Technical Note TN2033

How to use the ATSUI Low Level APIs to get glyph outlines

CONTENTS

Getting Glyph Information

Retrieving Glyph Outlines

Retrieving Glyph Metrics

Data Type Reference

API Reference

Downloadables

This document assumes you are familiar with the APIs and concepts for ATSUI (Apple Type Services for Unicode Imaging).

ATSUI is Apple's API for drawing Unicode text. In addition to the high-level ATSUI APIs, ATSUI has added a set of APIs that enable you to access text information at a lower level. These APIs contain functions that let you:

- Get information on the glyphs and glyph positions associated with an `ATSUTextLayout`. For each glyph, you can determine the glyph ID in the corresponding font, which character in the original text the glyph is derived from, and positioning information for the glyph.

- For a given glyph in a particular font, retrieve the outline data for the glyph. These are the curves that make up the glyph's shape.

- For a given glyph in a particular font, retrieve both the ideal (resolution-independent) metrics and the screen (resolution-dependent) metrics for that glyph.

- Given a set of glyphs and positions that have been obtained from an `ATSUTextLayout`, draw them, after having possibly modified the glyph positioning information.

This is functionality that was previously available on Mac OS 9 through a combination of special TrueType scaler calls or calls to APIs exported by Adobe® Type Manager. Neither of these API sets are available to native OS X applications. By calling these ATSUI low-level APIs, you can use one API set for both TrueType and Type 1 fonts, and your code can run both on CarbonLib and native on OS X.

[Nov 26 2001]

Getting Glyph Information

Getting the glyph information from an `ATSUTextLayout` is the starting point for all of the APIs described in this document. You need the glyph information in order to call all of the other APIs.

Even if you don't need information on individual glyph properties, having the glyph information allows you to reposition the glyphs before drawing them. You can draw text in ways ATSUI doesn't support directly, such as drawing text along a curve.

> **Note:**
> To draw text along a curve, you must obtain and draw the glyphs in small groups. If you are handling arbitrary Unicode text, you need to determine what those groups are while being sensitive to international issues. For example, Arabic is a cursive script, and breaking in the middle of a ligature or between two cursively connected glyphs will result in an incorrect appearance in your output.

The API function `ATSUGetGlyphInfo` returns the following information about each glyph in the range you specify:

- The `ATSUStyle` associated with the glyph.
- The ID of the glyph within the font specified by the `ATSUStyle`.
- The offset of the character in the original text with which this glyph is associated.
- The cross-stream shift (if any) of the glyph.
- The x offset in ideal (resolution-independent) coordinates from the beginning of the line the glyph is on.
- The x offset in screen (resolution-dependent) coordinates of the original of the glyph relative to the beginning of the line the glyph is on.
- The x offset in screen (resolution-dependent) coordinates from the beginning of the line where a leading caret for this glyph crosses the baseline.
- Additional fields whose contents are currently reserved by Apple.

It's important to note that the relationship between the original Unicode characters in the `ATSUTextLayout` and the glyphs returned by `ATSUGetGlyphInfo` is not necessarily one to one, and can often be quite complicated. Even the simple accented latin character é can be represented by an e with a combining ´ accent. In this case, two characters map to one glyph.

Common ligatures such as fi also form automatically for some fonts, causing two characters to map to one glyph. Right-to-left scripts such as Arabic, and complex scripts such as Devanagari or Thai have even more complicated mappings from characters to glyphs. Assuming a one to one mapping between characters and glyphs will make it incompatible with such languages, and perhaps not even compatible with European languages and Japanese in some cases.

For this reason it's best to use the high level ATSUI APIs whenever possible, and to let ATSUI lay out an entire paragraph of text at a time. Your application is then completely insulated from such issues.

Here is a code snippet for fetching the glyph information associated with the first glyph of an `ATSUTextLayout`. For details on the API, including how to draw a possibly modified `ATSUGlyphInfoArray`, see the API reference at the end of this document.

