Using and Writing Cortland Tools Preliminary Notes

Preliminary Note: 1/16/86

Writer: Willam H. Harris
Apple User Education

Changes Since Last Draft

This is the first draft of this document. The sources used to prepare this document are as follows:

Tool Locator ERS	12/3/85
QuickDraw II ERS	1/15/86
Memory Manager ERS	11/27/85
Event Manager ERS	11/25/85
Miscellaneous Tools	1/10/86

Contents

Preface

This manual is an introduction to the Cortland Tools for the application developer. It defines the terms used in describing the tools and provides background information. If you are planning on simply using the Cortland Tools that Apple provides, Chapters 1 and 2 will provide you with enough information. If you are planning on developing your own tools, you will need to read Chapter 3.

For a more complete description of some the Apple tools, refer to *Description of the Cortland Tools: Part I Preliminary Notes.*

Please note that this manual is only a preliminary document. The information may change for the final release. The *Preliminary Notes* give you an idea of the powerful capabilities of the Cortland Tools so that you can begin planning your application.

Chapter 1

Introducing the Tools

What Is a "Tool", Anyway?

A software "Tool", in the Cortland environment, is a collection of logically related routines (or functions) comprising one major capability. Each function is an "entry point" of the tool set that performs a fundamental operation and converts zero or more inputs to zero or more outputs and side effects. For example, the QuickDraw II Tool provides functions that handle graphics on the Cortland. Within that tool, **PenSize** and **PenMode** are functions that set the pen size and pen mode.

If you are familiar with Macintosh programming, this concept is similar to the Toolbox. In the Cortland implementation, the concept is even more important. Many of the capabilities of the Cortland (when it runs as a Cortland, and is not emulating an Apple II) are easily available through the tools. For example, even the Memory and Event Managers are considered to be tools on the Cortland.

What Can the Tools Do For Me?

The tools are fast and simple to use. They provide powerful capabilities that allow the application to concentrate on its specific business rather than having to do all of the background work . To use the tools in the simplest fashion, you don't need to know anything but the name for the tool and how to call it from the particular language (assembly, Pascal, or C) that you're using. (Calling information is in Chapter 2.)

A number of the tools are included in ROM. This approach makes the tools available to all programs without using disk space and without the need to link tool libraries to applications.

Other tools are available in RAM (at the moment, what the RAM tools will be is still being decided). However, the structure of the tools is such that you don't need to keep track of where a particular function is or even if it is in ROM or RAM. This magic is performed by a tool called the Tool Locator, which allows tools and applications to communicate. Because the Tool Locator does its work quietly, you won't even see it if you are simply using the Apple tools in your application.

There is another advantage of the tools and the Tool Locator. In addition to using the Apple tools, you can add on your own tools if you wish. You don't even need to replace the Apple tools with your tools; both can be available at once.

The Cortland tools are independent of the operating system being used. They are thus available for any Cortland application, whether the application is running under ProDOS, Pascal, or another operating system.

What Tools Will Apple Provide?

In this section, we simply list and define all of the tools that are currently planned. The listing does not contain extensive description, syntax, or any examples; for an in-depth look at an individual tool, refer to *Description of the Cortland Tools: Part I Preliminary Notes*.

Please remember that this list is still preliminary. Other functions or entire tools may be added as space permits, some functions may be removed, and the parameters for others may change. We can generally assure you that the Tools listed here will be available, and we are trying to provide enough information in these Preliminary Notes to give you a head start on developing your application.

Tool Locator

BootInit	Initializes the Tool Locator and all other ROM-based tool sets.
AppInit	Does nothing.
AppEnd	Does nothing.
Version	Returns the version of the Tool Locator.
GetTsPtr	Returns pointer to the Function Pointer Table of the specified tool set.
SetTSPtr	Installs the pointer to a Function Pointer Table in the appropriate Tool Pointer Table.

