** *fontName* [I-223] Function

Returns the font number of the font named in the string *fontName*. Use this trap instead of **tb:!GetFNum**. The constants representing the currently defined font numbers are shown below.

| | |
|---|---|
| **tb:!SystemFont** | [I-219] Constant |
| **tb:!ApplFont** | [I-219] Constant |
| **tb:!NewYork** | [I-219] Constant |
| **tb:!Geneva** | [I-219] Constant |
| **tb:!Monaco** | [I-219] Constant |
| **tb:!Venice** | [I-219] Constant |
| **tb:!London** | [I-219] Constant |
| **tb:!Athens** | [I-219] Constant |
| **tb:!SanFran** | [I-219] Constant |
| **tb:!Toronto** | [I-219] Constant |
| **tb:!Cairo** | [I-219] Constant |
| **tb:!LosAngles** | [I-219] Constant |
| **tb:!Times®** | [I-219] Constant |
| **tb:!Helvetica®** | [I-219] Constant |
| **tb:!Courier** | [I-219] Constant |
| **tb:!Symbol** | [I-219] Constant |
| **tb:!Mobile** | [I-219] Constant |

**tb:!SystemFont** is the number of the default system font such as is used in menu titles. **tb:!ApplFont** is the number of the default application font. The remaining constants represent the standard fonts.

NOTE: The presence of these constants are unrelated to the fonts which are actually installed on any given Macintosh. The Macintosh OS will substitute a font if the requested font is not installed.

To get the number of the NewYork font, do the following:

*Example:*

```
(GetFNum "NewYork") => 2
```

**tb:!GetFNum** *fontName* VAR *theNum* [I-223] Function

Modifies *theNum* to be the font number of the font named *fontName*.

*Example:*

```
(setf theNum 0)
(tb:!GetFNum "NewYork" (VAR theNum))
theNum => 2
```

**tb:!RealFont** *fontNum size* [I-223] Function

Returns true if the font *fontNum* exists in the particular font size *size*.

*Example:*

```
;;;Does New York font exist in 12 point?
(tb:!RealFont  tb:!NewYork 12) => T
```

## Keeping Fonts in Memory

7.4 This trap is used to prevent font information from being purged from memory.

**tb:!SetFontLock** *lockFlag* [I-223] Function

Prevents the purging of the most recently used font's resource if *lockFlag* is true. If *lockFlag* is false, purging is allowed.

## Advanced Routines

7.5 This routine is not normally used by an application directly, but may be of interest to advanced programmers who want to bypass the QuickDraw routines that deal with text.

**tb:!FMSwapFont** *inRec* [I-223] Function

This is an internally used trap. See *Inside Macintosh* for more details.

## Fractional Width Routines

7.6 These routines were added to the Font Manager to support fractional character widths.

**tb:!SetFScaleDisable** *scaleDis* [IV-32] Function

Tells the Font Manager whether to scale a font of another size if it cannot find one of the required size.

**tb:!FontMetrics** *theMetrics* [IV-32] Function

Modifies *theMetrics*, an instance of the **tb:FMetricRec** flavor, with information about the current font.

**tb:FMetricRec** [IV-32] Flavor

This flavor defines the data structure used to record font information (cf. **tb:!FontMetrics**).

**:AscentI** Method of **tb:FMetricRec**
**:AscentF** Method of **tb:FMetricRec**
**:DescentI** Method of **tb:FMetricRec**
**:DescentF** Method of **tb:FMetricRec**

Record the number of pixels the font extends above (ascent) and below (descent) the baseline. Each pixel count is represented by two integers: an integral count and a fractional count representing the 16 bits to the right of the decimal.

**:LeadingI** Method of **tb:FMetricRec**
**:LeadingF** Method of **tb:FMetricRec**

Record the number of pixels of white space between the descenders of one line and the ascenders of the next line down. Each pixel count is represented by two integers: an integral count and a fractional count representing the 16 bits to the right of the decimal.

**:WidMaxI** Method of **tb:FMetricRec**
**:WidMaxF** Method of **tb:FMetricRec**

Record the number of pixels of the widest character in the font. Each pixel count is represented by two integers: an integral count and a fractional count representing the 16 bits to the right of the decimal.

**:WTabHandle** Method of **tb:FMetricRec**

A handle to the global width table describing this font.

**tb:!SetFractEnable** *fractEnable* [IV-32] Function

Enables or disables fractional font widths.

# Chapter 8
# EVENT MANAGER

## Introduction

8.1 The Event Manager is your application's link to its user. When the user presses the mouse button, types on the keyboard, or inserts a disk in the disk drive, your application is notified by means of an event. A typical Macintosh application program is event-driven, meaning it decides what to do by asking the Event Manager for events and responding to them in the appropriate manner.

The Event Manager is probably the most used Toolbox Manager and **tb:EventRecord** the most used, and most useful, Toolbox flavor. All Macintosh applications are event-driven with a main event loop at their core. At the center of the main event loop are the Event Manager traps **tb:!GetNextEvent** and **tb:!WaitNextEvent**. These traps take a **tb:EventRecord** instance as an argument. If there is an event, information about it is returned in various fields of the **tb:EventRecord** instance. If you pass a **tb:EventRecord** instance to the trap, you can then access the various fields of the record by using the instance variables.

If a **tb:EventRecord** instance is passed to the trap **tb:!GetNextEvent** or **tb:!WaitNextEvent** and there is an event, the information returned in the instance depends on the type of event. The event type can be determined from the **:what** instance variable. The time the event was recorded is found in the **:when** instance variable. The current position of the mouse is found in **:V** and **:H**. The information returned in the message field depends on the event type. For window-related events, Update and Activate, the window pointer of the window in question is in **:messageWindow**. For key-down events, the character of the key pressed is in **:messageChar**.

**tb:EventRecord** [I-249] Flavor

This flavor records the information returned by **tb:!WaitNextEvent** and **tb:!GetNextEvent**.

**:What** Method of **tb:EventRecord**

Returns the event code as an integer. The defined event codes are represented by event code constants (e.g., **tb:!mouseDown**) documented in paragraph 8.2.

**:When** Method of **tb:EventRecord**

Returns the time of the event as an integer.

**:Message** Method of **tb:EventRecord**

Returns the variable message portion of the event as an integer. The meaning of this value depends totally upon the associated event code, **:What**. Therefore, the value returned by this method cannot be used until it is "interpreted" in the light of the event code. The alternate methods **:MessageWindow**, **:MessageChar**, **:MessageKey**, and **:MessageDrNum** described below return interpreted values.

**:MessageWindow** Method of **tb:EventRecord**

Assuming that the event record instance records an event related to a window, this method returns **:Message** interpreted as a **tb:Window** instance.

**:MessageDrNum** Method of **tb:EventRecord**

Assuming that the event record instance corresponds to an event code of **tb:!diskEvt**, then this method returns **:Message** interpreted as an integer drive number.

**:MessageChar** Method of **tb:EventRecord**

Assuming that the event record instance records an event related to a key, this method returns **:Message** interpreted as a Lisp character object.

**:MessageKey** Method of **tb:EventRecord**

Assuming that the event record instance records an event related to a key, this method returns **:Message** interpreted as an integer keyboard key code.

**:V** Method of **tb:EventRecord**
**:H** Method of **tb:EventRecord**

Return the coordinates of the mouse at the time the event occurred. If these coordinates are needed for a point argument to some function, then just pass the event record instance itself. The **tb:EventRecord** flavor mixes in the **tb:Point** flavor so that an event record instance can be used anywhere a point instance is required.

**:Modifiers** Method of **tb:EventRecord**

Returns the modifier flags associated with this event as an integer. The defined event modifiers are represented by event modifier masks (e.g., **tb:!activeFlag**) documented paragraph 8.2.

## Event Manager Traps

8.2 The most used trap in the Event Manager is **tb:!WaitNextEvent**. All applications have at their core a routine which repeatedly calls the trap **tb:!WaitNextEvent**. This trap modifies its EventRecord argument to be the next event in the event queue, provided there is one. The Main Event Loop (MEL) keeps calling **tb:!WaitNextEvent** until the trap returns true; then MEL calls the relevant event handler routine.

**tb:!GetNextEvent** *eventMask anEventRecordinstance* [I-257] Function
**tb:!EventAvail** *eventMask anEventRecordinstance* [I-259] Function

**tb:!GetNextEvent** is called to locate the next available event of the type specified by *eventMask*. If such an event exists, the trap returns true with information about the event in various fields of the event record. If the event was located in the event queue, **tb:!GetNextEvent** also removes it from the queue. Normally, one passes **tb:!everyEvent** in *eventMask*. This tells the Event Manager to return the next event in the event queue regardless of type.

NOTE: If using MultiFinder, **tb:!WaitNextEvent** should be used instead of **tb:!GetNextEvent**. All microExplorer applications use MultiFinder.

**tb:!EventAvail** is similar except that if it finds an event in the event queue, it leaves the event there instead of removing it.

**tb:!WaitNextEvent** *eventMask anEventRecordinstance sleep region* Function

Allows an application to use the CPU more efficiently. It helps reduce the null event traffic an application sees by allowing the caller to specify, in addition to *anEventRecord* and *eventMask*, a time value *sleep* for which to relinquish the processor if no events are pending, and a *region* (global coordinates) which describes the current cursor position.

The time value (in 1/60th of a second ticks) allows an application to sleep until a real event occurs or the specified time has passed. The region describing the current mouse position simplifies the application's cursor tracking; the application receives a "mouse-moved" event only when the mouse strays outside the given region. The global variable **tb:!nilrgn** (an empty region) is provided in case you want to default this argument.

It is recommended that any new application use **tb:!WaitNextEvent** whenever possible, enabling background events to get as much time as possible.

NOTE: If your application calls **tb:!WaitNextEvent** do not call the Desk Manager trap **tb:!SystemTask**.

Symbolic constants for the *eventMask* argument and for the event codes returned by the **:what** message to the event record are listed below.

*Example:*

```
;;; create eventrecord instance only once so main event
;;; loop doesn't need to create a new instance over and over
(defun initialize()
  ;; create event record
  (setf *event* (make-instance 'tb:EventRecord))
  ...other init code...)

(catch 'EVENT-LOOP-EXIT
  (loop
    (when (tb:!WaitNextEvent tb:!everyEvent *event* 0
                             tb:!nilrgn)
      (case (the fixnum (send *event* :what))
        (#.tb:!nullEvent nil)
        (#.tb:!mouseDown (MouseDownHandler))
        (#.tb:!keyDown   (KeyDownHandler))
        ...other event handlers...))))
```

To find out which window a mouse-down event is in, call the Window Manager trap **tb:!FindWindow.** See Chapter 9 on the Window Manager for details).

Most of the time we ignore mouse-up events generated when the mouse button is released. The only times you need to know about mouse-up events are when tracking a drag selection, highlighting, and tracking the mouse while the button is still down. In these cases it is better to use the other Event Manager mouse button traps like **tb:!StillDown, tb:!WaitMouseUp,** or **tb:!Button.** Use **tb:!StillDown** for tracking a drag selection.

To test for a double click in an object, see if the difference between **:When** and the previous click in the object is less than **tb:*DoubleTime***. Technically, the trap **tb:!GetDblTime** returns the user's latest choice for a double click interval, but calling this each time takes too much time communicating across the bus. For this reason **tb:*DoubleTime*** has the value at boot time.

*Example:*

```
(defun initialize()
  ...add this to the initialize routine...
  (setf *lasttime* 0)
   ...other init code...)

(defun MouseDownHandler ()
  "handler for all mouse down events"
  (let ((elapsed 0))
    ;; Double click occurs if this click occurred less than
    ;; tb:*DoubleTime* ticks since the last click.
    (setf elapsed (- (send *event* :When) *lasttime*))
    (if (<= elapsed tb:*DoubleTime*)
        (...then double click detected...)
        (...else single click detected...))
    (setf *lasttime* (send *event* :When))))
```

KeyDown events are generated whenever the user presses a key on the keyboard. AutoKey events (repeating KeyDown events) are generated when the user holds down a key for a specified period of time. The length of time is specified by the user with the control panel desk accessory.

KeyDown and AutoKey are almost always handled the same way. You get the character of the key depressed from the EventRecord by doing:

*Example:*

```
(setf theKey (send *event* :MessageChar))
```

There are two types of window related events: activate events and update events. There are two types of activate events: a deactivate event which effects the current active window, and an activate event which effects the window which is to become the active window. You can determine which of these two types the current activate event is by applying the **tb:!activeFlag** mask to the **:Modifiers** instance variable or calling **tb:!activeFlag-p** with the **:Modifiers** values as its argument. To make such a determination, do the following:

*Example:*

```
(if (tb:!activeFlag-p (send *event* :modifiers))
    (activateHandler)
    (deactivateHandler))
```

The following constants serve as masks for the value returned by the **:Modifiers** message to a **tb:EventRecord** instance. Alternately, the predicate functions apply the matching mask to their argument, an event record modifier value.

**tb:!activeFlag** [I-253] Constant
**tb:!activeFlag-p** *eventRecordModifier* Function

The constant is a mask of the event record modifier bit which is set if **tb:!activeEvt** event code represented an activate event; reset if it represented a deactive event. redicate function tests its argument, an event record modifier, for this bit.

**tb:!btnState** [I-253] Constant
**tb:!btnState-p** *eventRecordModifier* Function

The constant is a mask of he event record modifier bit which is set if the mouse button is still down. The predicate function tests its argument, an event record modifier, for this bit.

**tb:!cmdKey** [I-253] Constant
**tb:!cmdKey-p** *eventRecordModifier* Function

The constant is a mask of he event record modifier bit which is set if the Command Key down. The predicate function tests its argument, an event record modifier, for this bit.

**tb:!shiftKey** [I-253] Constant
**tb:!shiftKey-p** *eventRecordModifier* Function

The constant is a mask of he event record modifier bit which is set if the Shift Key is down. The predicate function tests its argument, an event record modifier, for this bit.

**tb:!alphaLock** [I-253] Constant
**tb:!alphaLock-p** *eventRecordModifier* Function

The constant is a mask of he event record modifier bit which is set if the Caps Lock key is down. The predicate function tests its argument, an event record modifier, for this bit.

**tb:!optionKey** [I-253] Constant
**tb:!optionKey-p** *eventRecordModifier* Function

The constant is a mask of he event record modifier bit which is set if the Option key is down. The predicate function tests its argument, an event record modifier, for this bit.

**tb:!controlKey** [I-253] Constant
**tb:!controlKey-p** *eventRecordModifier* Function

The constant is a mask of he event record modifier bit which is set if the Control key is down. The predicate function tests its argument, an event record modifier, for this bit.

The following constants are event masks used to in the *eventMask* argument to functions such as **tb:!WaitNextEvent**, **tb:!GetNextEvent**, and **tb:!FlushEvents**.

**tb:!mDownMask** [I-254] Constant
**tb:!mUpMask** [I-254] Constant
**tb:!keyDownMask** [I-254] Constant
**tb:!keyUpMask** [I-254] Constant
**tb:!autoKeyMask** [I-254] Constant
**tb:!updateMask** [I-254] Constant
**tb:!diskMask** [I-254] Constant
**tb:!activMask** [I-254] Constant
**tb:!networkMask** [I-254] Constant
**tb:!driverMask** [I-254] Constant
**tb:!app1Mask** [I-254] Constant
**tb:!app2Mask** [I-254] Constant
**tb:!app3Mask** [I-254] Constant

These are the event masks corresponding to the event codes described below (e.g., **tb:!mDownMask** is the mask for the **tb:!mouseDown** event code). These masks may be used individually or summed together to specify the events of interest (i.e., the *eventMask* argument) for functions such as **tb:!WaitNextEvent**, **tb:!GetNextEvent**, and **tb:!FlushEvents**. The mask for all possible events is **tb:!everyEvent**. (See the caution concerning **tb:!app4Mask**.)

**tb:!everyEvent** [I-254] Constant

An event mask specifying all possible events.

The following constants are event codes returned by **tb:!WaitNextEvent** and **tb:!GetNextEvent**.

**tb:!nullEvent** [I-249] Constant

Event code indicating that there is no event to process.

**tb:!mouseDown** [I-249] Constant

Event code indicating that the mouse button was pressed. The event record records where and when the mouse button was pressed. The event record itself can be passed to any mouse down handling code which requires a **tb:Point** instance since **tb:Point** is a mixin of **tb:EventRecord**.

**tb:!mouseUp** [I-249] Constant

Event code indicating that the mouse button was pressed. The event record records where the mouse was released. This event is seldom handled directly by application code. The meaning of a **tb:!mouseUp** event usually depends upon the particular **tb:!mouseDown** event which preceded it. Therefore, if the time and place the mouse button was released is important, then the **tb:!mouseDown** handler typically calls a specialized tracking handler which watches for **tb:!mouseUp** and acts accordingly.

**tb:!keyDown** [I-249] Constant

Event code indicating that a key was pressed. The **:messageChar** message to the event record will return the character object representing the key which was pressed.

**tb:!keyUp** [I-249] Constant

Event code indicating that a key was released. There is seldom any need for an application to handle this event since "repeat" keystrokes caused by the user holding down one key continuously is reported through the **tb:!autoKey** event code.

**tb:!autoKey** [I-249] Constant

Event code similar to **tb:!keyDown** except that it is really one of the "repeat" keys caused when the user holds a key down. This event is usually handled the same as **tb:!keyDown**.

**tb:!updateEvt** [I-249] Constant

Event code indicating that the window recorded in the event record needs to be refreshed. The **:messageWindow** message to the event record will return the window which needs updating.

This event is most commonly posted when a window was closed. Thereby, uncovering another window which then receives this **tb:!updateEvt** so that it can replace the black space left by the window which was just closed.

**tb:!diskEvt** [I-249] Constant

Event code indicating that a floppy disk was inserted.

**tb:!activeEvt** [I-249] Constant

Event code indicating that the window recorded in the event record was previously active and has now become inactive or it was previously inactive and has become active. Apply the **tb:!activeFlag** mask to the result of the **:Modifiers** message to the event record to distinguish the two.

**tb:!networkEvt** [I-249] Constant

Event code indicating network activity.

**tb:!driverEvt** [I-249] Constant

Event code indicating device driver activity.

**tb:!app1Evt** [I-249] Constant
**tb:!app2Evt** [I-249] Constant
**tb:!app3Evt** [I-249] Constant

Event codes for events signaled by an application via **tb:!PostEvent.**

**tb:!app4Evt** [I-249] Constant
**tb:!app4Mask** [I-254] Constant

This event code was originally reserved for the application's use, but it has since been preempted by the MultiFinder which is required for the operation of the microExplorer.

**CAUTION: microExplorer applications may *not* use tb:!app4Evt or tb:!app4Mask as their use will interfere with the operation of the MultiFinder.**

**tb:!GetMouse** *mouseLoc* [I-259] Function

Modifies *mouseLoc*, an instance of **tb:Point**, with the the current location of the mouse in the local coordinates of the current grafPort.

*Example:*

```
(setf mouseLocation (make-instance 'tb:Point))
(tb:!GetMouse mouseLocation)
mouseLocation => #<POINT x=99 y=127>
```

**tb:!Button** [I-259] Function

Returns true if the mouse button is pressed down.

**tb:!StillDown** [I-259] Function

Returns true if the mouse button is down and there are no other mouse events in the event queue.

**tb:!WaitMouseUp** [I-259] Function

This trap is the same as **tb:!StillDown** except that if the mouse button is not down, **tb:!WaitMouseUp** removes the preceding mouse-up event before returning false.

**tb:!GetKeys** *keyMap* [I-259] Function

Returns a keyMap of the current state of the keyboard. The keyMap is a 128-bit record. If you need to know the actual key pressed on the keyboard and not just the ASCII character equivalent, the key code can be extracted from the event record by doing:

*Example:*

```
(setf keyCode (send *event* :messageKey))
```

The key code mapping to the keyboard is given in *Inside Macintosh* pages I-251 and V-191, 192.

**tb:!TickCount** [I-260] Function

Returns the current number of ticks (1/60'ths of a second) since the system last started up.

NOTE: Don't rely on the tick count to be exact. It is usually accurate to within one tick but if you are accessing the disk or serial ports extensively, ticks can be lost.

**tb:!GetDblTime** [I-260] Function

Returns the current setting, in ticks (1/60th of a second), for the maximum time difference between mouse-down events to be considered a double-click. This value is set by the Control Panel desk accessory.

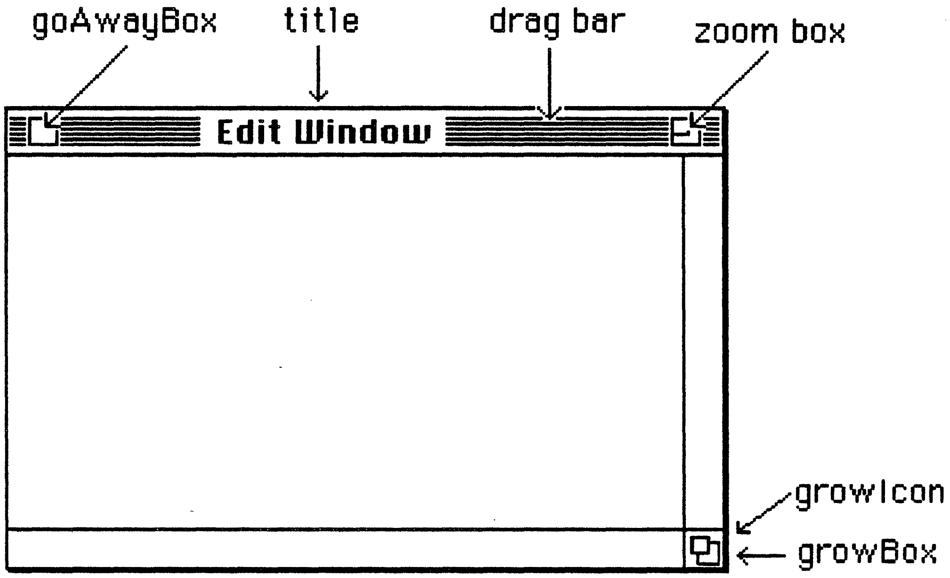
**tb:!GetCaretTime** [I-260] Function

Returns the time, in ticks (1/60th of a second), between blinks of the caret, i.e., the insertion point in a TextEdit record. (The "caret" is typically the I-Beam cursor.)

# Chapter 9
# WINDOW MANAGER

## Introduction

9.1 The diagram below illustrates the primary components of a window.



From a user's point of view, a window is the only means of viewing data. Actually, of course, a window is an illusion carefully maintained by the programmer. For every possible action the user can make, including dragging, growing, zooming, closing, scrolling through the contents, or switching to another window, the programmer must call the necessary functions to maintain this illusion.

In order to understand how to use these functions you need to understand the entire main event loop which encompasses not only the Window Manager, but the Event Manager, QuickDraw, the Control Manager, the Menu Manager, the Dialog Manager, and the Desk Manager.

## Initialization and Allocation

9.2 These routines are used to initialize windows and allocate the necessary memory in the Macintosh heap.

**tb:!InitWindows** [I-281] Function

Initializes the Window Manager. You should never need to call this function since windows are initialized for you when you launch a TbServer.

**tb:Window** [I-276] Flavor
**tb:CWindow** [V-199] Flavor

This flavor defines a color QuickDraw window data structure. **tb:CWindow** is effectively a synonym for **tb:Window**. The Toolbox Interface does not currently implement the old-style, non-color QuickDraw windows. All methods and initialization options of **tb:Window** also apply to **tb:CWindow**.

Furthermore, **tb:Window** and **tb:CWindow** both have **tb:cGrafPort** as a mixin. As such, they inherit all of the instance accessor methods belonging to color grafPorts, and can be used in any routine that requires a **tb:cGrafPort** instance.

**:wStorage** Init Option of **tb:Window**

This is nominally a pointer to where to store the window. In the current Toolbox Interface implementation, it should always be defaulted to **tb:!nilPtr**, the default, which causes a new instance to be created.

**:boundsRect** Init Option of **tb:Window**

This is a **tb:Rect** instance defining the bounds of new window in global coordinates. Defaults a something appropriate to the current screen size.

**:title** Init Option of **tb:Window**

This is the string to be used in the title bar. Defaults to "New Window". If the specified title is too long to fit in the title bar, it will be truncated.

**:visible** Init Option of **tb:Window**

If this option is true, then the new window will immediately become visible. The default is true.

**:behind** Init Option of **tb:Window**

If this option is a pointer to a window, then the new window will be created *behind* the specified window. If this option is **tb:!onePtr**, the default, then the new window will be created in front of all other windows.

**:goAwayFlag** Init Option of **tb:Window**

If this option is true, the default, then a GoAway box will be drawn in the window frame.

**:refCon** Init Option of **tb:Window**

This option represents a 32-bit integer of programmer-defined information which will be permanently associated with the new window. The default is 0. While this user hook is needed in C and Pascal environments, the preferred alternative on the microExplorer is to mix **tb:Window** into your own window flavor which has the extra instance variables you need.

**:procID** Init Option of **tb:Window**

This integer option determines what kind of window is created. Constants defining the available window types are shown below. The default is **tb:!zoomDocProc**.

**tb:Window** instances have the following instance accessor methods in addition to those it inherits from **tb:cGrafPort**.

| | | |
|---|---|---|
| • :WINDOWKIND | ;108 | [ integer ] |
| • :VISIBLE | ;110 | [ boolean ] |
| • :HILITED | ;111 | [ boolean ] |
| • :GOAWAYFLAG | ;112 | [ boolean ] |
| • :ZOOMFLAG | ;113 | [ boolean ] |
| • :STRUCRGN | ;114 | [ rgnhandle ] |
| • :CONTRGN | ;118 | [ rgnhandle ] |
| • :UPDATERGN | ;122 | [ rgnhandle ] |
| • :WINDOWDEFPROC | ;126 | [ handle ] |
| • :DATAHANDLE | ;130 | [ handle ] |
| • :TITLEHANDLE | ;134 | [ handle ] |
| • :TITLEWIDTH | ;138 | [ integer ] |
| • :CONTROLLIST | ;140 | [ controlhandle ] |
| • :NEXTWINDOW | ;144 | [ pointer ] |
| • :WINDOWPIC | ;148 | [ pichandle ] |

**tb:GetWMgrPort** [I-28] Function
**tb:GetCWMgrPort** [V-210] Function

These two traps return the Window Manager port as a grafPort or cGrafPort, respectively. The Window Manager port is generally off limits. It belongs strictly to the Window Manager. In fact, *Are You MultiFinder Friendly?* recommends that you, "Consider the call GetWMgrPort to be for amusement only."

**tb:!GetWMgrPort** *grafPointer* [I-28] Function
**tb:!GetCWMgrPort** *cGrafPointer* [V-210] Function

The Window Manager port is generally off limits. It belongs strictly to the Window Manager. In fact, *Are You MultiFinder Friendly?* recommends that you, "Consider the call GetWMgrPort to be for amusement only."

**tb:!NewWindow** *wStorage boundsrRect title visible procID behind goAwayFlag refCon* [I-281] Function
**tb:!NewCWindow** *wStorage boundsrRect title visible procID behind goAwayFlag refCon* [V-207] Function

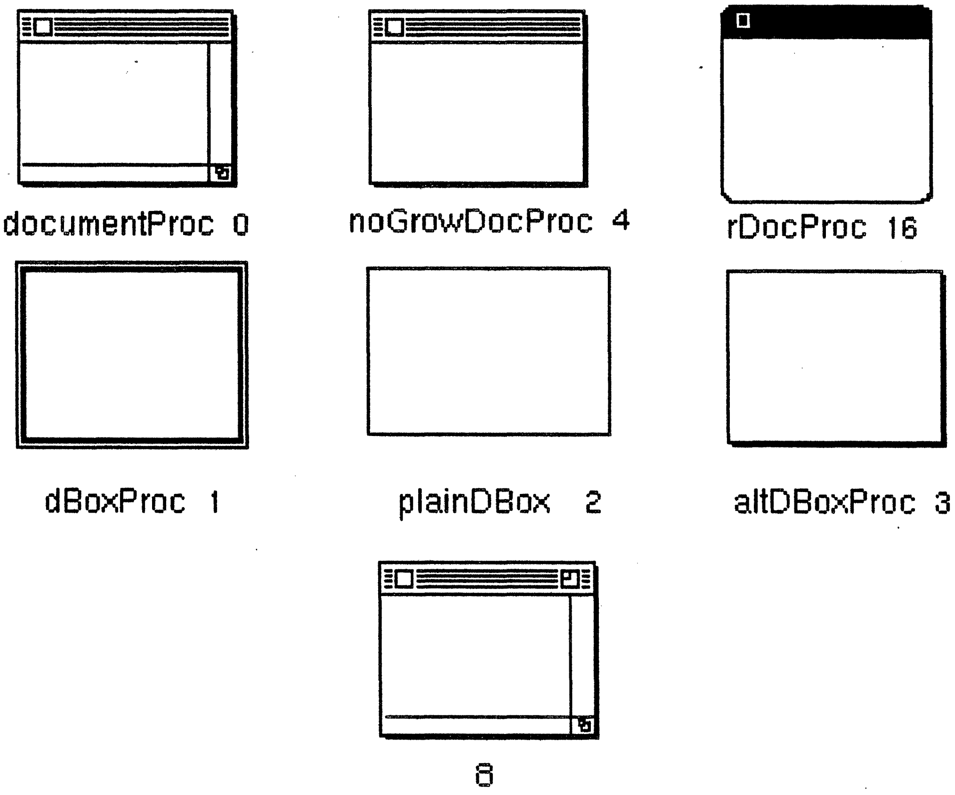
Create new windows, initialize the fields, create all the associated structures, and return a window pointer to the new window.

You should always leave *wStorage* set to the default value of **tb:!nilPtr**. The *boundsRect* is the rectangle that bounds the new window. The *procID* (an integer) indicates the type of window wanted and are defined by the constants shown below.

The argument *behind* is a window pointer and is used if you want the new window to be created in back of the window pointed at by *behind*. Normally, you would pass **tb:!onePtr** and the window would be created in front of all the other existing windows. The *refCon* (a 32-bit integer) field is a place to put information of relevance to the window. It is suggested that you do not use this field. Instead, create a new flavor with any additional fields as instance variables.

| | | |
|---|---|---|
| **tb:!documentProc** | [I-273] | Constant |
| **tb:!dBoxProc** | [I-273] | Constant |
| **tb:!plainDBox** | [I-273] | Constant |
| **tb:!altDBoxProc** | [I-273] | Constant |
| **tb:!noGrowDocProc** | [I-273] | Constant |
| **tb:!zoomDocProc** | [I-273] | Constant |
| **tb:!zoomNoGrow** | [I-273] | Constant |
| **tb:!rDocProc** | [I-274] | Constant |

These constants are used as the *procID* initialization option to **tb:Window flavors.** The general appearance of these windows is shown in the Standard Types of Windows figure below. Notice that the degree of rounding of **tb:!rDocProc** can be controlled by adding by "incrementing" this constant before using it as a *procID* initialization option. See *Inside Macintosh* I-274 for details.



**Standard Types of Windows**

*Example:*

```
(defflavor tb:TEWindow
      (text)
      (tb:Window)
  :gettable-instance-variables
  :settable-instance-variables
  :inittable-instance-variables)

(defmethod (tb:TEWindow :after :init) (init-options)
  (declare (ignore init-options))
  (let ((prtRect (send self :portRect)))
    (send prtRect :inset 3 3)
    (setf text
          (make-instance 'tb:TERec :viewRect prtRect
                                   :destRect prtRect))))

(defmethod (tb:TEWindow :after :dispose) ()
  (send text :dispose))
```

**tb:!GetNewWindow** *windowID wStorage behind* [I-283] Function
**tb:!GetNewCWindow** *windowID wStorage behind* [V-207] Function

These traps are the same as **tb:!NewWindow** and **tb:!NewCWindow** except most of the information about the new window is saved in a previously defined resource of type "WIND" which has a resource ID *windowID*. Additionally, for color windows a window color table resource of type "wctb" will be loaded if one is available with the ID *windowID*.

**tb:!CloseWindow** *window* [I-283] Function

You should never need to call this function. **tb:!CloseWindow** is used if you passed your own storage pointer in *wStorage* when creating the window.

**:dispose** Method of **tb:Window**
**tb:!DisposeWindow** *window* [I-284] Function

Dispose of the window if you passed **tb:!nilPtr** in *wStorage* when creating the window.

## Window Display

9.3 These routines control the display characteristics of a window determining whether it is visible or invisible, active or inactive, etc.

**:set-title** *string* Method of **tb:Window**
**tb:!SetWTitle** *window string* [I-284] Function

Set the title of the window.

**:title** Method of **tb:Window**
**tb:GetWTitle** *window* [I-284] Function
**tb:!GetWTitle** *window* VAR *string* [I-284] Function

**tb:GetWTitle** returns the title of the window as a string. **tb:!GetWTitle** is similar except that it updates *string* with the title string.

*Example:*

```
(tb:!GetWTitle win (VAR title))
title => "A New Title"
```

**:select** Method of **tb:Window**
**tb:!SelectWindow** *window* [I-284] Function

Make the selected window the active window by doing all the necessary highlighting and generating the appropriate activate events. See the example of a main event loop to understand when to use this trap.

**:hide** Method of **tb:Window**
**tb:!HideWindow** *window* [I-285] Function

Make the window invisible. If the window is the front window, it unhighlights the window, brings forward the next window, and generates the appropriate activate events.

**:show** Method of **tb:Window**
**tb:!ShowWindow** *window* [I-285] Function

Make the window visible.

**:erase** Method of **tb:Window**

Erases the content region of the window.

**tb:!ShowHide** *window showFlag* [I-285] Function

If *showFlag* (boolean) is true, **tb:!ShowHide** makes the window visible if the window is invisible. If the window is already visible, it does nothing. If *showFlag* is false, it makes the window invisible if the window is visible and does nothing if the window is already invisible.

NOTE: Unlike **tb:!HideWindow** or **tb:!ShowWindow**, this function never changes the highlighting or front to back ordering of windows.

**tb:!HiliteWindow** *window fHilite* [I-286] Function

Normally, you will not call this trap since the **:select** message will automatically highlight the window. Highlighting a nonactive window is contrary to Macintosh User Interface Guidelines.

**tb:!BringToFront** *window* [I-286] Function

Normally, you will not call this trap since the **:select** message will automatically bring the window to the front.

**tb:!SendBehind** *windowA windowB* [I-286] Function

Normally, you will not call this trap since the **:select** message will usually achieve the desired effect.

**tb:!FrontWindow** [I-286] Function

Returns the front-most window.

NOTE: Providing you have created a new window by making an instance of **tb:Window**, **tb:!FrontWindow** will return the same window instance. This can be very useful if you have stored additional local information in the window instance.

**tb:!DrawGrowIcon** *window* [I-287] Function

Redraws the GrowIcon and associated lines. Call this trap after receiving an activate or update event.

## Mouse Location

9.4 These routines are used to decipher the meaning of a mouse-down event.

**tb:FindWindow** *point* [I-287] Function
**tb:!FindWindow** *point* VAR *whichWindow* [I-287] Function

Given a point, **tb:FindWndow** returns two values: a partCode if the point is in a recognized window, and a windowPtr if it applies. Call this trap after receiving a mouse-down event from the Event Manager trap **tb:!WaitNextEvent**.