Listing 1. Retrieving glyph information.

```
char giaBuffer[128];

ByteCount giabc = 128;
ATSUGlyphInfoArray *gia = (ATSUGlyphInfoArray *)giaBuffer;
status = ATSUGetGlyphInfo(atsuLayout, 0, 1, &giabc, gia);
```

Back to top

Retrieving Glyph Outlines

One of the things you can do with a glyph when you know both its ID and the font it came from is to retrieve the curves that make up the shape of the glyph, known as the glyph outlines. ATSUI provides several APIs to assist in this. Using these APIs, you can:

- Determine the native curve type of the font in question. TrueType fonts use quadratic curves, and Type 1 (PostScript) fonts use cubic curves. Use `ATSUGetNativeCurveType` for this.
- Retrieve the curves making up one glyph's shape by passing in callback functions which get called for each segment of the curves and lines making up the glyph's shape.
- Retrieve raw quadratic curve data from a TrueType font or a Type 1 font (Type 1 cubic curves will have been converted to quadratic form).

There are two APIs for retrieving curves via callback, `ATSUGetQuadraticGlyphPaths` and `ATSUGetCubicGlyphPaths`. You can call `ATSUGetQuadraticGlyphPaths` even for a font whose native curve type is cubic, and you can call `ATSUGetCubicGlyphPaths` for a font whose native curve type is quadratic. In each case, the font's curves are converted to the desired format.

> **Note:**
> As of Mac OS X 10.1, all curves pass through a quadratic form before being returned to the caller. It is not possible to retrieve the cubic curves directly without passing through this intermediate conversion.

When retrieving quadratic paths, the callbacks used are for:

- The start of a new path.

- Each line in a path, giving its start and end points.
- Each quadratic curve in a path, giving its start point, end point, and off-curve control point.
- The end of the currently open path.

When retrieving cubic paths, the callbacks used are for:

- Each move of the pen position to a new location.
- Each line from the current pen position to an ending position.
- Each curve from the current pen position to an ending position, with two off-curve control points.
- The end of the currently open path.

Note that for cubic paths, the starting position for each curve or line is implicit from the current pen position. The start of a path is also implicit, and will be signalled by the initial move to establish the initial pen position.

Also note that the curves returned are the ones modified by hints present in the font. If you need unhinted outlines, use a very large point size (e.g., 1000 points), and scale the resultant curves down. Alternatively, you can set the `ATSUStyleRenderngOptions` of the `ATSUStyle` to zero.

The coordinates returned for the curves and lines are in QuickDraw coordinates, with the y axis pointing down.

See the API reference at the end of this document for further details.

Back to top

Retrieving Glyph Metrics

ATSUI lets you retrieve both the ideal (resolution-independent) and screen (resolution-dependent) metrics for a glyph.

For the ideal metrics, ATSUI returns:

- The advance of the glyph (the amount by which the pen is advanced after drawing the glyph).
- The side bearing of the glyph. This is the offset from the glyph origin to the beginning of the glyph image.
- The other side bearing of the glyph. This is the offset from the end of the glyph image to the end of the glyph advance.

For the screen metrics, ATSUI returns:

- The device advance. This is the number of pixels of the advance for the glyph as actually drawn on the screen.
- The top left point of the glyph in device coordinates.
- The height and width of the glyph in pixels. Note that glyphs may possibly overlap when drawn.
- The side bearing and other side bearing in pixels.

For further details, see the API reference below.

Back to top

Data Type Reference

Data Types and Constants

These new APIs make use of the following constants and data types:

```
/* Glyph outline path constants used in ATSUGetNativeCurveType */
enum { kATSCubicCurveType = 0x0001,
       kATSQuadCurveType = 0x0002,
       kATSOtherCurveType = 0x0003
     };
typedef UInt16 GlyphID;
typedef UInt16 ATSCurveType;
struct ATSGlyphIdealMetrics {
        /* used with API routine ATSGetGlyphIdealMetrics */
        Float32Point advance;
        Float32Point sideBearing;
        Float32Point otherSideBearing;
        };
struct ATSGlyphScreenMetrics {
        /* used with API routine ATSGetGlyphScreenMetrics */
        Float32Point deviceAdvance;
        Float32Point topLeft;
        UInt32 height;
        UInt32 width;
        Float32Point sideBearing;
        Float32Point otherSideBearing;
        };
```