QuickDraw II

Housekeeping Functions

QDBootInit	Initializes QuickDraw II at boot time.
QDApInit	Initializes QuickDraw II, sets the current port to the standard port, and clears the screen.
QDQuit	Frees up any buffers that were allocated.
QDVersion	Returns the version of QuickDraw II.

Global Environment Calls

GetStandardSCB	Returns a copy of the standard SCB in the low byte of the word.
SetMasterSCB	Sets the master SCB to the specified value (only the low byte is used).

GetMasterSCB	Returns a copy of the master SCB (only the low byte is valid).
InitColorTable	Returns a copy of the standard color table for the current mode.
SetColorTable	Sets the color table to specified values.
GetColorTable	Fills a color table with the contents of another color table.
SetColorEntry	Sets the value of a color in a specified color table.
GetColorEntry	Returns the value of a color in a specified color table .
SetSCB	Sets the scan line control byte (SCB) to a specified value.
GetSCB	Returns the value of a specified scan line control byte (SCB).
SetAllSCBs	Sets all scan line control bytes (SCBs) to a specified value.

GrafPort Functions

OpenPort	Initializes specified memory locations as a standard port and allocates new VisiRgn and ClipRgn.
InitPort	Initializes specified memory locations as a standard port.
ClosePort	Deallocates the memory associated with a port.
SetPort	Makes the specified port the current port.
GetPort	Retürns the handle to the current port.
SetPortInfo	Sets the current port's map information structure to the specified location information.
SetPortSize	Changes the size of the current GrafPort's PortRect.
MovePortTo	Changes the location of the current GrafPort's PortRect.
SetOrigin	Adjusts the contents of PortRect and BoundsRect so that the upper left corner of PortRect is set to the specified point.
SetClip	Sets the clip region to the region passed.
GetClip	Returns a handle to the current clip region.
ClipRect	Changes the clip region of the current GrafPort to a rectangle equivalent to a given rectangle.

Cursor-Handling Routines

SetCursor	Sets the cursor to the image passed in the cursor record.
GetCursorAdr	Returns a pointer to the current cursor record.
HideCursor	Decrements the cursor level. A cursor level of zero indicates the cursor is visible; a cursor level less than zero indicates the cursor is not visible.
ShowCursor	Increments the cursor level unless it is already zero. A cursor level of zero indicates the cursor is visible; a cursor level less than zero indicates the cursor is not visible.
ObscureCursor	Hides the cursor until the mouse moves. This tool is used to get the cursor out of the way of typing.

Pen, Pattern, and Drawing Mode Functions

HidePen	Decrements the pen level. A pen level of zero indicates drawing will occur; a pen level less than zero indicates drawing will not occur.
ShowPen	Increments the pen level unless it is already zero. A pen level of zero indicates that drawing will occur; a pen level less than zero indicates drawing will not occur.
GetPen	Returns the pen location.
SetPenState	Sets the pen state in the GrafPort to the values passed.
GetPenState	Returns the pen state from the GrafPort.
PenSize	Sets the current pen size to the specified pen size.
PenMode	Sets the current pen mode to the specified pen mode.
PenPat	Sets the current pen mode to the specified pen mode.
BackPat	Sets the background pattern to the specified pattern.
PenNormal	Sets the pen state to the standard state (PnSize = 1,1; PnMode = copy; PnPat = Black). The pen location is not changed.
MoveTo	Moves the current pen location to the specified point.
Move	Moves the current pen location by the specified horizontal and vertical displacements.
LineTo	Draws a line from the current pen location to the specified point.
Line	Draws a line from the current pen location to a new point specified by the horizontal and vertical displacements.

