OS/2 Intelligent Font Interface

OS/2 Intelligent Font Interface Font Driver Interface Definition Document

Release 10.2

Author: IBM Hursley

Date: November 1990

Revise 1: Release 10.1
IBM Boca Raton Tetsuro Nishimura

Date: April 1995

Revise 2: Release 10.2
IBM Austin Marc L. Cohen

Date: September 1997

Introduction

This document describes the functions and interfaces of Font Drivers for OS/2 3.0. Although Font Drivers that meet the OS/2 1.3 specification will continue to work, drivers written to this specification will run as flat model, 32-bit extensions and will thus be expected to provide better performance, as well as more functionality.

Font Drivers are dynalink libraries (DLLs) that provide the Graphics Engine (GRE) with bitmap and outline descriptions of text characters.

It is intended that any company with font rasterization technology can implement a font driver. A font driver will provide entry points for enumerating the typefaces available, to get the metrics for a typeface and to retrieve an outline or a bitmap for a font. The Graphics Engine (GRE), which is part of OS/2 Presentation Manager (PM) will request the Font Driver to load and unload fonts, provide metrics for a face or character, and to provide characters in either bitmap or outline form. The intent is that the data format of the fonts on disk and the technology used to turn this data into attractive text of any size is hidden from the Graphics Engine. The Graphics Engine will be able to interact with several such drivers simultaneously. This allows OS/2 to have high quality scaleable fonts without committing to any one format or technology.

Readers of this document are assumed to have good knowledge of OS/2 Presentation Manager features in areas related to Fonts, such as the API calls which access outline fonts, how fonts are installed and the difference between system and device fonts.

The following items are enhanced for version 21 which will be support by the OS/2 2.X and the WPOS OS/2 1.0 graphics engine.
Bitmap font support
New glyphlist names ("UNICODE", "PMJPN", "PMKOR", "PMCHT", "PMPRC")

Additional error codes

General Features of the IFI

Access to the Font Driver is restricted. Only the Graphics Engine will directly call the Font Driver Entry points. Printer or display drivers will use services provided by various GPI and Graphic Engine interfaces. The Font Drivers will be invisible to applications.
The code for each Font Driver is provided as a Dynalink Library with the standard dynalink extension of 'DLL'. Font driver DLLs have a single exported entry point called:

```
   FONT_DRIVER_DISPATCH_TABLE
```

The dispatch table points to a FDHEADER structure as follows:

```
typedef struct _FDHEADER { /* fdhdr */
   ULONG           cbLength;            // Length of FDHEADER
   UCHAR           strId[16];           // String 'OS/2 FONT DRIVER'
   UCHAR           szTechnology[40];    // Identifier of Font Driver technology
   ULONG           ulVersion;           // IFI version number (20)
   ULONG           ufDeviceCaps;        // Capabilities of device
   PFDDISPATCH     pfddisp;
} FDHEADER;
```

where:

cbLength	The length of the dispatch table, including the signature.
strId	'OS/2 Font Driver' *or 'IFI FONT Driver'*
szTechnology	The name of the font technology.
ulVersion	IFI version number supported by this font driver. The first version of this for 32-bit font drivers will be decimal 20. *The enhanced version for the OS/2 2.x and the WPOS OS/2 1.0 graphics engine will be decimal 21.*
ufDeviceCaps	This is a set of flags put in for future expansion capability. All bits should be set to 0.
pfddisp	A pointer to a table of Fd* entry points.

The table of Fd* entry points consists of a table of 10 entry points identified as follows:

```
typedef struct _FDDISPATCH { /* fdisp */
        PFDLFF      FdLoadFontFile;
        PFDQF       FdQueryFaces;
        PFDUFF      FdUnloadFontFile;
        PFDOFC      FdOpenFontContext;
        PFDSFC      FdSetFontContext;
        PFDCFC      FdCloseFontContext;
        PFDQFA      FdQueryFaceAttr;
        PFDQCA      FdQueryCharAttr;
        PFDCLF      FdClaimFontFile;
        PFDCFF      FdConvertFontFile;
        PFDQFF      FdQueryFullFaces;
} FDDISPATCH;
```

The functions performed by these entry points are discussed later in this document.
Font Drivers entry points are called as _syscall. The rules for this type of entry point are as follows *for OS/2 on the Intel platform:*

> The called function must preserve EBX, ESI, EDI, EBP and all segment registers.
> The called function must remove parameters from the stack.
> Return values are passed in EAX. ECX and EDX may be destroyed.