ATSUGetQuadraticGlyphPaths callbacks

For quadratic curves, pt1 is the starting point, pt2 is the ending point, and controlPt is the off-curve control point. In Chapter 1 of the TrueType reference manual, these are referred to as p0, p2, and p1 respectively. For both curves and lines, the starting point is passed explicitly to the callback function. Moves are implicit; there is no `MoveTo` callback.

```
typedef CALLBACK_API( OSStatus , ATSQuadraticNewPathProcPtr )
    (void *callBackDataPtr);
typedef CALLBACK_API( OSStatus , ATSQuadraticLineProcPtr )
    (const Float32Point *pt1, const Float32Point *pt2, void *callBackDataPtr);
typedef CALLBACK_API( OSStatus , ATSQuadraticCurveProcPtr )
    (const Float32Point *pt1, const Float32Point *controlPt,
     const Float32Point *pt2, void *callBackDataPtr);
typedef CALLBACK_API( OSStatus , ATSQuadraticClosePathProcPtr )
    (void *callBackDataPtr);
```

ATSUGetCubicGlyphPaths callbacks

For cubic curves, the points are defined as in the PostScript Language Reference Manual. p0, the starting point, is implicit from the current pen position. p3 is the ending point; p1 and p2 are the off-curve control points. For both curves and lines, the starting point is implicit; there is an explicit `MoveTo` callback to move the pen. Conversely, there is no `NewPath` callback for the cubic case.

```
typedef CALLBACK_API( OSStatus , ATSCubicMoveToProcPtr )
    (const Float32Point *pt, void *callBackDataPtr);
typedef CALLBACK_API( OSStatus , ATSCubicLineToProcPtr )
    (const Float32Point *pt, void *callBackDataPtr);
typedef CALLBACK_API( OSStatus , ATSCubicCurveToProcPtr )
    (const Float32Point *pt1, const Float32Point *pt2,
     const Float32Point *pt3, void *callBackDataPtr);
typedef CALLBACK_API( OSStatus , ATSCubicClosePathProcPtr )
    (void *callBackDataPtr);

struct Float32Point {
    Float32 x;
    Float32 y;
};
```

Note:
`Float32Point` is defined in MacTypes.h

The following data structures are used to return glyph outlines. The format of the data is the same as that in the glyph table of a TrueType font; this is documented in chapter 6 of the TrueType Reference Manual. The data is modified as follows before being returned to the caller: the control bits are padded to an integral number of longs, and all coordinates are converted from fixed to floating point. See the TrueType reference manual for further details.

```
struct ATSUCurvePath {
    UInt32 vectors;
    UInt32 controlBits[1];
    Float32Point vector[1];
};

struct ATSUCurvePaths {
    UInt32 contours;
    ATSUCurvePath contour[1];
};
```

The following data structures are used to return the glyph information associated with an `ATSUTextLayout`. For each glyph, the following information is returned: the 16 bit glyph identifier (unique to the associated font), the reserved fields "reserved" and "layoutFlags," the index of the character in the associated Unicode character stream from which this glyph is derived, the `ATSUStyle` used for this glyph, any cross-stream shift for the glyph, its ideal with-stream offset from the origin of this layout, its device-adjusted with-stream offset from the origin of this layout, and the position in device coordinates where a trailing caret for this glyph would intersect the baseline.

```
struct ATSUGlyphInfo {
    GlyphID glyphID;
    UInt16 reserved;
    UInt32 layoutFlags;
    UniCharArrayOffset charIndex;
    ATSUStyle style;
    Float32 deltaY;
    Float32 idealX;
    SInt16 screenX;
    SInt16 caretX;
};

struct ATSUGlyphInfoArray {
    ItemCount numGlyphs;
    ATSUTextLayout layout;
    ATSUGlyphInfo glyphs[1];
};
```

Back to top

API Reference

Metrics

ATSUGlyphGetIdealMetrics

```
OSStatus ATSUGlyphGetIdealMetrics (
    ATSUStyle iATSUStyle,
    ItemCount iNumOfGlyphs,
    GlyphID * iGlyphIDs,
    Sint32 iInputOffset,
    ATSGlyphIdealMetrics * oIdealMetrics );
```

This function is used to get the ideal metrics of glyphs specified by the array of input `glyphIDs` and an `ATSUStyle`. The ideal metrics are derived from the glyph outlines and specific style attributes such as the font, point size, and verticality. Since you can pass only a single style, at most one style run can be handled per call. `oIdealMetrics` must have been previously allocated by the caller to contain the requested number of returned metrics.

