Technical Note QD18

Drawing Icons the System 7 Way

CONTENTS

This Technical Note describes how to utilize the built-in System 7 icon drawing utility. Use this information to better conform to the System 7 visual human interface.

[May 01 1992]

Introduction

With the introduction of System 7 for the Macintosh, Apple has defined a new look and feel for many screen elements that better utilize color. Among those redefined elements are the icons drawn by the Finder and other system components. Until now, Apple has not documented how to draw icons the way the Finder does in System 7.

This Technical Note discusses the icon toolkit calls that the Finder uses to draw and manipulate the screen icons. Two of the calls, `PlotIconID` and `PlotCIconHandle`, are the ones you will probably use the most since they simply deal with drawing single icons to the screen. Some parts of the toolbox require that an icon family handle be passed to them to allow the drawing of color icons. The icon toolkit provides calls that allow you to create, draw, and manipulate these handles.

Back to top

The New 'ic' Type Resources

`PlotIconID` and `PlotCIconHandle` allow the use of standard `CIcons` as documented in *Inside Macintosh* Volume V. The `PlotIconID` call also permits the use of a new set of icon resources documented in *Inside Macintosh* Volume VI, Chapter 9. This new set is a collection of different icons, representing a single Finder object, into a family. Each member of the family has the same resource ID as the `'ICN#'`, and a resource type indicating the icon data it contains. Currently

Apple has defined three sizes of icons and three bit depths for each size. The sizes are large (32 by 32 pixels), small (16 by 16 pixels), and mini (12 by 12 pixels). The bit depths are 1, 4, and 8. The actual resource types are defined as:

```
Large1BitMask    =    'ICN#';
Large4BitData    =    'icl4';
Large8BitData    =    'icl8';
Small1BitMask    =    'ics#';
Small4BitData    =    'ics4';
Small8BitData    =    'ics8';
Mini1BitMask     =    'icm#';
Mini4BitData     =    'icm4';
Mini8BitData     =    'icm8';
```

The 1-bit-per-pixel member of each size also contains the mask data for all icons of that size (yes, this means that all your icons of a certain size must have the same mask). A 1-bit-per-pixel member must exist for each icon size that `PlotIconID` uses. The icon size used is determined by the size of the destination rectangle. If the destination rectangle is greater than 16 pixels on a side then the large icon will be used. If the rectangle is 13-16 pixels on both sides, the small icon will be used. If it is 12 or less on each side, the mini-icon will be used. The bit depth is determined by the device of the grafPort you plot into at drawing time. Be sure to create a color grafPort when you want to use color icons.

Back to top

Icon Families (or Suites and Caches As the Tool Set Refers to Them)

An icon family is simply a collection of icon handles that contain up to one image of each bit depth and size for a given icon. The family can be fully populated (every possible size or depth available), or it can have only those icons that exist or are needed. By using families, you remove the need to determine which size or depth of icon to use when drawing into a given rectangle. Several system routines, the Notification Manager for example, can take an icon family handle when an icon is requested. This permits them to use the proper color icons if available. In the case of a sparsely populated icon family, when the proper icon is not available, the icon toolkit will pick a substitute to produce the best results.

An icon cache is a family that also has a `ProcPtr` and a `refCon`. The main difference between a cache and a family is that the elements of the cache's array are sparsely populated. When using an icon cache, the system either will use the entry in the icon family portion of the cache or, if the desired element is empty, it will call the procedure pointed to by the `ProcPtr` and request the data for the icon. The procedure should have this interface:

```
FUNCTION IconGetter(theType: ResType;
                    yourDataPtr: Ptr): Handle;
```

This function should return either the icon data to be drawn or NIL to signify that this entry in the icon cache does not exist. Icon caches can be used with all icon family calls. A few extra calls are also available to manipulate icon caches.