For WPOS OS/2 1.0 on the PowerPC platform, the calling convention is T.B.D.

Error Handling

In general a return value of ~~EAX == -1~~ *-1 from the font driver* indicates that an error has occurred. It is the responsibility of the Font Driver to log the particular error code by calling WinSetErrorInfo(). The font driver should assume that the graphics engine does not make errors and should therefore not check (except possibly for debugging purposes) the values passed to it - for example font file handles and flag bits.

Normally errors fall into 3 categories:

PMERR_COORDINATE_OVERFLOW	arithmetic problems in generating font output
PMERR_INSUFFICIENT_MEMORY	insufficient resources
PMERR_BASE_ERROR	bad return code from an OS/2 base DOS... function. The error number is also logged in this case.

WinSetErrorInfo() is documented in the ~~OS/2 Device Drivers book volume 2.~~ OS/2 Technical Library Presentation Driver Reference.

Things a Font Driver does not need to do:

- A Font Driver need not be reentrant but must be serially reusable.
- A Font Driver does not normally need to be aware of processes. However see discussion of font files below.
- A Font Driver does not need to do caching. The Graphics Engine will manage caching of images, font metrics and font contexts.

Things a Font Driver should avoid:

Use of Floating Point. There are a number of technical difficulties to surmount in using floating point from a DLL.

WPOS OS/2 1.0 on PowerPC can use floating point instructions. The restriction is only for OS/2 2.X and WPOS OS/2 1.0 on the Intel platform.

Use of malloc(). Use SSAllocMem instead (see Memory Allocation below)

Font Files:

The preferred form of fonts is in a DLL. This allows the font data to be in discardable read-only memory. The Font Driver is given the opportunity to convert font files from the distribution format to an optimal format during installation of the font by the Control Panel.
See the section on FdLoadFontFile() for more details on access to font data in a DLL.

See the section on Memory Allocation for more details on access to font data in dynamically acquired memory.

Although the preferred form of fonts is in a module format as far as memory is concerned, a large character set font may not be designed in the module format, since there is a crucial binary resource limitation of 64K for OS/2 and WPOS OS/2 1.0.

Memory Allocation

A Font Driver is allowed to use the Graphics Engine's Selector Server to allocate dynamic memory for fonts and other data. Memory allocated this way is automatically addressable to all PM processes, thus eliminating the need of any process knowledge.

SSAllocMem

ULONG APIENTRY SSAllocMem(BaseAddress, ObjectSize, Flags)

```
PVOID       BaseAddress;  // A pointer to a variable to receive the
                          //  base address of the allocated memory.
ULONG       ObjectSize;   // Size, in bytes, of the object. The size
                          //  gets rounded up to the next page boundary.
ULONG       Flags;        // Reserved. Must be zero.
```

Purpose

This function allocates a shared memory object that is managed by the selector server component of the graphics engine. It ensures that the returned memory object is a global memory object which can be accessed from any process. Use SSFreeSeg() to free the storage.

Return Codes:

SSAllocMem returns a ULONG value:

NO_ERROR
ERROR_NOT_ENOUGH_MEMORY
ERROR_INVALID_PARAMETER

Other return codes listed under DosAllocSharedMem may also apply.

SSFreeMem

```
ULONG        APIENTRY    SSFreeMem(BaseAddress)

PVOID        BaseAddress;  // address of storage to be freed
```

Return Codes: SSFreeMem returns a ULONG value:

NO_ERROR
ERROR_NOT_ACCESS_DENIED
ERROR_INVALID_PARAMETER

SSAllocMem and SSFreeMem are also available on WPOS OS/2 1.0.

Abnormal Process Termination

The IFI has been designed so that font drivers do not need to be concerned about most aspects of abnormal process termination. Resources allocated by font drivers are all global, and should not be freed when a process ends.
Font drivers should protect against abnormal termination within their own code. For example if Font Driver needs semaphores it should use fast safe RAM semaphores and use DosExit-List processing to clear the semaphore, and should attempt to repair damage done to any data structures to which the semaphore was protecting access.