**tb:!FindWindow** is similar except that it modifies *whichWindow* to be the new partCode.

*Example:*

```
(defun mousedownHandler (thePt)
  "handler for all mouseDown events"
  (multiple-value-bind (partCode win)
      (tb:FindWindow thePt)
    (case (the fixnum partCode)
      (#.tb:!inMenuBar    (inMenuBarHandler thePt))
      (#.tb:!inSysWindow  (ignore)) ;handled by WaitNextEvent
      (#.tb:!inContents   (inContentHandler win thePt))
      (#.tb:!inDrag       (inDragHandler win thePt))
      (#.tb:!inGrow       (inGrowHandler win thePt))
      (#.tb:!inGoAway     (inGoAwayHandler win thePt))
      (#.tb:!inZoomIn     (inZoomInHandler win thePt))
      (#.tb:!inZoomOut    (inZoomOutHandler win thePt)))
      (otherwise          (ignore)))))
```

NOTE: The (the fixnum ...) form around **partCode** allows the compiler to use a microcoded dispatch function rather than a series of compares. Since we are dispatching on the numeric value of the symbols such as **tb:!inMenuBar**, we need the #. reader macro to force evaluation of the symbols (because case normally dispatches on the symbols themselves).

| | | |
|---|---|---|
| **tb:!inDesk** | [I-287] | Constant |
| **tb:!inMenuBar** | [I-287] | Constant |
| **tb:!inSysWindow** | [I-287] | Constant |
| **tb:!inContents** | [I-287] | Constant |
| **tb:!inDrag** | [I-287] | Constant |
| **tb:!inGrow** | [I-287] | Constant |
| **tb:!inGoAway** | [I-287] | Constant |
| **tb:!inZoomIn** | [I-287] | Constant |
| **tb:!inZoomOut** | [I-287] | Constant |

These constants collectively define the possible *partCodes* which may be returned by **tb:!FindWindow** or **tb:FindWindow**. This integer code identifies the part of the window on which the mouse was clicked. The mouseDown handler of the event loop would normally dispatch on this partCode to determine the appropriate response to the mouse click. Typical responses are as follows:

**tb:!inDesk** - This partCode can be safely ignored.

**tb:!inMenuBar** - Call **tb:!MenuSelect** (q.v.).

**tb:!inSysWindow** - The user clicked on a Desk Accessory. See the Desk Manager for details.

**tb:!inDrag** - Call **tb:!DrawWindow**.

**tb:!inGrow** - First call **tb:!GrowWindow** and then **tb:!SizeWindow**.

**tb:!inGoAway** - First call **tb:!TrackGoAway** and if it returns true, then dispose of the window.

**tb:!inZoomIn** or **tb:!inZoomOut** - Call **tb:!TrackBox** and if it returns true, then call **tb:!ZoomWindow**.

**tb:!inContents** - The action depends upon what controls, if any, your window has. In general, if your window does have controls, call **tb:!FindControl** to determine which control was selected and then implement a control-specific dispatch similar to this one.

If your window does not have associated controls, then treat this partCode as a non-specific mouse event.

**tb:!TrackGoAway** *window point* [I-288] Function

Called when there is a mouse-down event in the goAwayBox of a window. It highlights the goAwayBox until the mouse button is released and returns T if the mouse was still inside the goAwayBox when released. If **tb:!TrackGoAway** returns true, send the window a **:dispose** message.

**tb:!TrackBox** *window point partCode* [IV-50] Function

If the trap **tb:!FindWindow** returns a result of **tb:!inZoomIn** or **tb:!inZoomOut**, call **tb:!TrackBox** giving the current window *window*, the current mouse position *point*, and the *partCode* returned by the trap **tb:!FindWindow**. If the trap result is true, call the trap **tb:!ZoomWindow**.

**tb:!ZoomWindow** *window partCode front* [IV-50] Function

Zooms *window* according to *partCode* and will bring the window to the front if *front* is true.

**:inside-p** *point* Method of **tb:Window**

Returns true if *point* is inside the window.

## Window Movement and Sizing

9.5 These procedures control the movement and size of a window.

**:move** *h v* &optional (*front* t) Method of **tb:Window**
**tb:!MoveWindow** *window h v front* [I-289] Function

Move *window* to a point with coordinates (*h*,*v*) where *h* and *v* are expressed in global coordinates. If *front* is true (the default) and *window* is not the active window, **tb:!SelectWindow** is called to make it the active window.

**tb:!DragWindow** *window point boundsRect* [I-289] Function

Drags an outline of *window* starting at the point *point*, specified in global coordinates, limiting the drag area to *boundsRect*. (See *Inside Macintosh* for details).

**tb:!GrowWindow** *window point rect* [I-289] Function

Draws a grow image of *window*, with size *rect*, that tracks the mouse starting at *point*. *Point* should be in global coordinates. When the mouse button is released, the trap returns two values: the new height and width of *window*.

*Example:*

```
(defun inGrowHandler (win startPt)
  (let ((sizeRect (make-instance 'tb:rect
                   :top 100 :bottom 300
                   :left 100 :right 300)))
    (multiple-value-bind (newHeight newWidth)
       (tb:!GrowWindow win startPt sizeRect)
      (when (and (/= 0 newHeight)(/= 0  newWidth))
         ;; then new values aren't 0, so the size DID change
        (tb:!SizeWindow win newWidth newHeight t)))))
```

**:width** — Method of **tb:Window**
**:height** — Method of **tb:Window**

Return the width and height of the window, respectively.

**tb:!SizeWindow** *window width height fUpdate* — [I-290] Function

Resizes window to *width* and *height*. If *fUpdate* is true, any newly created part of the content's region is put into the update region.

## Update Region Maintenance

9.6 These routines control the areas that will be affected during an update event.

**:inval** — Method of **tb:Window**

Adds the entire portRect of **self** into the update region of the window whose grafPort is the current port.

**:inval** — Method of **tb:Rect**
**tb:!InvalRect** *rect* — [I-291] Function

Add *rect* into the update region of the window whose grafPort is the current port.

**:inval** — Method of **tb:Region**
**tb:!InvalRgn** *region* — [I-291] Function

Add *region* into the update region of the window whose grafPort is the current port.

**:valid** — Method of **tb:Rect**
**tb:!ValidRect** *rect* — [I-292] Function

Remove *rect* from the update region of the window whose grafPort is the current port.

**:valid** — Method of **tb:Region**
**tb:!ValidRgn** *region* — [I-292] Function

Remove *region* from the update region of the window whose grafPort is the current port.

**tb:!BeginUpdate** *window* [I-292] Function
**tb:!EndUpdate** *window* [I-293] Function

Call **tb:!BeginUpdate** upon receipt of an update event for *window*. Call **tb:!EndUpdate** when you are finished handling an update event for *window*.

## Miscellaneous Routines

9.7 The following section outlines the miscellaneous Window Manager routines.

**tb:!SetWRefCon** *window longInt* [I-293] Function
**tb:!GetWRefCon** *window* [I-293] Function

You should never need to use these traps. If you need to store additional information about a window, create a new flavor of window that contains any additional fields required.

**tb:!SetWindowPic** *window picture* [I-293] Function

Stores *picture* in the window record of *window* so that when the window's contents are to be drawn, the Window Manager draws *picture* instead of generating an update event.

**tb:!GetWindowPic** *window* [I-293] Function

Returns any picture handle that may be associated with *window*.

**:pin** Method of **tb:Rect**
**tb:!PinRect** *rect point* [I-293] Function

Returns two values indicating the vertical and horizontal coordinates of the point within the rectangle *rect* that is closest to the point *point*. The method is faster than the function.

**:dragGray** *point* &key **:limitRect :slopRect :axis :actionProc** Method of **tb:Region**
**tb:!DragGrayRgn** *region point limitRect slopRect axis actionProc* [I-294] Function

Pulls a dotted gray outline of the *region* around following the movements of the mouse until the mouse button is released. All points and rectangles are in the local coordinates of the current grafPort. The *axis* value should be one of the constants **tb:!noConstraint**, **tb:!hAxisOnly**, or **tb:!vAxisOnly** as described below. The *actionProc* should always be **tb:!nilPtr**. If the mouse button is released within *slopRect*, the function returns multiple values *dh* and *dv*. If the mouse button is released outside *slopRect* both returned values are -32768 (#x8000). Refer to *Inside Macintosh* for details.

### tb:!noConstraint [I-295] Constant
### tb:!hAxisOnly [I-295] Constant
### tb:!vAxisOnly [I-295] Constant

These three constant are used as *axis* arguments to Window Manager and Control Manager functions which may wish to constrain mouse movement in some way. The choices are unconstrained motion, horizontal motion only, or vertical motion only.

*Example:*

```
tb:           ; with this, we don't have to prefix everything with tb:
(defun test-draggrayrgn ()
  (let ((event (make-instance 'EventRecord))
        (pt    (make-instance 'Point))
        (w     (make-instance 'Window
                 :title "Press any key to exit"
                 :boundsrect (make-instance 'Rect
                               :left 50 :top 50
                               :right 350 :bottom 300)))
         slopr r drgrgn rgn)
    (!SetPort w)
    (setf slopr (send (send w :portrect) :inset 50 50))
    (send slopr :frame)

    (setf r (send (send (make-instance 'Rect) := slopr)
                  :inset 50 50))
    (setf rgn (send (make-instance 'Region) := r))
    (send rgn :union (send r :offset 25 25))
    (setf drgrgn (make-instance 'Region))
    (send rgn :fill)
    (catch 'EVENT-LOOP-EXIT
       (loop       ;throw to EVENT-LOOP-EXIT to exit this loop
        (when (!WaitNextEvent !everyEvent event 10
                              !nilRgn)
           ;;then we have an event we are supposed to process
          (case (the fixnum (send event :what))
            (#.!mouseDown
              (!GlobalToLocal (send pt := event))
              (when (send rgn :inside-p pt)
                 ;;then mouse clicked inside our region
                (send drgrgn := rgn)
                (multiple-value-bind (dy dx)
                    (send drgrgn :dragGray
                          pt :sloprect slopr)
                  (when (and (not (= 0 dx dy))
                             (not (= #x8000 dx dy)))
                      ;;then it was moved and it stayed in bounds
                    (send (send (send rgn :erase)
                                :offset dx dy) :fill)))))
            (#.!keyDown
               ;;a key was pressed, that's our signal to quit
              (send w :dispose)
              (throw 'EVENT-LOOP-EXIT nil))))))))
```

### tb:!GetGrayRgn [V-208] Function

Returns the current desktop region.

**tb:!SetDeskCPat** *pixPat* [V-210] Function

Sets the desktop pattern to the given pixel pattern.

## Low-Level Routines

9.8 These are all low-level Window Manager traps and are unlikely ever to be used. See *Inside Macintosh* for more details.

**tb:!CheckUpdate**
**tb:!ClipAbove**
**tb:!SaveOld**
**tb:!DrawNew**
**tb:!PaintOne**
**tb:!PaintBehind**
**tb:!CalcVis**
**tb:!CalcVisBehind**

## Color Window Manager Traps

9.9 These traps control the color characteristics of a given window.

**tb:!SetWinColor** *window CTabHandle* [V-207] Function

Sets the window's color table. If *window* has no auxiliary window record, a new one is created with *CTabHandle* and added to the head of the auxiliary window list. If *window* has an auxiliary window record, its contents are replaced by *CTabHandle*. After setting the window's color table, the window is automatically redrawn in the new colors.

**tb:WinCTab** [V-202] Flavor

This flavor defines a color window color table. All fields must be individually set after instantiation.

| | |
|---|---|
| **:set-content.value** *partCode* | Method of **tb:WinCTab** |
| **:set-frame.value** *partCode* | Method of **tb:WinCTab** |
| **:set-text.value** *partCode* | Method of **tb:WinCTab** |
| **:set-hilite.value** *partCode* | Method of **tb:WinCTab** |
| **:set-titlebar.value** *partCode* | Method of **tb:WinCTab** |

These methods initialize partCodes for the color table and must be set to the constants **tb:!wContentColor**, **tb:!wFrameColor**, **tb:!wTextColor**, **tb:!wHiliteColor**, and **tb:!wTitleBarColor** respectively.

| | |
|---|---|
| **:content.red** | Method of **tb:WinCTab** |
| **:content.blue** | Method of **tb:WinCTab** |
| **:content.green** | Method of **tb:WinCTab** |
| **:set-content.red** *16b-unsigned-integer* | Method of **tb:WinCTab** |
| **:set-content.blue** *16b-unsigned-integer* | Method of **tb:WinCTab** |
| **:set-content.green** *16b-unsigned-integer* | Method of **tb:WinCTab** |

These methods handle the RGB color for the window background.

**:frame.red** Method of **tb:WinCTab**
**:frame.blue** Method of **tb:WinCTab**
**:frame.green** Method of **tb:WinCTab**
**:set-frame.red** *16b-unsigned-integer* Method of **tb:WinCTab**
**:set-frame.blue** *16b-unsigned-integer* Method of **tb:WinCTab**
**:set-frame.green** *16b-unsigned-integer* Method of **tb:WinCTab**

These methods handle the RGB color for the window frame.

**:text.red** Method of **tb:WinCTab**
**:text.blue** Method of **tb:WinCTab**
**:text.green** Method of **tb:WinCTab**
**:set-text.red** *16b-unsigned-integer* Method of **tb:WinCTab**
**:set-text.blue** *16b-unsigned-integer* Method of **tb:WinCTab**
**:set-text.green** *16b-unsigned-integer* Method of **tb:WinCTab**

These methods set the RGB color for window text.

**:hilite.red** Method of **tb:WinCTab**
**:hilite.blue** Method of **tb:WinCTab**
**:hilite.green** Method of **tb:WinCTab**
**:set-hilite.red** *16b-unsigned-integer* Method of **tb:WinCTab**
**:set-hilite.blue** *16b-unsigned-integer* Method of **tb:WinCTab**
**:set-hilite.green** *16b-unsigned-integer* Method of **tb:WinCTab**

These methods set the RGB color for the hilite lines in the title bar when the window is highlighted.

**:titlebar.red** Method of **tb:WinCTab**
**:titlebar.blue** Method of **tb:WinCTab**
**:titlebar.green** Method of **tb:WinCTab**
**:set-titlebar.red** *16b-unsigned-integer* Method of **tb:WinCTab**
**:set-titlebar.blue** *16b-unsigned-integer* Method of **tb:WinCTab**
**:set-titlebar.green** *16b-unsigned-integer* Method of **tb:WinCTab**

These methods set the RGB colors for the (unhighlighted) title bar background.

**:ctsize** Method of **tb:WinCTab**
**:set-ctsize** *integer* Method of **tb:WinCTab**

The number of partCodes in the table less one.

**tb:!wContentColor** [V-204] Constant
**tb:!wFrameColor** [V-204] Constant
**tb:!wTextColor** [V-204] Constant
**tb:!wHiliteColor** [V-204] Constant
**tb:!wTitleBarColor** [V-204] Constant

These constants serve as partCode identifiers for the window color table structure. In particular, they are the initial values of the **:content.value**, **:frame.value**, **:text.value**, **:hilite.value**, and **:titlebar.value** instance variables of the **tb:winCTab** flavor, respectively.

**tb:!GetAuxWin** *window AuxWinRec* [V-207] Function

Sets *AuxWinRec* to be the window's auxiliary window record. If *window* has an auxiliary record, **tb:!GetAuxWin** returns true. If *window* does not have an auxiliary record, **tb:!GetAuxWin** returns false and sets *AuxWinRec* to the default auxiliary record. If *window* is **tb:!nilPtr**, **tb:!GetAuxWin** returns true and *AuxWinRec* becomes the default record.

**tb:AuxWinRec** [V-201] Flavor

Creates a new, uninitialized auxiliary window record object.

**:awnext** Method of **tb:AuxWinRec**
**:set-awnext** *AuxWinHandle* Method of **tb:AuxWinRec**

Handle of next record on the list.

**:awowner** Method of **tb:AuxWinRec**
**:set-awowner** *WindowPtr* Method of **tb:AuxWinRec**

Pointer to this window's owner window.

**:awctable** Method of **tb:AuxWinRec**
**:set-awctable** *CTabHandle* Method of **tb:AuxWinRec**

Handle to window's color table.

**:awrefcon** Instance Variable of **tb:AuxWinRec**

This instance variable is reserved for the application's use.

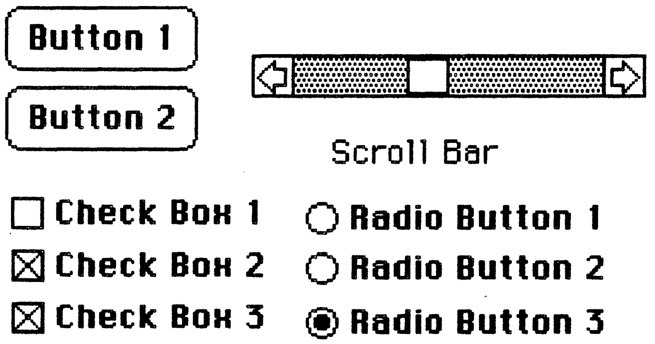
**tb:!GetWVariant** *window* [V-208] Function

Returns the variant code for *window*.

# Chapter 10
# CONTROL MANAGER

## Introduction

10.1 A *control* is an object in a window that is selected by pressing the mouse button while the cursor is within the bounds of the object. This either causes an immediate action or changes the value of a program parameter which will have some later effect. The Control Manager is used to create, change, and dispose of controls. There are four predefined controls: buttons, check boxes, radio buttons, and scroll bars. See the illustration below for examples of each of these items.



The available controls are identified by integer *procIDs* which have the following symbolic names in the Toolbox Interface:

**tb:!pushButProc** [I-315] Constant

A *button* is used when you want the reaction to occur immediately after the mouse button has been pressed.

**tb:!checkBoxProc** [I-315] Constant
**tb:!radioButProc** [I-315] Constant

*Check boxes ;* and *radio buttons* are generally arranged in groups and are used to display settings. The difference between them is that only one radio button of a group can be "on" at a given time, whereas any or all check boxes in a group can be "on" at the same time.

**tb:!scrollBarProc** [I-315] Constant

*Scroll bars ;* enable the user to change the part of the window that is displayed. They are used when the contents of a window are bigger than the window's display area.

**tb:!useWFont** [I-315] Constant

Add this constant to **tb:!pushButProc**, **tb:!checkBoxProc**, **tb:!radioButProc**, or **tb:!scrollBarProc** to create a procID which will use the window's grafPort font for annotating the control.

## Initialization and Allocation

10.2 These routines create and dispose of controls.

### **tb:ControlRecord** [I-317] Flavor

This flavor defines a new control according to its initialization options. An instance of this flavor may be used anywhere a ControlHandle is called for.

NOTE: Since controls belong to windows, make sure that when creating a control there is a window present.

**:theWindow** *pointer* Init Option of **tb:ControlRecord**
**:owner** Method of **tb:ControlRecord**

This is the pointer to the window which this control belongs to. The default is the frontmost window.

**:boundsRect** *rect* Init Option of **tb:ControlRecord**
**:top** Method of **tb:ControlRecord**
**:bottom** Method of **tb:ControlRecord**
**:left** Method of **tb:ControlRecord**
**:right** Method of **tb:ControlRecord**

This is a rectangle defined in the containing window's local coordinates of where this control will appear. The default is the rectangle defined by (50 50 100 100). The *rect* argument is a **tb:Rect** instance of a list of four integers defining the corners of the rectangle. The methods correspond to the **tb:Rect** instance variables.

**:title** *string* Init Option of **tb:ControlRecord**
**:title** Method of **tb:ControlRecord**
**:set-title** *StringPointer* Method of **tb:ControlRecord**

This is a string of up to 255 characters which becomes the title of the control. The string may be empty. If it is too long, it is truncated. The default is "Control Title".

**:visible** *visible-p* Init Option of **tb:ControlRecord**
**:vis** Method of **tb:ControlRecord**
**:set-vis** *byte* Method of **tb:ControlRecord**

If this option is true, then the control will be visible. For the initialization option, true is non-**nil**. For the methods, true is 1 and false is 0. The default is true.

**:value** *16b-integer* Init Option of **tb:ControlRecord**
**:min** *16b-integer* Init Option of **tb:ControlRecord**
**:max** *16b-integer* Init Option of **tb:ControlRecord**
**:value** Method of **tb:ControlRecord**
**:min** Method of **tb:ControlRecord**
**:max** Method of **tb:ControlRecord**
**:set-value** *16b-integer* Method of **tb:ControlRecord**
**:set-min** *16b-integer* Method of **tb:ControlRecord**
**:set-max** *16b-integer* Method of **tb:ControlRecord**

These specify the integer initial value, the maximum value, and the minimum value allowed for this control. Default is the range of 0..10 with an initial value of 0.

**:procID** *16b-integer* Init Option of **tb:ControlRecord**

This integer value defines the type of control. The standard types (push buttons, radio buttons, check boxes, and scroll bars) are represented by the constant symbols **tb:!pushButProc** *et al.* defined above. The default is a push button.

**:refCon** *32b-integer* Init Option of **tb:ControlRecord**
**:refCon** Method of **tb:ControlRecord**
**:set-refCon** *32b-integer* Method of **tb:ControlRecord**

This is a 32-bit integer reserved for the application's use. A hook such as this is needed in C or Pascal environments; but a better alternative on the microExplorer is to define a new flavor which uses **tb:ControlRecord** as a mixin and then add your extra instance variable to that.

**:defProc** Method of **tb:ControlRecord**
**:set-defProc** *handle* Method of **tb:ControlRecord**

This is a handle to the Macintosh function which defines this control.

**:controlAction** Method of **tb:ControlRecord**
**:set-controlAction** *procPointer* Method of **tb:ControlRecord**

This is a pointer to the control's default action procedure.

**:next** Method of **tb:ControlRecord**

This is a handle to the next control.

**:hilite** Method of **tb:ControlRecord**
**:set-hilite** *partCode* Method of **tb:ControlRecord**

This is the control partCode to be highlighted. A value of 255 means that all controls are shown as inactive.

**tb:!NewControl** *theWindow boundsRect title visible value* [I-319] Function
*min max procID refCon*

Creates a new control of type *procID*, associated with the window *theWindow*, and returns a handle *ControlHandle* to this new control. It is bound by the rectangle *boundsRect* and has the title name *title*. It can have a range of values from *min* to *max*, with its initial value being that specified in *value*.

To set up a simple push button, do the following:

*Example:*

```
(setf controlRect
     (make-instance 'tb:rect :left 23 :top 79
                              :right 95 :bottom 97))
(setf myControl (tb:!NewControl myWindow controlRect
                              "OK" t 100 0 100 0 0))
```

**tb:!GetNewControl** *controlID theWindow* [I-321] Function

This trap operates in the same manner as **tb:!NewControl** except that it gets the control definition information from a resource of type "CNTL" with a resource ID *controlID*.

**:dispose** Method of **tb:ControlRecord**
**tb:!DisposeControl** *theControl* [I-321] Function

Dispose of the control *theControl* and remove it from the control list and releases any memory it uses.

**tb:!KillControls** *theWindow* [I-321] Function

Disposes of all the controls associated with the window *theWindow*. The traps **tb:!CloseWindow** and **tb:!DisposeWindow** automatically dispose of any controls associated with the window.

## Control Display

10.3 These procedures affect the appearance of a control but not its size or location.

**tb:!SetCTitle** *theControl title* [I-321] Function

Set the title string of *theControl* to *title*. (See also **:set-title** method.)

**tb:GetCTitle** *theControl* [I-321] Function
**tb:!GetCTitle** *theControl* VAR *title* [I-321] Function

**tb:GetCTitle** returns the title string of the control *theControl*. **tb:!GetCTitle** is similar except that *title* is modified to be the title string. (See also **:title** method.)

*Example:*

```
(tb:!GetCTitle myControl (VAR title))
title => "A new title"
```

**:hide** Method of **tb:ControlRecord**
**tb:!HideControl** *theControl* [I-322] Function

Make *theControl* invisible.

**:show** Method of **tb:ControlRecord**
**tb:!ShowControl** *theControl* [I-322] Function

Make *theControl* visible.

**tb:!DrawControls** *theWindow* [I-322] Function

Draws all the controls associated with *theWindow*.

**tb:!Draw1Control** *theControl* [IV-53] Function

Draws *theControl* if it is visible within the window.

**tb:!HiliteControl** *theControl hiliteState* [I-322] Function

Highlights *theControl* according to the state specified in *hiliteState*. See *Inside Macintosh* for additional information on *hiliteState*.

**tb:!UpdtControl** *theWindow update* [IV-53] Function

Draws all the controls associated with the window *theWindow* that are in the update region *update*.

## Mouse Location

10.4 These routines handle the various responses to pressing a mouse button.

**tb:FindControl** *thePt theWindow* [I-323] Function
**tb:!FindControl** *thePt theWindow* VAR *whichControl* [I-323] Function

**tb:FindControl** is called when a mouse-down event is recorded in the content region of a window; this trap checks to see if *thePt* is inside any of the active controls associated with *theWindow*. If the event happened while the mouse was inside a control, the trap returns two values: the partCode for the part of the control the point is in and the control's handle.

**tb:!FindControl** is similar to **tb:FindControl** except that *whichControl* is modified to be the partCode and no value is returned.

The **tb:!FindControl** and **tb:FindControl** traps expect the mouse position in local coordinates, whereas the Window Manager **tb:!FindWindow** and **tb:FindWindow** traps expect the mouse position in global coordinates. You must convert the mouse position's coordinate system using the QuickDraw trap **tb:!GlobalToLocal**.

| | | |
|---|---|---|
| **tb:!inButton** | [I-316] | Constant |
| **tb:!inCheckBox** | [I-316] | Constant |
| **tb:!inUpButton** | [I-316] | Constant |
| **tb:!inDownButton** | [I-316] | Constant |
| **tb:!inPageUp** | [I-316] | Constant |
| **tb:!inPageDown** | [I-316] | Constant |
| **tb:!inThumb** | [I-316] | Constant |

These constants name the standard control types as returned by **tb:!FindControl** and **tb:FindControl**.

| | |
|---|---|
| **tb:!inButton** | - Simple push button. |
| **tb:!inCheckBox** | - Check box or radio button. |
| **tb:!inUpButton** | - The up arrow of a scroll bar. |
| **tb:!inDownButton** | - The down arrow of a scroll bar. |
| **tb:!inPageUp** | - The page-up region of a scroll bar. |
| **tb:!inPageDown** | - The page-down region of a scroll bar. |
| **tb:!inThumb** | - The thumb region of a scroll bar. |

**tb:!TrackControl** *theControl startPt actionProc* [I-323] Function

If **tb:!FindControl** or **tb:FindControl** returns a partCode, call the trap **tb:!TrackControl** to track the mouse. This involves calling a track action procedure, the type of action depending on what type of control *theControl* is.

For example, a mouse-down event in the thumb of a scroll bar, calls a procedure which outlines the thumb while the mouse button is still down. **tb:!TrackControl** returns when the user lets up the mouse button. **tb:!TrackControl** returns either the partCode returned by **tb:!FindControl**, or 0, which means the user moved the mouse out of the control before releasing the mouse button. In the latter case, the program should do nothing. Pass **tb:!nilPtr** in *actionProc*.

**tb:!TestControl** *theControl thePoint* [I-325] Function

Returns the partCode of the part of the control *theControl* that the point *thePoint* is in.

## Control Movement and Sizing

**10.5** These routines are called when moving, dragging, or resizing a control.

**tb:!MoveControl** *theControl h v* [I-325] Function

Moves the position of *theControl* to the point (*h*,*v*) in the local coordinate system of its window, and draws *theControl* in its new position.

**tb:!DragControl** *theControl startPt limitRect slopRect axis* [I-325] Function

Drags a dotted outline of *theControl* starting at the point *startPt* and draws *theControl* in its new position. This is similar to **tb:!DragGrayRgn**.

**tb:!SizeControl** *theControl width height* [I-326] Function

Changes the size of *theControl's* boundary rectangle to the new *width* and *height* specified and redraws *theControl*.

**tb:!SetCtlValue** *theControl theValue* [I-326] Function

Set the current value of *theControl* to *theValue* and redraw *theControl* with its new value. (See also **:set-value** method.)

**tb:!GetCtlValue** *theControl* [I-326] Function

Returns the current value of *theControl*. (See also **:value** method.)

**tb:!SetCtlMin** *theControl minValue* [I-326] Function

Sets the minimum value of *theControl* to *minValue* and redraws *theControl* with its new minimum value. (See also **:set-min** method.)

**tb:!GetCtlMin** *theControl* [I-327] Function

Returns the minimum value of *theControl*. (See also **:min** method.)

**tb:!SetCtlMax** *theControl maxValue* [I-327] Function

Sets the maximum value of *theControl* to *maxValue* and redraws *theControl* with its new maximum value. (See also **:set-max** method.)

**tb:!GetCtlMax** *theControl* [I-327] Function

Returns the maximum value of *theControl*. (See also **:max** method.)

## Miscellaneous Routines

**10.6** These routines set and return various fields of the control record.

**tb:!SetCRefCon** *theControl data* [I-327] Function

Sets the refCon field value of *theControl* to *data*.

**tb:!GetCRefCon** *theControl* [I-327] Function

Returns the value of the refCon field of *theControl*.

**tb:!SetCtlAction** *theControl actionProc* [I-328] Function

Sets the field that contains a pointer to an action procedure of *theControl* to *actionProc*.

**tb:!GetCtlAction** *theControl* [I-328] Function

Returns a pointer to the action procedure of *theControl.*

**tb:!GetCVariant** *theControl* [V-222] Function

Returns the variant code of the color control *theControl.*

## Control Manager Color Traps

10.7 When a new control is created inside a color window, a new color control is created and a color table is associated with it. The color table is created from the color table associated with the color window and can be modified using the trap **tb:!SetCtlColor**.

If the control was created using the trap **tb:!GetNewControl**, and there is a "cctb" (control color table) resource with the same resource ID as the "CNTL" resource used to created the control, then the control color table specified by the "cctb" resource is used to create the color table for the new control.

**tb:!SetCtlColor** *theControl newColorTable* [V-222] Function

Sets *theControl*'s color table to be *newColorTable.*

*Example:*

```
(setf cTab (tb:!GetCTable 127))
(tb:!SetCtlColor myControl cTab)
```

**tb:!GetAuxCtl** *theControl acHandl* [V-222] Function

Sets *acHandl* to be the auxiliary control record for the color control *theControl.* If *theControl* used the default colors, **tb:!GetAuxCtl** returns false. If *theControl* has its own color table or if *theControl* is **tb:!nilPtr**, **tb:!GetAuxCtl** returns true.

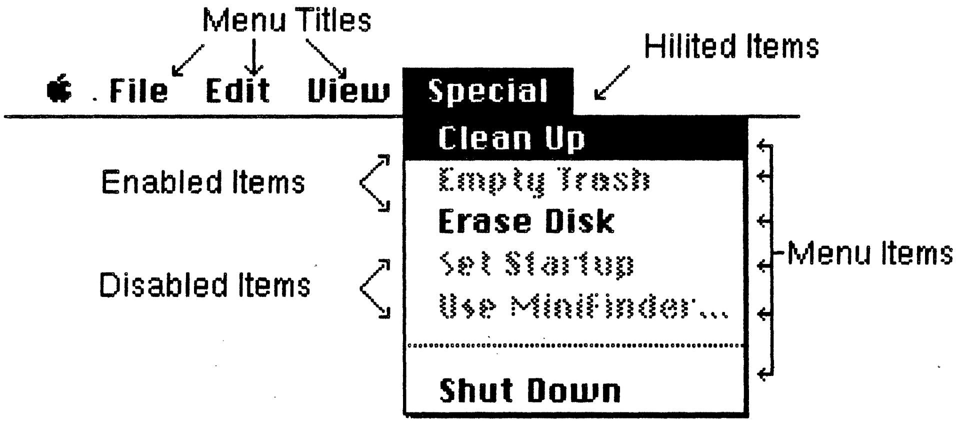
# Chapter 11
# MENU MANAGER

## Introduction

11.1 The Menu Manager is used to:

- Create menus
- Build menu bars
- Modify the properties of menus
- Modify the properties of menu items
- Dispose of the menus
- Allow the user to choose from a menu

Menu bars are formed from a list of menus. Menus consist of a title and a list of menu items.



## Initialization and Allocation

11.2 These routines create and dispose of menus.

**tb:!InitMenus** [I-351,V-243] Function

Initializes the Menu Manager. You will never need to call this trap as it is called for you when you launch a TbServer.

**tb:MenuInfo** [I-345] Flavor

This flavor defines a menu.

**:menuID** *16b-integer* Init Option of **tb:MenuInfo**
**:menuID** Method of **tb:MenuInfo**
**:set-menuID** *16b-integer* Method of **tb:MenuInfo**

This is the menu ID. It must be unique within an application. The ID may be the same as its own "MENU" resource ID, if any, but it must not be the same as any other resource ID being used. The default is 50.

| | |
|---|---|
| **:menuTitle** *string* | Init Option of **tb:MenuInfo** |

This is the title string of the menu. The default is "Menu".

| | |
|---|---|
| **:menuWidth** | Method of **tb:MenuInfo** |
| **:set-menuWidth** *pixels* | Method of **tb:MenuInfo** |
| **:menuHeight** | Method of **tb:MenuInfo** |
| **:set-menuHeight** *pixels* | Method of **tb:MenuInfo** |

These are the menu's width and height measured in pixels.

| | |
|---|---|
| **:menuProc** | Method of **tb:MenuInfo** |
| **:set-menuProc** *handle* | Method of **tb:MenuInfo** |

This is the handle to the menu's definition procedure.

| | |
|---|---|
| **:menuEnableFlags** | Method of **tb:MenuInfo** |
| **:set-menuEnableFlags** *32b-integer* | Method of **tb:MenuInfo** |

This is a 32-bit integer composed of 32 boolean flags. Bit 0 is set if the menu itself is enabled. Bits 1 though 31 are set if the corresponding menu item is enabled.

| | |
|---|---|
| **:menuData** | Method of **tb:MenuInfo** |
| **:set-menuData** *string* | Method of **tb:MenuInfo** |

This is a string of up to 255 characters containing the menu title and other data.

The following example demonstrates the creation of a new MenuInfo instance:

*Example:*

```
(setf myMenu (make-instance 'tb:menuinfo :menuID 128
                             :menuTitle "Sample Menu"))
```

| | |
|---|---|
| **tb:!NewMenu** *menuID menuTitle* | [I-351] Function |

Allocates memory for a new menu and returns a handle to it. The new menu has the title specified in the string *menuTitle* and the menu ID specified in the integer *menuID*. The preferred method for creating new menus is to make an instance of **tb:MenuInfo.**

To create a menu with a menu ID of 128 and a title "Sample Menu," do the following:

*Example:*

```
(setf myMenu (tb:!NewMenu 128 "Sample Menu"))
```

| | |
|---|---|
| **tb:!GetMenu** *resourceID* | [I-351,V-243] Function |

Uses the information in a "MENU" resource, with a resource ID specified by the integer *resourceID*, to create a new menu and returns a handle to the menu.

**:dispose** Method of **tb:MenuInfo**
**tb:!DisposeMenu** *menu* [I-352] Function

Dispose of menus created by **tb:!NewMenu**. For menus created by **tb:!GetMenu**, use **tb:!ReleaseResource**.

NOTE: Remove the menu from the menu list using the trap **tb:!DeleteMenu** before disposing of it.