Back to top

Drawing Modes or Transforms

In addition to various sizes and bit depths, icons can be drawn with different modes or transforms. Transforms are analogous to certain Finder states for the icons. For example, the transform that you would use to show an icon of a disk that has been ejected is `ttOffline`. Here is a list of the available transforms:

```
ttNone                =    $0;
ttDisabled            =    $1;
ttOffline             =    $2;
ttOpen                =    $3;
ttSelected            =    $4000;
ttSelectedDisabled    =    (ttSelected + ttDisabled);
ttSelectedOffline     =    (ttSelected + ttOffline);
ttSelectedOpen        =    (ttSelected + ttOpen);
```

The actual appearance of the icon drawn by each transform type may vary with future system software, so you should always use the transform that best fits the state it represents in your application. This way you will be consistent with future changes to the look and feel of regular system icons. Note the `ttSelected` transform can be added to any of the other transform types.

There are also transforms that use the Finder label colors to color the icon. To determine the proper label for a file's icon, you can check bits 1-3 of the `fdFlags` field in the file's Finder info. (See the Finder Interface chapter in *Inside Macintosh* Volume VI for more information). These bits contain a number from 0 to 7. Simply add the corresponding `ttLabel` value to the transform that you give the call. The label values are defined like this:

```
ttLabel1    =    $0100;
ttLabel2    =    $0200;
ttLabel3    =    $0300;
ttLabel4    =    $0400;
ttLabel5    =    $0500;
ttLabel6    =    $0600;
ttLabel7    =    $0700;
```

Back to top

Alignment

Most icons do not fully fill their rectangle, and it is sometimes necessary to draw an icon relative to other data (like menu text). In these instances it is nice to be able to have the icon move in its rectangle so that it will be at a predictable location in the destination rectangle. When drawing an icon you can pass one of these standard alignments in the alignment parameter or you can add a vertical alignment to a horizontal alignment to create a composite alignment value.

```
atNone           =    $0;
atVerticalCenter      =    $1;
atTop                 =    $2;
atBottom              =    $3;
atHorizontalCenter    =    $4;
atAbsoluteCenter      =    (atVerticalCenter + atHorizontalCenter);
atCenterTop           =    (atTop + atHorizontalCenter);
atCenterBottom        =    (atBottom + atHorizontalCenter);
atLeft                =    $8;
atCenterLeft          =    (atVerticalCenter + atLeft);
atTopLeft             =    (atTop + atLeft);
atBottomLeft          =    (atBottom + atLeft);
atRight               =    $C;
atCenterRight         =    (atVerticalCenter + atRight);
atTopRight            =    (atTop + atRight);
atBottomRight         =    (atBottom + atRight);
```

Back to top

And Now (Drum Roll Please) the Calls and What to Pass

Now that we have defined every major data type we can think of, here are the actual toolkit calls themselves. One word of caution: only the ForEachIconDo call protects the handle that is passed to it, so make your icon resources nonpurgeable.

Icon Family Calls

NewIconSuite returns an empty icon family handle with all members set to NIL.

```
FUNCTION AddIconToSuite(theIconData: Handle;
                        theSuite: Handle;
                        theType: ResType): OSErr;
```

This call will add the data in theIconData into the suite at the location reserved for theType of icon data. AddIconToSuite will replace any old data in that slot without disposing of it, so you may want to call GetIconFromSuite to obtain the old handle (if any) to dispose of it. This call will be used most often with the NewIconSuite call to fill the empty family after it's created.

```
FUNCTION GetIconFromSuite(VAR theIconData: Handle;
                          theSuite: Handle;
                          theType: ResType): OSErr;
```

This call will return a handle to the pixel data of the family member of theSuite specified by theType. If you intend to dispose of this handle, be sure to call AddIconToSuite with a NIL handle to zero out the family entry.

```
FUNCTION ForEachIconDo(theSuite: Handle;
                       selector: IconSelectorValue;
                       action: IconAction;
                       yourDataPtr: Ptr): OSErr;
```