Installation in OS/2 *and WPOS*

To install a font driver in OS/2 an entry is added to the OS2.INI file using the PRF... calls (using the HINI_USER handle).

The entry for a font driver is:

	Application	Key	Value
	---------------	------	-----
	PM_Font_Drivers	Filename	Fully qualified path and name
e.g.		"PMXXX.DLL"	"C:\OS2\DLL\PMXXX.DLL"

Following is no longer true. Font drivers will be called at ring 2 or 3, depending on how it is arrived at through the destination presentation driver.
Since the font driver will be called from ring 2, and therefore has I/O privilege, the CONFIG.SYS file must also be edited to add:

IOPL=YES

Since OS/2 2.0 or later always specify the statement of IOPL=YES in the CONFIG.SYS file, the installation of the font driver may not need the consideration.
On WPOS OS/2 1.0, the registry may be used to store the font driver entries instead of the OS2.INI file.

Facename management

The Graphics Engine will manage the Facename list much the same way as it does today. In addition to the lMatch, The Graphics Engine will treat all of the fonts in a single font data file as an array of fonts. This will allow GRE to make the proper selection for private and public fonts if there is facename collision.

Coordinate Systems

Numerical data passed back and forth between the Graphics Engine and Font Drivers is in different coordinate systems depending on the requirements. There are two systems used:

Pixel or bitmap Coordinates are used to pass information about the actual size of bitmaps, the character origin within the bitmap, etc.
Notional Coordinates are used to communicate resolution independent information about the font, for example the width of a character and pair-kerning amounts. Notional coordinates are defined by the number of Notional units in the 'M square' and are generally the units in which an outline font is defined.
In bitmap fonts, notional coordinates are defined by the pixel units of a bitmap font. The widths of characters and pair-kerning amounts will be passed in the pixel units for a bitmap font.

Note that OS/2 PM does not support very large Notional units per 'M'. For large values, there will be inaccuracies in the size of small characters on low resolution devices, although PM will still position the characters accurately.
Glyphlists

PM accesses glyphs via the IFI using a simple 2 byte index number. The mapping between this index and glyphs is called the font's Glyphlist.
PM applications access glyphs via the API using a 1 or 2 byte index. The mapping between this index and glyphs is called a codepage.

PM has two modes of operation. In the first mode, called translate mode, PM recognizes the Glyphlist of the font, and the codepage of the application and automatically translates the application's codepage index (called a codepoint) into a glyph index. The set of glyphlists and codepages supported in this mode by PM is fixed for a given release but may increase from release to release. In the second mode, called passthru mode, PM does not recognize the glyphlist of the font and simply performs a null translation between the application's codepoint and the font's glyph index.

Fonts accessed via the IFI identify their glyphlist by means of the glyphlistname field in the font directory. OS/2 release 1.3 supports fonts in translate mode if the glyphlistname starts with the 2 characters 'PM'. PM in OS/2 release 1.3 supports a glyphlist name of 331 characters called 'PM331'. Release 2.0 supports 383 characters with a glyphlistname of 'PM383'. Release 4.0 supports 504 characters with a glyphlistname of 'PM383'.
OS/2 2.X and WPOS OS/2 1.0 will support large character set fonts with glyphlist names of 'UNICODE', 'PMJPN', 'PMKOR', 'PMCHT' and 'PMPRC'. See the appendix for the glyph index definition of the glyphlists.

API note: Fonts supported in Translate mode are reported to applications as having a codepage of 0 which signals to applications that any PM-supported codepage can be used. Fonts supported in passthru mode are reported as being codepage 65400. Only the first 256 glyphs of a Passthru font are accessible by an application. A typical example of a passthru font is the Symbol font shipped with OS/2 1.3. *A font with the NULL glyphlist ('') will be recognized as a passthru font by the graphics engine.*

ABC Widths

ABC spacing defines the size of the character's image and the left and right side bearings of the character. The B space is the distance from the left most part of the actual character image to the right most part of the character image. The A space is the distance from x = 0 to the left edge of the B space and the C space is the corresponding distance on the right side. Note that the A, and C space will often be negative in the case of a italic or script font. The increment of a character is the sum of the 3 spaces. A + B + C = the width of the character.