## Forming the Menus

11.3 These procedures form new menus.

**:appendItem** *data* Method of **tb:MenuInfo**
**tb:!AppendMenu** *menu data* [I-352,V-243] Function

Append the string *data* to the menu indicated in *menu*. Call these traps repeatedly to add to menus. To add three menu items to a previously created menu named *myMenu*, do the following:

*Example:*

```
(tb:!AppendMenu myMenu "FirstItem;SecondItem;ThirdItem")
```

The trap also recognizes meta characters which control the appearance of the menu items. The presently defined meta characters are:

*x*;*y* separates menu items *x* and *y* in the data string. For example, the above example would have created a three line menu.

^*n* prefixes an icon number *n*, indicating that the icon should appear in the menu with the item.

- creates a dividing line between items.

!*c* indicates that the menu item is to be marked with the character *c* that follows.

<*c* indicates that the character *c* that follows specifies the character style of the menu item. The allowed character styles are:

| | |
|---|---|
| B | Bold |
| I | Italic |
| U | Underline |
| O | Outline |
| S | Shadow |

/*c* associates a keyboard equivalent with the character *c* that follows.

(*x* disables the following menu item *x* in the data string.

To add an item which is disabled, has icon 128, is in italics, and has command key equivalent M, do the following:

*Example:*
```
(tb:!AppendMenu myMenu "(Messy Item^128<I/M")
```

**tb:!AddResMenu** *menu resType* [I-353,V-243] Function

Adds to a menu, using the resource names of all the resources of *resType* in all the open resource files. To add to the standard Apple menu, which consists of all the available desk accessories, do the following:

*Example:*
```
(setf appleMenu (tb:!NewMenu 128 "Apple"))
(tb:!AddResMenu appleMenu "DRVR")
```

**tb:!InsertResMenu** *menu resType afterItem* [I-353,V-243] Function

This trap is the same as **tb:!AddResMenu** except it adds the resource names starting after the menu item with the index *afterItem* (an integer) in *menu*. If *afterItem* is 0, it adds before the first menu item. If *afterItem* is larger than the number of items in the menu, the new item is added after the last menu item.

## Forming the Menu Bar

**11.4** These procedures create, modify, and delete menus and menu bars.

**tb:!InsertMenu** *menu beforeID* [I-353,V-244] Function

Inserts *menu* into the menu list before the menu whose menu ID is *beforeID*. If *beforeID* is 0, *menu* is inserted at the end. To insert the menu created in the example for **tb:!AppendMenu** do:

*Example:*
```
(tb:!InsertMenu myMenu 0)
```

**tb:!DrawMenuBar** [I-354,V-244] Function

Redraws the menu bar and includes any changes that have been made since the last **tb:!DrawMenuBar**.

**tb:!DeleteMenu** *menuID* [I-354] Function

Removes the menu whose menu ID is *menuID* from the menu bar.

**tb:!ClearMenuBar** [I-354,V-247] Function

Removes all the menus from the menu list.

**tb:!GetNewMBar** *menuBarID* [I-354,V-247] Function

Creates a menu bar from a previously defined "MBAR" resource, which has a resource ID of *menuBarID*, and returns a handle to the new menu bar.

**tb:!GetMenuBar** [I-355] Function

Creates a copy of the current menu bar and returns a handle to it.

**tb:!SetMenuBar** *menuList* [I-355] Function

Sets the current menu list to the given menu list.

## Choosing From a Menu

11.5 These procedures control the functions related to the selection of menu items: exposing menus, highlighting menu items, etc.

**tb:!MenuSelect** *startPoint* [I-355,V-244] Function

This trap is called when there is a mouse-down event in the menu bar. The value of *startPoint* is extracted from the *where* field of the event record returned by the Event Manager trap **tb:!GetNextEvent**. The trap draws the menu and highlights the selected menu item. The menu ID and the menu item number are returned when the mouse button is released. **tb:!MenuSelect** is an unusual trap because it does a multiple value return. To call this trap, do:

*Example:*

```
(multiple-value-bind (menuID menuItemNumber)
    (tb:!MenuSelect *event*)
  (when (/= 0 menuID)
    (MenuItemHandler menuID menuItemNumber)))
```

The variable *menuID* is set to the selected menu's menu ID, and *menuItemNumber* to the selected menu item's item number. Notice we used ***event*** instead of a point, since **tb:Point** is a mixin of **tb:EventRecord**.

**tb:!MenuKey** *character* [I-356,V-244] Function

The Menu Manager allows you to associate a key on the keyboard with an item in the menu bar. Instead of having to select a menu item with the mouse, you can select a menu item by pressing the Command Key and the key associated with the menu item.

To handle keyboard equivalents (a command character key combination), call the trap **tb:!MenuKey** whenever you receive a KeyDown event and the **tb:!cmdKey** flag is set in the modifier field of the event record.

If there is a menu item associated with the key, the trap returns two values: the menu's menu ID and menu item number. If there is no menu item associated with the key, the trap returns a menu ID of zero.

*Example:*

```
(when (tb:!cmdKey-p (send *event* :modifiers))
  (multiple-value-bind (menuID menuItemNumber)
      (tb:!MenuKey (send *event* :messageChar)))
    (when (/= 0 menuID)
      (MenuItemHandler menuID menuItemNumber))))
```

**tb:!HiliteMenu** *menuID* [I-357,V-244] Function

Highlights the title of the menu specified in *menuID*. Call **tb:!HiliteMenu 0** after **tb:!MenuSelect** or **tb:!MenuKey** to dehighlight the selected menu.

**tb:!MenuChoice** [V-240] Function

Called if the **tb:!MenuSelect** trap returns 0. It determines if the mouse button was released while inside a disabled item. If so, it returns two values: the menuID and menuItem number of the disabled item.

**tb:!PopUpMenuSelect** *menu top left popUpItem* [V-241] Function

Draws the pop-up menu whose handle is *menu*, at the vertical position *top* and horizontal position *left* (in global coordinates), highlighting the menu item *popUpItem*. Returns the menu item and menu ID of the menu item selected.

## Controlling the Appearance of an Item

11.6 These routines create, modify, and delete individual items appearing on a given menu.

**tb:!SetItem** *menu item itemString* [I-357] Function

Changes the text of the menu item in *item* to *itemString*.

*Example:*

```
;;; Change the second item.
(tb:!SetItem myMenu 2 "New Item")
```

**tb:GetItem** *menu integer* [I-358] Function
**tb:!GetItem** *menu integer* VAR *itemString* [I-358] Function

**tb:GetItem** returns the text of the menu item in *item*. **tb:!GetItem** is similar except it modifies *itemString* to be the menu item text.

*Example:*

```
(tb:GetItem myMenu 2) => "New Item"
(tb:!GetItem myMenu 2 (VAR itemString))
itemString => "New Item"
```

**tb:!InsMenuItem** *menu itemString item* [IV-55] Function

Inserts the item *itemString* after the item number *item* in *menu*.

**:deleteItem** Method of **tb:MenuInfo**
**tb:!DelMenuItem** *menu item* [IV-56] Function

Delete the item numbered *item* from *menu*.

**tb:!DisableItem** *menu item* [I-358,V-246] Function

Disables (makes unselectable) the menu item number *item* in *menu*.

**tb:!EnableItem** *menu item* [I-358,V-246] Function

Enables (makes selectable) the menu item number *item* in *menu*.

**tb:!CheckItem** *menu item checked* [I-358] Function

Puts a checkmark on menu item *item* in *menu* if *checked* is true. It removes the checkmark if *checked* is false.

**tb:!commandMark** [I-220] Constant
**tb:!checkMark** [I-220] Constant
**tb:!diamondMark** [I-220] Constant
**tb:!appleMark** [I-220] Constant

These four characters are used as item marks in menus and elsewhere. Notice that these constants represent character objects on the microExplorer rather than character codes as in C.

**tb:!SetItemMark** *menu item character* [I-359,V-246] Function

Places *character* before the menu item *item* in *menu*.

To set the mark of the menu item numbered *myItem* in the menu *myMenu* to diamondMark, do the following:

*Example:*

```
(tb:!SetItemMark myMenu 3 tb:!diamondMark)
```

**tb:GetItemMark** *menu item* [I-359,V-246] Function
**tb:!GetItemMark** *menu item* VAR *character* [I-359,V-246] Function

**tb:GetItemMark** returns the marking character of the menu item *item* of *menu*. **tb:!GetItemMark** is similar except that it updates *character* with the marking character.

*Example:*

```
(tb:GetItemMark myMenu 3) => #\x
(tb:!GetItemMark myMenu 3 (VAR itemMark))
itemMark => #\x
```

**tb:!SetItemIcon** *menu item iconID* [I-359,V-246] Function

Searches the open resource files for the icon numbered *iconID* and sets the item icon of the menu item *item* in *menu* to the new icon.

**tb:GetItemIcon** *menu item* [I-360,V-246] Function
**tb:!GetItemIcon** *menu item* VAR *iconID* [I-360,V-246] Function

**tb:GetItemIcon** returns the icon number of the item icon of the menu item *item* in the menu *menu*. **tb:!GetItemIcon** is similar except that it modifies *iconID* to be the icon number.

*Example:*

```
(GetItemIcon myMenu 1) => 10
(tb:!GetItemIcon myMenu 1 (VAR icon))
icon => 10
```

**tb:!SetItemStyle** *menu item style* [I-360] Function

Changes the character style of the menu item *item* to *style*. The currently defined styles are: **tb:!Bold**, **tb:!Italic**, **tb:!Underline**, **tb:!Outline**, **tb:!Shadow**, **tb:!Condense**, and **tb:!Extend**. These styles can be summed to specify, say, bold italic.

To set an item to underline and italics, do the following:

*Example:*

```
(setf chStyle (+ tb:!Underline tb:!Italic))
(tb:!SetItemStyle myMenu 1 chStyle)
```

**tb:GetItemStyle** *menu item* [I-360,V-247] Function
**tb:!GetItemStyle** *menu item* VAR *style* [I-360,V-247] Function

**tb:GetItemStyle** returns the character style of menu item *item*. **tb:!GetItemStyle** is similar except that it modifies *style* with the character style.

*Example:*

```
(GetItemStyle myMenu 1) => 3
(tb:!GetItemStyle myMenu 1 (VAR chStyle))
chStyle => 3
```

**tb:GetItemCmd** *menu item* [V-240] Function
**tb:!GetItemCmd** *menu item* VAR *cmdChar* [V-240] Function

**tb:GetItemCmd** returns the Command Character (the Menu KeyBoard equivalent) of the menu item number *item* in the menu whose handle is *menu*. **tb:!GetItemCmd** is similar except that it modifies *cmdChar* to be the Command Character.

**tb:!SetItemCmd** *menu item cmdChar* [V-240] Function

Sets the Command Character (the Menu KeyBoard equivalent) of the menu item number *item* in the menu whose handle is *menu* to the character specified in *cmdChar*.

## Miscellaneous Routines

11.7 These procedures perform miscellaneous functions relating to menus.

**tb:!CalcMenuSize** *menu* [I-361] Function

Recalculates the dimensions of *menu*. This is an internally used trap.

**tb:!CountMItems** *menu* [I-361] Function

Returns the number of items in *menu*.

**tb:!GetMHandle** *menuID* [I-361] Function

Returns the handle of the menu specified by *menuID*.

**tb:!FlashMenuBar** *menuID* [I-361] Function

Inverts the title of the menu *menuID*. To flash the menu bar, do the following:

*Example:*

```
;;; Flash the menu bar.
(tb:!FlashMenuBar 0)      ; Invert to black.
(tb:!FlashMenuBar 0)      ; Return to normal.
```

**tb:!SetMenuFlash** *count* [I-361] Function

Sets the number of times a menu item blinks when selected. This is normally set from the Control Panel desk accessory.

## Menu Manager Color Traps

**11.8** These routines control the color characteristics of menus and menu bars.

**tb:!InitProcMenu** *resourceID* [V-238] Function

This trap should only be called if the application has a custom menu bar proc.

**tb:!DelMCEntries** *menuID menuItem* [V-238] Function

Deletes entries from the menu color information table for the given *menuID* and *menuItem*.

**tb:!GetMCInfo** [V-239] Function

Returns a copy of the current menu color information table.

NOTE: This is not the same type of structure as a color table. (See *Inside Macintosh* Volume V, pages 231-234.)

**tb:!SetMCInfo** *menuCTable* [V-239] Function

Sets the current menu's color information table to *menuCTable*.

**tb:!DispMCInfo** *menuCTbl* [V-239] Function

Disposes of the menu color information table *menuCTbl.*

**tb:!GetMCEntry** *menuID menuItem* [V-239] Function

Returns a pointer to the color information table entry for the menu item *menuItem* in the menu *menuID*.

**tb:!SetMCEntries** *numEntries menuCEntries* [V-239] Function

Takes the pointer *menuCEntries* to an array of *numEntries* number of color information records and adds the information to the current color information table.

# Chapter 12
# TEXTEDIT

## Introduction

12.1 TextEdit is a set of text editing routines. These routines allow you to write a simple text editor which supports cutting, copying, and pasting. The TextEdit data structure is called a TextEdit Record (a **tb:TERec**. This record contains all the information necessary to draw the text: the font, the font size, where to draw it, and the text characters.

## Initialization and Allocation

12.2 These routines initialize TextEdit, allocate handles for text, and dispose of unneeded memory.

**tb:!TEInit** [I-383] Function

Initializes TextEdit. You will never need to call this routine as it is called for you automatically when you launch a TbServer.

**tb:TERec** [I-377] Flavor

This flavor defines a TextEdit record data structure.

| | |
|---|---|
| **:destRect** *rect* | Init Option of **tb:TERec** |
| **:destRectTop** | Method of **tb:TERec** |
| **:destRectLeft** | Method of **tb:TERec** |
| **:destRectBottom** | Method of **tb:TERec** |
| **:destRectRight** | Method of **tb:TERec** |

This is the destination rectangle, the rectangle in which the text is drawn to fit. The coordinates are in the local coordinate system of the current grafPort. The default is (50 50 100 100).

| | |
|---|---|
| **:viewRect** *rect* | Init Option of **tb:TERec** |
| **:viewRectTop** | Method of **tb:TERec** |
| **:viewRectLeft** | Method of **tb:TERec** |
| **:viewRectBottom** | Method of **tb:TERec** |
| **:viewRectRight** | Method of **tb:TERec** |

This is the view rectangle, the area of the drawn text which is actually shown. The coordinates are in the local coordinate system of the current grafPort. The default is (50 50 100 100).

**tb:TERec** instances have the following additional instance accessor methods:

| | | |
|---|---|---|
| • :LINEHEIGHT | ;24 | [ integer ] |
| • :FONTASCENT | ;26 | [ integer ] |
| • :SELSTART | ;32 | [ integer ] |
| • :SELEND | ;34 | [ integer ] |
| • :WORDBREAK | ;38 | [ procptr ] |

| | | |
|---|---|---|
| :CLIKLOOP | ;42 | [ procptr ] |
| :JUST | ;58 | [ integer ] |
| :TELENGTH | ;60 | [ integer ] |
| :HTEXT | ;62 | [ handle ] |
| :CRONLY | ;72 | [ integer ] |
| :TXFONT | ;74 | [ integer ] |
| :TXFACE | ;76 | [ style ] |
| :TXMODE | ;78 | [ integer ] |
| :TXSIZE | ;80 | [ integer ] |
| :INPORT | ;82 | [ grafptr ] |
| :NLINES | ;94 | [ integer ] |

**tb:!TENew** *destRect viewRect* [I-383] Function

Returns a new TextEdit record which supports only a single font, size, style, and color and which has a destination rectangle *destRect* and a view rectangle *viewRect*. The *destRect* and *viewRect* are specified in the local coordinates of the current port.

The TextEdit record is associated with the current grafPort. Remember to set the current port to the port in which you want the text to appear.

**:dispose** Method of **tb:TERec**
**tb:!TEDispose** *hTE* [I-383] Function

Dispose of the TextEdit record *hTE*.

## Accessing the Text of an Edit Record

12.3 The following routines get and set the specified text.

**tb:!TESetText** *text length hTE* [I-383] Function

Sets the text of the TextEdit record *hTE* to the first *length* characters in the text buffer *text*.

**tb:!TEGetText** *hTE* [I-384] Function

Returns a handle to the text in the TextEdit record *hTE*.

## Insertion Point and Selection Range

12.4 These routines control the placement and highlighting of text selections.

**tb:!TEIdle** *hTE* [I-384] Function

Causes a blinking caret to appear at the TextEdit record insertion point. This trap should be called from the main event loop. You should call this trap only when there is a TextEdit record associated with the active window.

**tb:!TEClick** *pt extend hTE* [I-384] Function

Called when a mouse-down event is recorded in the content region of an active window containing a TextEdit record. Set *extend* to T if the Shift key is being held down.

The point *pt* should be in local coordinates, so call the QuickDraw trap **tb:!GlobalToLocal** for the point, which is gotten from either **tb:!GetMouse** or from the event record, before passing it to the trap.

**tb:!TESetSelect** *selStart selEnd hTE* [I-385] Function

Sets the selection range of the TextEdit record *hTE* to start at *selStart* and end at *selEnd*. To make an insertion point, make *selStart* equal to *selEnd*.

**tb:!TEActivate** *hTE* [I-385] Function

Called when you receive an Activate event for a window that has an associated TextEdit record.

**tb:!TEDeactivate** *hTE* [I-385] Function

Called when you receive a Deactivate event for a window that has an associated TextEdit record.

## Editing

12.5 These routines are used to cut, copy, paste, insert, and delete text.

**tb:!TEKey** *key hTE* [I-385] Function

Inserts the character key at the insertion point of the TextEdit record *hTE*. If *hTE*'s *selStart* is not equal to *selEnd* (that is, a range of text is highlighted), the text between *selStart* and *selEnd* is first deleted. This trap is called when you receive a key-down event, and the current active window has a TextEdit record associated with it.

*Example:*

```
(tb:!TEKey (send *event* :MessageChar) myTEHandle)
```

**tb:!TECut** *hTE* [I-385] Function

Cuts the text from the TextEdit record *hTE*, starting at *selStart* and ending at *selEnd*, and puts it in the TextEdit scrap.

**tb:!TECopy** *hTE* [I-386] Function

Copies the text from the TextEdit record *hTE*, starting at *selStart* and ending at *selEnd*, and puts it in the TextEdit scrap.

**tb:!TEPaste** *hTE* [I-386] Function

Pastes the contents of the TextEdit scrap into the TextEdit record *hTE* at its current insertion point. If *hTE*'s *selStart* is not equal to *selEnd* (that is, a range of text is highlighted), the text between *selStart* and *selEnd* is first deleted.

**tb:!TEDelete** *hTE* [I-387] Function

Deletes the text from *selStart* to *selEnd* in the TextEdit record *hTE*.

**tb:!TEInsert** *text length hTE* [I-387] Function

Inserts *length* number of characters from the buffer pointed to by *text* into the TextEdit record *hTE*.

*Example:*

```
;;; Output "hello world." to a TERec
(setf hndl                                ; get a handle to string
      (tb:!NewHandle "hello world."))
(tb:!hLock hndl)                        ; lock the handle, then...
(setf text-ptr (tb:deref hndl))         ; dereference it into a ptr
(tb:!TEInsert text-ptr 12 myTEHandle)       ; output the string
(tb:!DisposHandle hndl)                   ; dispose of our handle
```

If you had started with a handle to a string rather than the string itself, then lock and dereference that handle into a pointer as shown above. When you are finished with this temporary pointer, then unlock it with **tb:!hUnlock** rather than disposing of it.

## Text Display and Scrolling

12.6 These routines and constants control the display of text.

**tb:!teJustLeft** [I-376] Constant
**tb:!teJustCenter** [I-376] Constant
**tb:!teJustRight** [I-376] Constant
**tb:!teForceLeft** [I-376] Constant

These constants are used as the *just* argument values in TextEdit functions. They specify the justification of text.

**tb:!TESetJust** *hTE just* [I-387] Function

Sets the justification of the text in the TextEdit record *hTE*. The value of the *just* argument should be one of the following: **tb:!teJustLeft**, **tb:!teJustCenter**, **tb:!teJustRight**, or **tb:!teForceLeft**.

**tb:!TEUpdate** *rUpdate hTE* [I-387] Function

Called when an update event is received in the main event loop and there is a TextEdit record associated with the current active window.

**tb:!TextBox** *text length box just* [I-388] Function

Draws *length* number of characters from the text buffer *text* inside the rectangle *box* with justification *just*. The value of the *just* argument should be one of the following: **tb:!teJustLeft**, **tb:!teJustCenter**, **tb:!teJustRight**, or **b:!teForceLeft**.

**tb:!TEScroll** *hTE dh dv* [I-388] Function

Scrolls the text within *hTE*'s view rectangle a distance of *dh* pixels horizontally and *dv* pixels vertically.

**tb:!TEPinScroll** *dh dv h* [IV-57] Function

The same as **tb:!TEScroll** except it stops scrolling once the last line has scrolled into the view rectangle.

**tb:!TEAutoView** *auto hTE* [IV-57] Function

If *auto* is true, automatic scrolling is enabled. If *auto* is false, automatic scrolling is disabled.

**tb:!TESelView** *hTE* [IV-57] Function

If the selection range of the TextEdit record *hTE* is not in the TextEdit record's view rectangle, this trap scrolls the text.

## Scrap Handling

12.7 These routines control your application's scrap handling.

**tb:!TEFromScrap** [I-389] Function

Copies the desk scrap to the TextEdit scrap.

**tb:!TEToScrap** [I-389] Function

Copies the TextEdit scrap to the desk scrap. You must call the Scrap Manager trap **tb:!ZeroScrap** to clear the desk scrap first or this trap will not work properly.

**tb:!TEScrapHandle** [I-389] Function

Returns a handle to the TextEdit scrap.

**tb:!TEGetScrapLen** [I-389] Function

Returns the length of the TextEdit scrap.

**tb:!TESetScrapLen** *length* [I-390] Function

Sets the length of the TextEdit scrap to *length*.

## Advanced Routines

12.8 This routine is used in advanced applications only.

**tb:!SetWordBreak** *wBrkProc hTE* [I-390] Function

Installs in the **:wordBreak** instance variable of *hTE* a special routine which will call the word break routine pointed at by *wBrkProc*.

**tb:!SetClikLoop** *clikProc hTE* [I-390] Function

Installs in the **:clikLoop** instance variable of *hTE* a special routine which will call the click loop routine pointed at by *clikProc*.

**tb:!TECalText** *hTE* [I-390] Function

Recalculates the *linestarts* array of the TextEdit record *hTE*. This trap should be called after doing anything that affects the number of characters that can be displayed in a line, like resizing the *destRect* of *hTE*.

# Chapter 13
# DIALOG MANAGER

## Introduction

13.1 The Dialog Manager creates and manipulates a special type of window used to get information to or from the user. If the window requires the user to input information, it is know as a dialog box. If it provides the user with information, it is known as an alert. One example of a dialog box is the window that is brought up when you select the "Save As..." menu item in a standard "File" menu. This dialog box asks for the name of the new file.

Alerts are used to tell the user about errors or to provide some information that the user can act upon: whether or not you want to continue an operation, for example.

The specifications (templates) for dialog boxes and alerts are not easily built using the Macintosh Toolbox. They are usually created with a resource editor.

## Initialization

13.2 These procedures initialize the Dialog Manager, set the sound associated with alerts, and set the font that will be used on text appearing within a dialog box.

**tb:!InitDialogs** *restartProc* [I-411] Function

Initializes the Dialog Manager. You will never need to call this routine because it is called for you when you launch a TbServer.

**tb:!ErrorSound** *soundProc* [I-411] Function

Sets the sound made by alerts to that defined in *soundProc*. Passing **tb:!nilPtr** turns off the sound.

**tb:!SetDAFont** *fontNum* [I-412] Function

Sets the font appearing in the dialog box or alert to *fontNum*. This trap effects only the text (static or editable) displayed in the dialog. It does not effect the item titles.

## Creating and Disposing of Dialogs

13.3 These routines create and dispose of dialog boxes.

**tb:DialogRecord** [I-408] Flavor

This flavor creates a dialog record. An instance of this flavor may be used anywhere a dialog pointer is needed. This flavor mixes in the **tb:Window** flavor.

**:dStorage** *pointer* Init Option of **tb:DialogRecord**

This option controls memory allocation for the dialog box. If this value is **tb:!nilPtr**, the default, then a new dialog box is allocated on the heap. Otherwise, this option must be a pointer to at least 176 bytes of storage.

**:boundsRect** *rect* Init Option of **tb:DialogRecord**

This option is a rectangle which controls the size and location of the dialog box. The default is related to the screen size.

**:title** *string* Init Option of **tb:DialogRecord**

This string becomes the title of modeless dialog boxes. Specify an empty string as a title for modal boxes. The default is "New Dialog".

**:visible** *visible-p* Init Option of **tb:DialogRecord**

If this option is true, the dialog box will be visible when created. The default is true.

**:procID** *integer* Init Option of **tb:DialogRecord**

This option specifies the type of dialog box. Use one of the Window Manager procIDs such as **tb:!documentProc** (q.v.). The default is **tb:!documentProc**. See Standard Types of Windows figure below for an illustration of the available procIDs.

**:behind** *windowPtr* Init Option of **tb:DialogRecord**

This is a pointer to a window which this dialog box will appear behind. If this option is **tb:!onePtr**, which is the default, then the dialog box is the frontmost window.

**:goAwayFlag** *goAway-p* Init Option of **tb:DialogRecord**

If this option is true, then the modeless dialog boxes only will have a close box in the window frame. The default is false.

**:refCon** *32b-integer* Init Option of **tb:DialogRecord**

This option is a 32-bit integer which is reserved for the application and defaults to 0. A hook such as this is needed in C or Pascal environments, but on the microExplorer a better way to attach application-specific information to a dialog box is to mix **tb:DialogRecord** into your own flavor. You flavor then defines the extra instance variables you need.

**:items** *handle* Init Option of **tb:DialogRecord**
**:items** Method of **tb:DialogRecord**
**:set-items** *handle* Method of **tb:DialogRecord**

Handles to the dialog box's item list of controls. They default to an empty handle and usually load from a resource.

**:textH** Method of **tb:DialogRecord**
**:set-textH** *TEHandle* Method of **tb:DialogRecord**

Handles to the current editText item.

**:editField** Method of **tb:DialogRecord**
**:set-editField** *integer* Method of **tb:DialogRecord**

The editText item number -1. If there is no editText item in the dialog, this value is -1.

**:aDefItem** Method of **tb:DialogRecord**
**:set-aDefItem** *integer* Method of **tb:DialogRecord**

This is the default button item number for modal dialogs and alerts. The following is an example of how to create a DialogRecord object.

*Example:*

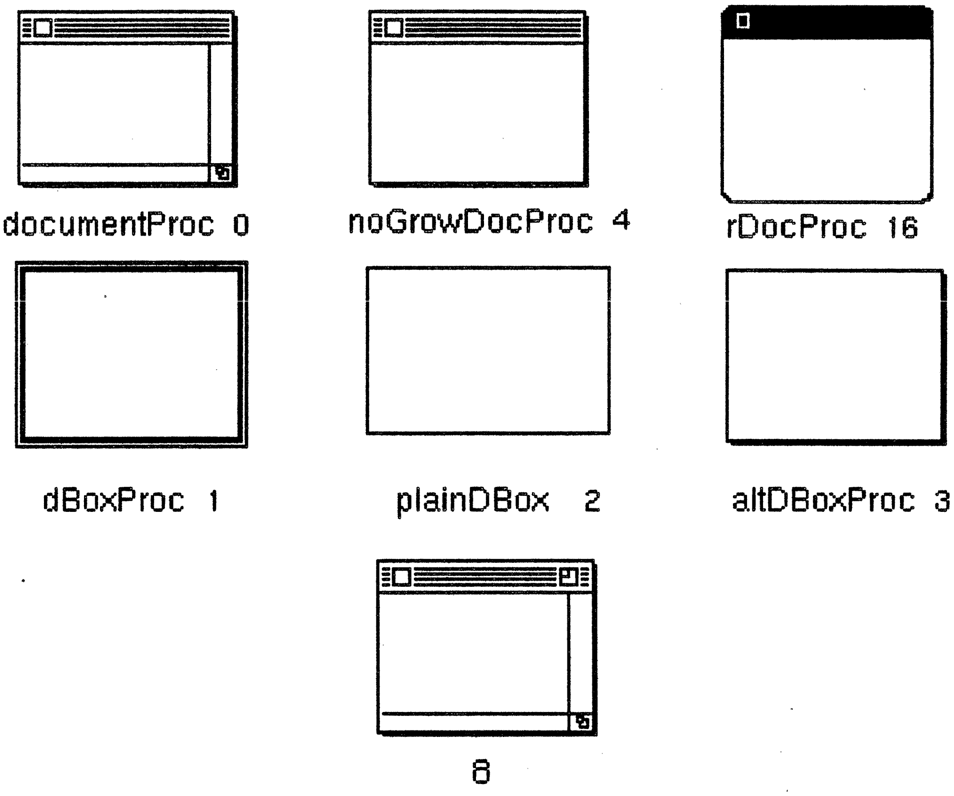
```
(setf boundsRect
      (make-instance 'tb:Rect :left 112 :top 55
                              :right 239 :bottom 108))
(setf myItems  (tb:!GetResource "DITL" 128))
(setf myDialog (make-instance 'tb:DialogRecord
                  :boundsrect boundsRect :items myItems))
```

**tb:!NewDialog** *dStorage boundsRect title visible procID behind goAwayFlag refCon items* [I-412] Function

Creates a new dialog box returning a dialog pointer. If you want to allocate the memory for the dialog box (which must be at least 176 bytes long), pass a pointer to this memory as *dStorage*. Most of the time you won't, so just pass **tb:!nilPtr**. The *boundsRect* is the rectangle that defines the boundary of the new dialog window. The *procID* specifies the type of dialog box required.

The pointer *behind* is used if you want the newly created dialog box to be created behind an already existing window. Normally, you pass **tb:!onePtr** and the dialog box is created in front of all the existing windows. *Items* is a handle to the dialog items (also known as controls or the item list) associated with the new dialog box. Items are usually created with a resource editor and read in with the Resource Manager.

The dialog window types available are:



**Standard Types of Windows**

**tb:!GetNewDialog** *dialogID dStorage behind* [I-413,V-284] Function

Creates a new dialog box using information in a previously defined "DLOG" resource which has a resource ID *dialogID*. If you want to allocate the memory for the dialog box (which must be at least 176 bytes long), pass a pointer to this memory as *dStorage*. Normally, you will not, so just pass **tb:!nilPtr**. The *behind* argument is only used if you want to display the new dialog box behind an existing window. The usual value is **tb:!onePtr**.

**tb:!CloseDialog** *dialog* [I-413] Function

Disposes of the dialog box *dialog*, but does not dispose of the dialog record or the item list. Use this trap if you passed a *dStorage* pointer when you created the dialog box.

**:dispose** Method of **tb:DialogRecord**
**tb:!DisposDialog** *dialog* [I-415] Function

Dispose of the dialog box *dialog* by calling **tb:!CloseDialog** and then release the memory occupied by the dialog's item list and dialog record. Use this trap if you did not pass a *dStorage* pointer when you created the dialog box.

**tb:!CouldDialog** *dialogID* [I-415,V-284] Function
**tb:!FreeDialog** *dialogID* [I-415,V-285] Function

**tb:!CouldDialog** ensures that the "DLOG" resource which has a resource ID *dialogID* is in memory and makes it unable to be purged.

**tb:!FreeDialog** undoes the effect of **tb:!CouldDialog**, allowing the "DLOG" resource with the resource ID *dialogID* to be purged.

## Handling Dialog Events

13.4 These routines control the handling of events which occur within a dialog window.

**tb:ModalDialog** *filterProc* [I-415] Function
**tb:!ModalDialog** *filterProc* VAR *itemHit* [I-415] Function

**tb:ModalDialog** repeatedly gets and handles events in a modal dialog window. When the trap detects a valid event inside a dialog item, it returns the number of the item that was hit. Always pass **tb:!nilPtr** in *filterProc.*

**tb:!ModalDialog** is similar except that it modifies *itemHit* with the number of the selected dialog item.

**CAUTION: This trap assumes that the frontmost window is a dialog window. It does not work if it is evaluated from the Lisp Listener (it crashes!).**

To handle a modal dialog box safely, do the following:

*Example:*

```
tb:
(defun myModalDialog ()
  (let* ((myDialogBox (!GetNewDialog dialogResID !nilPtr
                                               !onePtr))
         (itemHit    (ModalDialog !nilPtr)))
    (!DisposDialog myDialogBox)
    itemHit))
```

**tb:!IsDialogEvent** *theEvent* [I-416] Function

If your application includes any modeless dialog boxes, call this trap from the main event loop after calling the function **tb:!GetNextEvent** or **tb:!WaitNextEvent.** This trap returns true if the event specified in *theEvent* needs to be handled as part of a dialog. If the trap returns true, you should pass the event to the trap **tb:!DialogSelect** for it to handle. See *Inside Macintosh* for more details.

**tb:DialogSelect** *theEvent* [I-417] Function
**tb:!DialogSelect** *theEvent* VAR *dialog* VAR *itemHit* [I-417] Function

**tb:DialogSelect** returns three values. The first value is true if *theEvent* is associated with an enabled dialog box. If the first value is true, the second value is the dialog pointer of the associated dialog box. The last value returned is the number of the selected dialog item in the dialog box. If the first value is false, the second and third values have no meaning.

**tb:!DialogSelect** also returns true if *theEvent* is associated with an enabled dialog box. However, it modifies *dialog* and *itemHit* to be the dialog box and item selected.

This trap does not handle keyboard equivalents for menu items. If you wish to support keyboard equivalents, check for a key-down event. If the event was a key-down event, call the Menu Manager trap **tb:!MenuKey** before proceeding.

*Example:*

```
;;; ... in the main event loop ...
(when (tb:!WaitNextEvent tb:!everyEvent *event* 0
                         tb:!nilRgn))
  ;; Check for menu keyboard equivalent
  (setf MenuKey-p
        (and (= tb:!keyDown (send *event* :What))
             (tb:!cmdKey-p (send *event* :modifiers))))
  (if (and (not MenuKey-p) (tb:!IsDialogEvent *event*))
    ;;then this is a dialog event without a command key
    (multiple-value-bind (result whichDialog itemHit)
        (tb:DialogSelect *event*)
      ...process the dialog selection...)
    ;;else process a normal event
    ...))
```

**tb:!DlogCut** *theDialog* [I-418] Function

Applies the TextEdit routine **tb:!TECut** to the currently selected edit text item in theDialog if it has one.

**tb:!DlogCopy** *theDialog* [I-418] Function

Applies the TextEdit routine **tb:!TECopy** to the currently selected edit text item in theDialog if it has one.

**tb:!DlogPaste** *theDialog* [I-418] Function

Applies the TextEdit routine **tb:!TEPaste** to the currently selected edit text item in theDialog if it has one.

**tb:!DlogDelete** *theDialog* [I-418] Function

Applies the TextEdit routine **tb:!TEDelete** to the currently selected edit text item in theDialog if it has one.

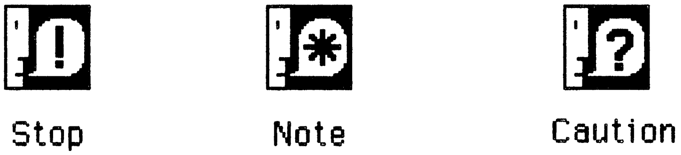
**tb:!DrawDialog** *dialog* [I-418] Function

Draws the dialog box *dialog*.

## Invoking Alerts

13.5 Alerts are used to report errors or give warnings to the user. They display a Standard Alert Icon and an OK button, in addition to any other items in the alert template.