This routine will call your IconAction procedure (see below) for each icon in the family specified by selector and theSuite. The selector parameter is a bit-level flag that specifies which family members to operate on; they can be added together to create composite selectors that work on several different family members. The values for selector are:

```
svLarge1Bit   =    $00000001;
svLarge4Bit        =    $00000002;
svLarge8Bit        =    $00000004;
svSmall1Bit        =    $00000100;
svSmall4Bit        =    $00000200;
svSmall8Bit        =    $00000400;
svMini1Bit         =    $00010000;
svMini4Bit         =    $00020000;
svMini8Bit         =    $00040000;
svAllLargeData     =    $000000FF;
svAllSmallData     =    $0000FF00;
svAllMiniData      =    $00FF0000;
svAll1BitData      =    (svLarge1Bit + svSmall1Bit + svMini1Bit);
svAll4BitData      =    (svLarge4Bit + svSmall4Bit + svMini4Bit);
svAll8BitData      =    (svLarge8Bit + svSmall8Bit + svMini8Bit);
svAllAvailableD    =    $FFFFFFFF;
```

The action procedure that gets called for each icon type selected for the family is a Pascal type function with the following interface:

```
FUNCTION IconAction(theType: ResType;
                    VAR theIcon: Handle;
```

The parameter `theIcon` is passed by reference here so that your routine can modify the contents of the suite directly. The `yourDataPtr` parameter is the value passed when you called `ForEachIconDo`. It allows you to easily communicate with your application. The action procedure returns an `OSErr`. If any value other than `noErr` is returned, `ForEachIconDo` will stop processing immediately and return the error passed. (Note: This implies that the icons selected may only be partially operated on. There is no guaranteed order in which the icons get operated on.)

```
FUNCTION GetIconSuite(VAR theIconSuite: Handle;
                      theResID: INTEGER;
                      selector: IconSelectorValue): OSErr;
```

`GetIconSuite` will create a new icon family and fill it from the current resource chain with the icons of resource ID `theResID` and types indicated by `selector`. This is the call you will probably use most often to create an icon family. Note that if you `SetResLoad(False)` before making this call, the suite will be filled with unloaded resource handles.

```
FUNCTION PlotIconSuite(theRect: Rect;
                       align: IconAlignmentType;
                       transform: IconTransformType;
                       theIconSuite: Handle): OSErr;
```

This call renders the proper icon image from the passed icon family based on the bit depth of the display you are using and the rectangle that you have passed. The parameters align and transform are applied to the icon selected for drawing and then the icon is plotted into the current grafPort. `PlotIconSuite` chooses the appropriate icon based primarily on size. Once the proper icon size is determined (based on the destination rectangle), the present member of that size with the deepest bit depth that the current device can use is selected. A size category is considered present if the black-and-white member (with mask), `'ICN#'`, `'ics#'`, or `'icm#'`, is present. `PlotIconSuite` can be used for both picture accumulation and printing.

```
FUNCTION DisposeIconSuite(theIconSuite: Handle;
                          disposeData: BOOLEAN): OSErr;
```

This call disposes the icon family handle itself. In addition, if `disposeData` is true, any of the icon data handles that do not belong to a resource fork will also be disposed.

This call allows you to specify a label to draw an icon of this suite when no label is specified in `PlotIconSuite`. This is used primarily to ensure that a family passed to a system routine gets drawn with the proper label. The default label can be overridden by specifying a label in `PlotIconSuite`.

`GetSuiteLabel` returns the label previously set with `SetSuiteLabel`.

Icon Cache Calls

In addition to the icon family calls, icon caches have these additional calls:

```
FUNCTION MakeIconCache(VAR theHandle: Handle;
                       makeIcon: IconGetter;
                       yourDataPtr: UNIV Ptr): OSErr;
```

This call creates an empty icon cache similar to `NewIconSuite`, and associates the additional icon loading procedure and data value with the family.

```
FUNCTION LoadIconCache(theRect: Rect;
                       align: IconAlignmentType;
                       transform: IconTransformType;
                       theIconCache: Handle): OSErr;
```

This call allows preflight loading of certain elements of your icon cache. This is handy if you suspect that certain drawing operations may occur at a time not convenient for loading your icon data (e.g., when your resource fork might not be in open chain). `LoadIconCache` takes the same parameters as `PlotIconSuite` and uses the same criteria to select the icon to load. The grafPort must be set properly before making this call since it is one of the criteria for determining the icon to load.