If kerning is used by the application then a suitable value for the character increment between the characters giFirst and giSecond (the first and second glyphs in a kerning pair) is the kerning amount returned for that pair PLUS A + B + C.

Font Driver Entry Points

This section describes each entry point into a Font Driver. Font Drivers must implement all entry points.

FdConvertFontFile

LONG FdConvertFontFile(pszSrc, pszDestDir, pszNewName);

```
        PSZ pszSrc;                 //Fully qualified file name of file to be
                                    //  converted
        PSZ pszDestDir;             //Directory where converted file is to be
                                    //  placed
        PSZ pszNewName;             //name of converted file
```

Purpose:

If **pszDestDir** and **pszNewName** both point to valid strings, then this call requests that the font driver install the distribution font data file to the directory specified in **pszDestDir**. At this time, the font driver will be able to do any conversions necessary to prepare the file for use. The font driver should return the new name of the file in the 256 byte buffer pointed to by ~~pszName~~ ***pszNewName***.
If **pszDestDir** and **pszNewName** are both NULL, then this call requests the font driver to remove the installed files at **pszSrc**. The engine will not call the font driver with only one of **pszDestDir** and ~~pszName~~ ***pszNewName*** NULL.

Returns:

0	OK
-1	Error
-2	The target file already exists and is not the same as the source file.
-3	*Short of source files.*
-4	*Installation is cancelled by the user.*

FdLoadFontFile

HFF FdLoadFontFile (PSZ pszFileName);

```
PSZ  pszFileName              // fully qualified file name
```

Purpose:

This call informs the font driver that a particular file may be used for opening font contexts. The returned HFF value must be process independent.
The Graphics Engine may make several calls with the same filename, so the Font Driver must count how many times **FdLoadFont** is called for each font file. For each **FdUnloadFont** call, the Font Driver must decrement this count, and not actually unload the font file until the reference count is zero.
It is the responsibility of the font driver to quickly and safely identify valid font files. If the font driver does not recognize the file extension, it should reject the file immediately. If it does then some further check should be made to verify that the font file belongs to the font driver.
Font file names are fully qualified. However, when comparing names for being the 'same' file case should be ignored.

Implementation Notes

The font driver should maintain a record of the loading of font files containing (at least) the following information PER FILE:
File name
Count of the number of times **FdLoadFontFile** has been called for this file minus the number of times **FdUnloadFontFile** has been called for it.
If the font driver calls other DLLs, then it is the font driver's responsibility to ensure that addresses it uses from that DLL are valid. There are 3 ways:

(inefficient) is to issue **DosLoadModule** and **DosGetProcAddr** at each call and (optionally) **DosFreeModule** after the call.
During DLL initialization:

Issue **DosLoadModule** and **DosGetProcAddr** and save the procedure address (because this is executed on the shell process, which never dies, the DLL is now permanently loaded and procedure addresses are fixed.) Note due to an OS/2 restriction, the loaded DLL cannot have a DLL initialization routine when **DosLoadModule** is issued from an initialization routine.
On each call:
~~Issue LAR instruction to check for accessibility.~~
~~If LAR fails, issue DosGetResource2.~~ *T.B.D.*
If that fails issue **DosLoadModule** and **DosGetProcAddr**.

Use the linker/loader facility. This can only be used if the required DLL is guaranteed always to be present in the system, and its name is known at link time.

If the Font Driver obtains font data from DLLs then it is the font driver's responsibility to ensure that the resource addresses it uses from that DLL are valid: There are two ways:

At **FdLoadFontFile** time:
Check whether this file has been loaded before.
If it has, return the same HFF value as was returned on the original request.
If not:
DosLoadModule
DosGetResource2 for every resource (to guarantee address will stay same) and save the resource addresses.
return a globally unique HFF
On each use of a resource in the DLL:
~~Issue LAR instruction to check for accessibility.~~
~~If LAR fails, issue DosGetResource2.~~ *T.B.D.*
If that fails issue **DosLoadModule** and **DosGetResource2**.
At **FdLoadFontFile** time:
DosLoadModule
Check whether this file has been loaded before ON THIS PROCESS
If it has, return the same HFF value as was returned on the original request.
If not, return a globally unique HFF
On each use of a resource in the DLL:
Issue **DosGetResource2**.
If that fails issue **DosLoadModule** and **DosGetResource2**.