Standard Alert Icons



**tb:!Alert** *alertID filterProc* [I-418,V-284] Function

Creates and displays an alert defined in the "ALRT" resource which has a resource ID *alertID*.

**tb:!stopIcon** [I-420] Constant
**tb:!noteIcon** [I-420] Constant
**tb:!cautionIcon** [I-420] Constant

"ALRT" resource resource IDs for the standard alert icons.

**tb:!StopAlert** *alertID filterProc* [I-419,V-284] Function

Acts in the same manner as **tb:!Alert** except that it draws a Stop icon in the top left hand corner before drawing the remainder of the alert window.

**tb:!NoteAlert** *alertID filterProc* [I-420,V-284] Function

Acts in the same manner as **tb:!Alert** except that it draws a Note icon in the top left hand corner before drawing the remainder of the alert window.

**tb:!CautionAlert** *alertID filterProc* [I-420,V-284] Function

Acts in the same manner as **tb:!Alert** except it draws a Caution icon in the top left hand corner before drawing the remainder of the alert window.

**tb:!CouldAlert** *alertID* [I-420,V-285] Function

Ensures that the "ALRT" resource with a resource ID *alertID* is in memory and makes it unable to be purged.

**tb:!FreeAlert** *alertID* [I-420,V-285] Function

Undoes the effect of **tb:!CouldAlert**, allowing the "ALRT" resource with the resource ID *alertID* to be purged.

## Manipulating Items in Dialogs and Alerts

13.6 These routines modify the dialog items within a dialog box or an alert window.

**tb:!ParamText** *param0 param1 param2 param3* [I-421] Function

Provides a means of changing the text in statText items by allowing you to substitute the strings *param0* to *param3* for the special strings "^0" to "^3".

**tb:GetDItem** *dialog itemNo* [I-421] Function
**tb:!GetDItem** *dialog itemNo* VAR *type item box* [I-421] Function

**tb:GetDItem** returns three values: the type number, item handle, and enclosing rectangle of the dialog item number *item* in the dialog box *dialog*.

**tb:!GetDItem** is similar except that it modifies *type*, *item*, and *box* to be the type number, item handle, and enclosing rectangle. Notice that *item* must be initialized to a handle and *box* must be initialized to a rectangle.

*Example:*

```
(multiple-value-bind (type myItem box)
     (GetDItem myDialog 1)
  ...code using type, myItem, and box...)

(setf type   0)
(setf myItem (make-instance 'tb:mac-handle))
(setf box    (make-instance 'tb:Rect))
(tb:!GetDItem myDialog 1 (VAR type) myItem box)
...code using type, myItem, and box...
```

**tb:!SetDItem** *dialog itemNo type item box* [I-421] Function

Sets the type, item, and box of the dialog item number *item* in the dialog box *dialog*.

**tb:!HideDItem** *dialog itemNo* [IV-59] Function
**tb:!ShowDItem** *dialog itemNo* [IV-59] Function

Hides or shows the item numbered *itemNo* in the dialog box *dialog*.

**tb:!UpdtDialog** *dialog updateRgn* IV-60] Function

Draws all the items of the dialog box *dialog* that are in the update region *updateRgn*.

**tb:!FindDItem** *dialog thePoint* [IV-60] Function

Returns the item number minus one of the dialog box *dialog* that the point *thePoint* is in. If the point is not inside any dialog item, the trap returns -1.

**tb:GetIText** *item* [I-422] Function
**tb:!GetIText** *item* VAR *text* [I-422] Function

**tb:GetIText** returns the text of the item if the dialog item *item* is a static or editable text item. **tb:!GetIText** is similar except it modifies *text* to be the item text.

To get the text from the editable text item with a handle *myItem*, do the following:

*Example:*

```
(tb:GetIText myItem) => "Sample Item"
(tb:!GetIText myItem (VAR text))
text => "Sample Item"
```

**tb:!SetIText** *item text* [I-422] Function

Sets the text of the item to *text,* a string, if the dialog item *item* is a static or editable text item.

**tb:!SelIText** *dialog itemNo strtSel endSel* [I-422] Function

Sets the selection range of the text starting at *strtSel* and ending at *endSel* if the item number *item* of the dialog box *dialog* is a text item. To select all the text of an editable item, pass 0 for *strtSel* and 32767 (#x7FFF) for *endSel.*

**tb:!GetAlrtStage** [I-422] Function

Returns the stage of the last alert.

**tb:!ResetAlrtStage** [I-423] Function

Sets the alert stage to zero.

## Dialog Manager Color Traps

13.7 The Macintosh II Dialog Manager has been expanded to support color dialog boxes. A color dialog box can be explicitly created using the trap **tb:!NewCDialog**. If you use the traps **tb:!GetNewDialog** or **tb:!Alert** to create the dialog or alert, you can specify the creation of a color dialog or alert by having a color table resource of the same resource ID as the resource specifying the dialog box or alert. For example, if a dialog box was created with the trap **tb:!GetNewDialog**, then the dialog color table resource "dctb" should have the same resource ID as the "DLOG" resource that specifies the dialog box.

If an alert was created using the traps **tb:!Alert**, **tb:!StopAlert**, **tb:!NoteAlert**, or **tb:!CautionAlert**, the alert color table resource "actb" should have the same resource ID as the "ALRT" resource that specifies the alert.

**tb:!NewCDialog** *dStorage boundsRect title visible procID behind goAwayFlag refCon items* [V-283] Function

This trap creates a new color dialog box. The arguments are the same as for the trap **tb:!NewDialog**.

If you want to allocate the memory for the dialog box (which must be at least 176 bytes long), pass a pointer to this memory as *dStorage*. If not, pass **tb:!nilPtr**. The *boundsRect* is the rectangle that defines the boundary of the new dialog window. The *procID* specifies the type of dialog box required.

The pointer *behind* is used if you want the newly created dialog box to appear behind an already existing window. Normally, you pass **tb:!onePtr** and the dialog box is created in front of all the existing windows. *Items* is a handle to the dialog items (controls) associated with the new dialog box.

This trap returns a color dialog pointer *CDialogPtr*.

*Example:*

```
(setf boundsRect
      (make-instance 'tb:Rect :left 112 :top 55
                              :right 239 :bottom 108))
(setf myItems  (tb:!GetResource "DITL" 130))
(setf myDialog (tb:NewCDialog tb:!nilPtr boundsRect "" t
                              1 tb:!onePtr nil 0 myItems))
```

# Chapter 14
# DESK MANAGER

## Introduction

14.1 The Desk Manager traps are used to support desk accessories. To use desk accessories inside a program you only need to use four traps:

- **tb:!SystemTask** - called from the Main Event Loop.
- **tb:!OpenDeskAcc** - called when a desk accessory has been selected from the Apple menu.
- **tb:!SystemClick** - called if the Window Manager trap **tb:!FindWindow** returns **tb:!inSysWindow**.
- **tb:!SystemEdit** - called if a standard Edit menu item was selected.

## Desk Manager Traps

14.2 These traps open and close desk accessories and respond to mouse-down events.

**tb:!OpenDeskAcc** *theAcc* [I-440] Function

Opens the desk accessory with the name *theAcc* and displays it.

*Example:*

```
(defun AppleMenuHandler (menuItem)
  (if (= 1 menuItem)
      ;;then handle the "About..." dialog box
     ...)
      ;;else open the selected desk accessory
     (tb:!OpenDeskAcc
         (tb:GetItem AppleMenu menuItem))))
```

**tb:!CloseDeskAcc** *refNum* [I-440] Function

Closes a desk accessory. You can get the *refNum* from the desk accessory's window record. The *refNum* is kept in the **:windowKind** instance variable.

**tb:!SystemClick** *theEvent theWindow* [I-441] Function

When a mouse-down event occurs and the Window Manager trap **tb:!FindWindow** returns **tb:!inSysWindow**, your application should call this trap to handle the event.

**tb:!SystemEdit** *editCmd* [I-441] Function

Called when there is a mouse-down event in the menu bar and one of the five standard edit functions was selected. If this function returns **nil**, your application should do the required editing.

**tb:!Undo** [I-441] Constant
**tb:!Cut** [I-441] Constant
**tb:!Copy** [I-441] Constant
**tb:!Paste** [I-441] Constant
**tb:!Clear** [I-441] Constant

These are the standard edit functions *editCmd* values.

**tb:!SystemTask** [I-442] Function

Performs all the periodic events for any open desk accessories. This trap should be called once in the main event loop if you wish to support desk accessories. If your application calls **tb:!WaitNextEvent** instead of **tb:!GetNextEvent**, do not call this function.

**tb:!SystemEvent** *theEvent* [I-442] Function

This is an internal trap used by the Event Manager.

**tb:!SystemMenu** *menuResult* [I-443] Function

This is an internal trap used by the Menu Manager.

# Chapter 15
# SCRAP MANAGER

## Introduction

15.1 The Scrap Manager is used for accessing and manipulating the desktop scrap file. The only trap of note in the Scrap Manager is InfoScrap, which returns a pointer to information about the desk scrap.

## Getting Desk Scrap Information

15.2 These routines provide information about the data that is stored in the desktop scrap file.

### tb:ScrapStuff — [I-457] Flavor

This flavor records information about the desk scrap.

### :scrapSize — Method of tb:ScrapStuff

This is the size of the scrap in bytes.

### :scrapHandle — Method of tb:ScrapStuff

This is a handle to the scrap if it is in memory or a NIL handle if it is not.

### :scrapCount — Method of tb:ScrapStuff

This integer changes every time **tb:!ZeroScrap** is called. If this count changes, then you can assume that the desk scrap has changed.

### :scrapState — Method of tb:ScrapStuff

This value indicates where the desk scrap is:

- positive => in memory
- zero => on disk
- negative => hasn't been initialized by **tb:!ZeroScrap**

### :scrap — Method of tb:ScrapStuff

This is a pointer to a string naming the scrap (typically "Clipboard File").

### tb:!_InfoScrap — [I-457] Function

The trap **tb:!_InfoScrap** returns a pointer to information about the desk scrap. Use **tb:!InfoScrap**, it's easier.

**tb:!InfoScrap** [I-457] Function
**tb:!_InfoScrap** [I-457] Function

**tb:!InfoScrap** returns an instance of type **tb:ScrapStuff** which contains information about the desk scrap. **tb:!_InfoScrap** is similar except it returns a pointer to the instance.

*Example:*

```
(setf myScrapInfo (tb:!InfoScrap))
(send myScrapInfo :scrapSize)  => 0
(send myScrapInfo :scrapCount) => 9
```

## Keeping the Desk Scrap on the Disk

15.3 These functions load and unload desk scrap from memory.

**tb:!UnloadScrap** [I-458] Function

Writes the desk scrap in memory to the ScrapFile on disk.

**tb:!LoadScrap** [I-458] Function

Loads the desk scrap from the ScrapFile on disk into memory.

## Writing to the Desk Scrap

15.4 These routines write to the desk scrap, add new data, or clear the scrap.

**tb:!ZeroScrap** [I-458] Function

Clears the scrap in memory and changes the scrap count. Call this function before putting anything into the desk scrap.

*Example:*

```
(send (tb:!InfoScrap) :scrapCount) => 9
(tb:!ZeroScrap)                    => noErr
(send (tb:!InfoScrap) :scrapCount) => 17
```

**tb:!PutScrap** *length theType source* [I-459] Function

Puts the data pointed to by *source*, of type *theType* with a length *length*, into the desk scrap.

## Reading From the Desk Scrap

15.5 This trap reads information from the desk scrap.

**tb:GetScrap** *hDest theType* [I-459] Function

Gets the scrap of type *theType* from the desk scrap, which is at an offset of *offset* into the desk scrap, and copies it to the handle *hDest*.

The trap returns the size of the copied data. See *Inside Macintosh* for details. Use this trap instead of **tb:!GetScrap**.

**tb:!GetScrap** *hDest theType offset* [I-459] Function

Gets the scrap of type *theType* from the desk scrap, which is at an offset of *offset* into the desk scrap, and copies it to the handle *hDest*. The trap returns the size of the copied data. See *Inside Macintosh* for details.

# Chapter 16
# TOOLBOX UTILITIES

## Introduction

16.1 The Toolbox Utilities are a collection of traps that are used for:

- Fixed-point arithmetic
- String manipulation
- Byte and bit manipulation
- Graphics utilities

None of the arithmetic traps apply to the microExplorer environment. These are documented for completeness only.

## Fixed-Point Arithmetic

16.2 The Toolbox fixed-point numbers can be considered 32-bit integers. A fixed-point number is essentially two 16-bit integers packed into a 32-bit integer. See *Inside Macintosh,* page I-79, for more details. To get at the two fields of a fixed-point number use the traps **tb:!HiWord** and **tb:!LoWord**. It is best to use Lisp functions to perform these operations.

**tb:!FixRatio** *numer denom* [I-467] Function

Returns the fixed-point quotient of the two integers *numer* and *denom.*

**tb:!FixMul** *a b* [I-467] Function

Returns the result of multiplying the two fixed-point numbers *a* and *b.*

**tb:!FixRound** *x* [I-467] Function

Rounds the fixed-point number *x* up to the nearest integer.

## String Manipulation

16.3 The following functions manipulate strings. It is important not to confuse the Pascal types StringHandle and Str255.

**tb:!NewString** *theString* [I-468] Function

Creates a relocatable string StringHandle containing a he text string of up to 255 characters. To get a string handle to the string "sample string", do the following:

*Example:*

```
(setf sampleStringHandle
      (tb:!NewString "sample string"))
```

**tb:!SetString** *h theString* [I-468] Function

Sets the relocatable string *h* to the string *theString* where the string is limited to 255 characters.

**tb:!GetString** *stringID* [I-468] Function

Returns a handle to a "STR " type resource (there is a space after the R), which has the resource ID *stringID*.

**tb:!GetIndString** *strListID index* [I-468] Function

Returns the *index*th string from a "STR#" resource with the resource ID *strListID*.

**tb:mx-string-to-mac-string** *theString* Function

Copies *theString* from the microExplorer to a relocatable string in Macintosh memory and returns a handle to the Macintosh string. This is the same as **tb:!NewString**.

**tb:mac-string-to-mx-string** *theString* Function

Copies the string pointed at by *theString* from Macintosh memory to the microExplorer and returns a Lisp string. *TheString* can be either a **tb:mac-pointer** or a **tb:mac-handle**.

## Byte Manipulation

16.4 The following three traps are very involved. Refer to *Inside Macintosh* for details.

**tb:!Munger** *h offset ptr1 len1 ptr2 len2* [I-468] Function

See *Inside Macintosh.*

**tb:!PackBits** *srcPtr dstPtr srcBytes* [I-470] Function
**tb:!UnPackBits** *srcPtr dstPtr dstBytes* [I-470] Function

Used for packing and unpacking MacPaint® documents.

## Bit Manipulation

16.5 The bit numbering scheme used for bit oriented traps is the most significant bit of the first 32-bit word is zero, the most significant bit of the next 32-bit word is 32, and so on. It is better to use Lisp functions to perform these operations.

**tb:!BitTst** *bytePtr bitNum* [I-471] Function

Returns true if the bit number *bitNum* from the pointer *bytePtr* is set, and false if it is not.

**tb:!BitSet** *bytePtr bitNum* [I-471] Function
**tb:!BitClr** *bytePtr bitNum* [I-471] Function

Set or clear the bit number *bitNum* from the pointer *bytePtr*.

**tb:!BitAnd** *value1 value2* [I-471] Function
**tb:!BitOr** *value1 value2* [I-471] Function
**tb:!BitXor** *value1 value2* [I-471] Function

Perform a logical AND, logical OR, or logical XOR, respectively, on the two 32-bit integers *value1* and *value2*.

**tb:!BitNot** *value* [I-471] Function

Performs a logical NOT on the 32-bit integer *value*.

**tb:!BitShift** *value count* [I-472] Function

Shifts the 32-bit integer *value* count bits to the left if *count* is positive, or *count* bits to the right if *count* is negative.

## Other Operations on Long Integers

16.6 These routines also perform operations on long integers. It is better to use Lisp functions to perform these operations.

**tb:!HiWord** *x* [I-472] Function
**tb:!LoWord** *x* [I-472] Function

Return the integer in the most significant or least significant 16 bits of the 32-bit integer *x*, respectively.

## Graphic Utilities

16.7 These routines act on icons, cursors, patterns, and pictures.

**tb:!ScreenRes** [I-473] Function

Returns two values indicating the resolution of the Macintosh being used. The number of horizontal pixels per inch is the first value returned and the number of vertical pixels per inch is the second.

*Example:*

```
(multiple-value-bind (scrnHRes scrnVRes)
    (tb:!ScreenRes)
scrnHRes => 72
scrnVRes => 72
```

**tb:!GetIcon** *iconID* [I-473] Function

Returns a handle to the "ICON" resource with the resource ID *iconID*.

**tb:!PlotIcon** *theRect theIcon* [I-473] Function

Draws the icon *theIcon* inside the rectangle *theRect.*

**tb:!GetPattern** *patID* [I-473] Function

Returns a handle to the "PAT " resource (there is a space after the T), which has the resource ID *patID.*

**tb:!GetIndPattern** *patListID index* [I-473] Function

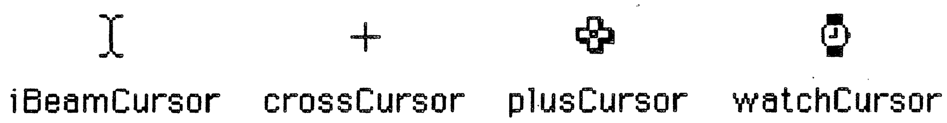
Returns the *index*th pattern from the "PAT#" resource that has the resource ID *patListID.*

**tb:!GetCursor** *cursorID* [I-474] Function

Returns a handle to the "CURS" resource with the resource ID *cursorID.*

The Standard Cursor resource ID's are: **tb:!IBeamCursor**, **tb:!CrossCursor**, **tb:!PlusCursor**, and **tb:!WatchCursor**.

Standard Cursors



**tb:!ShieldCursor** *shieldRect offsetPt* [I-474] Function

Hides the cursor if the cursor and *shieldRect* intersect.

**tb:!GetPicture** *picID* [I-475] Function

Returns a handle to the "PICT" resource with the resource ID *picID.*

## Miscellaneous Utilities

16.8 The following traps perform miscellaneous utility functions. It is better to use Lisp functions to perform these operations.

**tb:!DeltaPoint** *ptA ptB* [I-475] Function

Subtracts the point *ptB* from the point *ptA* and returns the resulting point as a 32-bit integer. The vertical coordinate of the point is the high order 16 bits and the horizontal coordinate of the point is the low order 16 bits.

**tb:!SlopeFromAngle** *angle* [I-475] Function
**tb:!AngleFromSlope** *slope* [I-476] Function

Converts between an angle *angle* and a slope dh/dv as a fixed-point number.

## Fixed-point Arithmetic

16.9 arithmetic is better handled by the microExplorer than the Macintosh. Therefore, the following traps should not be used:

**tb:!Long2Fix** *x* [IV-65] Function
**tb:!Fix2Long** *x* [IV-65] Function

Converts *x* between a longInt and a fixed-point number.

**tb:!Fix2Frac** *x* [IV-65] Function
**tb:!Frac2Fix** *x* [IV-65] Function

Converts *x* between a fixed-point and a fractional number.

**tb:!Fix2X** *x* [IV-65] Function
**tb:!X2Fix** *x* [IV-65] Function

Converts *x* between a fixed-point and an extended number.

**tb:!Frac2X** *x* [IV-65] Function
**tb:!X2Frac** *x* [IV-65] Function

Converts *x* between a fractional and an extended number.

**tb:!FracSin** *x* [IV-64] Function
**tb:!FracCos** *x* [IV-64] Function

Returns the sine or cosine respectively of the fixed radian argument *x*. *x* is of type fixed and the result is of type fract.

**tb:!FracSqrt** *x* [IV-64] Function

Returns the square root of *x*, with *x* interpreted as an unsigned fract in the range of 0 through $4-(2^{-30})$, inclusive.

**tb:!FracMul** *x y* [IV-64] Function

Returns *x* multiplied by *y*. See *Inside Macintosh* for details.

**tb:!FracDiv** *x y* [IV-64] Function

Returns *x* divided by *y*. See *Inside Macintosh* for details.

**tb:!FixATan2** *x y* [IV-65] Function

Returns the arctangent of *y* divided by *x* in radians.

**tb:!FixDiv** *x y* [IV-64] Function

Returns *x* divided by *y*. See *Inside Macintosh* for details.

# Chapter 17
# PACKAGE MANAGER

## Introduction

17.1 The Package Manager provides access to packages, the sets of data structures, and routines that are stored as resources and brought into memory only when needed.

## International Utilities Package

17.2 The routines in this package access country-dependent information such as the formats for numbers, currency, dates, and times. Use of this package will enable you to make your application country-independent.

| | | |
|---|---|---|
| **tb:IUDateString** *dateTime form* | [I-504] | Function |
| **tb:!IUDateString** *dateTime form* VAR *result* | [I-504] | Function |
| **tb:IUDatePString** *dateTime form intlParam* | [I-505] | Function |
| **tb:!IUDatePString** *dateTime form* VAR *result intlParam* | [I-505] | Function |

**tb:IUDateString** returns the date contained in *dateTime* (which is returned by the trap **tb:!GetDateTime**) as a string. The format of this string is determined by *form* which is one of the constants described below.

**b:!IUDateString** is similar to **tb:IUDateString** except it modifies *result* with the string.

**tb:IUDatePString** is similar to **tb:IUDateString** except that the *form* argument is overridden by the data format in *intlParam.*

**tb:!IUDatePString** is similar to **tb:!IUDateString** except that the *form* argument is overridden by the data format in *intlParam.*

| | | |
|---|---|---|
| **tb:shortDate** | [I-504] | Constant |
| **tb:longDate** | [I-504] | Constant |
| **tb:abbrevDate** | [I-504] | Constant |

These are constants for use in the data format *form* argument to **tb:IUDateString** and **tb:!IUDateString.**

| | | |
|---|---|---|
| **tb:IUTimeString** *dateTime wantSeconds* | [I-505] | Function |
| **tb:!IUTimeString** *dateTime wantSeconds* VAR *result* | [I-505] | Function |
| **tb:IUTimePString** *dateTime wantSeconds intlParam* | [I-505] | Function |
| **tb:!IUTimePString** *dateTime wantSeconds* VAR *result* *initParam* | [I-505] | Function |

**tb:IUTimeString** returns the time contained in *dateTime* (which is returned by the trap **tb:!GetDateTime**) as a string. If *wantSeconds* is true, the time of day is returned with seconds included. If *wantSeconds* is false, only the hours and minutes are returned.

**tb:!IUTimeString** is similar to **tb:IUTimeString** except that it modifies *result* to the formatted string.

**tb:IUTimePString** is similar to **tb:IUTimeString** except the format is determined by the resource *intlParam* rather than the default resource.

**tb:!IUTimePString** is similar to **tb:!IUTimeString** except the format is determined by the resource *intlParam* rather than the default resource.

**tb:!IUMetric** [I-505] Function

Returns true if international resource 0 specifies that the metric system is to be used.

**tb:!IUGetIntl** *theID* [I-505] Function

Returns a handle to the international resource (resource type "INTL") which has a resource ID *theID*.

**tb:!IUSetIntl** *refNum theID intlParam* [I-506] Function

Sets the international resource (resource type "INTL") resource ID *theID* in the resource file with a reference number *refNum* to the data specified by *intlParam*.

**tb:!IUMagString** *aPtr bPtr aLen bLen* [I-506] Function
**tb:!IUMagIDString** *aPtr bPtr aLen bLen* [I-507] Function

**tb:!IUMagString** compares a string starting at *aPtr* and having a length *aLen*, with a string starting at *bPtr* and having a length *bLen*, using both primary and secondary ordering. Returns -1 if *aPtr* string is less than *bPtr* string, 0 if *aPtr* string is equal to *bPtr* string, and 1 if *aPtr* string is greater than *bPtr* string.

**tb:!IUMagIDString** is similar except it uses only the primary ordering.

## Standard File Package

17.3 The routines in this package present the standard user interface when a file is to be saved or opened. These routines use the SFReply data structure.

**tb:SFReply** [I-519] Flavor

This flavor defines a user's reply to a query for an input file namestring. **tb:SFReply** objects are true microExplorer instances.

**:good** Method of **tb:SFReply**

If true, then the user specified a filename and then clicked on Open or Save. If false, then the user clicked on Cancel.

**:fType** Method of **tb:SFReply**

This is a four-character string identifying the file type of the user's selection. This field is unrelated to the Lisp pathname type component.

**:vRefNum** Method of **tb:SFReply**

This is nominally the volume reference number of the user's selection. In the hierarchical file system as is used on the microExplorer, it effectively identifies the volume and directory.

**:version** Method of **tb:SFReply**

This is nominally the file version number. However, since many Macintosh utilities assume version 0, this field is not particularly important yet.

**:fName** Method of **tb:SFReply**

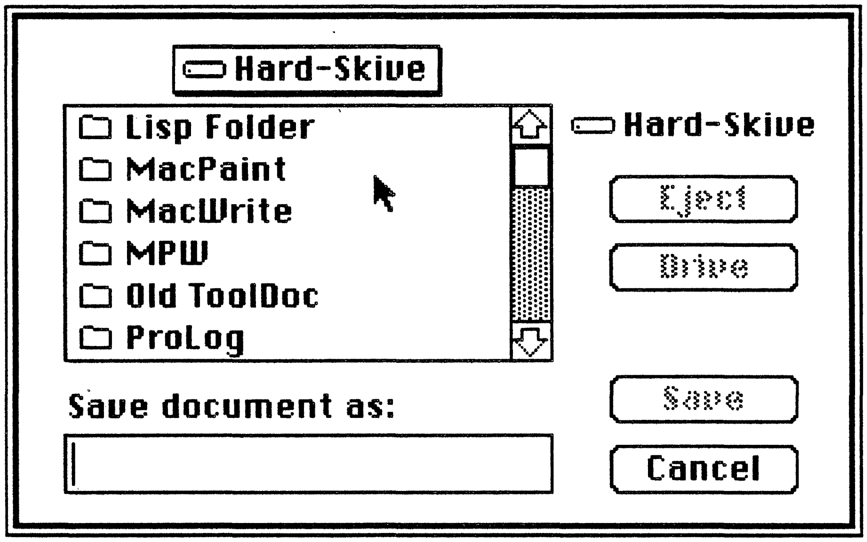
This is a string naming the file on the Macintosh file system.

> NOTE: For files with Explorer-style names, this string is a concatenation of the Lisp pathname name and type components separated by a period (e.g., "FOO.LISP").

**tb:!SFPutFile** *where prompt origName dlgHook reply* [I-519] Function
**tb:!SFPPutFile** *where prompt origName dlgHook reply dlgID filterProc* [I-523] Function

**tb:!SFPutFile** displays a "Save as..." dialog box at the point *where* with a prompting string *prompt* and a default fName reply *origName*. The dialog result is returned in *reply*, an instance of **tb:SFReply**. The *dlgHook* and *filterProc* are not available now, so pass **tb:!nilPtr**.

**tb:!SFPPutFile** is similar except it allows you to define your own dialog box, which has a resource ID *dlgID*.



*Example:*

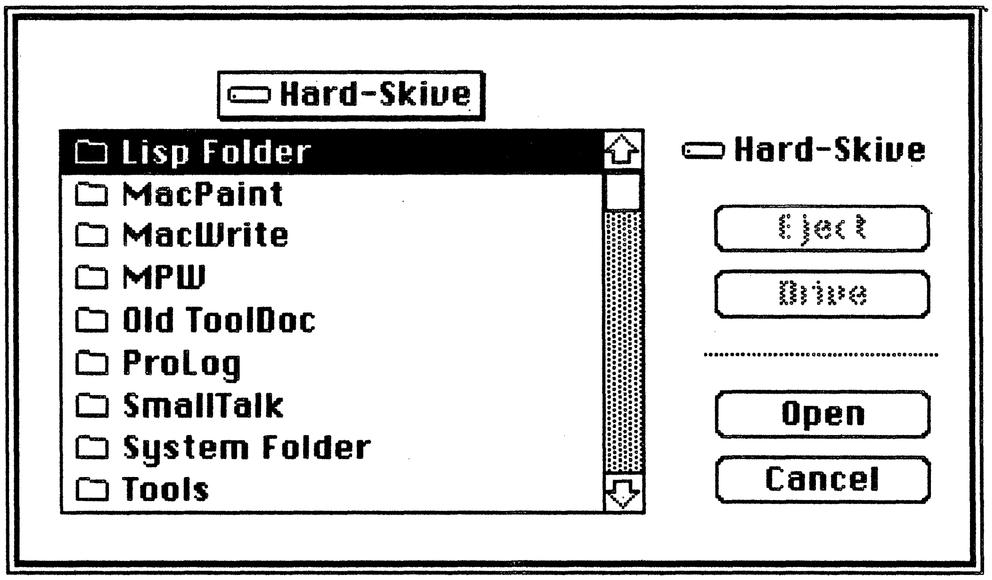
```
(setq where (make-instance 'tb:Point :H 133 :V 78))
(setq reply (make-instance 'tb:SFReply))
(tb:!SFPutFile where "Save document as:"  "" tb:!nilPtr
               reply)
```

**tb:!SFGetFile** *where prompt fileFilter numTypes typeList dlgHook reply* [I-523] Function

**tb:!SFPGetFile** *where prompt fileFilter numTypes typeList dlgHook reply dlgID filterProc* [I-526] Function

**tb:!SFGetFile** displays an "Open" dialog box, at the point *where*, with a prompt string *prompt* (not displayed in the dialog box). The dialog box will display a scrolling window containing the names of all files of the types (four-character OS type names) in *typeList* which is a lot of file types (see example) on the current disk volume. If you pass **nil** for *typeList* and 0 for *numTypes*, then all files will be displayed. The dialog result is returned in *result*. The *dlgHook* and *filterProc* arguments are not available now, so pass **tb:!nilPtr.**

**tb:!SFPGetFile** is similar except it allows you to define your own dialog box, which has a resource ID *dlgID*.



*Example:*

```
(setq where (make-instance 'tb:Point :H 133 :V 78))
(setq typeList '("TEXT"))
(setq reply (make-instance 'tb:SFReply))
(tb:!SFGetFile where "" tb:!nilPtr 1 typeList tb:!nilPtr
               reply)
```

NOTE: The file types in *typeList* are Macintosh OSTypes and are unrelated to Lisp pathname type components. In particular, the fact that the Macintosh OS file type of "TEXT" in this example means the same as a Lisp file type of "TEXT" is just a coincidence.

# Chapter 18
# MEMORY MANAGER

## Introduction

18.1 The Memory Manager is used to create and manipulate blocks of memory in the Macintosh memory. There are two types of memory blocks: relocatable and non-relocatable. Relocatable blocks of memory are created with the trap **tb:!NewHandle**. Non-relocatable blocks of memory are created with the trap **tb:!NewPtr**. It is best to stay away from non-relocatable blocks because they give rise to fragmented heaps - which quickly degrades the performance of and amount of memory available to a Macintosh program.

The relocatable blocks are referred to by handles which are really indirect pointers. The handle is a pointer to a place in memory where the actual pointer to the block is kept. The reasoning behind this is that when the Memory Manager is allocating new blocks of memory it sometimes finds it necessary to move blocks around in memory to make room for the new block. Having handles (indirect pointers) means it can move the blocks around without invalidating all the references to the other blocks.

Some of the more useful Memory Manager OSUtility traps are also documented at the end of this section.

## Initialization and Allocation

18.2 These routines initialize, modify, and create new heap zones.

**tb:!SetApplBase** *startPtr* [II-28] Function

Sets the base of the application heap to *startPtr*.

**CAUTION: This trap is very dangerous and should not be used.**

**tb:!InitZone** *ptr* [II-29] Function

Creates and initializes a new heap zone. *Ptr* points to a block of memory containing the four variables startPtr, limitPtr, cMoreMasters, and pGrowZone. See *Inside Macintosh* for details.

CAUTION: This trap is very dangerous. Use it only if you know exactly what you are doing.

**tb:!GetApplLimit** [II-29] Function

Returns the current application heap's *limitPtr*, the highest address to which it can grow.

**tb:!SetApplLimit** *zoneLimit* [II-30] Function

Sets the highest address of the current application heap to *zoneLimit*.

**CAUTION: This trap is very dangerous. Never use this trap.**

**tb:!MoreMasters** [II-31] Function

Allocates another block of master pointers. This should be done at the beginning of a program if you expect to be creating a lot of relocatable memory blocks.

## Heap Zone Access

**18.3** These routines provide access to heap zones.

**tb:!GetZone** [II-31] Function

Returns two values: a pointer to the current heap zone and an operating system result code.

**tb:!SetZone** *hz* [II-31] Function

Sets the current heap zone to the zone pointed to by *hz*.

**CAUTION: This trap is very dangerous. Use this trap only if you know exactly what you are doing.**

**tb:!SystemZone** [II-32] Function

Returns a pointer to the System Zone, the zone that the Macintosh Toolbox uses as its own private memory.

**tb:!ApplicZone** [II-32] Function

Returns a pointer to the original Application Zone.

## Allocating and Releasing Relocatable Blocks

**18.4** These routines allocate new blocks of relocatable memory and release those that are no longer needed.

**tb:!NewHandle** *logicalSize* [II-32] Function

Returns two values: a handle to a relocatable object in memory with the size specified in *logicalSize*, and an OSErr.

**tb:!nilHndl** Constant

This constant is an instance of **tb:mac-handle** is a **:handle** instance variable of zero. Use this constant wherever the *inside Macintosh* documentation says to use a (Pascal) NIL for a handle argument.

**tb:!DisposHandle** *h* [II-33] Function

Disposes of a handle to a relocatable block in memory that has the handle *h*.

**CAUTION: Once a handle has been disposed of, all references to the handle become invalid. If you try to dispose of an already disposed handle, it will damage the master pointer block.**

**tb:!GetHandleSize** *h* [II-33] Function

Returns the logical size of the handle *h* as an integer.

**tb:!SetHandleSize** *h newSize* [II-34] Function

Sets the logical size of the handle *h* to the value *newSize*.

**tb:!HandleZone** *h* [II-34] Function

Returns two values: a zone pointer to the heap zone that contains the handle *h*, and an OSErr.

**tb:!RecoverHandle** *p* [II-35] Function

Returns a handle to the relocatable object pointed to by *p*.

**tb:!ReallocHandle** *h logicalSize* [II-35] Function

Allocates a new relocatable block with a size *logicalSize* and updates the handle *h* to point to this new block. This trap is used when a handle has been purged.

## Allocating and Releasing Non-Relocatable Blocks

18.5 These routines allocate new blocks of non-relocatable memory and release those that are no longer needed.

**tb:!NewPtr** *logicalSize* [II-36] Function

Returns two values: a pointer to a non-relocatable object in memory with a size of *logicalSize*, and an OSErr. Creating non-relocatable blocks causes fragmentation of the heap. When the heap becomes fragmented, large areas become unusable and the machine slows down. (See *Inside Macintosh* for details).

When you need a block of memory, it is best to create it with **tb:!NewHandle** and then temporarily lock it in memory when it is needed.