The following four calls are provided to change `theData` or `theProc` associated with an icon cache:

```
FUNCTION GetIconCacheData(theCache: Handle;
                          VAR theData: Ptr): OSErr;
FUNCTION SetIconCacheData(theCache: Handle;
                          theData: Ptr): OSErr;
FUNCTION GetIconCacheProc(theCache: Handle;
                          VAR theProc: IconGetter): OSErr;
FUNCTION SetIconCacheProc(theCache: Handle;
                          theProc: IconGetter): OSErr;
```

Plotting Icons Not Part of a Suite

The next calls are grouped because they are similar. They let you plot an icon to the screen without your creating an icon suite. They are also good if you have a `'cicn'` instead of an icon family.

```
FUNCTION PlotIconID(theRect: Rect;
                    align: IconAlignmentType;
                    transform: IconTransformType;
                    theResID: INTEGER): OSErr;

FUNCTION PlotCIconHandle(theRect: Rect;
                         align: IconAlignmentType;
                         transform: IconTransformType;
                         theCIcon: CIconHandle): OSErr;

FUNCTION PlotIconMethod(theRect: Rect;
                        align: IconAlignmentType;
                        transform: IconTransformType;
                        theMethod: IconGetter;
                        yourDataPtr: UNIV Ptr): OSErr;

FUNCTION PlotIconHandle(theRect: Rect;
                        align: IconAlignmentType;
                        transform: IconTransformType;
                        theIcon: Handle): OSErr;

FUNCTION PlotSICNHandle(theRect: Rect;
                        align: IconAlignmentType;
                        transform: IconTransformType;
                        theSICN: Handle): OSErr;
```

All these routines share the following parameters: `theRect` is the destination rectangle to draw the indicated icon into; `align` is the alignment method to use if the icon does not fit the rectangle given; `transform` indicates the desired appearance of the icon on the screen.

In PlotIconID, the parameter theResID is the resource ID of the family of 'ic' type resources to use. If the correct bit depth or the size required is not defined, the closest-fitting one will be used.

The PlotCIconHandle parameter theCIcon is a handle that you get to a standard QuickDraw color icon. Unlike PlotCIcon, PlotCIconHandle does not honor the current foreground and background colors. Call GetCIcon to load the icon. Dispose of it when you are done, since they can take up quite a bit of memory.

PlotIconMethod calls your IconGetter procedure, discussed earlier, to check for the existence of icon data.

PlotIconHandle will plot the data from an 'ICN#' or 'ICON' resource from its handle. It is a new version of PlotIcon.

PlotSICNHandle plots the data of a 'SICN' resource from its handle. Only 'SICN' resources with a single member, or one in which the second member is a mask for the first, will plot correctly.

All the functions return an error code if things did not go well with the drawing or, in the case of the PlotIconID call, if the indicated icon family could not be used.

Miscellaneous Calls

```
FUNCTION GetLabel(labelNumber: INTEGER;
                  VAR labelColor: RGBColor;
                  VAR labelString: Str255): OSErr;
```

This call returns the actual color and string used in the label menu of the Finder and the label's Control Panel. This information is provided in case you wish to include the label text or color when displaying a file's icon in your application.

```
FUNCTION IconSuiteToRgn(theRgn: RgnHandle;
                        iconRect: Rect;
                        align: IconAlignmentType;
                        theIconSuite: Handle): OSErr;

FUNCTION IconIDToRgn(theRgn: RgnHandle;
                     iconRect: Rect;
                     align: IconAlignmentType;
                     iconID: INTEGER): OSErr;

FUNCTION IconMethodToRgn(theRgn: RgnHandle;
                         iconRect: Rect;
                         align: IconAlignmentType;
                         theMethod: IconGetter;
                         yourDataPtr: Ptr): OSErr;
```