Calculations With Rectangles

SetRect	Sets the rectangle pointed to by RectPtr to the specified values.
OffsetRect	Offsets the rectangle pointed to by RectPtr by the specified displacements.
InsetRect	Insets the rectangle pointed to by RectPtr by the specified displacements.
SectRect	Calculates the intersection of two rectangles and places the intersection in a third rectangle.
UnionRect	Calculates the union of two rectangles and places the union in a third rectangle.
PtInRect	Detects whether a specified point is in a specified rectangle.
Pt2Rect	Copies one point to the upper left of a specified rectangle and another point to the lower right of the rectangle.
EqualRect	Compares two rectangles and returns TRUE or FALSE.
EmptyRect	Returns whether or not a specified rectangle is empty.

Rectangle Functions

FrameRect	Draws the boundary of the specified rectangle with the current pattern and pen size.
PaintRect	Paints (fills) the interior of the specified rectangle with the current pen pattern.
EraseRect	Paints (fills) the interior of the specified rectangle with the background pattern.
InvertRect	Inverts the pixels in the interior of the specified rectangle.
FillRect	Paints (fills) the interior of the specified rectangle with the specified pattern.

Pixel Transfer Calls

ScrollRect	Shifts the pixels inside the intersection of a specifed rectangle, VisRgn, ClipRgn, PortRect, and BoundsRect.
PaintPixels	Transfers a region of pixels.

Calculations With Points

AddPt	Adds two specified points together and leaves the result in the destination point.
SubPt	Subtracts the source point from the destination point and leaves the result in the destination point.
SetPt	Sets a point to specified horizontal and vertical values.
EqualPt	Returns a boolean result indicating whether two points are equal.
LocalToGlobal	Converts a point from local coordinates to global coordinates.
GlobalToLocal	Converts a point from global coordinates to local coordinates.

Miscellaneous Utilities

Random	Returns a pseudo-random number in the range –32768 to 32767.
SetRandSeed	Sets the seed value for the **Random** function.
GetPixel	Returns the pixel below and to the right of the specifed point.

Calculations With Regions

NewRgn	Allocates space for a new region and initializes it to the empty region. This is the only way to create a new region.
DisposeRgn	Deallocates space for the specified region.
CopyRgn	Copies the contents of a region from one region to another.
SetEmptyRgn	Destroys the previous region information by setting it to the empty region.
SetRectRgn	Destroys the previous region information by setting it to a rectangle described by the inputs.
RectRgn	Destroys the previous region information by setting it to a rectangle described by the input.
OpenRgn	Tells QuickDraw II to allocate temporary space and start saving lines and framed shapes for later processing as a region definition.

CloseRgn	Tells QuickDraw II to stop processing information and to return the region that has been created.
OffsetRgn	Moves the region on the coordinate plane a distance of dh horizontally and dv vertically.
InsetRgn	Shrinks or expands a region.
SectRgn	Calculates the intersection of two regions and places the intersection in the third region.
UnionRgn	Calculates the union of two regions and places the union in the third region.
DiffRgn	Calculates the difference of two regions and places the difference in the third region.
XorRgn	Calculates the difference between the union and the intersection of two regions and places the result in the third region.
PtInRgn	Checks to see whether the pixel below and to the right of the point is within the specified region.
RectInRgn	Checks whether a given rectangle intersects a specified region.
EqualRgn	Compares the two regions and returns TRUE if they are equal or FALSE if not.
EmptyRgn	Checks to see if a region is empty.

Graphic Operations on Region Calls

FrameRgn	Draws the boundary of the specified region with the current pattern and current pen size.
PaintRgn	Paints (fills) the interior of the specified region with the current pen pattern.
EraseRgn	Fills the interior of the specified region with the backgound pattern.
InvertRgn	Inverts the pixels in the interior of the specified region.
FillRgn	Fills the interior of the specified region with the specfied pattern.

Memory Manager

Housekeeping Functions

MMInit	Called at boot time. Does nothing.
MMStartUp	Initializes the Memory Manager.
MMShutDown	Turns off the Memory Manager.
MMVersion	Returns the version of the Memory Manager.

Memory Allocating Functions

NewHandle	Creates a new block.
ReAllocHandle	Reallocates a block that was purged.