The File Manager trap **tb:!Read** requires that you pass it a pointer to the buffer where it will place the read information. To create this buffer, you first create a new relocatable block of the required size:

*Example:*

```
(setf myBufferHandle (tb:!NewHandle myBufferSize))
```

Now, lock it by doing:

```
(tb:!HLock myBufferHandle)
```

This stops the Memory Manager from moving it while you are using it. Next, get a pointer to the buffer (the handle is a pointer to a pointer):

```
(setf myBufferPtr (tb:deref myBufferHandle))
```

**myBufferPtr** now points to the start of the buffer. After filling the required fields of the block, pass the pointer to the trap:

```
(send paramBlock :ioBuffer myBufferPtr)
(tb:!Read paramBlock)
```

After calling the trap, do not forget to unlock the handle. Leaving it locked has the same effect as creating a non-relocatable block, it fragments the heap. So do the following:

```
(tb:!HUnLock myBufferHandle)
```

**tb:!nilPtr** Constant

This constant is an instance of **tb:mac-pointer** with a **:pointer** instance variable of zero. Use this constant wherever the *Inside Macintosh* documentation says to use a (Pascal) NIL as a pointer argument.

**tb:!onePtr** Constant

This constant is an instance of **tb:mac-pointer** with a **:pointer** instance variable of #xFFFFFF. Use this constant wherever the *Inside Macintosh* documentation says to use a (Pascal) -1 as a pointer argument.

**tb:!DisposPtr** *p* [II-36] Function

Disposes of a non-relocatable block in memory which is pointed to by *p*.

**CAUTION: Once a pointer has been disposed of, all references to the pointer become invalid. If you try to dispose of an already disposed pointer it will damage the master pointer block.**

**tb:!GetPtrSize** *p* [II-37] Function

Returns the logical size of the pointer *p*.

**tb:!SetPtrSize** *p newSize* [II-37] Function

Sets the logical size of the pointer *p* to a new value *newSize*.

**tb:!PtrZone** *p* [II-38] Function

Returns the heap zone pointer of the heap that contains the non-relocatable block *p*.

## Freeing Space in the Heap

**18.6** These routines free space in the heap.

**tb:!FreeMem** [II-38] Function

Returns the amount of free memory in the current heap zone.

**tb:!MaxMem** [II-38] Function

This trap compacts the current heap zone, purges everything purgeable, and returns two values: the size of the largest contiguous free block and the maximum number of bytes that the heap zone can be grown to.

**tb:!CompactMem** *cbNeeded* [II-39] Function

Compacts the current heap by moving all relocatable blocks towards the bottom of the heap (it does not purge any!) until a contiguous space of a least *cbNeeded* is available.

**tb:!ResrvMem** *cbNeeded* [II-39] Function

Creates a free space for a block with a size of *cbNeeded* bytes. When you need to create a handle that will exist for a long time, call the trap **tb:!ResrvMem** early in your program. The trap will reserve the required memory near the bottom of the heap. This reduces any heap fragmentation problems.

**tb:!PurgeMem** *cbNeeded* [II-40] Function

Purges purgeable blocks from the current heap zone until *cbNeeded* contiguous bytes are free or all the purgeable blocks have been purged.

**tb:!EmptyHandle** *h* [II-40] Function

Purges the handle *h*.

## Properties of Relocatable Blocks

**18.7** These routines lock and unlock handles and make them purgeable or unpurgeable.

**tb:!HLock** *h* [II-41] Function
**tb:!HUnlock** *h* [II-41] Function

**tb:!HLock** locks the handle *h*. This stops the Memory Manager from moving the block while it is being used. **b:!HUnlock** removes the lock.

To reduce the probability of heap fragmentation, call the trap **tb:!MoveHHi** before you lock a handle. This trap will move the handle to the edge of the nearest non-relocatable block assuring it doesn't add to fragmentation..

**tb:!HPurge** *h* [II-41] Function
**tb:!HNoPurge** *h* [II-42] Function

Makes the handle *h* purgeable or not purgeable.

**tb:!HSetRBit** *h* [IV-79] Function
**tb:!HClrRBit** *h* [IV-79] Function

Sets or clears the resource bit in the handle *h*.

**tb:!HGetState** *h* [IV-79] Function

Returns the flags of the handle *h*.

**tb:!HSetState** *h* *theState* [IV-80] Function

Sets the flags of the handle *h* to *theState*.

## Grow Zone Operations

**18.8** These routines perform operations which affect the grow zone.

**tb:!SetGrowZone** *growZone* [II-42] Function

Sets the current heap grow zone procedure to that defined in *growZone*.

CAUTION: This trap is very dangerous.

**tb:!GZSaveHnd** [II-43] Function

Returns a handle that must not be moved by the growZone function.

## Miscellaneous Routines

18.9 These routines perform miscellaneous Memory Manager operations.

**tb:!BlockMove** *sourcePtr destPtr byteCount* [II-44] Function

Moves a block of *byteCount* bytes starting at *sourcePtr* to *destPtr*.

**tb:block-move** *source dest count* [II-44] Function

A more generalized version of **tb:!BlockMove**. Moves *count* elements from *source* to *dest*. *source* and *dest* can be pointers, handles or Lisp arrays Note that *count* is a number of elements, not bytes. However, when both *source* and *dest* are either pointers or handles, *count* is assumed to be in bytes.

**tb:!TopMem** [II-44] Function

Returns a pointer to the end of memory.

**tb:!MemError** [II-44] Function

Returns the last Memory Manager error number.

**tb:!MaxBlock** [IV-77] Function

Returns the free space available without purging the current heap zone.

**tb:!PurgeSpace** [IV-78] Function

Returns two values: the total number of bytes that could be obtained by purging all purgeable blocks, and the largest contiguous space in bytes that would exist after the purge. This is done without actually purging anything.

**tb:!MaxApplZone** [II-30] Function

Grows the application heap zone to ApplLimit.

**tb:!MoveHHi** *h* [II-44] Function

Floats the handle *h* as high up the current heap as is possible.

**tb:!StackSpace** [IV-78] Function

Returns the amount of space between the current stack pointer and HeapEnd.

**tb:!NewEmptyHandle** [IV-78] Function

Creates a new empty handle.

## Memory Manager Utilities

**18.10** These are a collection of memory management utilities used for converting and manipulating handles and pointers. The only traps you are likely to use are:

- **tb:!HandToHand** to duplicate a handle.
- **tb:!HandAndHand** to append one handle onto another.
- **tb:!PtrAndHand** to add data onto the end of a handle.

**tb:!HandToHand** *theHandle* [II-374] Function

Makes a copy of the handle in *theHandle* and returns two values: the handle to the copy, and an Operating System result code. If you need to make a copy of the handle **myHandle**, you could do the following:

*Example:*

```
(setf myHandle (tb:!NewHandle 10))
(multiple-value-bind (newHandle result)
    (tb:!HandToHand myHandle)
  ...)
```

A copy of the handle **myHandle** will be returned in **newHandle**.

**tb:!PtrToHand** *srcPtr size* [II-375] Function

Returns two values: a new handle which is a copy of the *size* bytes starting at *srcPtr*, and an OSErr. To make a relocatable copy of a non-relocatable block **myPointer**, do the following:

*Example:*

```
(setq pointerSize (tb:!GetPtrSize myPointer))
(multiple-value-bind (newHandle result)
    (tb:!PtrToHand myPointer pointerSize)
  ...)
```

A handle to the new relocatable block will be returned in **newHandle**.

**tb:!PtrToXHand** *srcPtr dstHndl size* [II-375] Function

Takes an existing handle *dstHndl* and makes it a copy of the *size* bytes starting at *srcPtr* and returns an error code as a result. To set the existing handle **myHandle** to the contents of the pointer **myPointer**, do the following:

*Example:*

```
(setf pointerSize (tb:!GetPtrSize myPointer))
(setf result
      (tb:!PtrToXHand myPointer myHandle pointerSize))
```

**tb:!HandAndHand** *aHndl bHndl* [II-375] Function

Appends the data in *aHndl* onto the end of the handle *bHndl* and returns an error code as a result.

**tb:!PtrAndHand** *ptr hndl size* [II-376] Function

Appends the data (*size* number of bytes) starting at *ptr*, onto the handle *hndl* and returns an error code as a result.

## Accessing Memory

**18.11** These routines are used for accessing the contents of Macintosh memory directly.

**tb:deref** *theHandle* Function

Dereferences *theHandle* once and returns the resulting pointer. Remember to lock *theHandle* before doing this or the resulting pointer may become invalid. The most common use of **tb:deref** is in the File Manager. Many File Manager routines require pointers to buffers. You can allocate the buffer with **tb:!NewHandle**. Then, when using a File Manager routine, you can lock the buffer and then use **tb:deref** to get a pointer to the buffer. After the File Manager routine is finished, the buffer can be unlocked. This will help prevent heap fragmentation.

The following routines allow you to access Macintosh memory directly. Make sure when attempting to access 16-bit or 32-bit values that the location you specify lies on an even address, or you will cause an address error on the Macintosh.

**tb:fetch** *thePointer offset* Function
**tb:fetchword** *thePointer offset* Function
**tb:fetchbyte** *thePointer offset* Function

Returns the 32-bit, 16-bit, or 8-bit integer value, respectively, at the location which is *offset* bytes beyond the location pointed at by *thePointer*.

**tb:fetchhandle** *theHandle offset* Function
**tb:fetchwordhandle** *theHandle offset* Function
**tb:fetchbytehandle** *theHandle offset* Function

Returns the 32-bit, 16-bit, or 8-bit integer value, respectively, at the location which is *offset* bytes beyond the location indirectly pointed at by *theHandle*.

**tb:fetchrect** *thePointer offset rect* Function

Sets *rect* to be the rectangle which is *offset* bytes beyond the location pointed at by *thePointer*.

**tb:fetchrecthandle** *theHandle offset rect* Function

Sets *rect* to be the rectangle which is *offset* bytes beyond the location indirectly pointed at by *theHandle*.

**tb:stow** *thePointer offset theValue* Function
**tb:stowword** *thePointer offset theValue* Function
**tb:stowbyte** *thePointer offset theValue* Function

Stores *theValue* as a 32-bit, 16-it, or 8-bit integer value, respectively, into the location which is *offset* bytes beyond the location pointed at by *thePointer*.

**tb:stowhandle** *theHandle offset theValue* Function
**tb:stowwordhandle** *theHandle offset theValue* Function
**tb:stowbytehandle** *theHandle offset theValue* Function

Stores *theValue* as a 32-bit, 16-bit, or 8-bit integer value, respectively, into the location which is *offset* bytes beyond the location indirectly pointed at by *theHandle*.

# Chapter 19
# SEGMENT LOADER

## Introduction

**19.1** The Segment Loader traps are advanced and obscure. You are advised to read the Segment Loader documentation in *Inside Macintosh* before even attempting to use any of them.

## Segment Loader Traps

**19.2** These routines are included in the documentation for completeness only. Please refer to *Inside Macintosh* for details.

| | |
|---|---|
| **tb:CountAppFiles** | [II-57] Function |
| **tb:!appOpen** | [II-58] Constant |
| **tb:!appPrint** | [II-58] Constant |

This trap should be called when your application is started. If your application is launched from the MultiFinder (by a user double-clicking on its icon or by double-clicking on one of your application's documents), the MultiFinder passes to your application information about any documents that were selected to be opened with your application. **tb:!CountAppFiles** returns two values. The first is either the constant **tb:!appOpen** or **tb:!appPrint** and indicates whether the documents are to be opened or printed, respectively. The second is the number of documents selected.

**tb:launch** *name* [II-57] Function

Launches a Macintosh application. *Name* is a full pathname specifying the application to launch. **tb:launch** returns a Toolbox OSErr which tells whether or not the launch was successful.

# Chapter 20
# OPERATING SYSTEM EVENT MANAGER

## Introduction

**20.1** These traps will almost never be used. The Toolbox Event Manager has all the traps necessary to handle any possible event. The only OS Event Manager trap you might possibly use is **tb:!PostEvent**, which places a user-defined event in the event queue.

## Operating System Event Manager Routines

**20.2** These routines post, remove, and access events.

**tb:!PostEvent** *eventCode eventMsg* [II-68] Function

Returns an OSErr. Event types **tb:!app1Evt**, **tb:!app1Evt**, and **tb:!app3Evt** are available as user-definable events. Do not use **tb:!app4Evt** as it is now used by MultiFinder. To support your own event type it is necessary to have an event handler procedure for that event number in the main event loop. Then, when you want to generate an event, call the trap **tb:!PostEvent**.

To generate an event which has the event code of **tb:!app1Evt** and the current time as the message, do the following:

*Example:*

```
(setf secs (tb:!GetDateTime))
(tb:!PostEvent tb:!app1Evt secs)
```

**tb:!FlushEvents** *eventMask stopMask* [II-69] Function

Removes events from the event queue as specified by the given event masks. Returns 0 if all events were removed from the queue. Otherwise, it returns the event code of the event that caused the flush to stop.

*Example:*

```
;;;flush out all events from the queue.
(tb:!FlushEvents tb:!everyEvent 0)
```

**tb:!GetOSEvent** *theEvent eventMask* [II-69] Function
**tb:!OSEventAvail** *theEvent eventMask* [II-70] Function

Given a **tb:EventRecord** instance, **tb:!GetOSEvent** updates that instance to be the next available event of a specified type or types, and removes it from the event queue. If an event is found in the event queue, **tb:!GetOSEvent** returns -1; otherwise, it returns 0.

**tb:!OSEventAvail** is similar except it does not remove the event from the queue.

NOTE: Since **tb:EventRecords** are true instances on the microExplorer, you cannot pass a pointer to an **tb:EventRecord** object. Instead, allocate a block of memory the same size as an **tb:EventRecord** (16 bytes) and pass a pointer to this block. You can access the various fields of the **tb:EventRecord** using fetch.

*Example:*

```
(setf eventPtr (tb:!NewPtr 16))
(tb:!GetOSEvent eventPtr tb:!everyEvent)
```

**tb:!SetEventMask** *theMask* [II-70] Function

Sets the system event mask to the event mask specified.

**tb:!GetEVQHdr** [II-71] Function

Returns a pointer to the event queue.

# Chapter 21
# FILE MANAGER

## Introduction

21.1 The File Manager is the part of the Macintosh Toolbox that deals with the creation and manipulation of volumes, files, and directories. With the File Manager you can:

- Open files and working directories.
- Read and write files.
- Create new files and directories.
- Get and set information about volumes, files, and directories.

Almost all of the File Manager traps take one argument--a parameter block. A parameter block is a data structure containing fields that control the actions of the trap. There are seven types of parameter blocks used by the File Manager, each represented by a flavor:

- **tb:ioParam**
- **tb:fileParam**
- **tb:volumeParam**
- **tb:CInfoPBRec**
- **tb:CMovePBRec**
- **tb:WDPBRec**
- **tb:FCBPBRec**

NOTE: When one of the above flavors is instantiated, a significant amount of Macintosh memory is allocated. Therefore, to keep the Macintosh memory from filling up, always send your parameter block instance a **:dispose** message when you exit the block in which the instance is bound.

**tb:ParamBlockRec** [II-98] Flavor

This is the base flavor upon which all of the other File Manager parameter block flavors are built. This flavor is never directly instantiated.

**:ioCompletion** Method of **tb:ParamBlockRec**
**:set-ioCompletion** *procPtr* Method of **tb:ParamBlockRec**

This is a pointer to a procedure to be executed at the end of an asynchronous call. It is automatically set to **nil** for synchronous calls.

**:ioResult** Method of **tb:ParamBlockRec**

This is the result code of the operation. For asynchronous calls, it is set to 1 upon receipt of the call and then set to the final result code upon completion.

**:ioNamePtr** Method of **tb:ParamBlockRec**
**:set-ioNamePtr** *StringPtr* Method of **tb:ParamBlockRec**

This is a pointer to the volume namestring or to the file namestring optionally prefixed with the volume.

**:ioVRefNum** Method of **tb:ParamBlockRec**
**:set-ioVRefNum** *16b-integer* Method of **tb:ParamBlockRec**

Depending upon the operation, this is either a volume reference number or a drive number.

**tb:ioParam** [II-100] Flavor

This flavor defines the parameter block needed for I/O on open files. This flavor is built on **tb:ParamBlockRec**.

**:ioRefNum** Method of **tb:ioParam**
**:set-ioRefNum** *16b-integer* Method of **tb:ioParam**

This is the file's path reference number.

**:ioVersNum** Method of **tb:ioParam**
**:set-ioVersNum** *8b-integer* Method of **tb:ioParam**

This is the file version number. This field is nominally the same as the Lisp pathname version component, but major pieces of Macintosh software ignore it so it is normally zero.

**:ioPermssn** Method of **tb:ioParam**
**:set-ioPermssn** *8b-integer* Method of **tb:ioParam**

This instance variable controls the file access permission. Its value is one of the following constants.

| | |
|---|---|
| **tb:!fsRdPrem** | [IV-120] Constant |
| **tb:!fsWrPerm** | [IV-120] Constant |
| **tb:!fsRdWrPerm** | [IV-120] Constant |
| **tb:!fsRdWrShPerm** | [IV-120] Constant |
| **tb:!fsCurPerm** | [IV-120] Constant |

These constants provide the values for the **:ioPermssn** instance variable of the **tb:ioParam** flavor. These constants represent read-only, write-only, exclusive read-write, shared read-write, and whatever is currently allowed, respectively.

**:ioMisc** Method of **tb:ioParam**
**:set-ioMisc** *pointer* Method of **tb:ioParam**

The values of these instance variables vary with the operation and are normally **tb:!nilPtr**.

**:ioBuffer** — Method of **tb:ioParam**
**:set-ioBuffer** *pointer* — Method of **tb:ioParam**

These are pointers to the read-write data buffer.

**:ioReqCount** — Method of **tb:ioParam**
**:set-ioReqCount** *32b-integer* — Method of **tb:ioParam**

These are the number of bytes to be read, written, or allocated.

**:ioActCount** — Method of **tb:ioParam**

This 32-bit integer is the number of bytes actually read, written, or allocated.

**:ioPosMode** — Method of **tb:ioParam**
**:set-ioPosMode** *16b-integer* — Method of **tb:ioParam**

This controls the positioning of the file for reads and writes. Its value should be one of the following constants.

**tb:!fsAtMark** — [IV-100] Constant
**tb:!fsFromStart** — [IV-100] Constant
**tb:!fsFromLEOF** — [IV-100] Constant
**tb:!fsFromMark** — [IV-100] Constant

These constants provide the value for the **:ioPosMode** instance variable on the **tb:ioParam** flavor. These constants represent position at current mark, relative to start of file, relative to logical EOF, and relative to current mark, respectively. (See also **tb:!rdVerify**.)

**tb:!rdVerify** — [IV-100] Constant

If this value is added to any of the position mode constants above, then it indicates that a verify should be done for writes.

**:ioPosOffset** — Method of **tb:ioParam**
**:set-ioPosOffset** *32b-integer* — Method of **tb:ioParam**

This specifies the byte offset relative to the position specified by **:ioPosMode**.

**tb:FileParam** — [II-101] Flavor

This flavor defines the parameter block needed needed to change information about files. This flavor is built on **tb:ParamBlockRec**.

**:ioFRefNum** — Method of **tb:fileParam**
**:set-ioFRefNum** *16b-integer* — Method of **tb:fileParam**

This is the file's path reference number.

**:ioFVersNum** Method of **tb:fileParam**<br>
**:set-ioFVersNum** *8b-integer* Method of **tb:fileParam**

This is the file version number. This field is nominally the same as the Lisp pathname version component, but major pieces of Macintosh software ignore it, so it is normally zero.

**:ioFDirIndex** Method of **tb:fileParam**

This 16-bit integer is a unique sequence number of the file on the volume. It can be used for indexing files on the volume.

**:ioFlAttrib** Method of **tb:fileParam**

If bit 0 of this 8-bit integer is 0, the file is locked.

**:ioFlVersNum** Method of **tb:fileParam**<br>
**:set-ioFlVersNum** *8b-integer* Method of **tb:fileParam**

This is the file version number. This field is nominally the same as the Lisp pathname version component, but major pieces of Macintosh software ignore it, so it is normally zero.

**:ioFlFndrInfoFdType** Method of **tb:fileParam**<br>
**:set-ioFlFndrInfoFdType** *8b-integer* Method of **tb:fileParam**

This four-character string is the OS file type. This field is unrelated to the Lisp pathname type component.

**:ioFlFndrInfoCreator** Method of **tb:fileParam**<br>
**:set-ioFlFndrInfoCreator** *8b-integer* Method of **tb:fileParam**

This four-character string identifies the creator of the file. When you double-click in a file, the Finder™ uses this field to determine which application should be launched.

**:ioFlFndrInfoFdFlags** Method of **tb:fileParam**

These are flags used by the Finder. This field may be interrogated by using the constant masks defined below.

**tb:!fsHasBundle** [II-85] Constant<br>
**tb:!FInvisible** [II-85] Constant

These two constants are used as masks for **:ioFlFndrInfoFdFlags**. Their respective bits indicate that the associated file has a bundle and that the file Icon is invisible.

**:ioFlFndrInfoFdLocationV** Method of **tb:fileParam**<br>
**:set-ioFlFndrInfoFdLocationV** *16b-integer* Method of **tb:fileParam**<br>
**:ioFlFndrInfoFdLocationH** Method of **tb:fileParam**<br>
**:set-ioFlFndrInfoFdLocationH** *16b-integer* Method of **tb:fileParam**

These coordinates represent the point at which the file's icon is located. Initialize these values to 0 when creating a file.

**:ioFlFndrInfoFdFldr** Method of **tb:fileParam**

This is an integer indicating where the file's icon will appear. A positive number represents the folder with that number. Non-positive values have meanings indicated by the following constants.

**tb:!fTrash** [II-85] Constant
**tb:!fDesktop** [II-85] Constant
**tb:!fdisk** [II-85] Constant

These constants are used in the **:ioFlFndrInfoFdFldr** field to indicate that the icon is in the trash, on the desktop, or on disk respectively.

**:ioFlStBlk** Method of **tb:fileParam**
**:ioFlLgLen** Method of **tb:fileParam**
**:ioFlPyLen** Method of **tb:fileParam**

These integers represent the first allocation block of the data fork (16 bits) and its logical and physical EOF (32 bits).

**:ioFlRStBlk** Method of **tb:fileParam**
**:ioFlRLgLen** Method of **tb:fileParam**
**:ioFlRPyLen** Method of **tb:fileParam**

These integers represent the first allocation block of the resource fork (16 bits) and its logical and physical EOF (32 bits).

**:ioFlMdDat** Method of **tb:fileParam**
**:ioFlCrDat** Method of **tb:fileParam**

These 32-bit integers are the modification and creation dates of the file in universal time.

**:ioDirID** Method of **tb:fileParam**

This is the directory ID as a 32-bit integer.

**tb:volumeParam** [II-102] Flavor

This flavor is built upon **tb:ParamBlockRec** and defines a parameter block suitable for dealing directly with volumes. **tb:volumeParam** instances have the following instance accessor methods:

| | | |
|---|---|---|
| • :IOVOLINDEX | ;28 | [ integer ] |
| • :IOVCRDATE | ;30 | [ longint ] |
| • :IOVLSBKUP | ;34 | [ longint ] |
| • :IOVATRB | ;38 | [ integer ] |
| • :IOVNMFLS | ;40 | [ integer ] |
| • :IOVDIRST | ;42 | [ integer ] |
| • :IOVBLLN | ;44 | [ integer ] |
| • :IOVNMALBLKS | ;46 | [ integer ] |
| • :IOVALBLKSIZ | ;48 | [ longint ] |
| • :IOVCLPSIZ | ;52 | [ longint ] |

| | | |
|---|---|---|
| • :IOALBLST | ;56 | [ integer ] |
| • :IOVVNXTFNUM | ;58 | [ longint ] |
| • :IOVFRBLK | ;62 | [ integer ] |
| • :IOVSIGWORD | ;64 | [ integer ] |
| • :IOVDRVINFO | ;66 | [ integer ] |
| • :IOVDREFNUM | ;68 | [ integer ] |
| • :IOVFSID | ;70 | [ integer ] |
| • :IOVBKUP | ;72 | [ longint ] |
| • :IOVSEQNUM | ;76 | [ integer ] |
| • :IOVWRCNT | ;78 | [ longint ] |
| • :IOVFILCNT | ;82 | [ longint ] |
| • :IOVDIRCNT | ;86 | [ longint ] |
| • :IOVFNDRINFO1 | ;90 | [ longint ] |
| • :IOVFNDRINFO2 | ;94 | [ longint ] |
| • :IOVFNDRINFO3 | ;98 | [ longint ] |
| • :IOVFNDRINFO4 | ;102 | [ longint ] |
| • :IOVFNDRINFO5 | ;106 | [ longint ] |
| • :IOVFNDRINFO6 | ;110 | [ longint ] |
| • :IOVFNDRINFO7 | ;114 | [ longint ] |
| • :IOVFNDRINFO8 | ;118 | [ longint ] |

### tb:CInfoPBRec Flavor

This flavor is built upon **tb:ParamBlockRec** and defines a parameter block for use with **tb:!GetCatInfo** and **tb:!SetCatInfo** to get and set information about files and directories within a directory. Notice that there are several pairs of instance accessor methods which access the same field of the block. This is because **tb:!GetCatInfo** and **tb:!SetCatInfo** work with either files or directories. One set of instance accessors is for file information and the other set is for directory information. **tb:CInfoPBRec** instances have the following instance accessor methods:

| | | |
|---|---|---|
| • :IOFVERSNUM | ;26 | [ signedbyte ] |
| • :FILLER1 | ;27 | [ signedbyte ] |
| • :IOFDIRINDEX | ;28 | [ integer ] |
| • :IOFLATTRIB | ;30 | [ signedbyte ] |
| • :FILLER2 | ;31 | [ signedbyte ] |
| • :IOFLFNDRINFOFDTYPE | ;32 | [ ostype ] |
| • :IODRUSRWDSFRRECTTOP | ;32 | [ integer ] |
| • :IODRUSRWDSFRRECTLEFT | ;34 | [ integer ] |
| • :IOFLFNDRINFOFDCREATOR | ;36 | [ ostype ] |
| • :IODRUSRWDSFRRECTBOTTOM | ;36 | [ integer ] |
| • :IODRUSRWDSFRRECTRIGHT | ;38 | [ integer ] |
| • :IOFLFNDRINFOFDFLAGS | ;40 | [ integer ] |
| • :IODRUSRWDSFRFLAGS | ;40 | [ integer ] |
| • :IOFLFNDRINFOFDLOCATIONV | ;42 | [ integer ] |
| • :IODRUSRWDSFRLOCATIONV | ;42 | [ integer ] |
| • :IOFLFNDRINFOFDLOCATIONH | ;44 | [ integer ] |
| • :IODRUSRWDSFRLOCATIONH | ;44 | [ integer ] |
| • :IOFLFNDRINFOFDFLDR | ;46 | [ integer ] |
| • :IODRUSRWDSFRVIEW | ;46 | [ integer ] |
| • :IODIRID | ;48 | [ longint ] |
| • :IODRDIRID | ;48 | [ longint ] |

| | | | |
|---|---|---|---|
| • | :IOFLSTBLK | ;52 | [ integer ] |
| • | :IODRNMFLS | ;52 | [ integer ] |
| • | :IOFLLGLEN | ;54 | [ longint ] |
| • | :IOFLPYLEN | ;58 | [ longint ] |
| • | :IOFLRSTBLK | ;62 | [ integer ] |
| • | :IOFLRLGLEN | ;64 | [ longint ] |
| • | :IOFLRPYLEN | ;68 | [ longint ] |
| • | :IOFLCRDAT | ;72 | [ longint ] |
| • | :IODRCRDAT | ;72 | [ longint ] |
| • | :IOFLMDDAT | ;76 | [ longint ] |
| • | :IODRMDDAT | ;76 | [ longint ] |
| • | :IOFLBKDAT | ;80 | [ longint ] |
| • | :IODRBKDAT | ;80 | [ longint ] |
| • | :IOFLXFNDRINFOFDICONID | ;84 | [ integer ] |
| • | :IODRFNDRINFOFRSCROLLV | ;84 | [ integer ] |
| • | :IOFLXFNDRINFOFDUNUSED1 | ;86 | [ integer ] |
| • | :IODRFNDRINFOFRSCROLLH | ;86 | [ integer ] |
| • | :IOFLXFNDRINFOFDUNUSED2 | ;88 | [ integer ] |
| • | :IODRFNDRINFOFROPENCHAIN | ;88 | [ longint ] |
| • | :IOFLXFNDRINFOFDUNUSED3 | ;90 | [ integer ] |
| • | :IOFLXFNDRINFOFDUNUSED4 | ;92 | [ integer ] |
| • | :IODRFNDRINFOFRUNUSED | ;92 | [ integer ] |
| • | :IOFLXFNDRINFOFDCOMMENT | ;94 | [ integer ] |
| • | :IODRFNDRINFOFRCOMMENT | ;94 | [ integer ] |
| • | :IOFLXFNDRINFOFDPUTAWAY | ;96 | [ longint ] |
| • | :IODRFNDRINFOFRPUTAWAY | ;96 | [ longint ] |
| • | :IOFLPARID | ;100 | [ longint ] |
| • | :IODRPARID | ;100 | [ longint ] |
| • | :IOFLCLPSIZ | ;104 | [ longint ] |

This flavor is built upon **tb:ParamBlockRec** and defines a parameter block which is used with the trap **tb:!CatMove**, which is used to move files from one directory to another. **tb:CMovePBRec** instances have the following instance accessor methods:

| | | | |
|---|---|---|---|
| • | :FILLER1 | ;24 | [ longint ] |
| • | :IONEWNAME | ;28 | [ pointer ] |
| • | :FILLER2 | ;32 | [ longint ] |
| • | :IONEWDIRID | ;36 | [ longint ] |
| • | :FILLER3-1 | ;40 | [ longint ] |
| • | :FILLER3-2 | ;44 | [ longint ] |
| • | :IODIRID | ;48 | [ longint ] |

**tb:WDPBRec** Flavor

This flavor is built upon **tb:ParamBlockRec** and defines a parameter block which is used with traps that deal specifically with working directories. **tb:WDPBRec** instances have the following instance accessor methods:

| | | | |
|---|---|---|---|
| • | :IOVREFNUM | ;22 | [ integer ] |
| • | :FILLER1 | ;24 | [ integer ] |
| • | :IOWDINDEX | ;26 | [ integer ] |

| | | |
|---|---|---|
| :IOWDPROCID | ;28 | [ longint ] |
| :IOWDVREFNUM | ;32 | [ integer ] |
| :FILLER2-1 | ;34 | [ integer ] |
| :FILLER2-2 | ;36 | [ integer ] |
| :FILLER2-3 | ;38 | [ integer ] |
| :FILLER2-4 | ;40 | [ integer ] |
| :FILLER2-5 | ;42 | [ integer ] |
| :FILLER2-6 | ;44 | [ integer ] |
| :FILLER2-7 | ;46 | [ integer ] |
| :IOWDDIRID | ;48 | [ longint ] |

### tb:FCBPBRec Flavor

This flavor is built upon **tb:ParamBlockRec** and defines a parameter block which is used with the trap **tb:!GetFCBInfo**. **tb:FCBPBRec** instances have the following instance accessor methods:

| | | |
|---|---|---|
| :IOREFNUM | ;24 | [ integer ] |
| :FILLER | ;26 | [ integer ] |
| :IOFCBINDX | ;28 | [ longint ] |
| :IOFCBFLNM | ;32 | [ longint ] |
| :IOFCBFLAGS | ;36 | [ integer ] |
| :IOFCBSTBLK | ;38 | [ integer ] |
| :IOFCBEOF | ;40 | [ longint ] |
| :IOFCBPLEN | ;44 | [ longint ] |
| :IOFCBCRPS | ;48 | [ longint ] |
| :IOFCBVREFNUM | ;52 | [ integer ] |
| :IOFCBCLPSIZ | ;54 | [ longint ] |
| :IOFCBPARID | ;58 | [ longint ] |

The uses of each field of each parameter block are discussed in the traps that use the block. For simple file I/O, you will normally be interested in the following seven fields in an ioParam parameter block:

**:ioNamePtr** - A pointer to a string that contains the name of a file.
**:ioVRefNum** - The volume on which the file resides.
**:ioRefNum** - The reference number of the open file.
**:ioBuffer** - A buffer in memory which is used for read and write operations.
**:ioPosMode** - The positioning mode in the file which determines from where the read/write operation is to start.
**:ioPosOffset** - The offset in the file of the read/write operation. (This is related to **:ioPosMode**.)
**:ioReqCount** - The number of bytes to read/write.

## Initializing the File I/O Queue

21.2 The following routine initializes the File Manager.

### tb:!InitQueue [II-103] Function

Clears the File Manager queue of all calls except the current one.

## Accessing Volumes

21.3 These routines provide access to volumes.

**tb:!MountVol** *volumeParam* [IV-128] Function

Mounts the volume in the drive specified by **:ioVRefNum.** If no volumes are already mounted, this one becomes the default volume.

*Example:*

```
;;; Mount a volume in drive number drvNum and
;;;return the drive's reference number.
(defun mount-volume (drvNum)
  (declare (values drive-refNum))
  (let ((paramBlock (make-instance 'tb:volumeParam)))
    (send paramBlock :ioVRefNum drvNum)
    (tb:!MountVol paramBlock))
    (prog1 (send paramBlock :ioVRefNum)
           (send paramBlock :dispose))))
```

**tb:!GetVolInfo** *volumeParam* [IV-129] Function
**tb:!HGetVinfo** *volumeParam* [IV-130] Function

**tb:!GetVolInfo** returns information about the volume specified by **:ioVolIndex, ::ioVRefNum,** and **:ioNamePtr.** If **:ioVolIndex** is positive, the File Manager attempts to use it to find the volume. For instance, if::**ioVolIndex** is 2, the File Manager will look for the second mounted volume. If **:ioVolIndex** is negative, the File Manager uses **:ioNamePtr** and **:ioVRefNum** to find the volume. If **:ioVolIndex** is 0, the File Manager uses **:ioVRefNum** only.

**tb:!HGetVinfo** is similar except it returns more information.

*Example:*

```
;;; Return a volumeParam with info about the volume named name.
(defun get-volume-info (theName)
  (declare (values tb:volumeParam))
  (let ((paramBlock     (make-instance 'tb:volumeParam))
        (theNameHandle (tb:!NewString theName))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (tb:!GetVolInfo paramBlock)
    paramBlock))
```

**tb:!SetVolInfo** *volumeParam* [IV-131] Function

Modifies information about the volume specified by **:ioVRefNum.** Precede this trap by a call to **tb:!HGetVInfo** to fill in the fields of the **tb:volumeParam,** then change the fields you want modified. Finally, call this trap to write out the modifications.

**tb:!GetVol** *volumeParam* [IV-131] Function

Returns the default volume's reference number and name. The default volume's reference number is returned in **:ioVRefNum.** A pointer to the default volume's name is returned in **:ioNamePtr** if **:ioNamePtr** is not **tb:!nilPtr.** If a default directory was set, a pointer to its name and its working directory number will be returned in **:ioVRefNum** and **:ioNamePtr.**

*Example:*

```
;;; Return a tb:volumeParam block with the default volume's
;;; reference number and a pointer to its name.
(defun get-volume ()
  (declare (values tb:volumeParam))
  (let ((paramBlock (make-instance 'tb:volumeParam)))
    (tb:!GetVol paramBlock))
    paramBlock))
```

**tb:!HGetVol** *WDPBRec* [IV-132] Function

Returns the default volume and default directory last set by either **tb:!SetVol** or **tb:!HSetVol.** The volume reference number of the default volume will be returned in **:ioVRefNum.** The volume reference number on which the default directory exists is returned in **:ioWDVRefNum.** The directory ID of the default directory is returned in **:ioWDDirID.**

*Example:*

```
;;; Return a tb:WDPBRec with information about the default volume
;;; and the default directory.
(defun h-get-volume ()
  (declare (values tb:WDPBRec))
  (let ((paramBlock (make-instance 'tb:WDPBRec)))
    (tb:!HGetVol paramBlock))
    paramBlock))
```

**tb:!SetVol** *volumeParam* [IV-132] Function

Sets the default volume to the mounted volume specified by **:ioNamePtr** and **:ioVRefNum.** This also sets the root of the volume as the default directory.