If vertical metrics are needed, set `kATSUVerticalCharacterTag` in the `ATSUStyle`.

Input Parameters

`ATSUStyle iATSUStyle`	The opaque ATSUI style object, needed to specify the font and other information that can affect the metrics.
`ItemCount iNumOfGlyphs`	The number of glyph metrics requested and hence the number of `glyphIDs` provided as input.
`GlyphID *iGlyphIDs`	The address of the first element of an array of `glyphID` values whose metrics are desired.
`Sint32 iInputOffset`	The number of bytes for `ATSGetGlyphFractionalMetrics` to increment or decrement to find each successive `glyphID` provided as input starting at `iGlyphIDs`. This is useful when the `glyphIDs` are imbedded in an array of data structures. If you are passing in just an array of `glyphIDs`, set this to `sizeof(GlyphID)`.

Output Parameters

`ATSGlyphIdealMetrics *oIdealMetrics`	The pointer to the caller supplied allocated block of memory filled with the requested number of metrics.

Return Values

`noErr`	success
`paramErr`	`oIdealMetrics` and `iGlyphIDs` must not be NULL.
Memory manager errors	

ATSUGetNativeCurveType

```
OSStatus ATSUGetNativeCurveType(
    ATSUStyle iATSUStyle,
    CurveType * oCurveType);
```

This function is used to get the native curve type of the font referred to by the `ATSUStyle` - cubic or quadratic.

Input Parameters

`ATSUStyle iATSUStyle`	The opaque ATSUI style object.

Output Parameters

`CurveType * oCurveType`	The format in which the curves of the selected font are stored - cubic or quadratic.

Glyph Paths

ATSUGlyphGetCubicPaths & ATSUGlyphGetQuadraticPaths

```
OSStatus ATSUGlyphGetCubicPaths(
ATSUStyle iATSUStyle,
GlyphID iGlyphID,
ATSCubicMoveToUPP iMoveToProc,
ATSCubicLineToUPP iLineToProc,
ATSCubicCurveToUPP iCurveToProc,
ATSCubicClosePathUPP iClosePathProc,
void *iCallBackDataPtr,
OSStatus *oCallBackResult);
```

This function returns the cubic outline paths for a single glyph via callback functions. The outlines that are returned are the hinted outlines at the size specified; to obtain the effect of unhinted outlines, request a very large size (e.g., 1000 points) and scale the curves back down to the desired size.

Note:
As of Mac OS X 10.1, the curves returned by this function are derived from quadratic curves, irrespective of the native curve type of the font.

Some fonts restrict access to their glyph paths; in that case, `ATSUGlyphGetCubicPaths` will return the error code `kATSNotPublicOutlinesErr`.

Input Parameters

`ATSUStyle iATSUStyle`	The opaque ATSUI style object.
`GlyphID iGlyphID`	The `glyphID` for which to retrieve outline data. This must be a valid glyph in the font specified by `iATSUStyle`.
`ATSCubicMoveToUPP iMoveToProc`	The pointer to a function that will handle pen movement.
`ATSCubicLineToUPP iLineToProc`	The pointer to a function that will handle the LineTo operation.
`ATSCubicCurveToUPP iCurveToProc`	The pointer to a function that will handle the `CurveTo` operation.
`ATSCubicClosePathUPP iClosePathProc`	The pointer to a function that will handle the `ClosePath` operation.
`void *iCallBackDataPtr`	This pointer is passed through to callback functions. The caller can use this pointer to pass data to the functions.

Output Parameters

`OSStatus * oCallBackResult`	When a callback function returns a non-zero result, `ATSGlyphGetCubicPaths` stops parsing the path and returns the result `kATSOutlineParseAbortedErr`; the value returned by the callback is returned in `oCallBackResult`.