Memory Freeing Functions

DisposHandle	Purges a specified unlocked block and deallocates the handle.
DisposAll	Discards all of the handles for a specific owner.
PurgeHandle	Purges a specified unlocked block, but does not deallocate the handle.
PurgeAll	Purges all of the purgeable blocks for a specific owner.

Block Information Functions

GetHandleSize	Returns the size of a block.
SetHandleSize	Changes the size of a block.
FindHandle	Returns the handle of the block containing a specified address.

Locking and Unlocking Functions

HLock	Locks a block specified by a handle.
HLockAll	Locks all of the blocks owned by an owner.
HUnLock	Unlocks a block specified by a handle.
HUnLockAll	Unlocks all of the blocks owned by an owner.

Purge Level Functions

SetPurge	Sets the purge level of a block specified by a handle.
SetPurgeAll	Sets the purge level of all blocks owned by a specified owner.

Free Space Functions

FreeMem	Returns the total number of free bytes in memory.
MaxBlock	Returns the size of the largest free block in memory.

Event Manager

Housekeeping Functions

EMBootInit	Called at boot time. Does nothing.
EMStartUp	Initializes the ToolBox and Operating System Event Managers.
EMShutDown	Turns off the ToolBox and Operating System Event Managers.
EMVersion	Returns the version of the ToolBox and Operating System Event Managers.
DoWindows	Returns the address of the zero-page work area used by the ToolBox and Operating System Event Managers.

Accessing Events Through the ToolBox Event Manager

GetNextEvent	Returns the next available event of a specified type or types and, if the event is in the event queue, removes it from the queue.

Reading the Mouse

GetMouse	Returns the current mouse location.
Button	Returns the current state of the mouse button.
StillDown	Tests whether the mouse button is still down.
WaitMouseUp	Tests whether the mouse button is still down, and, if the button is not still down from the original press, removes the preceding mouse-up event before returning FALSE.

Miscellaneous ToolBox Event Manager Routines

GetDblTime	Returns the suggested maximum difference (in ticks) between mouse-up and mouse-down events in order for the mouse clicks to be considered a double click.
GetCaretTime	Returns the time (in ticks) between blinks of the "caret" (usually a vertical bar) marking the insertion point in text.
SetSwitch	Informs the ToolBox Event Manager of a pending switch event. **SetSwitch** is called by the Control Manager and should not be called by an application.

Posting and Removing Events

PostEvent	Places an event in the event queue.
FlushEvents	Removes events in the event queue.

Accessing Events Through the OS Event Manager

GetOSEvent	Returns the next available event of a specified type or types and, if the event is in the event queue, removes it from the queue.
OSEventAvail	This tool works the same as GetOSEvent, except that **OSEventAvail** leaves the event in the event queue (if the event was there in the first place).

Miscellaneous OS Event Manager Routines

SetEventMask	Sets the system event mask to the specified event mask.
GetEvQHdr	Returns a pointer to the header of the event queue.

Font and Text Calls

SetFont	Sets the current font to the specified font.
TextMode	Sets the text mode.
SpaceExtra	Sets the space extra field in the GrafPort to the value specified.
DrawChar	Draws the character at the current point.
DrawString	Draws the string at the current point.
DrawText	Draws the specified text at the current pen location.
CharWidth	Returns the size of a specified character.

StringWidth	Returns the size of the string.
TextWidth	Returns the width in pixels of the specified text.
GetFontInfo	Returns ?????? about the font.
SetForeColor	Sets the foreground color to the value specified.
SetBackColor	Sets the background color to the value specified.
GetForeColor	Returns the foreground color.
GetBackColor	Returns the background color.

SANE

The ROM Tools for the Cortland will provide all of the functions found in the Standard Apple Numeric Environment (SANE). The SANE Tools can be called by using the normal call mechanism. For more information regarding the capabilities of SANE, refer to the *Apple Numerics Manual* and the *SANE Tool Set Preliminary Notes*.