These routines will create a region from the mask of the icon selected by the iconRect and align values passed. They will allow you to do accurate hit testing and outline dragging of an icon in your application. The RgnHandle must have been previously allocated before you make this call.

```
FUNCTION RectInIconSuite(testRect: Rect;
                         iconRect: Rect;
                         align: IconAlignmentType;
                         theIconSuite: Handle): BOOLEAN;

FUNCTION RectInIconID(testRect: Rect;
                      iconRect: Rect;
                      align: IconAlignmentType;
                      iconID: INTEGER): BOOLEAN;

FUNCTION RectInIconMethod(testRect: Rect;
                          iconRect: Rect;
                          align: IconAlignmentType;
                          theMethod: IconGetter;
                          yourDataPtr: Ptr): BOOLEAN;

FUNCTION PtInIconSuite(testPt: Point;
                       iconRect: Rect;
                       align: IconAlignmentType;
                       theIconSuite: Handle): BOOLEAN;

FUNCTION PtInIconID(testPt: Point;
                    iconRect: Rect;
                    align: IconAlignmentType;
                    iconID: INTEGER): BOOLEAN;

FUNCTION PtInIconMethod(testPt: Point;
                        iconRect: Rect;
                        align: IconAlignmentType;
                        theMethod: IconGetter;
                        yourDataPtr: Ptr): BOOLEAN;
```

These calls hit test the passed Point or Rect against the icon indicated. The parameters `iconRect` and `align`, and the grafPort should be the same as when the icon was last drawn. The functions return true if the point is in the icon mask or if the rectangle intersects the icon mask.

Back to top

Error Codes

The Icon Utilities will pass back any errors encountered during execution, so you can expect to see Memory Manager, Resource Manager, and other normal errors.

There is one error code defined specifically for the Icon Utilities routines that may be returned by the Icon plotting routines.

```
{ Pascal }
CONST
    noMaskFound = -1000;
END;

/* C */
#define noMaskFound    -1000
```

This error will be returned if the Icon Utilities package could not find or create a mask for the icon family. The Icon Utilities will use the correct mask for each icon size, if one is available. If no mask for a specific size is available, a mask will be created from any mask in the family, but if there are no 1 bit images and no mask in the family, plotting calls will fail with this error.

Back to top

Type(def)s and Glue for Pascal and C

The Pascal and C interfaces are provided here to copy and paste since the current MPW standard interface files do not contain the glue for these calls.