Return Values

`noErr`	success
`paramErr`	`iATSUStyle` must be a pointer to a valid `ATSUStyle ATSUStyle`. The `iGlyphID` must be valid.
Memory manager errors	

```
OSStatus ATSUGlyphGetQuadraticPaths(
    ATSUStyle iATSUStyle,
    GlyphID iGlyphID,
    ATSQuadraticNewPathUPP iNewPathProc,
    ATSQuadraticLineUPP iLineToProc,
    ATSQuadraticCurveUPP iCurveToProc,
    ATSQuadraticClosePathUPP iClosePathProc,
    void *iCallBackDataPtr,
    OSStatus *oCallBackResult);
```

This function returns the quadratic outline paths for a single glyph via callback functions. The outlines that are returned are the hinted outlines at the size specified; to obtain the effect of unhinted outlines, request a very large size (e.g., 1000 points) and scale the curves back down to the desired size.

Some fonts restrict access to their glyph paths; in that case, `ATSGlyphGetQuadraticPaths` will return the error code `kATSNotPublicOutlinesErr`.

Input Parameters

`ATSUStyle iATSUStyle`	The opaque ATSUI style object.
`GlyphID iGlyphID`	The `glyphID` for which to retrieve outline data. This must be a valid glyph in the font specified by `iATSUStyle`.

`ATSQuadraticNewPathUPP iNewPathProc`	The pointer to a function that will handle the `NewPath` operation.
`ATSQuadraticLineUPP iLineToProc`	The pointer to a function that will handle the `LineTo` operation.
`ATSQuadraticCurveUPP iCurveToProc`	The pointer to a function that will handle the `CurveTo` operation.
`ATSQuadraticClosePathUPP iClosePathProc`	The pointer to a function that will handle the `ClosePath` operation.
`void *iCallBackDataPtr`	This pointer is passed through to callback functions. The caller can use this pointer to pass data to the functions.

Output Parameters

`OSStatus * oCallBackResult`	When a callback function returns a non-zero result, `ATSGlyphGetQuadraticPaths` stops parsing the path and returns the result `kATSOutlineParseAbortedErr`; the value returned by the callback is returned in `oCallBackResult`.

Return Values

`noErr`	success
`paramErr`	`iATSUStyle` must be a pointer to a valid `ATSUStyle ATSUStyle`. The `iGlyphID` must be valid.
Memory manager errors	

ATSUGlyphGetCurvePaths

```
OSStatus ATSUGlyphGetCurvePaths (
    ATSUStyle iATSUStyle,
    GlyphID iGlyphID,
    ByteCount *ioBufferSize,
    ATSUCurvePaths *oPaths);
```

This function returns the outline paths for a single glyph via data structures. The outlines that are returned are the hinted outlines at the size specified; to obtain the effect of unhinted outlines, request a very large size (e.g., 1000 points) and scale the curves back down to the desired size.

This function will only return quadratic paths. Apple recommends that you use the callback APIs, `ATSUGlyphGetCubicPaths` and `ATSUGlyphGetQuadraticPaths`, if possible.

Some fonts restrict access to their glyph paths; in that case, `ATSUGetCurvePaths` will return the error code `kATSNotPublicOutlinesErr` and NULL for oPaths.

Input Parameters

`ATSUStyle iATSUStyle`	The opaque ATSUI style object.
`GlyphID iGlyphID`	The `glyphID` for which to retrieve outline data. This must be a valid glyph in the font specified by `iATSUStyle`.
`ByteCount *ioBufferSize`	If `oPaths` is NULL, on output this parameter will contain the size of the buffer that would need to be allocated. If `oPaths` is not NULL, on input this is the maximum size of the passed in `ATSUCurvePaths` struct array; on output the actual size used.
`ATSUCurvePaths oPaths`	The glyph paths.