Desk Manager

No information available at this time.

Sound Manager

No information available at this time.

Control Manager

No information available at this time.

Dialog Manager

No information available at this time.

Menu Manager

No information available at this time.

Window Manager

No information available at this time.

File Operations

No information available at this time.

Text Edit

No information available at this time.

Chapter 2

Using the Apple Tools

Initializing Tools At Application Start-Up

There will be a simple mechanism for asking for the correct tools when the application starts. However, design of the mechanism is not yet complete.

Calling the Correct Function

Calling a Function From Assembly Language

We have provided macros in order to make calling a function as simple as possible. You make an assembly-language call as follows:

1. If the function has any output, push room for it on the stack.
2. Push the inputs in the specified order listed.
3. Invoke the appropriate macro.
4. Pull the output, if any, from the top of the stack.

Calling a Function From Pascal

The exact mechanism for this call is not yet known. Appropriate coding conventions will be established so that the tools will be available.

Calling a Function From C

The exact mechanism for this call is not yet known. Appropriate coding conventions will be established so that the tools will be available.

Passing Parameters

Generally, there are several ways to pass parameters:

- in the stack
- in a parameter block
- in the A, X, and Y registers

Method 1 is the most common method used by high-level languages. Method 2 is also very flexible, because the parameter block may be anywhere in memory and may contain additional pointers to anywhere in memory. Method 3 is also useful for small or few parameters but, because the tool dispatcher does not preserve the registers going into a function, it can only be used for one-way communication.

The parameters and parameter-passing method are defined by each function, and listed in their individual descriptions in *Description of the Cortland Tools*, Parts I and II.

Return From the Function

The function itself defines its handling of all parameters. In the most common case, stack parameters are handled by pulling off any input parameters and leaving any results returned by the function on the stack for the calling program to handle.

Return From the Call

Upon completion of the call, the function call returns control directly back to the calling routine. Some tools support returning errors on some functions. If they do, the convention is as follows:

C Flag indicates error
A-register contains error code

The state of all flags and registers is summarized as follows:

N flag	As set by function
V	As set by function
m	Unchanged (must be 0)
x	Unchanged (must be 0)
D	Set to 0
I	Unchanged
Z	As set by function
C	As set by function or error flag
E	Unchanged (must be 0)

A register	As set by function or A=0 successful call, A≠0 error code
X	As set by function
Y	As set by function
S	Parameters have been removed from stack
D	Unchanged
P	See list of flags above
DB	Unchanged
PB	Unchanged
PC	Address following call

Note that "unchanged" means that the value is the same as it was just before the function call.

Error codes $*XX*01-$*XX*0F are reserved for use by the function dispatcher. (*XX* indicates that the upper byte of the 16-bit A register is undefined.) Remaining error codes are defined by the tool itself.

Chapter 3

Writing Your Own Tool Sets

The Tool Locator System is so flexible that individual application writers can write their own tool sets to use in their applications. The Tool Locator System supports both System Tools and User Tools. This chapter will eventually provide the background information necessary to write your own Tool Sets. For the moment, here are some guidelines:

- Tool sets must use Full Native mode.
- Work space must be dynamically assigned. New tool sets should not use any fixed RAM locations for work space, but must obtain work space from the Memory Manager. This avoids memory conflicts such as those caused by fixed usage of "screen holes." A limited set of exceptions to this rule will be published in the final release of this manual.
- A simple interrupt environment must be provided. All new functions must either be reentrant or must disable interrupts during execution. Because each approach has significant costs, the designer must consider this decision very carefully. Most functions, especially those that execute in less than 500µs, will probably choose to disable interrupts. More time-consuming functions should probably also choose to disable interrupts, especially if they are executed rarely.
- Functions must restore the caller's execution environment before returning control to the caller.
- Functions may not assume the presence of any operating system unless the operating system is directly relevant; for example, a Pascal function that can only be used in a Pascal environment.