MPW C, Pascal, and Assembler files also have been submitted to AppleLink and the Developer CD, but their paths were not known when this note was written.

```
{ Pascal Types }

IconAction         =  ProcPtr;
    {FUNCTION IconAction(
        theType: ResType;
        VAR theIcon: Handle;
        yourDataPtr: Ptr): OSErr;}
IconGetter         =  ProcPtr;
    {FUNCTION IconGetter(
        theType: ResType;
        yourDataPtr: Ptr): Handle;}
IconSelectorValue  =  LONGINT;
IconAlignmentType  =  INTEGER;
IconTransformType  =  INTEGER;

{ Pascal Glue }

FUNCTION PlotIconID(
    theRect: Rect;
    align: IconAlignmentType;
    transform: IconTransformType;
    theResID: INTEGER): OSErr;
         INLINE  $303C, $0500, $ABC9;

FUNCTION NewIconSuite(
    VAR theIconSuite: Handle): OSErr;
         INLINE  $303C, $0207, $ABC9;

FUNCTION AddIconToSuite(
    theIconData: Handle;
    theSuite: Handle;
    theType: ResType): OSErr;
         INLINE  $303C, $0608, $ABC9;

FUNCTION GetIconFromSuite(
    VAR theIconData: Handle;
    theSuite: Handle;
    theType: ResType): OSErr;
         INLINE  $303C, $0609, $ABC9;

FUNCTION ForEachIconDo(
    theSuite: Handle;
    selector: IconSelectorValue;
    action: IconAction;
    yourDataPtr: Ptr): OSErr;
         INLINE  $303C, $060A, $ABC9;

FUNCTION GetIconSuite(
    VAR theIconSuite: Handle;
    theResID: INTEGER;
    selector: IconSelectorValue): OSErr;
         INLINE  $303C, $0501, $ABC9;
```

```
FUNCTION DisposeIconSuite(
    theIconSuite: Handle;
    disposeData: BOOLEAN): OSErr;
         INLINE  $303C, $0302, $ABC9;

FUNCTION PlotIconSuite(
    theRect: Rect;
    align: IconAlignmentType;
    transform: IconTransformType;
    theIconSuite: Handle): OSErr;
        INLINE  $303C, $0603, $ABC9;

FUNCTION MakeIconCache(
    VAR theHandle: Handle;
    makeIcon: IconGetter;
    yourDataPtr: UNIV Ptr): OSErr;
         INLINE  $303C, $0604, $ABC9;

FUNCTION LoadIconCache(
    theRect: Rect;
    align: IconAlignmentType;
    transform: IconTransformType;
    theIconCache: Handle): OSErr;
         INLINE  $303C, $0606, $ABC9;

FUNCTION PlotIconMethod(
    theRect: Rect;
    align: IconAlignmentType;
    transform: IconTransformType;
    theMethod: IconGetter;
    yourDataPtr: UNIV Ptr): OSErr;
         INLINE  $303C, $0805, $ABC9;

FUNCTION GetLabel(
    labelNumber: INTEGER;
    VAR labelColor: RGBColor;
    VAR labelString: Str255): OSErr;
         INLINE  $303C, $050B, $ABC9;

FUNCTION PtInIconID(
    testPt: Point;
    iconRect: Rect;
    align: IconAlignmentType;
    iconID: INTEGER): BOOLEAN;
         INLINE  $303C, $060D, $ABC9;

FUNCTION PtInIconSuite(
    testPt: Point;
    iconRect: Rect;
    align: IconAlignmentType;
    theIconSuite: Handle): BOOLEAN;
         INLINE  $303C, $070E, $ABC9;

FUNCTION PtInIconMethod(
    testPt: Point;
    iconRect: Rect;
    align: IconAlignmentType;
    theMethod: IconGetter;
    yourDataPtr: Ptr): BOOLEAN;
         INLINE  $303C, $090F, $ABC9;

FUNCTION RectInIconID(
    testRect: Rect;
    iconRect: Rect;
```

```
    align: IconAlignmentType;
    iconID: INTEGER): BOOLEAN;
         INLINE  $303C, $0610, $ABC9;

FUNCTION RectInIconSuite(
    testRect: Rect;
    iconRect: Rect;
    align: IconAlignmentType;
    theIconSuite: Handle): BOOLEAN;
         INLINE  $303C, $0711, $ABC9;

FUNCTION RectInIconMethod(
    testRect: Rect;
    iconRect: Rect;
    align: IconAlignmentType;
    theMethod: IconGetter;
    yourDataPtr: Ptr): BOOLEAN;
         INLINE  $303C, $0912, $ABC9;

FUNCTION IconIDToRgn(
    theRgn: RgnHandle;