*Example:*

```
;;; Make a volume the default volume.
(defun set-volume (theName VRefNum)
  (declare (values OSErr))
  (let ((paramBlock   (make-instance 'tb:volumeParam))
        (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (prog1 (tb:!SetVol paramBlock)          ; return OSErr
           (tb:!DisposHandle theNameHandle)
           (send paramBlock :dispose))))
```

**tb:!HSetVol** *WDPBRec* [IV-133] Function

Sets both the default volume and default directory, which are specified by **:ioNamePtr**, **:ioVRefNum**, and **:ioWDDirID**.

*Example:*

```
;;; Make a volume the default volume and a directory the
;;;  default directory
(defun h-set-vol (theName VRefNum WDDirID)
  (declare (values tb:WDPBRec))
  (let ((paramBlock   (make-instance 'tb:WDPBRec))
        (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (send paramBlock :ioWDDirID WDDirID)
    (prog1 (tb:!HSetVol paramBlock)          ; return OSErr
           (tb:!DisposHandle theNameHandle)
           (send paramBlock :dispose))))
```

**tb:!FlushVol** *volumeParam* [IV-133] Function

Flushes the volume (writes descriptive information, the volume buffer, and all access path buffers) specified by **:ioNamePtr** and **:ioVRefNum.**

*Example:*

```
;;; Example of flushing a volume.
(defun flush-volume (theName VRefNum)
  (declare (values OSErr))
  (let ((paramBlock   (make-instance 'tb:volumeParam))
        (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (prog1 (tb:!FlushVol paramBlock)         ; return OSErr
           (tb:!DisposHandle theNameHandle)
           (send paramBlock :dispose))))
```

**tb:!UnmountVol** *volumeParam* [IV-134] Function

Unmounts the volume specified by **:ioNamePtr** or **:ioVRefNum** by flushing it, closing all open files on the volume, and releasing the memory used by the volume.

CAUTION: Do not unmount the startup volume.

*Example:*

```
(defun unmount-volume (theName VRefNum)
  (declare (values OSErr))
  (let ((paramBlock   (make-instance 'tb:volumeParam))
        (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (prog1 (tb:!UnmountVol paramBlock)       ; return OSErr
           (tb:!DisposHandle theNameHandle)
           (send paramBlock :dispose))))
```

**tb:!OffLine** *volumeParam* [IV-134] Function

Places the volume specified by **:ioNamePtr** or **:ioVRefNum** off-line by calling PBFlushVol and releasing the memory used by the volume.

*Example:*

```
(defun flush-volume (theName VRefNum)
  (declare (values OSErr))
  (let ((paramBlock   (make-instance 'tb:volumeParam))
       (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
         (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (progl (tb:!OffLine paramBlock))        ; return OSErr
          (tb:!DisposHandle theNameHandle)
          (send paramBlock :dispose))))
```

**tb:!Eject** *volumeParam* [IV-135] Function

Flushes the volume specified by **:ioNamePtr** or **:ioVRefNum**, places it off-line and ejects the volume.

*Example:*

```
(defun eject (theName VRefNum)
  (declare (values OSErr))
  (let ((paramBlock   (make-instance 'tb:volumeParam))
       (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
         (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (progl (tb:!Eject paramBlock))          ; return OSErr
          (tb:!DisposHandle theNameHandle)
          (send paramBlock :dispose))))
```

## Accessing Files

21.4 These routines are for creating, modifying, and deleting files.

**tb:!Create** *fileParam* [II-107] Function
**tb:!HCreate** *fileParam* [IV-146] Function

**tb:!Create** creates a file with the name specified in the **:ioNamePtr** instance variable on the volume specified by **:ioVRefNum**. **tb:!HCreate** is similar except it allows directory ID to be specified in **:ioDirID**.

*Example:*

```
;;; Create a file named theName on the volume VRefNum and return
;;; the File Manager result code.
(defun create-file (theName VRefNum)
  (declare (values OSErr)
  (let ((paramBlock   (make-instance 'tb:fileParam))
       (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
         (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (progl (tb:!Create paramBlock))         ; return OSErr
          (tb:!DisposHandle theNameHandle)
          (send paramBlock :dispose))))
```

```
;;; Create a file named theName on the volume VRefNum in the
;;; directory DirID and return the File Manager result code.
(defun h-create-file (theName VRefNum DirID)
  (declare (values OSErr))
  (let ((paramBlock    (make-instance 'tb:fileParam))
        (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (send paramBlock :ioDirID DirID)    ; specify directory ID
    (prog1 (tb:!Create paramBlock))     ; return OSErr
           (tb:!DisposHandle theNameHandle)
           (send paramBlock :dispose))))
```

**tb:!DirCreate** *fileParam* [IV-146] Function

Creates a new directory with name specified by **:ioNamePtr** on the volume specified by **:ioVRefNum.** The parent of the new directory is specified in **:ioDirID** if it is 0, the new directory will be placed in the root directory. The directory ID of the new directory is returned in **:ioDirID.**

*Example:*

```
;;; Create a directory named theName on the volume VRefNum in the
;;; directory DirID and return the new directory's directory ID.
(defun create-directory (theName VRefNum DirID)
  (declare (values ioDirID))
  (let ((paramBlock    (make-instance 'tb:fileParam))
        (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (send paramBlock :ioDirID DirID)
    (tb:!DirCreate paramBlock))
    (tb:!DisposHandle theNameHandle)
    (prog1 (send paramBlock :ioDirID)         ; return ioDirID
           (send paramBlock :dispose))))
```

**tb:!Open** *ioParam* [II-108] Function
**tb:!HOpen** *ioParam* [IV-136] Function

**tb:!Open** opens an access path to the file specified in :io*NamePtr*, on the volume **:ioVRefNum.** The path reference number is returned in **:ioRefNum.** The field **:ioPermssn** specifies what access permission is to be allowed. (See **:ioPermssn** method of **tb:ioParam.**)

The field **:ioMisc** specifies the access path buffer. If **:ioMisc** is **tb:**nilPtr, the normal case, the File Manager allocates the buffer itself.

**tb:!HOpen** is similar except it allows a directory ID to be specified in **:ioDirID.**

*Example:*

```
;;; Open a file named theName on the volume VRefNum and return
;;; its refNum.
(defun open-file (theName VRefNum)
  (declare (values ioRefNum))
  (let ((paramBlock   (make-instance 'tb:ioParam))
        (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (tb:!Open paramBlock)
    (tb:!DisposHandle theNameHandle)
    (prog1 (send paramBlock :ioRefNum)    ; return ioRefNum
           (send paramBlock :dispose))))

;;; Open a file named theName on the volume VRefNum in the
;;; directory DirID and return its refNum.
(defun h-open-file (theName VRefNum DirID)
  (declare (values ioRefNum))
  (let ((paramBlock   (make-instance 'tb:ioParam))
        (theNewHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (send paramBlock :ioDirID DirID)  ; specify directory ID
    (tb:!Open paramBlock)
    (tb:!DisposHandle theNameHandle)
    (prog1 (send paramBlock :ioRefNum)    ; return ioRefNum
           (send paramBlock :dispose))))
```

**tb:!OpenRF** *fileParam* [II-109] Function
**tb:!HOpenRF** *fileParam* [II-109] Function

**tb:!OpenRF** is the same as **tb:!Open** except it opens the resource fork instead of the data fork. This trap should not be used for opening resource files. Use the Resource Manager trap **tb:!OpenResFile** instead.

**tb:!HOpenRF** is similar except it allows a directory ID to be specified in **:ioDirID**.

**CAUTION: Do not put anything except resources in the resource fork or you may risk causeing the File Manager to die.**

**tb:!Read** *ioParam* [II-110] Function

Tries to read **:ioReqCount** bytes from the file whose access path reference number is **:ioRefNum**, and puts them in a buffer pointed to by **:ioBuffer**. The starting position of the operation is specified by **:ioPosMode** and **:ioPosOffset**. The values for **:ioPosModes** are defined with that method.

The number of bytes actually read is returned in **:ioActCount**. The position of the mark at the end of the read is returned in **:ioPosMode.**

To read from a file with a reference number **myRefNum** into a handle, do the following:

*Example:*

```
;;; Read from a file into a handle and return the number of bytes
;;; actually read.
(defun read-handle (myRefNum myBufferHandle myReqCount
  (declare (values ioActCount))
  (let ((paramBlock   (make-instance 'tb:ioParam)))
    (tb:!SetHandleSize myBufferHandle myReqCount)
    (tb:!HLock myBufferHandle)
    (send paramBlock :ioRefNum myRefNum)
    (send paramBlock :ioBuffer
          (tb:deref myBufferHandle))
    (send paramBlock :ioReqCount myReqCount)
    (tb:!Read paramBlock)
    (tb:!HUnlock myBufferHandle)
    (prog1 (send paramBlock :ioActCount)
           (send paramBlock :dispose))))
```

**tb:!Write** *ioParam* [II-110] Function

Tries to write **:ioReqCount** bytes from the buffer pointed to by **:ioBuffer**, and puts them in a file whose access path reference number is **:ioRefNum**. The starting position of the operation is specified by **:ioPosMode** and **:ioPosOffset**. The number of bytes actually written is returned in **:ioActCount**. The position of the mark at the end of the write is returned in **:ioPosMode**.

To write the contents of a buffer whose handle is **myBufferHandle** onto the end of a file with a refNum **myRefNum**, do the following:

*Example:*

```
;;; Write to a file from a handle and return the number of bytes
;;; actually written.
(defun write-handle (myRefNum myBufferHandle)
  (declare (values ioActCount))
  (let ((paramBlock   (make-instance 'tb:ioParam))
        (myBufferSize (tb:!GetHandleSize
                        myBufferHandle)))
    (tb:!HLock myBufferHandle)
    (send paramBlock :ioRefNum myRefNum)
    (send paramBlock :ioBuffer
          (tb:deref myBufferHandle))
    (send paramBlock :ioReqCount myBufferSize)
     ;; Write relative to the end of the file.
    (send paramBlock :ioPosMode tb:!fsFromLEOF)
    (tb:!Write paramBlock)
    (tb:!HUnlock myBufferHandle)
    (prog1 (send paramBlock :ioActCount)
           (send paramBlock :dispose))))
```

**tb:!GetFPos** *ioParam* [II-111] Function

Returns the position of the mark of the file with a reference number **:ioRefNum** in **:ioPosOffset**. To get the mark of the file with a reference number **myRefNum**, do the following:

*Example:*

```
;;; Return the mark of the file myRefNum.
(defun get-file-pos (myRefNum)
  (declare (values ioPosOffset))
  (let ((paramBlock (make-instance 'tb:ioParam)))
    (send paramBlock :ioRefNum myRefNum)
    (tb:!GetFPos paramBlock)
    (prog1 (send paramBlock :ioPosOffset)
           (send paramBlock :dispose))))
```

**tb:!SetFPos** *ioParam* [II-111] Function

Sets the position of the mark of the file with a reference number **:ioRefNum** to the position specified by **:ioPosMode** and **:ioPosOffset.**

*Example:*

```
;;; Set the mark of file myRefNum to fPos.
(defun set-file-pos (myRefNum fPos)
  (declare (values ignore))
  (let ((paramBlock (make-instance 'tb:ioParam)))
    (send paramBlock :ioRefNum myRefNum)
    (send paramBlock :ioPosMode tb:!fsFromStart)
    (send paramBlock :ioPosOffset fPos)
    (prog1 (tb:!SetFPos paramBlock)
           (send paramBlock :dispose))))
```

**tb:!GetEOF** *ioParam* [II-112] Function

Returns in **:ioMisc** the logical EOF of the file with a reference number **:ioRefNum.**

*Example:*

```
(defun get-eof (myRefNum)
  (declare (values logical-EOF))
  (let ((paramBlock (make-instance 'tb:ioParam)))
    (send paramBlock :ioRefNum myRefNum)
    (tb:!GetEOF paramBlock)
    ;; Since :ioMisc is a pointer, convert it to a number.
    (prog1 (send (send paramBlock :ioMisc) :pointer)
           (send paramBlock :dispose))))
```

**tb:!SetEOF** *ioParam* [II-112] Function

Sets the logical EOF of the file with a reference number **:ioRefNum** to the value in **:ioMisc.**

*Example:*

```
(defun set-eof (myRefNum myEOF)
  (declare (values logical-EOF))
  (let ((paramBlock (make-instance 'tb:ioParam)))
    (send paramBlock :ioRefNum myRefNum)
    ;; Convert myEOF to a pointer for ioMisc.
    (send paramBlock :ioMisc
          (make-instance 'tb:mac-pointer
                         :pointer myEOF))
    (prog1 (tb:!GetEOF paramBlock)
           (send paramBlock :dispose))))
```

**tb:!Allocate** *ioParam* [II-113] Function

Adds **:ioReqCount** bytes to the file with a reference number **:ioRefNum**, and sets the physical EOF to one byte beyond the last block allocated. To add **myAllocSize** bytes to the file with an **:ioRefNum** of **myRefNum**, do the following:

*Example:*

```
(defun allocate (myRefNum myAllocSize)
  (declare (values ioActCount))
  (let ((paramBlock (make-instance 'tb:ioParam)))
    (send paramBlock :ioRefNum myRefNum)
    (send paramBlock :ioReqCount myAllocSize)
    (tb:!Allocate paramBlock)
    (prog1 (send paramBlock :ioActCount)
           (send paramBlock :dispose))))
```

**tb:!FlushFile** *ioParam* [II-114] Function

Writes the access path buffer of the file with a reference number **:ioRefNum** to the volume.

*Example:*

```
(defun flush-file (myRefNum)
  (declare (values ignore))
  (let ((paramBlock (make-instance 'tb:ioParam)))
    (send paramBlock :ioRefNum myRefNum)
    (tb:!FlushFile paramBlock)
    (send paramBlock :dispose))
```

**tb:!Close** *ioParam* [II-114] Function

Closes the file with a reference number **:ioRefNum** and disposes of the access path. To close the file with a reference number **myRefNum**, do the following:

*Example:*

```
(defun close-file (myRefNum)
  (declare (values ignore))
  (let ((paramBlock (make-instance 'tb:ioParam)))
    (send paramBlock :ioRefNum myRefNum)
    (tb:!Close paramBlock)
    (send paramBlock :dispose))
```

## Changing Information About Files

21.5 These traps set and return information in files and affect various aspects of the file itself (its name, version number, locked status, etc.).

**tb:!GetFileInfo** *fileParam* [II-115] Function

**tb:!GetFileInfo** returns file information about the specified file on a volume specified by the volume reference number **:ioVRefNum**. The file can be specified by two methods: by index in **:ioDirIndex** or by name in **:ioNamePtr**. See *Inside Macintosh* page II-115 for the fields the trap returns, and II-101 for the contents of the returned fields.

**tb:!GetFileInfo** is most often used to set up a file info parameter block to use with the trap **tb:!SetFileInfo**. See the example for **tb:!SetFileInfo**.

**tb:!HGetFInfo** is similar except it allows a directory ID to be specified in **:ioDirID**.

*Example:*

```
;; Return a fileParam block containing info about the file theName.
(defun get-file-info (theName VRefNum)
  (declare (values fileParam))
  (let ((paramBlock     (make-instance 'tb:fileParam))
        (theNameHandle  (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (tb:!GetFileInfo paramBlock))
    paramBlock))
```

**tb:!SetFileInfo** *fileParam* [II-116] Function

**tb:!SetFileInfo** sets the file information of a specified file on a volume specified by the volume reference number **:ioVRefNum**. The file can be specified in two manners: by index in **:ioDirIndex** or by name in **:ioNamePtr**. See *Inside Macintosh* page II-116 for the fields the trap requires, and II-101 for the contents of the passed fields.

This trap is most often used to set the file type and creator of a newly created file. First, set up the parameter block using the trap **tb:!GetFileInfo**. Then, set the **:iofdType** and **:iofdCreator** field to the desired values. Finally, pass the parameter block to **tb:!SetFileInfo**.

**tb:!HSetFInfo** is similar except it allows a directory ID to be specified in **:ioDirID**.

*Example:*

```
;;; Set a file's type and creator.
(defun set-file-info (theName VRefNum theType
                              theCreator)
  (declare (values ignore))
  (let ((paramBlock (tb:get-file-info theName VRefNum)))
    (send paramBlock :ioFlFndrInfofdType theType)
    (send paramBlock :ioFlFndrInfofdCreator theCreator)
    (tb:!SetFileInfo paramBlock)
    (send paramBlock :dispose)))
```

**tb:!SetFilLock** *fileParam* [II-116] Function
**tb:!RstFilLock** *fileParam* [II-117] Function

**tb:!SetFilLock** and **b:!RstFilLock** lock and unlock respectively the file specified by **:ioNamePtr** on the volume which has a volume reference number **:ioVRefNum**.

**tb:!HSetFLock** and **tb:!RstFilLock** are similar except they allow a directory ID to be specified in **:ioDirID**.

*Example:*

```
(defun lock-file (theName VRefNum)
  (declare (values OSErr))
  (let ((paramBlock    (make-instance 'tb:fileParam))
        (theNameHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (prog1 (tb:!SetFilLock paramBlock))     ; return OSErr
           (tb:!DisposHandle theNameHandle)
           (send paramBlock :dispose))))

(defun unlock-file (theName VRefNum)
  (declare (values OSErr))
  (let ((paramBlock    (make-instance 'tb:fileParam))
        (theNameHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (prog1 (tb:!RstFilLock paramBlock))     ; return OSErr
           (tb:!DisposHandle theNameHandle)
           (send paramBlock :dispose))))
```

**tb:!SetFilType** *:ioParam* [II-117] Function

Sets the version number of the file specified by **:ioNamePtr** on the volume with a volume reference number **:ioVRefNum**.

NOTE: Using the file version number is not a good idea. The Resource Manager, the Segment Loader, and the Standard File Package will only work with files whose version number is 0. Changing the version number, or using anything but the default version number 0, can create some very insidious bugs.

**tb:!Rename** *ioParam* [II-118] Function

Renames the file specified by **:ioNamePtr** on the volume with a volume reference number **:ioVRefNum** to the name pointed to by **:ioMisc**.

*Example:*

```
;;; Change a file name from theName to newName.
(defun rename-file (theName newName VRefNum)
  (declare (values OSErr))
  (let ((paramBlock    (make-instance 'tb:ioParam))
        (theNameHandle (tb:!NewString theName))
        (newNameHandle (tb:!NewString newName)))
    (tb:!HLock theNameHandle)
    (tb:!HLock newNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (send paramBlock :ioMisc (tb:deref newNameHandle))
    (prog1 (tb:!Rename paramBlock))       ; return OSErr
           (tb:!DisposHandle theNameHandle)
           (tb:!DisposHandle newNameHandle)
           (send paramBlock :dispose))))
```

**tb:!Delete** *fileParam* [II-119] Function

**tb:!Delete** deletes the file specified by **:ioNamePtr** on the volume with a volume reference number **:ioVRefNum.**

**tb:!HDelete** is similar except it allows a directory ID to be specified in **:ioDirID**. **tb:!HDelete** can be used to delete empty directories as well as files. To delete the file named **theName** on the volume with a volume reference number **VRefNum**, do the following:

*Example:*

```
;;; Delete the file named theName.
(defun delete-file (theName VRefNum)
  (let ((paramBlock     (make-instance 'tb:fileParam))
        (theNameHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (progl (tb:!Delete paramBlock))           ; return OSErr
           (tb:!DisposHandle theNameHandle)
           (send paramBlock :dispose))))
```

## Hierarchical Directory Routines

21.6 These routines are for examining and changing information about directories.

**tb:!GetCatInfo** *CInfoPBRec* [IV-155] Function

Gets information about the files and directories in a file catalog. See *Inside Macintosh* for details.

**tb:!SetCatInfo** *CInfoPBRec* [IV-155] Function

Sets information about the files and directories in a file catalog. See *Inside Macintosh* for details.

**tb:!CatMove** *CMovePBRec* [IV-155] Function

Moves files or directories from one directory or another. The name of the file or directory to be moved is specified by **:ioNamePtr.** **:ioVRefNum** specifies either the volume reference number or working directory reference number which contains the file or directory to be moved. The name and directory ID of the directory to which the file or directory is to be moved is specified by **:ioNewName** and **:ioNewDirID.**

*Example:*

```
;;; Move a file or directory named theName from directory oldDirID
;;; into the directory named newName with ID newID.
(defun move-directory
       (theName oldDirID newName newDirID VRefNum)
  (declare (values ignore))
  (let ((paramBlock    (make-instance 'tb:CMovePBRec))
        (theNameHandle (tb:!NewString theName))
        (newNameHandle (tb:!NewString newName)))
    (tb:!HLock theNameHandle)
    (tb:!HLock newNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (send paramBlock :ioNewName
          (tb:deref newNameHandle))
    (send paramBlock :ioNewDirID newDirID)
    (send paramBlock :ioDirID oldDirID)
    (progl  (tb:!CatMove paramBlock))     ; return OSErr
           (tb:!DisposHandle theNameHandle)
           (tb:!DisposHandle newNameHandle)
           (send paramBlock :dispose))))
```

## Working Directory Routines

21.7 These routines open and close working directories.

**tb:!OpenWD** *WDPBRec* [IV-155] Function

Takes the directory specified by **:ioVRefNum**, **:ioWDDirID**, and **:ioWDProcID** and makes it a working directory. It returns a working directory reference number in **:ioVRefNum**.

*Example:*

```
;;; Make directory specified by theName, VRefNum, directory ID
;;; WDDirID, and ioWDProcID a working directory.  The working
;;; directory reference number is returned.
(defun open-wd (theName VRefNum WDDirID WDProcID)
  (declare (values iovrefNum))
  (let ((paramBlock    (make-instance 'tb:WDPBRec))
        (theNameHandle (tb:!NewString theName)))
    (tb:!HLock theNameHandle)
    (send paramBlock :ioNamePtr
          (tb:deref theNameHandle))
    (send paramBlock :ioVRefNum VRefNum)
    (send paramBlock :ioWDProcID WDProcID)
    (send paramBlock :ioWDDirID WDDirID)
    (tb:!OpenWD paramBlock))
    (progl (send paramBlock :ioVRefNum)
           (tb:!DisposHandle theNameHandle)
           (send paramBlock :dispose))))
```

**tb:!CloseWD** *WDPBRec* [IV-155] Function

Releases the working directory whose working directory reference number is specified by **:ioVRefNum**.

*Example:*

```
;;; Close working directory WDRefNum.
(defun close-wd (WDRefNum)
  (let ((paramBlock (make-instance 'tb:WDPBRec))
    (send paramBlock :ioVRefNum WDRefNum)
    (prog1 (tb:!CloseWD paramBlock)
           (send paramBlock :dispose))))
```

**tb:!GetWDInfo** *WDPBRec* [IV-155] Function

Returns information about a working directory. The working directory is specified by **:ioVRefNum** and **:ioWDIndex**. If **:ioWDIndex** is 0, the **:ioVRefNum** is assumed to contain the working directory reference number. Otherwise, **:ioWDIndex** should contain the index number of the directory. In this case, if **:ioVRefNum** is not 0, it is assumed to be a volume specification and only directories on that volume will be indexed. If **:ioWDProcID** is not 0, only working directories with that ID are indexed; otherwise all working directories are indexed.

**:ioWDVRefNum** always returns the volume reference number. **:ioVRefNum** returns a working directory reference number if a working directory reference number is passed in that field; otherwise it returns a volume reference number. The volume name is returned in **:ioNamePtr**.

## Accessing Queues

21.8 The following are very low-level traps. You will never need to use them and they can be very dangerous.

**tb:!GetFSQHdr** [II-125] Function

Returns a pointer to the header of the File I/O Queue.

**tb:!GetVCBQHdr** [II-126] Function

Returns a pointer to the header of the Volume Control Block Queue.

**tb:!GetDrvQHdr** [II-128] Function

Returns a pointer to the header of the Drive Queue.

## File Control Blocks

21.9 THis is a very low-level traps. You should never need to use. It can be very dangerous.

**tb:!GetFCBInfo** *FCBPBRec* [IV-179] Function

Returns information about the specified open file. See *Inside Macintosh* for details.

# Chapter 22
# PRINTING MANAGER

## Introduction

22.1 The Printing Manager is used to print files with a printer, usually the ImageWriter®.

## Initialization and Termination

22.2 These routines open and close the Printing Manager.

### tb:!PrOpen

[II-157] Function

Prepares the Printing Manager for use.

### tb:!PrClose

[II-157] Function

Shuts down the Printing Manager and releases any memory it uses.

## Print Records and Dialogs

22.3 These routines control print records and dialog boxes.

### tb:THPrint

[II-149] Flavor

This flavor defines a print record with the following instance accessor methods:

| | |
|---|---|
| • :IPRVERSION | [ integer ] |
| • :IDEV | [ integer ] |
| • :IVRES | [ integer ] |
| • :IHRES | [ integer ] |
| • :RPAGETOP | [ integer ] |
| • :RPAGELEFT | [ integer ] |
| • :RPAGEBOTTOM | [ integer ] |
| • :RPAGERIGHT | [ integer ] |
| • :RPAPERTOP | [ integer ] |
| • :RPAPERLEFT | [ integer ] |
| • :RPAPERBOTTOM | [ integer ] |
| • :RPAPERRIGHT | [ integer ] |
| • :WDEV | [ integer ] |
| • :IPAGEV | [ integer ] |
| • :IPAGEH | [ integer ] |
| • :BPORT | [ byte ] |
| • :FEED | [ byte ] |
| • :PRINFOPT1 | [ integer ] |
| • :PRINFOPT2 | [ integer ] |
| • :PRINFOPT3 | [ integer ] |
| • :PRINFOPT4 | [ integer ] |
| • :PRINFOPT5 | [ integer ] |

| | |
|---|---|
| • :PRINFOPT6 | [ integer ] |
| • :PRINFOPT7 | [ integer ] |
| • :IROWBYTES | [ integer ] |
| • :IBANDV | [ integer ] |
| • :IBANDH | [ integer ] |
| • :IDEVBYTES | [ integer ] |
| • :IBANDS | [ integer ] |
| • :BPATSCALE | [ byte ] |
| • :BULTHICK | [ byte ] |
| • :BULOFFSET | [ byte ] |
| • :SCAN | [ integer ] |
| • :BXINFOX | [ byte ] |
| • :IFSTPAGE | [ integer ] |
| • :ILSTPAGE | [ integer ] |
| • :ICOPIES | [ integer ] |
| • :BJDOCLOOP | [ byte ] |
| • :FFROMAPP | [ byte ] |
| • :PIDLEPROC | [ pointer ] |
| • :PFILENAME | [ pointer ] |
| • :IFILEVOL | [ integer ] |
| • :BFILEVERS | [ byte ] |

### **tb:!PrintDefault** *hPrint* [II-158] Function

Sets the fields of *hPrint* to the default values kept in the printer resource file.

*Example:*

```
(setf hPrint (make-instance 'tb:THPrint))
(tb:!PrintDefault hPrint)
```

### **tb:!PrValidate** *hPrint* [II-158] Function

Checks if the print record *hPrint* is consistent with the current version of the Printing Manager and the currently installed printer. It returns **nil** if this is correct.

NOTE: Never call **tb:!PrValidate** between the printing of pages of a document.

*Example:*

```
(setf hPrint (make-instance 'tb:THPrint))
(tb:!PrValidate hPrint)
```

### **tb:!PrStlDialog** *hPrint* [II-158] Function

Displays the printing style dialog box (Page SetUp...); any changes to the default values are saved in *hPrint*. The trap returns true if the OK button was chosen, false if the Cancel button was chosen.

*Example:*

```
(setf hPrint (make-instance 'tb:THPrint))
(when (tb:!PrStlDialog hPrint)
   ...continue printing with information from user...)
```

**tb:!PrJobDialog** *hPrint* [II-158] Function

Displays the job dialog box (Print...). Any changes to the default values are saved in *hPrint*. The trap returns true if the OK button was chosen, false if the Cancel button was chosen.

*Example:*

```
(set hPrint (make-instance 'tb:THPrint))
(when (tb:!PrJobdialog hPrint)
  ...continue printing with information from user...)
```

**tb:!PrJobMerge** *hPrintSrc hPrintDst* [II-159] Function

Copies all the information set by **tb:!PrJobDialog** from the print record *hPrintSrc* to the print record *hPrintDst*.

## Printing

22.4 These routines open and close printing grafPorts, start and finish the printing of a specified page, and control the printing of a previously spooled document.

**tb:!PrOpenDoc** *hPrint pPrPort pIOBuf* [II-159] Function

Sets up a new printing grafPort using information in the print record *hPrint* and returns the grafPort. The pointers *pPrPort* and *pIOBuf* are set to **tb:!nilPtr.**

**tb:!PrOpenPage** *pPrPort pPageFrame* [II-159] Function

Starts a new page. If it is spool printing, *pPageFrame* is the rectangle used for scaling. This trap completely reinitializes the current grafPort so be sure to set the desired properties, such as font type and size, before proceeding. If you do not want the page scaled, pass the rPage rectangle from your print record in *pPageFrame*. See the example for **tb:!PrPicFile**.

**CAUTION: Do not call the QuickDraw trap tb:!OpenPicture while a printing page is open.**

**tb:!PrClosePage** *pPrPort* [II-160] Function

Finishes printing the current page.

**tb:!PrCloseDoc** *pPrPort* [II-160] Function

Closes the printing grafport *pPrPort*.

**tb:!PrPicFile** *hPrint pPrPort pIOBuf pDevBuf prStatus* [II-160] Function

Prints a previously spooled document about which there is information in *hPrint*. Usually **tb:!nilPtr** is passed for *pPrPort*, *pIOBuf*, and *pDevBuf*. Pass an instance of **tb:TPrStatus** in *prStatus*.

**tb:TPrStatus** [II-616] Flavor

This flavor defines a print status record with the following instance accessor methods:

| | |
|---|---|
| • :ITOTPAGES | [ integer ] |
| • :ICURPAGE | [ integer ] |
| • :ITOTCOPIES | [ integer ] |
| • :ICURCOPY | [ integer ] |
| • :ITOTBANDS | [ integer ] |
| • :ICURBAND | [ integer ] |
| • :FPGDIRTY | [ boolean ] |
| • :FIMAGING | [ boolean ] |

*Example:*

```
;;; Print a one page document.
tb:
(defun print-1-page ()
  (let ((hPrint     (make-instance 'THPrint))
        (myPrPort   nil)
        (myPrStatus nil))
    (!PrintDefault hPrint)
    (when (and (!PrStlDialog hPrint)
               (!PrJobDialog hPrint))
      (setf myPrPort
            (!PrOpenDoc hPrint !nilPtr !nilPtr))
      (!PrOpenPage myPrPort
        (make-instance 'rect
          :left    (send hPrint :rpageleft)
          :top     (send hPrint :rpagetop)
          :right   (send hPrint :rpageright)
          :bottom  (send hPrint :rpagebottom)))
      ...draw the document here...
      (!PrClosePage myPrPort)
      (!PrCloseDoc  myPrPort)
      (when (= !bSpoolLoop (send hPrint :bJDocLoop))
        (setf myPrStatus (make-instance 'TPrStatus))
        (!PrPicFile hPrint !nilPtr !nilPtr !nilPtr
                    myPrStatus)))))
```

## Error Handling

22.5 These routines control error handling within the File Manager.

**tb:!PrError** [II-161] Function

Returns the result code of the last Printing Manager routine.

**tb:!PrSetError** *iErr* [II-161] Function

Sets the Printing Manager errorCode to *iErr*. This is useful for cancelling a printing operation.

## Low-Level Driver Access

22.6 Using the low-level printer traps is not a good idea because it will make your code dependent on the printer and the printer driver. See *Inside Macintosh* for details on these traps.

| | | |
|---|---|---|
| **tb:!PrDrvrOpen** | [II-163] | Function |
| **tb:!PrDrvrClose** | [II-163] | Function |
| **tb:!PrCtlCall** *iWhichCtl lParam1 lParam2 lParam3* | [II-163] | Function |
| **tb:!PrDrvrDCE** | [II-163] | Function |
| **tb:!PrDrvrVers** | [II-163] | Function |

# Chapter 23
# DEVICE MANAGER

## Introduction

23.1 The Device Manager is mostly used for directly calling the low-level device drivers like the Serial Driver or Printer Driver. The only time you will be likely to use it is to read or write data through the serial ports.

All the Device Manager traps, like the File Manager traps, use parameter blocks. All the Device Manager traps, with the exception of **tb:!Control** and **tb:!Status**, use the standard File Manager **tb:ioParam** parameter block instance. The traps **tb:!Control** and **tb:!Status** use a **tb:controlParam** instance.

## Device Manager Traps

23.2 These routines open, close, read from, write to, get information from, and send information to the device driver.

**tb:!Open** *paramBlock* [II-178] Function

Opens the device driver referred to by **:ioNamePtr**, with a read/write permission specified by **:ioPermssn**, and returns a reference number in **:ioRefNum**. The drive number, if there is one, is specified by **:ioVRefNum**. To open the print port (port B) serial driver for output, do the following:

*Example:*

```
;;; Use the Device Manager to open the Sound Driver.
(let ((paramBlock (make-instance 'tb:ioParam))
      (drvrhnd    (tb:!NewString ".Sound"))
      (drvrName   nil)))
  (tb:!HLock drvrhnd)
  (setf drvrName (tb:deref drvrhnd))
  (send paramBlock :ioNamePtr drvrName)
  (send paramBlock :ioPermssn tb:!fsCurPerm)
  (tb:!Open paramBlock))
```

**tb:!Close** *paramBlock* [II-178] Function

Closes the device driver with the reference number *ioRefNum*.

**tb:!Read** *paramBlock* [II-178] Function

Tries to read **:ioReqCount** bytes from the device driver with a reference number **:ioRefNum**, and puts them in a buffer pointed to by **:ioBuffer**. The drive number, if there is one, is specified by **:ioVRefNum**. The actual number of bytes read is returned in **:ioActCount**.