Returns:

HFF hff;	Font File Handle - note 0 is invalid.
-1	Error or not recognized as a font file for this font driver.

FdUnloadFontFile

LONG FdUnloadFontFile(hff);

```
HFF hff;            //Font File Handle
```

Purpose
Informs the Font Driver that the specified font file will no longer be used for opening new Font Contexts.
Implementation Notes
The storage associated with the HFF should be freed. The Engine will not issue this call until all HFCs associated with this HFF have been Closed.

Returns:

0	Successfully unloaded
-1	Failed to unload

FdQueryFaces

LONG FdQueryFaces (hff, pifiMetrics, cMetricLen, cFontCount, cStart)

```
HFF             hff            // Font File handle
PIFIMETRICS            pifiMetrics;    // Buffer for the metrics
ULONG           cMetricLen;    // Length of the metrics structure
ULONG           cFontCount;    // # of fonts wanted.
ULONG           cStart;        // index of the font to start with
```

Purpose
The font driver provides a list of all typefaces available as an array of FontMetrics. To get FontMetrics for a set of Faces, cStart is used to index the array with the first entry being entry 0, cFontCount to specify number of elements requested and ~~paMetrics~~ *pifiMetrics* to point to buffer to copy items requested. Only first cMetricLen bytes of IFIMETRICS structure are copied.
If cMetricLen is 0 then only the count is returned.
Note that the Graphics Engine sometimes asks for less than the total number of bytes in the IFIMETRICS structure. E.g. when processing GpiQueryFontFileDescriptions it asks for the first 64 bytes of the metrics information.
Conversely, future versions of the Graphics Engine may ask for more data than the Font Driver knows about. In this case, the Font Driver should skip over the additional data areas before starting data for the following metrics structure.
The version 21 Font Driver may return bitmap font metrics in which IFI_METRICS_OUTLINE flag is turn off in the fsType field. The Graphics Engine will handle the bitmap font as a Graphics Engine bitmap font properly.
The various fields of the IFIMETRICS structure are described at the end of this document.

Return:

#	Number of fonts
-1	Error

FdOpenFontContext

HFC FdOpenFontContext(HFF hff, ULONG ulFont);

```
HFF        hff;        // Font File handle of file to get font from
ULONG      ulFont;     // index of required font in facename directory
```

Purpose
Requests that a Font Context be opened. The required font is indicated by ulFont. This is a zero-origin index into the array of fonts as returned by FdQueryFaces.
This call makes the context automatically available to all PM processes.
Implementation Notes
It intended that a Font Driver can rely on the Engine to ensure that only a single thread is using the font driver at any one time. However during development it may be desirable for font drivers to check this behavior by marking the driver as busy when in use and returning an error if so. This checking probably should not be performed in the shipped product.
The Font Driver should avoid placing a limit on the number of HFCs that may be created.

Returns:

HFC hfc;	Font context handle. Note 0 is invalid
-1	Error

FdSetFontContext

LONG FdSetFontContext(HFC hfc, PCONTEXTINFO pci)

```
HFC              hfc;        // Font Context
PCONTEXTINFO     pci;        // Context definition, see below..
```

Purpose
Sets or resets the transforms (etc.) for a font.
Implementation Notes
The Font Driver should determine which of the font context parameters have changed, and take whatever actions it needs to do in order to render the new font.
Note that the current PM Engine only ever opens a single font context. So to support applications (such as DTP applications) that switch fonts frequently, the font driver should be designed to make this call efficient even when all parameters of the font context (including the actual facename) are changed.
The contents of the CONTEXTINFO structure are as follows:

```
typedef struct _CONTEXTINFO /* ci */
{
        ULONG      cb;           /* Length in bytes of this structure. */
        ULONG      fl;           /* Flags. */
        SIZEL      sizlPPM;      /* Device resolution in pels/meter. */
        POINTFX    pfxSpot;      /* Spot size in pels. */
        MAT2       matXform;     /* Notional to Device transform. */
} CONTEXTINFO;
```