    iconRect: Rect;
    align: IconAlignmentType;
    iconID: INTEGER): OSErr;
         INLINE  $303C, $0913, $ABC9;

FUNCTION IconSuiteToRgn(
    theRgn: RgnHandle;
    iconRect: Rect;
    align: IconAlignmentType;
    theIconSuite: Handle): OSErr;
         INLINE  $303C, $0914, $ABC9;

FUNCTION IconMethodToRgn(
    theRgn: RgnHandle;
    iconRect: Rect;
    align: IconAlignmentType;
    theMethod: IconGetter;
    yourDataPtr: Ptr): OSErr;
         INLINE  $303C, $0915, $ABC9;

FUNCTION SetSuiteLabel(
    theSuite: Handle;
    theLabel: INTEGER): OSErr;
         INLINE  $303C, $0316, $ABC9;

FUNCTION GetSuiteLabel(
    theSuite: Handle): INTEGER;
         INLINE  $303C, $0217, $ABC9;

FUNCTION GetIconCacheData(
    theCache: Handle;
    VAR theData: Ptr): OSErr;
         INLINE  $303C, $0419, $ABC9;

FUNCTION SetIconCacheData(
    theCache: Handle;
    theData: Ptr): OSErr;
         INLINE  $303C, $041A, $ABC9;

FUNCTION GetIconCacheProc(
    theCache: Handle;
    VAR theProc: IconGetter): OSErr;
         INLINE  $303C, $041B, $ABC9;
```

```
FUNCTION SetIconCacheProc(
    theCache: Handle;
    theProc: IconGetter): OSErr;
         INLINE  $303C, $041C, $ABC9;

FUNCTION PlotIconHandle(
    theRect: Rect;
    align: IconAlignmentType;
    transform: IconTransformType;
    theIcon: Handle): OSErr;
         INLINE  $303C, $061D, $ABC9;

FUNCTION PlotSICNHandle(
    theRect: Rect;
    align: IconAlignmentType;
    transform: IconTransformType;
    theSICN: Handle): OSErr;
         INLINE  $303C, $061E, $ABC9;

FUNCTION PlotCIconHandle(
    theRect: Rect;
    align: IconAlignmentType;
    transform: IconTransformType;
    theCIcon: CIconHandle): OSErr;
         INLINE  $303C, $061F, $ABC9;


/* C Typedefs */

typedef pascal OSErr   (*IconAction)(
    ResType theType,
    Handle *theIcon,
    void *yourDataPtr);

typedef pascal Handle  (*IconGetter)(
    ResType theType,
    void *yourDataPtr);

typedef unsigned long  IconSelectorValue;
typedef short          IconAlignmentType;
typedef short          IconTransformType;

/* C Glue */

pascal OSErr PlotIconID(
    const Rect *theRect,
    IconAlignmentType align,
    IconTransformType transform,
    short theResID)
       =  {0x303C, 0x0500, 0xABC9};

pascal OSErr NewIconSuite(
    Handle *theIconSuite)
       =  {0x303C, 0x0207, 0xABC9};

pascal OSErr AddIconToSuite(
    Handle theIconData,
    Handle theSuite,
    ResType theType)
       =  {0x303C, 0x0608, 0xABC9};

pascal OSErr GetIconFromSuite(
    Handle *theIconData,
```

```
    Handle theSuite,
    ResType theType)
       =  {0x303C, 0x0609, 0xABC9};

pascal OSErr ForEachIconDo(
    Handle theSuite,
    IconSelectorValue selector,
    IconAction action,
    void *yourDataPtr)
       =  {0x303C, 0x080A, 0xABC9};

pascal OSErr GetIconSuite(
    Handle *theIconSuite,
    short theResID,
    IconSelectorValue selector)
       =  {0x303C, 0x0501, 0xABC9};

pascal OSErr DisposeIconSuite(
    Handle theIconSuite,
    Boolean disposeData)
       =  {0x303C, 0x0302, 0xABC9};

pascal OSErr PlotIconSuite(
    const Rect *theRect,
    IconAlignmentType align,
    IconTransformType transform,
    Handle theIconSuite)
       =  {0x303C, 0x0603, 0xABC9};

pascal OSErr MakeIconCache(
    Handle *theHandle,
    IconGetter makeIcon,
    void *yourDataPtr)
       =  {0x303C, 0x0604, 0xABC9};

pascal OSErr LoadIconCache(
    const Rect *theRect,
    IconAlignmentType align,
    IconTransformType transform,
    Handle theIconCache)
       =  {0x303C, 0x0606, 0xABC9};

pascal OSErr PlotIconMethod(
    const Rect *theRect,