Return Values

`noErr`	success
`paramErr`	the `ioBufferSize` must not be NULL. `iATSUStyle` must be a valid `ATSUStyle`. The `iGlyphID` must be a valid `glyphID` for the font selected by the `ATSUStyle`.
Memory manager errors	

Glyph Data Access

ATSUGetGlyphInfo

```
OSStatus ATSUGetGlyphInfo(
    ATSUTextLayout iTextLayout,
    UniCharArrayOffset iLineStart,
    UniCharCount iLineLength,
    ByteCount *ioBufferSize,
    ATSGlyphInfoArray *oGlyphInfoPtr );
```

This function returns the glyph information from an `ATSUTextLayout`. The data is a copy of the internal information used by ATSUI, and can be modified. However, if you are going to use this `ATSGlyphInfoArray` when calling `ATSUIDrawGlyphInfo`, changing any information other than the following can cause undefined behavior when you attempt to draw: deltaY, idealX, screenX, caretX.

You should not change or dispose of the `ATSUTextLayout` associated with the returned glyph information until you are done with that glyph information.

Input Parameters

`ATSUTextLayout iTextLayout`	The opaque ATSUI text layout object.
`UniCharArrayOffset iLineStart`	The offset of the first Unicode character to get glyph data from.
`UniCharCount iLineLength`	Number of Unicode characters to get glyph data from.
`ByteCount *ioBufferSize`	If `oGlyphInfoPtr` is NULL, on output this parameter will contain the size of the buffer that needs to be allocated. As input, the size of the passed in `ATSGlyphInfoArray` struct, as output the actual size of `oGlyphInfoPtr`.
`ATSGlyphInfoArray oGlyphInfoPtr`	Pointer to the glyph info array for output.

Return Values

`noErr`	success
`paramErr`	`ioBufferSize` must not be NULL. `iTextLayout` must be a pointer to a valid `ATSUTextLayout`.
Memory manager errors	

ATSUDrawGlyphInfo

```
OSStatus ATSUDrawGlyphInfo(
    ATSGlyphInfoArray *iGlyphInfoArray,
    Float32Point iLocation);
```

This function is used to draw the glyphs in an `ATSGlyphInfoArray`. If the caller has modified data in `iGlyphInfoArray` other than the following, the results are undefined: deltaY, idealX, screenX, caretX.

Input Parameters

`ATSGlyphInfoArray iGlyphInfoArray`	Pointer to the glyph info data
`Float32Point iLocation`	The location in the current graphics environment at which to begin drawing all the glyphs in the glyph info array.

Return Values

`noErr`	success
`paramErr`	the `iGlyphInfoArray` must not be NULL.
Memory manager errors	

Device Specific Routines

ATSUGlyphGetScreenMetrics

```
OSStatus ATSUGlyphGetScreenMetrics (
    ATSUStyle iATSUStyle,
    ItemCount iNumOfGlyphs,
    GlyphID * iGlyphIDs,
    Sint32 iInputOffset,
    ATSGlyphScreenMetrics * oScreenMetrics );
```

This function is used to get the screen metrics of glyphs specified by an array of input `glyphIDs` and an `ATSUStyle`. The screen metrics are derived from the supplied style's font scaler's calculations based on the current QuickDraw grafPort's resolution and bit-depth, whether anti-aliasing is set for the style's point size, the style transform, and other set style attributes. Since there is only a single style input, only a single style run can be measured with each call.

Screen metrics are useful when you are working at the level of pixels, e.g., if you need to know exactly how many pixels a particular glyph will occupy.

`oScreenMetrics` must have been previously allocated by the caller to contain the requested number of returned metrics.

Input Parameters

`ATSUStyle iATSUStyle`	The ATSUI opaque style.
`ItemCount iNumOfGlyphs`	The number of glyph metrics requested and hence the number of `glyphIDs` provided as input.
`GlyphID *iGlyphIDs`	The address to the first `glyphID` for `ATSGetGlyphRenderingMetrics` to access its metrics.
`Sint32 iInputOffset`	The number of bytes for `ATSGetGlyphRenderingMetrics` to increment or decrement to find each successive `glyphID` provided as input starting at `iGlyphIDs`.

Output Parameters

`ATSGlyphScreenMetrics *oScreenMetrics`	The pointer to the caller supplied allocated block of memory filled with the requested number of metrics.

Return Values

`noErr`	success
`paramErr`	the `oScreenMetrics` and `iGlyphIDs` must not be NULL.
Memory manager errors	

Back to top

Downloadables

	Acrobat version of this Note (56K)	Download

Back to top