*Example:*

```
(let ((paramBlock     (make-instance 'tb:ioParam))
      (myBufferHandle (tb:!NewHandle myReqCount))
      (myBufferPtr    (tb:deref myBufferHandle)))
  (tb:!Hlock myBufferHandle)
  (send paramBlock :ioRefNUm myRefNum)
  (send paramBlock :ioBuffer myBufferPtr)
  (send paramBlock :ioReqCount myReqCount)
  (prog1 (tb:!Read paramBlock))
        (tb:!HUnlock myBufferHandle)))
```

**tb:!Write** *paramBlock* [II-179] Function

Tries to write **:ioReqCount** bytes from a buffer pointed at by **:ioBuffer,** to the device driver with a reference number :ioRefNum. The drive number, if there is one, is specified by **:ioVRefNum.** The actual number of bytes read is returned in **:ioActCount.**

*Example:*

```
;;; Use the Device Manager to tell the Sound Driver to sound a
;;; 440Hz tone for one second
(let ((paramBlock (make-instance 'tb:ioParam))
      (buff       (tb:!NewHandle 100))
      (buffPtr    nil))
  (tb:!HLock buff)
  (setf buffPtr (tb:deref buff))
  (send paramBlock :ioBuffer buffPtr)
  (tb:stowword buffPtr 0 -1)
  (tb:stowword buffPtr 2 1780)
  (tb:stowword buffPtr 4 255)
  (tb:stowword buffPtr 6 60)
  (send paramBlock :ioReqCount 8)
  (tb:!Write paramBlock))
```

**tb:!Control** *paramBlock* [II-179] Function

Sends control information to the device driver with a reference number **:ioRefNum.** The drive number, if any, is put in **:ioVRefNum.** The type of information is specified by **:csCode** and the information is passed in **:csParam.**

For example, the Disk Driver trap **tb:!DiskEject** is actually a **tb:!Control** trap with a csCode = ejectCode (7). We can define a function DiskEject which takes care of everything as follows:

*Example:*

```
;;; Eject the disk in drive drvNum
(defun DiskEject (drvNum)
  (let ((paramBlock (make-instance 'tb:controlParam)))
    (send paramBlock :ioVRefNum drvNum)
    (send paramBlock :ioCRefNum -5)
    (send paramBlock :csCode 7)
    (tb:!Control paramBlock)))
```

**tb:!Status** *paramBlock* [II-179] Function

Returns control information about the device driver with a reference number **:ioRefNum.** The drive number, if any, is put in **:ioVRefNum.** The type of information returned is specified by :csCode and the information is passed in **:csParam.**

**tb:!KillIO** *paramBlock* [II-179] Function

Stops any current I/O requests from being processed. It also removes all pending I/O requests of the device driver which has a reference number **:ioRefNum.**

# Chapter 24
# DISK DRIVER

## Introduction

24.1 The Disk Driver is a Macintosh device driver used for storing and retrieving information on Macintosh $3^1/_2$- inch disks.

## Disk Driver Traps

24.2 These traps should only be used if you are writing a low-level disk editor like FEdit or a copy protection scheme.

**tb:!DiskEject** *drvNum* [II-214] Function

Ejects the disk from the drive *drvNum*.

**tb:!SetTagBuffer** *buffPtr* [II-214] Function

Sets the file tag buffer to *buffPtr*.

**tb:!DriveStatus** *drvNum* [II-215] Function

Returns a **tb:DrvSts** instance containing the status of the drive *drvNum*.

**tb:DrvSts** [II-215] Flavor

This flavor defines a drive status record with the following instance accessor methods:

| | |
|---|---|
| • :TRACK | [ integer ] |
| • :WRITEPROT | [ byte ] |
| • :DISKINPLACE | [ byte ] |
| • :INSTALLED | [ byte ] |
| • :SIDES | [ byte ] |
| • :QLINK | [ pointer ] |
| • :QTYPE | [ integer ] |
| • :DQDRIVE | [ integer ] |
| • :DQREFNUM | [ integer ] |
| • :DQFSID | [ integer ] |
| • :TWOSIDEFMT | [ byte ] |
| • :NEEDSFLUSH | [ byte ] |
| • :DISKERRS | [ integer ] |

*Example:*

```
(setf status (tb:!DriveStatus 1))
(send status :writeProt)
```

## Advanced Disk Driver Traps

24.3 Drive numbers are now dynamically assigned and there are three new Disk Driver control calls.

- Return the Disk Drive's media Icon
- Return the Disk Drive's physical Icon
- Return information about a Disk Drive

When a Disk Driver **tb:!Control** call is made with a :csCode of 21, a pointer to a data structure is returned in the **:csParam** field of the parameter block. This data structure consists of an icon, a mask icon, and a Pascal string all describing the disk drive whose logical drive number is in the **:ioRefNum** field. The data structure pointed to typically describes the disk media.

When a Disk Driver **tb:!Control** call is made with a :csCode of 22, a pointer to an icon and a mask icon for the disk drive whose logical drive number is in the **:ioRefNum** field is returned in the **:csParam** field of the parameter block. The icon pointed to typically describes the physical drive.

When a Disk Driver **tb:!Control** call is made with a :csCode of 23, a 32-bit value is returned containing status information about the disk drive whose logical drive number is in the **:ioRefNum**.

The low-order byte of the value returned specifies the drive type. The types currently defined are specified by the following bits:

| *Bit* | *Meaning* | |
|---|---|---|
| • 0 | No such drive | |
| • 1 | Unspecified drive | |
| • 2 | 400K byte drive | |
| • 3 | 800K byte drive | |
| • 5 | 3.2M byte drive | RESERVED |
| • 6 | 6.4M byte drive | RESERVED |
| • 7 | HD-20 | |

Bits 8 through 11 of the value returned specify the drive attributes. These attributes are indicated by bit flags. The bit flags defined are:

| *Bit* | *Meaning* |
|---|---|
| • 8 | Set for primary drive, clear for secondary drives. |
| • 9 | Set if SCSI drive, clear if IWM. |
| • 10 | Set if drive is fixed, clear if removable. |
| • 11 | Set for external drive, clear for internal. |

# Chapter 25
# SERIAL DRIVER

## Introduction

25.1 The Serial Driver allows you to transmit and receive data through the two serial ports. You use the Device Manager traps to open the ports, and to read and write data through the ports. The Serial Driver traps are used to get and set the status of the serial ports, and to reconfigure the ports.

The driver names and reference numbers of the serial drivers are:

| *Driver* | *Driver Name* | *Reference Number* |
|---|---|---|
| • Modem Port Input | .AIn | -6 |
| • Modem Port Output | .AOut | -7 |
| • Printer Port Input | .BIn | -8 |
| • Printer Port Output | .BOut | -9 |

## Changing Serial Driver Information

25.2 These routines enable you to initialize and reset driver information.

**tb:!SerReset** *refNum serConfig* [II-250] Function

Resets and initializes the driver *refNum* with the configuration *serConfig*. The configuration is built up by adding four values together: the baud rate, the number of stop bits, the parity, and the number of data bits. The values to be added together are as follows:

**Baud Rate:**

| *Description* | *Constant* | *Value* |
|---|---|---|
| • 300 baud | **tb:!Baud300** | 380 |
| • 600 baud | **tb:!Baud600** | 189 |
| • 1200 baud | **tb:!Baud120** | 94 |
| • 1800 baud | **tb:!Baud1800** | 62 |
| • 2400 baud | **tb:!Baud2400** | 46 |
| • 3600 baud | **tb:!Baud3600** | 30 |
| • 4800 baud | **tb:!Baud4800** | 22 |
| • 7200 baud | **tb:!Baud7200** | 14 |
| • 9600 baud | **tb:!Baud9600** | 10 |
| • 19200 baud | **tb:!Baud19200** | 4 |
| • 57600 baud | **tb:!Baud57600** | 0 |

**Stop Bits:**

| *Description* | *Constant* | *Value* |
|---|---|---|
| • 1 stop bit | **tb:!Stop10** | #x4000 |
| • 1.5 stop bits | **tb:!Stop15** | #x8000 |
| • 2 stop bits | **tb:!Stop20** | #xC000 |

**Parity:**

| *Description* | *Constant* | *Value* |
|---|---|---|
| no parity | **tb:!NoParity;** | 0 |
| odd parity | **tb:!OddParity;** | 4096 |
| even parity | **tb:!EvenParity** | 12288 |

**Data Bits:**

| *Description* | *Constant* | *Value* |
|---|---|---|
| 5 data bits | **tb:!Data5** | 0 |
| 6 data bits | **tb:!Data6** | 2048 |
| 7 data bits | **tb:!Data7** | 1024 |
| 8 data bits | **tb:!Data8** | 3072 |

To set the modem out serial port to a baud rate of 9600, one stop bit, with no parity bits, and 7 data bits, do the following:

*Example:*

```
(tb:!SerReset -7 (+ tb:!Baud9600
                    tb:!Stop10
                    tb:!NoParity
                    tb:!Data7))
```

**tb:!SerSetBuf** *refNum serBPtr serBLen* [II-251] Function

Sets the buffer of the driver *refNum* to *serBptr*, which has a length *serBLen.*

*Example:*

```
(setf myBufHandle (tb:!NewHandle 128))
(tb:!HLock myBufHandle)
(setf myBufPtr (deref myBufHandle))
(tb:!SerSetBuf -7 myBufPtr 128)
```

**tb:!SerHShake** *refNum flags* [II-251] Function

Sets the handshake and other information for the driver *refNum*. The *flags* parameter should be a **tb:SerShk** instance.

**tb:SerShk** [II-253] Flavor

This flavor defines a serial driver handshake data structures with the following instance accessor methods:

| | |
|---|---|
| :FXON | [ byte ] |
| :FCTS | [ byte ] |
| :XON | [ char ] |
| :XOFF | [ char ] |
| :ERRS | [ byte ] |
| :EVTS | [ byte ] |
| :FINX | [ byte ] |

**tb:!swOverrunErr** [II-254] Constant
**tb:!swOverrunErr-p** [II-254] Function
**tb:!hwOverrunErr** [II-252, 254] Constant
**tb:!hwOverrunErr-p** [II-252, 254] Function
**tb:!parityErr** [II-252, 254] Constant
**tb:!parityErr-p** [II-252, 254] Function
**tb:!framingErr** [II-252, 254] Constant
**tb:!framingErr-p** [II-252, 254] Function

The constants represent masks which may be applied to the **:errs** instance variable of a **tb:SerShk** instance or the **:cumErrs** instance variable of a **tb:SerStaRec** instance to determine what kind of errors occurred. **tb:!swOverrunErr** applies only to **:cumErrs**.

The functions are predicates which take the relevant instance variable and return true if the associated mask bits are true.

**tb:!ctsEvent** [II-252, 254] Constant
**tb:!ctsEvent-p** [II-252, 254] Function
**tb:!breakEvent** [II-252, 254] Constant
**tb:!breakEvent-p** [II-252, 254] Function

The constants represent masks which may be applied to the **:evts** instance variable of a **tb:SerShk** instance to determine whether a change in CTS or break status will cause the serial driver to post an event.

The functions are predicates which take the **:evts** instance variable value and return true if the associated status change will post an event.

*Example:*

```
(setf mySerShk (make-instance 'tb:SerShk))
(tb:!SerHShake -7 mySerShk)
```

**tb:!SerSetBrk** *refNum* [II-252] Function
**tb:!SerClrBrk** *refNum* [II-253] Function

Sets or clears break mode in the driver *refNum*.

## Getting Serial Driver Information

25.3 These routines return the size and status of a specified driver.

**tb:!SerGetBuf** *refNum* [II-253] Function

Returns the size of the driver *refNum*'s buffer. If an error occurs, **tb:!SerGetBuf** either signals or returns the OSErr depending upon the value of **tb:*signal-mac-oserr***.

### **tb:!SerStatus** *refNum* [II-253] Function

Returns a **tb:SerStaRec** instance with the status of the driver *refNum*. If an error occurs, **tb:!SerStatus** either signals or returns the OSErr depending upon the value of **tb:*signal-mac-oserr***.

### **tb:SerStaRec** [II-253] Flavor

This flavor defines a serial driver status record with the following instance accessor methods:

| | |
|---|---|
| • :CUMERRS | [ byte ] |
| • :XOFFSENT | [ byte ] |
| • :RDPEND | [ byte ] |
| • :WRPEND | [ byte ] |
| • :CTSHOLD | [ byte ] |
| • :XOFFHOLD | [ byte ] |

# Chapter 26
# SOUND MANAGER

## Introduction

26.1 The Sound Manager has been totally rewritten for the Macintosh II. The Macintosh II contains a custom sound chip so most of the sound generation processing has been offloaded from the CPU.

The new Sound Manager is downwardly compatible with the old Sound Driver. All the old Sound Driver traps are supported by the new Sound Manager, but the organization and theory of the new Sound Manager is quite different, reflecting the complete difference in hardware.

The new Sound Manager is based around two new objects: the synthesizer and the channel. A synthesizer is a driver that accepts sound generation or modification commands and translates them into sound. A channel is simply a queue of commands that are associated with a particular synthesizer. There are four types of synthesizers supported by the Sound Manager:

- *Note Synthesizer* - The Note Synthesizer generates simple sounds. A simple monophonic melody of notes can be played with the note synthesizer. Each note has a specified frequency, amplitude, and duration. The timbre of the sound can be changed at any time during the melody.

- *Wave Table Synthesizer* - The Wave Table Synthesizer generates sounds using wave tables. The timbre of the sound is specified by a table of 8-bit samples. This table specifies one cycle of the sound. Each sample in the table is a signed byte.

- *MIDI Synthesizer* - The MIDI Synthesizer generates MIDI data to drive external MIDI sound generators.

- *Sampled Sound Synthesizer* - The Sampled Sound Synthesizer generates sound from a sample buffer of 8-bit signed bytes.

The Sound Manager allows modifiers, small routines that modify commands, to be associated with a channel. These modifiers can modify, expand, or block commands passed to the channel.

## Sound Manager Commands

26.2 To create a new sound command, make an instance of **tb:SndCommand**.

**tb:SndCommand** [V-483] Flavor

This flavor defines a sound command.

**:cmd** Method of **tb:SndCommand**
**:set-cmd** *16b-integer* Method of **tb:SndCommand**

This is an integer identifying the command. The Macintosh sound command *names* (such as "InitCmd") have been turned into Lisp constants in the MACTOOLBOX package (e.g., **tb:!InitCmd**). Therefore, while the following paragraphs document the individual command numbers in the manner of *Inside Macintosh* [V-486], you may use the equivalent symbolic constants when setting this instance variable.

**:param1** Method of **tb:SndCommand**
**:param2** Method of **tb:SndCommand**
**:set-param1** *16b-integer* Method of **tb:SndCommand**
**:set-param2** *32-bit integer* Method of **tb:SndCommand**

These instance variables in the sound command hold miscellaneous arguments such as duration or pitch values. The exact meaning varies with the command.

If the high order of **:cmd** is set, then **:param2** contains a pointer to some memory location.

NOTE: If the synthesizer is sent a command it cannot act upon, it ignores it.

### General Commands

**26.2.1** These are the general commands.

**tb:!NullCmd**

cmd = 0 param1 = 0 param2 = NIL

Has no effect because the Sound Manager does not pass them on to the synthesizer.

**tb:!InitCmd**

cmd = 1 param1 = 0 param2 = *init*

Sent to a synthesizer or modifier by the Sound Manager when it is first linked to a channel. If the application passed an *init* parameter when calling the trap **tb:!SndNewChannel**, this information is passed in param2. The *init* parameter has the following masking currently defined:

**tb:!initChanLeft** [V-486] Constant
**tb:!initChanRight** [V-486] Constant

These InitCmd **:param2** *init* values specify left and right stereo channels.

**tb:!initChan0** [V-486] Constant
**tb:!initChan1** [V-486] Constant
**tb:!initChan2** [V-486] Constant
**tb:!initChan3** [V-486] Constant

These InitCmd:**param2** *init* values specify channels 0-3, respectively, for wave table only.

**tb:!initSRate22k** [V-486] Constant
**tb:!initSRate44k** [V-486] Constant

These InitCmd:**param2** *init* values specify sample rates.

**tb:!initMono** [V-486] Constant
**tb:!initStereo** [V-486] Constant

These InitCmd **:param2** *init* values specify monophonic and stereophonic channels, respectively.

**tb:!FreeCmd**

cmd = 2 param1 = 0 param2 = NIL

Causes the synthesizer and modifiers to stop processing commands after the current sound has finished playing.

**tb:!QuietCmd**

cmd = 3 param1 = 0 param2 = NIL

Causes the immediate termination of generation of the current sound.

**tb:!FlushCmd**

cmd = 4 param1 = 0 param2 = NIL

Causes all commands to be immediately flushed from the channel.

## Synchronization Commands

26.2.2 Sound Manager channels can be synchronized by using the CallBack command or the Synch command. When a channel is created by using the trap **tb:!SndNewChannel**, a CallBack routine can be specified. This routine can be used to synchronize the channel with some event or command.

The Sync command causes the Sound Manager to stop all processing on a channel until the same command is received on one or more other channels. When all the channels have reached the same Sync command, they all proceed.

### tb:!WaitCmd

cmd = 10 param1 = *duration* param2 = NIL

Causes a pause for the specified *duration* in the processing of commands.

### tb:!PauseCmd

cmd = 11 param1 = 0 param2 = NIL

Causes the processing of a command to pause for an indefinite amount of time.

### tb:!ResumeCmd

cmd = 12 param1 = 0 param2 = NIL

Causes the continuation of the processing of commands for a channel that was halted by a PauseCmd.

### tb:!CallBackCmd

cmd = 13 param1 = *user-defined1* param2 = *user-defined2*

Calls the channel's CallBack procedure, passing the two command arguments to the routine.

### tb:!SyncCmd

cmd = 14 param1 = *count* param2 = *identifier*

Causes the channel to wait for a Sync command with the same value as *identifier* from *count* other channels. When the conditions are met, the channel proceeds.

### tb:!EmptyCmd

cmd = 15 param1 = 0 param2 = NIL

Sent only by the Sound Manager.

## Modifier Control Commands

26.2.3 These commands control the modifiers.

### tb:!TickleCmd

cmd = 20 param1 = 0 param2 = NIL

Sent regularly by the Sound Manager to synthesizers and modifiers that require periodic actions.

### tb:!RequestNextCmd

cmd = 21  param1 = *count*  param2 = NIL

Sent by the Sound Manager when a modifier returns a result of T, that is, the modifier requests another command. The value *count* is the number of times in succession that this modifier has asked to send another command.

### tb:!HowOftenCmd

cmd = 22  param1 = *period*  param2 = *pointer*

Tells the Sound Manager to send a Tickle command every *period* to the modifier that is pointed to by *pointer*.

### tb:!WakeUpCmd

cmd = 23  param1 = *period*  param2 = *pointer*

Tells the Sound Manager to send a Tickle command after *period* amount of time has elapsed, to the modifier that is pointed to by *pointer*.

## Scaling and Note Commands

26.2.4 These are the scaling and note commands.

### tb:!NoteCmd

cmd = 40  param1 = *duration*  param2 = *amp+frequency*

Plays a note with the specified *amp* and *frequency* for *duration* amount of time. If the channel is monophonic, all processing stops until the note finishes. If the channel is polyphonic, processing continues without interruption.

### tb:!RestCmd

cmd = 41  param1 = *duration*  param2 = NIL

Causes the channel to rest for *duration* amount of time. This command may not result in complete silence as previous notes may still be decaying. This command differs from WaitCmd because it causes the currently sounding note to go into the release and decay stages, whereas WaitCmd causes a complete pause on the channel.

### tb:!FreqCmd

cmd = 42   param1 = 0      param2 = *frequency*

Changes the frequency of the currently sounding note to *frequency.* If no note is sounding, a note is triggered.

### tb:!AmpCmd

cmd = 43   param1 = *amplitude*     param2 = NIL

Sets the amplitude of the current note to *amplitude.* If no note is playing, then the amplitude of the next note triggered will be *amplitude.*

### tb:!TimbreCmd

cmd = 44   param1 = *timbre*          param2 = NIL

Sets the timbre of the channel to the timbre indicated by the timbre code *timbre.*

## Wave Table Synth Commands

**26.2.5** The following command affects the wave table.

### tb:!WaveTableCmd

cmd = 60   param1 = *length*          param2 = *pointer*

Specifies the wave table to be used with the succeeding note commands. The wave table is pointed to by *pointer* and its length is specified by *length.*

## Sampled Sound Synth Commands

**26.2.6** These are sampled sound synthesizer commands.

### tb:!SoundCmd

cmd = 80   param1 = 0      param2 = *pointer*

Specifies the sound to be played by successive note commands. The *pointer* argument points to the sound description.

**tb:!BufferCmd**

cmd = 81   param1 = 0     param2 = *pointer*

Plays the sound pointed to by the argument *pointer* with the most recent frequency and amplitude settings.

**tb:!RateCmd**

cmd = 82   param1 = 0     param2 = *rate*

Sets the playback rate of succeeding buffer commands. The argument *rate* is a multiplier of the original sample rate.

## Original Sound Driver Traps

26.3 These traps were a part of the old Sound Driver, but are still supported by the current version.

**tb:!swMode** [II-225] Constant
**tb:!ftMode** [II-225] Constant
**tb:!ffMode** [II-225] Constant

These are mode constants used in synthesizer records to identify the synthesizer as a square-wave, four-tone, or free-from synthesizer, respectively.

**tb:!StartSound** *synthRec numBytes completionRtn* [II-231] Function

Starts generating the sound described by the buffer *synthRec*, which has a size *numBytes*. If *completionRtn* is **tb:!onePtr**, the sound will be produced synchronously. If *completionRtn* is **tb:!nilPtr**, the sound will be produced asynchronously.

*Example:*

```
;;; Define a function which plays a square wave  at 440 Hz for
;;; 2 seconds.
(defun 440hz ()
   (let ((SWSynthRecPtr (tb:!NewPtr 8))); alloc space=1 tone
     (tb:StowWord SWSynthRecPtr 0 tb:!swMode);mode=swMode
     (tb:StowWord SWSynthRecPtr 2 1780)     ; count=440HZ
     (tb:StowWord SWSynthRecPtr 4 255)      ; amplitude=max
      (tb:StowWord SWSynthRecPtr 6 (* 60 2)); duration=2 secs
     (tb:!StartSound SWSynthRecPtr 8 tb:!nilPtr) ; start it
     (tb:!DisposPtr SWSynthRecPtr)))
```

**tb:!StopSound** [II-232] Function

Immediately stops the current **tb:!StartSound** call and then executes the completion routine if there is one. It also cancels all other **tb:!StartSound** calls that have been queued.

**tb:!GetSoundVol** [II-232] Function

Returns the current speaker volume. The value returned can range from 0 (no sound) to 7 (the loudest possible sound).

**tb:!SetSoundVol** *level* [II-233] Function

Sets the speaker volume to the desired value (from 0 to 7).

## Sound Manager Traps

26.4 These traps deal with sound channels.

**tb:SndChannel** [V-481] Flavor

This flavor defines a sound channel. After creating the **tb:SndChannel** instance, call **tb:!SndNewChannel** to initialize it.

**:nextChan** Method of **tb:SndChannel**
**:firstMod** Method of **tb:SndChannel**
**:callBack** Method of **tb:SndChannel**

These are the pointers to the next channel, the first modifier, and the call back procedure for a channel, respectively.

**:userInfo** Method of **tb:SndChannel**
**:set-userInfo** *32b-integer* Method of **tb:SndChannel**

This value is reserved for the use of the application.

**tb:!SndPlay** *chan sndHdl async* [V-477] Function

Plays the "snd " resource specified by *sndHdl.* The "snth" resource is added to the channel for each synthesizer and modifier in the resource list. The commands in the "snd " resource are then passed to the channel. If *chan* is **tb:!nilPtr** and there are no modifiers in the resource list, a note synthesizer is created.

**tb:!SndNewChannel** *chan synth init userRoutine* [V-477] Function

Creates a new channel between the application and a synthesizer. If the argument *chan* is **tb:!nilPtr**, the Sound Manager will allocate memory for the channel. The *synth* argument specifies which synthesizer is to be used. If the *synth* argument is 0, a note synthesizer is created. The following synthesizers with their respective *synth* are presently supported:

| | |
|---|---|
| • note Synth | 1 |
| • wave Table Synth | 3 |
| • sampled Synth | 5 |
| • MIDI Synth In | 7 |
| • MIDI Synth Out | 9 |

The *init* value is used as the argument for the InitCmd command. The value will depend on the synthesizer used. The *userRoutine* argument is a pointer to a routine that is called when a CallBackCmd is sent. If *userRoutine* is **tb:!nilPtr**, any CallBackCmds are ignored.

# Chapter 27
# OPERATING SYSTEM UTILITIES

## Introduction

27.1 The Operating System Utilities are a collection of useful routines that:

- Manipulate handles and pointers (memory management utilities)
- Convert strings
- Manipulate date and time operations

## Pointer and Handle Manipulation

27.2 These routines are a collection of memory management utilities used for converting and manipulating handles and pointers. The only traps you will be likely to use are:

- **tb:!HandToHand**, used to duplicate a handle.
- **tb:!HandAndHand**, used to append two handles.
- **tb:!PtrAndHand**, used to add data to the end of a handle.

**tb:!HandToHand** *theHandle* [II-374] Function

Makes a copy of the handle in *theHandle*, and returns two values: the handle to the copy and an OSErr. If you need to make a copy of the handle **myHandle**, you could do the following:

*Example:*

```
(setf myHandle (tb:!NewHandle 10))
(multiple-value-bind (newHandle result)
    (tb:!HandToHand myHandle)
  ...code using newHandle and result...)
```

A copy of the handle **myHandle** will be returned in **newHandle**.

**tb:!PtrToHand** *srcPtr size* [II-375] Function

Returns two values: a new handle which is a copy of the *size* bytes starting at *srcPtr*, and an Operating System result code. To make a relocatable copy of a non-relocatable block **myPointer**, do the following:

*Example:*

```
(setf pointerSize (tb:!GetPtrSize myPointer))
(multiple-value-bind (newHandle result)
    (tb:!PtrToHand myPointer pointerSize)
  ...code using newHandle and result...)
```

A handle to the new relocatable block will be returned in **newHandle**.

**tb:!PtrToXHand** *srcPtr dstHandl e size* [II-375] Function

Takes an existing handle *dstHandle* and sets it to a copy of the *size* bytes starting at *srcPtr* and returns an error code as a result. To set the existing handle **myHandle** to the contents of the pointer **myPointer**, do the following:

*Example:*

```
(setf pointerSize (tb:!GetPtrSize myPointer))
(tb:!PtrToXHand myPointer myHandle pointerSize) => OSErr
```

**tb:!HandAndHand** *aHndl bHndl* [II-375] Function

Appends the data in *aHndl* to the end of the handle *bHndl* and returns an error code as a result.

**tb:!PtrAndHand** *ptr hndl size* [II-376] Function

Appends the data, *size* number of bytes starting at *ptr*, to the handle *hndl* and returns an error code as a result.

## Date and Time Operations

27.3 The Date and Time traps enable you to get and set the time and date. You will usually use the traps **tb:!GetTime** and **tb:!SetTime** to do this.

**tb:DateTimeRec** [II-378] Flavor

This flavor defines a date and time structure.

| | |
|---|---|
| **:year** | Method of **tb:DateTimeRec** |
| **:set-year** | Method of **tb:DateTimeRec** |
| **:month** | Method of **tb:DateTimeRec** |
| **:set-month** | Method of **tb:DateTimeRec** |
| **:day** | Method of **tb:DateTimeRec** |
| **:set-day** | Method of **tb:DateTimeRec** |
| **:hour** | Method of **tb:DateTimeRec** |
| **:set-hour** | Method of **tb:DateTimeRec** |
| **:minute** | Method of **tb:DateTimeRec** |
| **:set-minute** | Method of **tb:DateTimeRec** |
| **:second** | Method of **tb:DateTimeRec** |
| **:set-second** | Method of **tb:DateTimeRec** |
| **:dayOfWeek** | Method of **tb:DateTimeRec** |
| **:set-dayOfWeek** | Method of **tb:DateTimeRec** |

These values represent the year (1904-2040), month (1-12), day (1-31), hour (0-23), minute (0-59), second (0-59), and day of the week (1-7, where 1=Sunday).

**tb:!ReadDateTime** *ptr* [II-378] Function

This is an internal trap. *Ptr* is a pointer to a longint.

**tb:!GetDateTime** &optional VAR *secs* [II-378] Function

Returns the number of seconds since midnight, 1 January 1904 in the local time zone.

Note that universal time in Common Lisp is defined as the number of seconds since midnight, 1 January 1900 Greenwich Mean Time (GMT). Therefore, universal time on the microExplorer differs from universal time on the Macintosh by four years plus the local time zone. Since the microExplorer universal time is relative to GMT, then all microExplorers in the world who are set to the correct local time should return identical universal times. Macintosh systems, on the other hand, willl return universal times which vary with the local time zone.

**tb:!SetDateTime** *secs* [II-379] Function

Sets the clock time to *secs*, the number of seconds since midnight, 1 January 1904.

**tb:!Date2Secs** *ptr* [II-379] Function

Given a **tb:DateTimeRec** instance pointed at by *ptr*, returns the number of seconds since midnight, 1 January 1904.

*Example:*

```
(setf myDate (make-instance 'tb:DateTimeRec))
(send myDate :set-year 1988)
(send myDate :set-month 7)
(send myDate :set-day 27)
(tb:!HLock myDate)
(setf secs (tb:!Date2Secs (tb:deref myDate)))
(tb:!HUnlock myDate)
secs => 2668809600
```

**tb:!Secs2Date** *secs ptr* [II-380] Function

Converts *secs*, the number of seconds since midnight, 1 January 1904, into a **tb:DateTimeRec** pointed at by *ptr*.

*Example:*

```
(setf myDate (make-instance 'tb:DateTimeRec))
(tb:!HLock myDate)
(tb:!Secs2Date 1000000 (tb:deref myDate))
(tb:!HUnlock myDate)
(send myDate :year)  => 1904
(send myDate :month) => 1
(send myDate :day)   => 12
```

**tb:!GetTime** [II-380] Function

Gets the number of seconds since midnight, 1 January 1904 (from **tb:!GetDateTime**) and converts the value into a **tb:DateTimeRec** instance which is returned.

*Example:*

```
(setf myDate (tb:!GetTime))
(send myDate :year)  => 1988
(send myDate :month) => 8
(send myDate :day)   => 30
```

**tb:!SetTime** *date* [II-380] Function

Takes the day and date in *date* and converts the values into the number of seconds since midnight, 1 January 1904, and sets the clock chip to the number of seconds.

## Parameter RAM Operations

27.4 These routines are used to read from and write to parameter RAM.

**tb:!InitUtil** [II-380] Function

An internally called trap.

**tb:!GetSysPPtr** [II-381] Function

Returns a pointer to the copy of the parameter RAM kept in memory.

## Queue Manipulation

27.5 Normally, you will not need to use these utilities. They are included in this documentation for completeness only.

**tb:!Enqueue** *qEntry theQueue* [II-382] Function

Adds the entry pointed to by *qEntry* to the queue *theQueue*.

**tb:!Dequeue** *qEntry theQueue* [II-383] Function

Removes the element *qEntry* from the queue *theQueue*.

## Trap Dispatch Table Utilities

27.6 These traps are used for getting and setting the address of trap routines (usually in ROM). There will be never be any need to use these unless you are interested in doing low-level coding such as disassembling the ROM.

**tb:!SetTrapAddress** *trapAddr trapNum* [II-384] Function
**tb:!SetOSTrapAddress** *trapaddr trapnum*

Install a routine with address *trapAddr* in the trap dispatch table. The routine is installed under the trap number *trapNum*. These traps are identical.

**tb:!GetTrapAddress** *trapNum* [II-384] Function
**tb:!GetOSTrapAddress** *trapnum*

Returns the address of the routine currently installed in the trap dispatch table under the trap number *trapNum*. These two traps are identical.

**tb:!SetToolTrapAddress** *trapaddr trapnum*

Installs a routine with address *trapAddr* in the trap dispatch table. The routine is installed as a Toolbox trap under the trap number *trapNum*.

**tb:!GetToolTrapAddress** *trapnum*

Returns the address of the Toolbox trap routine currently installed in the trap dispatch table under the trap number *trapNum*.

## Miscellaneous Utilities

27.7 These traps perform miscellaneous operating system utility functions.

**tb:!Delay** *numTicks* [II-384] Function

Waits for *numTicks* number of ticks (1/60ths of a second) and returns the time in ticks since the Macintosh was turned on.

*Example:*

```
;;; wait for 1 second
(tb:!Delay 60) => 121634
```

**tb:!SysBeep** *duration* [II-385] Function

Causes the system beep sound to be made for *duration* number of ticks.

NOTE: On the Macintosh II, *duration* is ignored and the sound played is the default error sound.

**tb:!GetMMUMode** [V-592] Function

Returns the address translation mode currently in use.

When the Macintosh II boots up, it defaults to a 24-bit addressing mode, the same as the Macintosh, Macintosh Plus, and Macintosh SE.

The addressing-mode constants are defined as:

- **tb:!false32b** 24-bit addressing mode
- **tb:!true32b** 32-bit addressing mode

# Chapter 28
# LIST MANAGER PACKAGE

## Introduction

28.1 The List Manager Package is a tool for storing and updating data elements within a list and for displaying the list in a rectangle within a window. The lists handled by the List Manager are not Lisp lists. Do not confuse the two.

## Creating and Disposing of Lists

28.2 The routines that follow are used to create and dispose of lists. To create a new ListRec object, use **make-instance.**

**tb:ListRec** [IV-263] Flavor

This flavor defines a list structure. All List Manager functions defined to take a list handle will accept instances of this flavor.

**:rView** *rect* Init Option of **tb:ListRec**

This is the list's display rectangle (excluding scroll bars) in local coordinates. The default is (50 50 100 100).

**:dataBounds** *pseudo-rect* Init Option of **tb:ListRec**

This is the boundary of list cells. A rectangle specification is used where the values represent *cell* coordinates rather than pixels number as usual. The default is (0 0 5 10).

**:cSize** *pseudo-point* Init Option of **tb:ListRec**

This is the size of a cells in pixels. A point specification is used where the two v-h "coordinates" actually represent the length of the sides of the cells. The default is a "zero" size cell which actually tells the List Manager to figure it out.

**:theProc** *procID* Init Option of **tb:ListRec**

This defaults to 0 indicating a standard text-only list.

**:theWindow** *windowPtr* Init Option of **tb:ListRec**

This is the window owning the list. The default is the frontmost window.

**:drawIt** *visible-p* Init Option of **tb:ListRec**

If this option is true (the default), then the list is drawn on *theWindow*.

**:hasGow** *growBox-p* Init Option of **tb:ListRec**

If this option is true (the default), the window will have a grow box.

**:scrollHoriz** *scroll-p* Init Option of **tb:ListRec**
**:scrollVert** *scroll-p* Init Option of **tb:ListRec**

If true (the default), the list will have a horizontal or vertical scroll bar, respectively.

**tb:ListRec** instances have the following instance accessor methods:

| | |
|---|---|
| • :RVIEWTOP | [ integer ] |
| • :RVIEWLEFT | [ integer ] |
| • :RVIEWBOTTOM | [ integer ] |
| • :RVIEWRIGHT | [ integer ] |
| • :PORT | [ pointer ] |
| • :INDENTH | [ integer ] |
| • :INDENTV | [ integer ] |
| • :CELLSIZEH | [ integer ] |
| • :CELLSIZEV | [ integer ] |
| • :VISIBLETOP | [ integer ] |
| • :VISIBLELEFT | [ integer ] |
| • :VISIBLEBOTTOM | [ integer ] |
| • :VISIBLERIGHT | [ integer ] |
| • :VSCROLL | [ handle ] |
| • :HSCROLL | [ handle ] |
| • :SELFLAGS | [ byte ] |
| • :LACTIVE | [ boolean ] |
| • :LISTFLAGS | [ byte ] |
| • :CLICKTIME | [ longint ] |
| • :CLIKLOCH | [ integer ] |
| • :CLIKLOCV | [ integer ] |
| • :MOUSELOCH | [ integer ] |
| • :MOUSELOCV | [ integer ] |
| • :LCLIKLOOP | [ pointer ] |
| • :LASTCLICKH | [ integer ] |
| • :LASTCLICKV | [ integer ] |
| • :REFCON | [ pointer ] |
| • :LISTDEFPROC | [ handle ] |
| • :USERHANDLE | [ handle ] |
| • :DATABOUNDSTOP | [ integer ] |
| • :DATABOUNDSLEFT | [ integer ] |
| • :DATABOUNDSBOTTOM | [ integer ] |
| • :DATABOUNDSRIGHT | [ integer ] |
| • :CELLS | [ handle ] |
| • :MAXINDEX | [ integer ] |

**tb:!LNew** *rView dataBounds cSize theProc theWindow drawIt hasGrow scrollHoriz scrollVert* [IV-270] Function

Creates a new list and returns a handle to it. The new list's grafPort is set to *theWindow's* port. The list will be displayed in the rectangle *rView*. *dataBounds* is a rectangle specifying the array dimensions of the list. *cSize* is a point giving the width and height of each cell, in pixels. If *drawIt* is true, the list will be displayed. *ScrollHoriz* and *scrollVert* are boolean values. If they are true, a horizontal scroll bar and a vertical scroll bar will appear. If *hasGrow* is true, the scroll bars are sized to allow room for a size box.