cb	This is the length of the CONTEXTINFO structure in bytes. It is presumed that future versions of the CONTEXTINFO structure could have more fields than shown here. If so, a future driver could distinguish between old and new versions of the structure by its length. The present Font Driver should check that the structure contains at least all the fields listed above. If it contains more, the extra fields should be ignored.
fl	Reserved for future use.
sizlPPM	This is the X and Y resolution in pels per meter.
pfxSpot	This is the spot size in pel units in the X and Y direction. This is included since the size of a single drawn pel may be much larger than the pel resolution would imply. The SIZEFX structure is a fractional version of the SIZEL structure, and is defined as follows:

```
typedef struct _POINTFX { /* ptfx */
        FIXED x;
        FIXED y;
} POINTFX;
typedef POINTFX FAR *PPOINTFX;
```

matXform This is the multiplicative part of the Notional to Device coordinate transform. The transform includes the shear angle of the font and the baseline direction of the char. Device coordinates are always in pels. The MAT2 structure is as follows:

```
typedef struct _MAT2 /* mat */
{
        FIXED   eM11;
        FIXED   eM12;
        FIXED   eM21;
        FIXED   eM22;
} MAT2;
```

Device coordinates (x',y') are derived from World coordinates (x,y) by the formula:

```
x' = x * eM11 + y * eM21;
```

```
                y' = x * eM12 + y * eM22;
```

Arbitrary scaling, rotation, and shears are allowed, as are singular transforms. The full transform would also include an offset, but we assume that the Font Driver does not need that additional information.
For bitmap fonts, the Graphics Engine will pass only the unit transform (1,0,0,1) on to the Font Driver. Arbitrary scaling, rotation and shears are not passed.

Returns

0	Success
-1	Failure

FdCloseFontContext

BOOL FdCloseFontContext(hfc);

```
    HFC hfc;            // Font Context
```

Purpose
Closes, i.e. deletes, the Font Context.
Implementation Notes
See **FdOpenFontContext**. The HFC value can be immediately reused for new Open Contexts after this call.
The Engine will ensure that all references to the context, in all processes that have used the context, are erased before calling this function. This may mean that **FdCloseFontContext** is called during abnormal termination of a process.
Returns:

0	Success
-1	Failure

FdQueryFaceAttr

LONG FdQueryFaceAttr(hfc,iQuery,pBuffer,cb,pagi,giStart)

```
    HFC       hfc;        // Font Context
    ULONG     iQuery;     // Query type.
    PBYTE     pBuffer;    // Buffer for returned data.
    ULONG     cb;         // Size of buffer, in bytes.
    PGLYPH    pagi;       // Glyph index list, for widths.
    GLYPH     giStart;    // Glyph index to start at, for widths.
```

Purpose
Requests that data to be written to given buffer, depending on value of iQuery. The Font driver must provide as many items as the buffer has room for. If pBuffer is NULL only the number of items to be copied is returned.

Note that the quantities returned here do not depend on the setting of the font context. In particular they do not depend on the current font transform.

iQuery == **FD_QUERY_ABC_WIDTHS**

Writes a range of character ABC widths to the buffer. ABC widths are returned in consecutive ABC_TRIPLETS structures in the memory pointed to by pBuffer. The ABC_TRIPLETS structure looks like:

```
typedef struct _ABC_TRIPLETS /*abc*/
{
        LONG        lA;
        ULONG       ulB;
        LONG        lC;
}ABC_TRIPLETS;
```

If pagi is NULL, then cb/sizeof(ABC_TRIPLETS) sets of triplets are returned for consecutive glyphs starting at giStart. Otherwise cb/sizeof(ABC_TRIPLETS) sets of triplets are returned for the list of glyphs pointed to by pagi.
The ABC widths should be returned in Notional Coordinates.
Currently OS/2 only calls the font driver with pagi == NULL.

iQuery == **FD_QUERY_KERNINGPAIRS**

Writes kerning pairs to the buffer. The arguments pagi and giStart are ignored. The kerning pairs are written as an array of *FD_KERNINGPAIRS* structures *in the memory pointed to by pBuffer,* defined as follows:

```
typedef struct _FD_KERNINGPAIRS /* krnpr */
{
        GLYPH       giFirst;
        GLYPH       giSecond;
        LONG        eKerningAmount;
} FD_KERNINGPAIRS;
```

The kerned pair consists of giFirst and giSecond. The kerning amount indicates how much the inter-character spacing should be adjusted. A positive number means the characters should be moved further apart. The kerning should be returned in Notional Coordinates. The graphics engine will sort the kerning pairs appropriately before handing them to the application program.