    IconAlignmentType align,
    IconTransformType transform,
    IconGetter theMethod,
    void *yourDataPtr)
       =  {0x303C, 0x0805, 0xABC9};

pascal OSErr GetLabel(
    short labelNumber,
    RGBColor *labelColor,
    Str255 labelString)
       =  {0x303c, 0x050B, 0xABC9};

pascal Boolean PtInIconID(
    Point testPt,
    Rect *iconRect,
    IconAlignmentType alignment,
    short iconID)
       =  {0x303c, 0x060D, 0xABC9};

pascal Boolean PtInIconSuite(
```

```
    Point testPt,
    Rect *iconRect,
    IconAlignmentType alignment,
    Handle theIconSuite)
       =  {0x303c, 0x070E, 0xABC9};

pascal Boolean PtInIconMethod(
    Point testPt,
    Rect *iconRect,
    IconAlignmentType alignment,
    IconGetter theMethod,
    void *yourDataPtr)
       =  {0x303c, 0x090F, 0xABC9};

pascal Boolean RectInIconID(
    Rect *testRect,
    Rect *iconRect,
    IconAlignmentType alignment,
    short iconID)
       =  {0x303c, 0x0610, 0xABC9};

pascal Boolean RectInIconSuite(
    Rect *testRect,
    Rect *iconRect,
    IconAlignmentType alignment,
    Handle theIconSuite)
       =  {0x303c, 0x0711, 0xABC9};

pascal Boolean RectInIconMethod(
    Rect *testRect,
    Rect *iconRect,
    IconAlignmentType alignment,
    IconGetter theMethod,
    void *yourDataPtr)
       =  {0x303c, 0x0912, 0xABC9};

pascal OSErr IconIDToRgn(
    RgnHandle theRgn,
    Rect *iconRect,
    IconAlignmentType alignment,
    short iconID)
       =  {0x303c, 0x0613, 0xABC9};

pascal OSErr IconSuiteToRgn(
    RgnHandle theRgn,
    Rect *iconRect,
    IconAlignmentType alignment,
    Handle theIconSuite)
       =  {0x303c, 0x0714, 0xABC9};

pascal OSErr IconMethodToRgn(
    RgnHandle theRgn,
    Rect *iconRect,
    IconAlignmentType alignment,
    IconGetter theMethod,
    void *yourDataPtr)
       =  {0x303c, 0x0915, 0xABC9};

pascal OSErr SetSuiteLabel(
    Handle theSuite,
    short theLabel)
       =  {0x303C, 0x0316, 0xABC9};

pascal short GetSuiteLabel(
```

```
    Handle theSuite)
       =  {0x303C, 0x0217, 0xABC9};

pascal OSErr GetIconCacheData(
    Handle theCache,
    void **theData)
       =  {0x303C, 0x0419, 0xABC9};

pascal OSErr SetIconCacheData(
    Handle theCache,
    void *theData)
       =  {0x303C, 0x041A, 0xABC9};

pascal OSErr GetIconCacheProc(
    Handle theCache,
    IconGetter *theProc)
       =  {0x303C, 0x041B, 0xABC9};

pascal OSErr SetIconCacheProc(
    Handle theCache,
    IconGetter theProc)
       =  {0x303C, 0x041C, 0xABC9};

pascal OSErr PlotIconHandle(
    const Rect *theRect,
    IconAlignmentType align,
    IconTransformType transform,
    Handle theIcon)
       =  {0x303C, 0x061D, 0xABC9};

pascal OSErr PlotSICNHandle(
    const Rect *theRect,
    IconAlignmentType align,
    IconTransformType transform,
    Handle theSICN)
       =  {0x303C, 0x061E, 0xABC9};

pascal OSErr PlotCIconHandle(
    const Rect *theRect,
    IconAlignmentType align,
    IconTransformType transform,
    CIconHandle theCIcon)
       =  {0x303C, 0x061F, 0xABC9};
```

Back to top

References

Inside Macintosh , Volume V, QuickDraw chapter

Inside Macintosh , Volume VI, Finder Interface chapter

Change History

May-01-1992	In this Note, we replaced the C and Pascal interface files and corrected the related text. So much text was tweaked that change bars are used only on code changes.
Oct-01-1991	Originally written.

Back to top

Downloadables

Acrobat version of this Note (K). Download

Technical Notes by Date | Number | Technology | Title
Developer Documentation | Technical Q&As | Development Kits | Sample Code