*Example:*

```
;;; Make a new list.  rView is big enough to show five lines.
(setf rView (make-instance 'tb:Rect
                           :left 10   :top 10
                           :right 200 :bottom 90))
;;; The list is one column by ten rows.
(setf dataBounds (make-instance 'tb:Rect
                                :left 0  :top 0
                                :right 1 :bottom 10))
;;; Leave cSize unspecified.
(setf cSize (make-instance 'tb:Point))
;;; Give the list a horizontal and a vertical scroll bar.
(setf myList (tb:!LNew rView dataBounds cSize 0 myWindow
                       nil nil t t))
```

**tb:!LDispose** *lHandle* [IV-271] Function

Disposes of the list data structure.

## Adding and Deleting Rows and Columns

28.3 These routines insert new rows and columns and delete existing rows and columns.

**tb:!LAddColumn** *count colNum lHandle* [IV-271] Function

Inserts *count* number of columns starting at the column specified by *colNum.*

*Example:*

```
;;; Add 1 column.
(tb:!LAddColumn 1 1 myList)
```

**tb:!LAddRow** *count rowNum lHandle* [IV-271] Function

Inserts *count* number of rows starting at the row specified by *rowNum.*

*Example:*

```
;;; Add 10 rows.
(tb:!LAddRow 10 11 myList)
```

**tb:!LDelColumn** *count colNum lHandle* [IV-271] Function

Deletes *count* number of columns starting at the column specified by *colNum.*

**tb:!LDelRow** *count rowNum lHandle* [IV-272] Function

Deletes *count* number of rows starting at the row specified by *rowNum.*

## Operations on Cells

28.4 These routines perform operations on cells.

**tb:!LAddToCell** *dataPtr dataLen theCell lHandle* [IV-272] Function

Appends the data pointed to by *dataPtr*, of length *dataLen*, to the cell specified by *theCell* in *lHandle*.

*Example:*

```
;;; Add 10 bytes of data pointed at by dataPtr to cell (1, 3)
;;; in myList.
(send theCell := 1 3)
(tb:!LAddToCell dataPtr 10 theCell myList)
```

**tb:!LClrCell** *theCell lHandle* [IV-272] Function

Clears the contents of *theCell*.

**tb:LGetCell** *dataPtr dataLen theCell lHandle* [IV-272] Function
**tb:!LGetCell** *dataPtr* VAR *dataLen theCell lHandle* [IV-272] Function

**tb:LGetCell** copies the data in *theCell* to the location specified by *dataPtr*, with *dataLen* specifying the maximum number of bytes allowed and returns the actual number of bytes copied.

**tb:!LGetCell** is similar except it modifies *dataLen* to contain the number of bytes copied.

*Example:*

```
;;; Get the data in cell (1, 3).
(send theCell := 1 3)
(tb:LGetCell dataPtr 10 theCell myList) => 10
```

**tb:!LSetCell** *dataPtr dataLen theCell lHandle* [IV-272] Function

Places the data pointed to by *dataPtr*, with length *dataLen*, into the specified cell *theCell*.

*Example:*

```
;;; Set the data in cell (1, 3) to the 10 bytes pointed at by dataPtr.
(send theCell := 1 3)
(tb:!LSetCell dataPtr 10 theCell myList)
```

**tb:!LCellSize** *cSize lHandle* [IV-273] Function

Sets the cellSize field in the list record.

**tb:!LGetSelect** *next theCell lHandle* [IV-273] Function

If *next* is false, **tb:!LGetSelect** returns true if *theCell* is selected, or false if it is not. If *next* is true, **tb:!LGetSelect** modifies *theCell* to be the cell coordinates of the next selected cell in the row that is greater than or equal to *theCell*. If there are no more cells in the row, it returns in *theCell* the cell coordinates of the next selected cell in the next row. If there are no more rows, **nil** is returned.

*Example:*

```
;;; Any cells selected?
(send theCell := 0 0)
(tb:!LGetSelect t theCell myList) => NIL
```

**tb:!LSetSelect** *setIt theCell lHandle* [IV-273] Function

If *setIt* is true, **tb:!LSetSelect** selects *theCell,* and redraws it if it is visible and was previously unselected. If *setIt* is false, it deselects the cell *theCell* and redraws if necessary.

*Example:*

```
;;; Select cell (1, 3).
(send theCell := 1 3)
(tb:!LSetSelect t theCell myList)
```

## Mouse Location

28.5 These routines respond to a click of the mouse button.

**tb:!LClick** *point modifiers lHandle* [IV-273] Function

Called when there is a mouse-down event in the destination rectangle or its scroll bars, this routine keeps control until the mouse button is released. *Point* is the mouse location in local coordinates. *modifiers* is the modifiers word from the event record. *lHandle* is the list to be tracked. The result is true if a double-click occurred.

*Example:*

```
(setf dblClkFlag (tb:!LClick *event*
                              (send *event* :modifiers)
                             myList))
```

**tb:!LLastClick** *lHandle* [IV-273] Function

Returns the cell coordinates of the last cell clicked in as two values. If no cell has been clicked in since **tb:!LNew**, the value returned is negative.

*Example:*

```
;;; Set theCell to the last cell clicked in.
(multiple-value-bind (v h)
    (tb:!LLastClick myList)
  (send theCell := h v)) => #<POINT x=-1 y=-1>
```

## Accessing Cells

28.6 These routines search for, find, or return cells and cell information.

**tb:LFind** *theCell lHandle* [IV-274] Function
**tb:!LFind** VAR *offset* VAR *len theCell lHandle* [IV-274] Function

**b:LFind** returns two values. Given a cell in *theCell,* it returns the offset and length in bytes of the cell's data. **tb:!LFind** is similar except it modifies *offset* and *len* to be the offset and length values.

**tb:!LNextCell** *hNext vNext theCell lHandle* [IV-274] Function

Given a cell in *theCell*, returns in *theCell* the next cell in the list.

*Example:*

```
;;; Get the cell below (1, 3)
(send theCell := 1 3)
(tb:!LNextCell nil t theCell myList)
theCell => #<POINT x=1 y=4>

;;; Get the cell to the right of (1, 3).
(send theCell := 1 3)
(tb:!LNextCell t t theCell myList)
theCell => #<POINT x=0 y=4>
```

**tb:!LRect** *cellRect theCell lHandle* [IV-274] Function

Returns the local (QuickDraw) coordinates of *theCell* in *cellRect*. If an invalid cell is specified, (0,0) (0,0) is returned in *cellRect*.

**tb:!LSearch** *dataPtr dataLen searchProc theCell lHandle* [IV-274] Function

Searches for the first cell greater than or equal to *theCell* that contains the specified data. If such a cell is found, true is returned and the cell coordinates are returned in *theCell*. If *searchProc* is NIL, the International Utilities Package function **tb:!IUMagIDString** is called to compare the specified data with the contents of each cell. If *searchProc* is not NIL, the routine pointed to by *searchProc* is called.

**tb:!LSize** *listWidth listHeight lHandle* [IV-274] Function

Causes the bottom right of the list to be adjusted so that the list is the height and width indicated by *listWidth* and *listHeight*. The contents of the list and the scroll bars are adjusted and redrawn as necessary. This routine is usually called immediately after the Window Manager procedure **tb:!SizeWindow**.

*Example:*

```
;;; Change the size of the myList's vRect to 200 wide by 150 tall.
(tb:!LSize 200 150 myList)
```

## List Display

28.7 These routines affect the manner in which lists are displayed.

**tb:!LDraw** *theCell lHandle* [IV-275] Function

Makes the List Manager's grafPort the current port, sets the clipping region to the cell's rectangle, and calls the list definition procedure to draw the cell. It restores the clipping region and port before exiting.

**tb:!LDoDraw** *drawIt lHandle* [IV-275] Function

Sets the List Manager's drawing mode to the state specified by *drawIt*. If *drawIt* is true, changes made by most List Manager calls will cause some sort of drawing to take place. If *drawIt* is false, all cell drawing is disabled.

**tb:!LScroll** *cCols dRows lHandle* [IV-275] Function

Scrolls the given list by the number of columns and rows specified by *cCols* and *dRows*.

**tb:!LAutoScroll** *lHandle* [IV-275] Function

Scrolls the list until the first cell is visible.

**tb:!LUpdate** *theRgn lHandle* [IV-275] Function

Redraws any visible cells in *lHandle* that intersect *theRgn* and redraws the controls, if necessary.

**tb:!LActivate** *act lHandle* [IV-276] Function

Activates or deactivates the list specified by *lHandle*. The *act* parameter should be set to true to activate the list, or false to deactivate the list. Call this trap when receiving an activate event for a window which contains a list.

# Chapter 29
# ERROR HANDLING

## Introduction

29.1 The Macintosh Toolbox uses several different methods to signal errors. The File Manager, for example, returns OSErrs (negative numbers) or 0 to indicate that no errors exist. In contrast, the Resource Manager stores the error code for the last Resource Manager call in a low memory location that is accessed by **!ResError**.

In order to simplify proper error checking, the Toolbox Interface is linked into the Texas Instruments error handling system. For more information on the error handling system see in the Texas Instruments *Explorer Lisp Reference* manual.

All Toolbox calls that return an OSErr automatically check for a zero result. If an error is detected and if **tb:*signal-mac-oserr*** is true, it signals an **tb:OSErr** condition which displays the *Inside Macintosh* name and comment for the particular error. After the execution of any Resource Manager trap that returns a ResError, the error handling mechanism checks for an error. If an error exists, a signal is generated in the standard Common Lisp manner. This checking is done internally without the overhead of calling **!ResError**.

**tb:*signal-mac-oserr*** Variable

If this variable is true, then the traps which are documented to return result codes will signal a **tb:OSErr** condition is that result code is negative. If the result code is non-negative, then it is returned.

If this variable is false, the these traps unconditionally return their result codes regardless of value.

## Signaling an Error

29.2 The following routine allows you to explicitly signal an error condition. Notice that none of the following flavors, methods, or functions observe **tb:*signal-mac-oserr***. This trap function code uses **tb:*signal-mac-oserr*** to decides whether to call **tb:signal-oserr** or not, but **tb:signal-oserr** and all processing it initiates ignores the variable.

**tb:toolbox-error** Condition

This is the flavor on which all Toolbox Interface error signals are built. This flavor is built on **lisp:error**.

**tb:toolbox-warn** Condition

This is the flavor on which all Toolbox Interface warning signals are built. This flavor is built on **sys:warn**.

**tb:OSErr** Condition

This is the flavor that records Toolbox Interface result code errors. It is built on **tb:toolbox-error**. When this flavor is instantiated, the **:oserr** initialization option is used to look up the associated signal names and error message in **tb:*OSErr-alist***. This table lookup approach avoids having to define all Macintosh result codes with **defsignal** forms.

This condition offers a **:no-action** proceed type which causes the signal to simply return with the original result code.

**:oserr** Init Option of **tb:OSErr**
**:oserr** Method of **tb:OSErr**

This value is the non-zero integer result code which caused **tb:OSErr** to be signaled. This is a required initialization option.

**:trap-symbol** Init Option of **tb:OSErr**
**:trap-symbol** Method of **tb:OSErr**

This value is the name of the Toolbox Interface function which signaled the **tb:OSErr** as a symbol. This symbol can be used to disambiguate identical result codes which are signaled by different traps.

**tb:*OSErr-alist*** Variable

This association list is used to associate the integer result codes returned by various traps with error signal names and a default error message. Each entry in this list has the form:

(*oserr signal-name message*)

where *oserr* is the integer result code, *signal-name* is the symbol or list of symbols representing the Macintosh result code mnemonic symbols, and *message* is a brief test string describing the error.

The list of signal names in one entry typically includes some additional symbols which classify the type of the signal. For example, all result code symbols associated with file system errors will carry the additional signal name of **tb:!FS-Error**. These classification signal names can be used in error handlers to intercept whole families of errors without having to enumerate each individual signal.

**tb:find-oserr** *id* Function

This function allows convenient interrogation of the OSErr database represented by **tb:*OSErr-alist***. If *id* is sufficient to identify one or more entries in the alist, then a list of those entries is returned. Otherwise, it returns **nil**. An entry is formatted as follows:

(*oserr signal-name message-string*)

where *signal-name* is a symbol or list of symbols.

If *id* is an integer, then it is taken to be an OSErr and the unique entry corresponding to that OSErr is returned as a one element list. If *id* is a symbol, then it is taken to be a signal name and all entries which include that signal name are returned in a list. If *id* is a string, then it is taken to be a substring of an error message and all entries which include that substring in their message are returned in a list.

**tb:signal-oserr** *oserr trap-symbol* &optional *format-string* &rest *args* Function

If *oserr* is non-negative, the function does nothing and returns zero. Otherwise, the function signals a **tb:OSErr** error condition with appropriate auxiliary information. *oserr* is used to look up the signal names and default message strings from **tb:*oserr-alist***. If *format-string* is specified, it overrides the default message in **tb:*oserr-alist***. *Trap-symbol* is the name of the Macintosh trap which returned *oserr* and is available in the condition object to clarify duplicate result codes. If an error is signaled and a handler chooses the **:no-action** proceed type, then *oserr* is returned.

## Suppressing Errors

29.3 Many times you will want to bypass the automatic error handling provided and handle some OSErrs with your own code. To make handling these errors easier, the following macros are provided.

**tb:suppress-oserr** &body *body* Macro
**tb:suppress-oserr-if** *condition-form* &body *body* Macro

If an **tb:OSErr** error condition is signaled inside of *body*, then **tb:suppress-oserr** automatically responds with a proceed type of **:no-action**. That is, the processing of *body* proceeds without signaling errors.

**tb:suppress-oserr-if** is similar except it resumes only if *condition-form* is true.

**tb:suppress-some-oserrs** *error-list* &body *body* Macro

If an **tb:OSErr** error condition on *error-list* is signaled inside of *body*, then *body* is automatically resumed with a proceed type of **:no-action**.

In the example below, if **!GetVolInfo** returns the error code **tb:!nsvErr**, it will not be signaled.

*Example:* `(suppress-some-oserrs (tb:!nsvErr) (tb:!GetVolInfo pb))`

## Restarting From an Error

29.4 The following macros allow you to restart after an error.

**tb:oserr-restart** *format-string format-args* &body *body* Macro
**tb:oserr-restart-if** *cond-form format-string format-args* &body *body* Macro

**tb:oserr-restart** executes *body*, with a restart for **tb:OSErr** in effect that will try *body* over. *format-string* and *format-args* are used to identify this proceed option, enabling the user to decide whether or not to use the restart.

If the user chooses to go to the restart provided, **tb:oserr-restart** throws back to the top of *body* and *body* is executed again. If *body* returns normally, the values of the last form in *body* are returned from the **tb:oserr-restart**.

**tb:oserr-restart-if** is similar except that the proceed option is offered only if *cond-form* is true.

# Appendix A
# RESOURCE TYPES

The following resource types have been predefined. Notice that in resource type strings, both case and blanks are significant.

| | |
|---|---|
| "actb" | Alert color table |
| "ADBS" | Apple Desktop Bus™ service routine |
| "ALRT" | Alert template |
| "atpl" | Internal AppleTalk® resource |
| "bmap" | Bit maps used by the Control Panel |
| "BNDL" | Bundle |
| "boot" | Copy of boot blocks |
| "CACH" | RAM cache code |
| "cctb" | Control color table |
| "CDEF" | Control definition function |
| "cicn" | Color Macintosh icon |
| "clst" | Cached icon lists used by Chooser and Control Panel |
| "clut" | Color look-up table |
| "CNTL" | Control template |
| "CODE" | Application code segment |
| "crsr" | Color cursor |
| "ctab" | Used by the Control Panel |
| "CURS" | Cursor |
| "dctb" | Dialog color table |
| "DITL" | Item list in a dialog or alert |
| "DLOG" | Dialog template |
| "DRVR" | Desk accessory or other device driver |
| "DSAT" | System startup alert table |
| "fctb" | Font color table |
| "finf" | Font information |
| "FKEY" | Command-Shift-number routine |
| "FMTR" | 3 1/2-inch disk formatting code |
| "FOND" | Font family record |
| "FONT" | Font |
| "FREF" | File reference |
| "FRSV" | IDs of fonts reserved for system use |
| "FWID" | Font widths |
| "gama" | Color correction table |
| "ICN#" | Icon list |
| "ICON" | Icon |
| "ictb" | Color table dialog item |
| "INIT" | Initialization resource |
| "insc" | Installer script |
| "INTL" | International resource |
| "INT#" | List of integers owned by Find File |
| "itl0" | Date and time formats |

| | |
|---|---|
| "itl1" | Names of days and months |
| "itl2" | International Utilities Package sort hooks |
| "itlb" | International Utilities Package script bundles |
| "itlc" | International configuration for Script Manager |
| "KCAP" | Physical layout of keyboard (used by Key Caps desk accessory) |
| "KCHR" | ASCII mapping (software) |
| "KMAP" | Keyboard mapping (hardware) |
| "KSWP" | Keyboard script table |
| "LDEF" | List definition procedure |
| "lmem" | Low memory globals |
| "MBAR" | Menu bar |
| "MBDF" | Default menu definition procedure |
| "mcky" | Mouse tracking |
| "mctb" | Menu color information table |
| "MDEF" | Menu definition procedure |
| "MENU" | Menu |
| "mitq" | Internal memory requirements for MakeITable |
| "MMAP" | Mouse tracking code |
| "mppc" | AppleTalk configuration code |
| "NBPC" | AppleTalk bundle |
| "NFNT" | 128K ROM font |
| "nrct" | Rectangle positions |
| "PACK" | Package |
| "PAT " | Pattern (the space is required) |
| "PAT#" | Pattern list |
| "PDEF" | Printing code |
| "PICT" | Picture |
| "pltt" | Color palette |
| "ppat" | Pixel pattern |
| "PREC" | Print record |
| "PRER" | Device type for Chooser |
| "PRES" | Device type for Chooser |
| "PTCH" | ROM patch code |
| "RDEV" | Device type for Chooser |
| "ROvr" | Code for overriding ROM resources |
| "ROv#" | List of ROM resources to override |
| "SERD" | RAM Serial Driver |
| "SICN" | Script symbol |
| "snd " | Sound (the space is required) |
| "snth" | Synthesizer |
| "STR " | String (the space is required) |
| "STR#" | String list |
| "wctb" | Window color table |
| "WDEF" | Window definition function |
| "WIND" | Window template |

# Appendix B
# RESULT CODES

When a result code is signaled as a **tb:OSErr** condition (e.g., by **tb:signal-oserr**), the symbolic result code name is attached to that error condition instance as a signal name. For example, using **tb:signal-oserr** to signal a result code of -1 would result in the error condition **tb:!qErr** being signaled. This error signal could be handled by any condition handler listing **tb:!qErr** as one of its conditions.

Some result codes have several signal names associated with them. A condition handler listing any of these alternative signal names can handle such a signal. For example, result codes -17 though -22 will signal errors **tb:!controlErr** through **tb:!unitEmptyErr** as described above. Furthermore, there will also be an additional signal name of **tb:!DM-Error** associated with each of these signals meaning that a condition handler for **tb:!DM-Error** will intercept all driver errors.

The mapping of result code numbers, associated signal names, and error messages is maintained in the association list **tb:*OSErr-alist***.

NOTE: Result codes are not necessarily unique. For example, -1 represents both **tb:!qErr**, "Queue element not found during deletion", and **tb:!iPrSavPFil**, "Problem saving print file". Therefore, signaling a result code of -1 will result in a condition instance with both signal names attached. The **:trap-symbol** instance variable of the condition instance may help to disambiguate the result code.

## General System Errors (VBL Mgr, Queueing, Etc.)

| | | |
|---|---|---|
| tb:!noErr | 0 | No error |
| tb:!qErr | -1 | Queue element not found during deletion |
| tb:!vTypErr | -2 | Invalid queue element |
| tb:!corErr | -3 | Core routine number out of range |
| tb:!tunimpErr | -4 | Unimplemented core routine |
| tb:!seNoDB | -8 | No debugger installed to handle Debugger command |

## IO System Errors

| | | |
|---|---|---|
| tb:!controlErr | -17 | Driver can't respond to this control call |
| tb:!statusErr | -18 | Driver can't respond to this status call |
| tb:!readErr | -19 | Driver can't respond to this read call |
| tb:!writErr | -20 | Driver can't respond to this write call |
| tb:!badUnitErr | -21 | Driver reference number doesn't match unit table |
| tb:!unitEmptyErr | -22 | Driver reference number specifies NIL handle in unit table |
| tb:!openErr | -23 | Requested r/w permission doesn't match driver's open permission |
| tb:!closErr | -24 | |
| tb:!!dRemovErr | -25 | Tried to remove an open driver |
| tb:!tdInstErr | -26 | DrvrInstall couldn't find driver in resources |
| tb:!abortErr | -27 | IO call aborted by KillIO |
| tb:!iIOAbortErr | -27 | IO abort error (Printing Manager) |

| | | |
|---|---|---|
| tb:!notOpenErr | -28 | Couldn't rd/wr/ctl/sts because driver was not opened |

## File System Error Codes

| | | |
|---|---|---|
| tb:!dirFulErr | -33 | Directory full |
| tb:!dskFulErr | -34 | Disk full |
| tb:!nsvErr | -35 | No such volume |
| tb:!ioErr | -36 | IO error (bummers) |
| tb:!bdNamErr | -37 | There may be no bad names in the final system! |
| tb:!fnOpnErr | -38 | File not open |
| tb:!eofErr | -39 | End of file |
| tb:!posErr | -40 | Tried to position to before start of file (r/w) |
| tb:!mFulErr | -41 | Memory full (open) or file won't fit (load) |
| tb:!tmfoErr | -42 | Too many files open |
| tb:!fnfErr | -43 | File not found |
| tb:!wPrErr | -44 | Diskette is write-protected |
| tb:!fLckdErr | -45 | File is locked |
| tb:!vLckdErr | -46 | Volume is locked |
| tb:!fBsyErr | -47 | File is busy (delete) |
| tb:!dupFNErr | -48 | Duplicate filename (rename) |
| tb:!opWrErr | -49 | File already open with write permission |
| tb:!paramErr | -50 | Error in user parameter list |
| tb:!rfNumErr | -51 | Refnum error |
| tb:!gfpErr | -52 | Get file position error |
| tb:!volOffLinErr | -53 | Volume not on line error (was ejected) |
| tb:!permErr | -54 | Permissions error (on file open) |
| tb:!volOnLinErr | -55 | Drive volume already on-line at MountVol |
| tb:!nsDrvErr | -56 | No such drive (tried to mount a bad drive num) |
| tb:!noMacDskErr | -57 | Not a Macintosh diskette (signature bytes are wrong) |
| tb:!extFSErr | -58 | Volume in question belongs to an external fs |
| tb:!fsRnErr | -59 | File system internal error: during rename the old entry was deleted but could not be restored |
| tb:!badMDBErr | -60 | Bad master directory block |
| tb:!wrPermErr | -61 | Write permissions error |

## Font Manager Error Codes

| | | |
|---|---|---|
| tb:!fontDecError | -64 | Error during font declaration |
| tb:!fontNotDeclared | -65 | Font not declared |
| tb:!fontSubErr | -66 | Font substitution occurred |

## Disk, Serial Ports, Clock Specific Errors

| | | |
|---|---|---|
| tb:!firstDskErr | -84 | First in the range of low-level disk errors |
| tb:!lastDskErr | -64 | Last in the range of low-level disk errors |
| tb:!noDriveErr | -64 | Drive not installed |
| tb:!offLinErr | -65 | R/w requested for an off-line drive |
| tb:!noNybErr | -66 | Couldn't find 5 nibbles in 200 tries |
| tb:!noAdrMkErr | -67 | Couldn't find valid addr mark |
| tb:!dataVerErr | -68 | Read verify compare failed |
| tb:!badCksmErr | -69 | Addr mark checksum didn't check |
| tb:!badBtSlpErr | -70 | Bad addr mark bit slip nibbles |
| tb:!noDtaMkErr | -71 | Couldn't find a data mark header |
| tb:!badDCksum | -72 | Bad data mark checksum |

| | | |
|---|---|---|
| tb:!badDBtSlp | -73 | Bad data mark bit slip nibbles |
| tb:!wrUnderrun | -74 | Write underrun occurred |
| tb:!cantStepErr | -75 | Step handshake failed |
| tb:!tk0BadErr | -76 | Track 0 detect doesn't change |
| tb:!initIWMErr | -77 | Unable to initialize IWM |
| tb:!twoSideErr | -78 | Tried to read 2nd side on a 1-sided drive |
| tb:!spdAdjErr | -79 | Unable to correctly adjust disk speed |
| tb:!seekErr | -80 | Track number wrong on address mark |
| tb:!sectNFErr | -81 | Sector number never found on a track |
| tb:!fmt1Err | -82 | Can't find sector 0 after track format |
| tb:!fmt2Err | -83 | Can't get enough sync |
| tb:!verErr | -84 | Track failed to verify |
| tb:!clkRdErr | -85 | Unable to read same clock value twice |
| tb:!clkWrErr | -86 | Time written did not verify |
| tb:!prWrErr | -87 | Parameter RAM written didn't read-verify |
| tb:!prInitErr | -88 | InitUtil found the parameter RAM uninitialized |
| tb:!rcvrErr | -89 | SCC receiver error (framing, parity, OR) |
| tb:!breakRecd | -90 | Break received (SCC) |

## Scrap Manager Error Codes

| | | |
|---|---|---|
| tb:!noScrapErr | -100 | No scrap exists error |
| tb:!noTypeErr | -102 | No object of that type in scrap |

## Storage Allocator Error Codes

| | | |
|---|---|---|
| tb:!memFullErr | -108 | Not enough room in heap zone |
| tb:!nilHandleErr | -109 | Handle was NIL in **tb:!HandleZone** or other |
| tb:!memWZErr | -111 | **tb:!WhichZone** failed (applied to free block) |
| tb:!memPurErr | -112 | Trying to purge a locked or non-purgeable block |
| tb:!memAdrErr | -110 | Address was odd or out of range |
| tb:!memAZErr | -113 | Address in zone check failed |
| tb:!memPCErr | -114 | Pointer Check failed |
| tb:!memBCErr | -115 | Block Check failed |
| tb:!memSCErr | -116 | Size Check failed |
| tb:!memLockedErr | -117 | Trying to move a locked block (**tb:!MoveHHi**) |

## New System Error Codes

| | | |
|---|---|---|
| tb:!dirNFErr | -120 | Directory not found |
| tb:!tmwdoErr | -121 | No free WDCB available |
| tb:!badMovErr | -122 | Move into offspring error |
| tb:!wrgVolTypErr | -123 | Wrong volume type error (operation not supported for MFS) |
| tb:!volGoneErr | -124 | Server volume has been disconnected |

### Resource Manager Error Codes (Other than I/O errors)

| | | |
|---|---|---|
| tb:!resNotFound | -192 | Resource not found |
| tb:!resFNotFound | -193 | Resource file not found |
| tb:!addResFailed | -194 | **tb:!AddResource** failed |
| tb:!addRefFailed | -195 | AddReference failed |
| tb:!rmvResFailed | -196 | **tb:!RmveResource** failed |
| tb:!rmvRefFailed | -197 | RmveReference failed |
| tb:!resAttrErr | -198 | Attribute inconsistent with operation |
| tb:!mapReadErr | -199 | Map inconsistent with operation |

### Miscellaneous Result Codes

| | | |
|---|---|---|
| tb:!evtNotEnb | 1 | Event not enabled at **tb:!PostEvent** |

### Color Quickdraw and Color Manager Errors

| | | |
|---|---|---|
| tb:!cMatchErr | -150 | **tb:!Color2Index** failed to find an index |
| tb:!cTempMemErr | -151 | Failed to allocate memory for temporary structures |
| tb:!cNoMemErr | -152 | Failed to allocate memory for structure |
| tb:!cRangeErr | -153 | Range error on colorTable request |
| tb:!cProtectErr | -154 | ColorTable entry protection violation |
| tb:!cDevErr | -155 | Invalid type of graphics device |
| tb:!cResErr | -156 | Invalid resolution for **tb:!MakeITable** |

### Errors for Color2Index/ITabMatch

| | |
|---|---|
| tb:!iTabPurgErr | -9 |
| tb:!noColMatch | -10 |

### Errors for MakeITable

| | |
|---|---|
| tb:!qAllocErr | -11 |
| tb:!tblAllocErr | -12 |
| tb:!overRun | -13 |
| tb:!noRoomErr | -14 |

### Errors for SetEntry

| | |
|---|---|
| tb:!seOutOfRange | -15 |
| tb:!seProtErr | -16 |
| tb:!i2CRangeErr | -17 |
| tb:!gdBadDev | -18 |
| tb:!reRangeErr | -19 |
| tb:!seInvRest | -20 |
| tb:!seNoMemErr | -21 |

### More Errors

| | | |
|---|---|---|
| tb:!unitTblFullErr | -29 | Unit table has no more entries |
| tb:!dceExtErr | -30 | DCE extension error |
| tb:!dsBadSlotInt | 51 | Unserviceable slot interrupt |
| tb:!dsBadSANEopcode | 81 | Bad opcode given to SANE Pack4 |
| tb:!memROZWarn | -99 | Soft error in ROZ |
| tb:!memROZError | -99 | Hard error in ROZ |
| tb:!updPixMemErr | -125 | Insufficient memory to update a pixmap |

## Menu Manager

| | | |
|---|---|---|
| tb:!mBarNFnd | -126 | System error code for MBDF not found |
| tb:!hMenuFindErr | -127 | Could not find HMenu's parent in tb:!MenuKey |

## Sound Manager Error Returns

| | | |
|---|---|---|
| tb:!noHardware | -200 | No hardware support for this synthesizer |
| tb:!notEnoughHardware | -201 | No more channels for this synthesizer |
| tb:!queueFull | -203 | No room in the queue |
| tb:!resProblem | -204 | Problem loading resource |
| tb:!badChannel | -205 | Invalid channel queue length |
| tb:!badFormat | -206 | Handle to "snd " resource was invalid |

## Errors Specific to the Start Manager

The following errors may be generated during system Init. If they are, they will be logged into the sInfo array and returned each time a call to the slot manager is made (for the card which generated the error).

| | | |
|---|---|---|
| tb:!smSDMInitErr | -290 | Error, SDM could not be initialized |
| tb:!smSRTInitErr | -291 | Error, Slot Resource Table could not be initialized |
| tb:!smPRAMInitErr | -292 | Error, Slot Resource Table could not be initialized |
| tb:!smPriInitErr | -293 | Error, Cards could not be initialized |
| tb:!smEmptySlot | -300 | No card in slot |
| tb:!smCRCFail | -301 | CRC check failed for declaration data |
| tb:!smFormatErr | -302 | FHeader Format is not Apple's |
| tb:!smRevisionErr | -303 | Wrong revision level |
| tb:!smNoDir | -304 | Directory offset is Nil |
| tb:!smLWTstBad | -305 | Long Word test field <> #x5A932BC7 |
| tb:!smNosInfoArray | -306 | No sInfoArray. Memory Mgr error |
| tb:!smResrvErr | -307 | Fatal reserved error. Reserved field <> 0 |
| tb:!smUnExBusErr | -308 | Unexpected BusError |
| tb:!smBLFieldBad | -309 | ByteLanes field was bad |
| tb:!smFHBlockRdErr | -310 | Error occurred during _sGetFHeader |
| tb:!smFHBlkDispErr | -311 | Error occurred during _sDisposePtr (Dispose of FHeader block) |
| tb:!smDisposePErr | -312 | _DisposePointer error |
| tb:!smNoBoardsRsrc | -313 | No Board sResource |
| tb:!smGetPRErr | -314 | Error occurred during _sGetPRAMRec (See SIMStatus) |
| tb:!smNoBoardId | -315 | No Board Id |
| tb:!smIntStatVErr | -316 | The InitStatusV field was negative after primary or secondary init |
| tb:!smIntTblVErr | -317 | An error occurred while trying to initialize the Slot Resource Table |
| tb:!smNoJmpTbl | -318 | SDM jump table could not be created |
| tb:!smBadBoardId | -319 | BoardId was wrong, re-init the PRAM record |
| tb:!smBusErrTO | -320 | BusError time out |

The following errors may be generated at any time after system Init and will not be logged into the sInfo array.

| | | |
|---|---|---|
| tb:!smBadRefId | -330 | Reference Id not found in List |
| tb:!smBadsList | -331 | Bad sList: Id1 < Id2 < Id3 ... format is not followed |
| tb:!smReservedErr | -332 | Reserved field not zero |
| tb:!smCodeRevErr | -333 | Code revision is wrong |
| tb:!smCPUErr | -334 | Code revision is wrong |
| tb:!smsPointerNil | -335 | LPointer is nil (From sOffsetData. If this error occurs, check sInfo rec for more information.) |
| tb:!smNilsBlockErr | -336 | Nil sBlock error (Don't allocate and try to use a nil sBlock) |
| tb:!smSlotOOBErr | -337 | Slot out of bounds error |
| tb:!smSelOOBErr | -338 | Selector out of bounds error |
| tb:!smNewPErr | -339 | _NewPtr error |
| tb:!smBlkMoveErr | -340 | _BlockMove error |
| tb:!smCkStatusErr | -341 | Status of slot = fail |
| tb:!smGetDrvrNamErr | -342 | Error occurred during _sGetDrvrName |
| tb:!smDisDrvrNamErr | -343 | Error occurred during _sDisDrvrName |
| tb:!smNoMoresRsrcs | -344 | No more sResources |
| tb:!smsGetDrvrErr | -345 | Error occurred during _sGetDriver |
| tb:!smBadsPtrErr | -346 | Bad pointer was passed to sCalcsPointer |
| tb:!smByteLanesErr | -347 | NumByteLanes was determined to be zero |
| tb:!smOffsetErr | -348 | Offset was too big (temporary error, should be fixed) |
| tb:!smNoGoodOpens | -349 | No opens were successful in the loop |
| tb:!smSRTOvrFlErr | -350 | SRT overflow |
| tb:!smRecNotFnd | -351 | Record not found in the SRT |

## Device Manager Slot Support Error

| | | |
|---|---|---|
| tb:!slotNumErr | -360 | Invalid slot # error |

## SysEnvirons Errors

| | | |
|---|---|---|
| tb:!envNotPresent | -5500 | Returned by glue |
| tb:!envBadVers | -5501 | Version non-positive |
| tb:!envVersTooBig | -5502 | Version bigger than call can handle |

# INDEX