Returns:

#	Number of items filled in

-1 Error

FdQueryCharAttr

LONG FdQueryCharAttr(hfc, pCharAttr, pbmm)

```
HFC             hfc;        // Font Context
PCHARATTR       pCharAttr;  // Char Attr info
PBITMAPMETRICS  pbmm;       // Char metrics info

typedef struct _CHARATTR
{
        ULONG       cb;         // Length of structure.
        ULONG       iQuery;     // Query type.
        GLYPH       gi;         // Glyph index in font.
        PBYTE       pBuffer;    // Bitmap buffer.
        ULONG       cbLen;      // Size of buffer in bytes.
        ULONG       fl;         // Boundary of bitmap buffer.
} CHARATTR;
```

Purpose
Requests data to be written to the given buffer pointed by pBuffer, depending on the value of iQuery. iQuery is a combination of bits, each of which requests different information be returned. Multiple bit flags may be set to request multiple pieces of information.
fl is added for the version 21 Font Driver.

FD_CHARATTR_ALIGNED_8	*The width of the bitmap returned is aligned in 8 pixels.*
FD_CHARATTR_ALIGNED_16	*The width of the bitmap returned is aligned in 16 pixels.*
FD_CHARATTR_ALIGNED_32	*The width of the bitmap returned is aligned in 32 pixels.*
FD_CHARATTR_NO_CACHE	*The bitmap should not be cached by the graphics engine since the bitmap image is volatile (may be changed later). This option is added to support the user defined characters.*

Version 20 Font Driver will not return the fl field, therefore cb will be 20. The width of the bitmap returned from the version 20 Font Driver will be only 32 pixels aligned. The version 21 Font Driver may return the fl field, and cb must be 24 in that case.

(iQuery & **FD_QUERY_BITMAPMETRICS**) == TRUE

Writes the BitmapMetrics data to the buffer. The engine will request both the BitMapMetrics and the actual bitmap in the same call. The BITMAPMETRICS structure contains information about the bitmap returned from FD_QUERY_CHARIMAGE, and is defined as follows:

```
typedef struct _BITMAPMETRICS /* bmm */
{
        SIZEL       sizlExtent;
        ULONG       cyAscent;
        POINTFX     pfxOrigin;
    } BITMAPMETRICS;
```

where:

sizlExtent The width and height of the bitmap returned in pixels. Note that although the bitmap must be padded to 32 pixels wide, sizlExtent.cx gives the REAL width.

If (iQuery & **FD_QUERY_CHARIMAGE**) == FALSE then sizlExtent.cx, sizlExtent.cy can be greater than the actual values (i.e. estimates at least as large as the real values).

cyAscent This field is reserved and should be zero.

pfxOrigin The position of the top left-hand corner of the bitmap relative to the character origin in device coordinates (pixels). If the character origin is at (x0, y0) then the first row of pixels returned is placed at: (pfxOrigin.x + x0, pfxOrigin.y + y0)
(pfxOrigin.x + x0, pfxOrigin.y - 1 + y0)
(pfxOrigin.x + x0, pfxOrigin.y - MAKEFIXED(sizlExtent.cy - 1, 0))
The origin is the 'center' of the pixel. Thus any origin returned is added to the current position (both fractional) and the result rounded to the 'nearest' integer to find the position of the top leftmost pixel. If (iQuery & FD_QUERY_CHARIMAGE) == FALSE then pfxOrigin is not required to be returned.

(iQuery & **FD_QUERY_OUTLINE**) == TRUE
Given the description of the device, cell size, transforms, and selected font in hfc, the Font Driver is asked to draw the outline of a single character image. This allows the Graphics Engine to put the character outline into a path for many interesting uses: clip paths, multicolored effects in the characters, etc. The outline is provided as a list of lines, polygons and splines in the given buffer. The Polygons are made up of primitives whose vertices are given relative to the character origin. A Polygon is a self describing record with a header POLYGONHEADER containing the following fields:

```
    typedef struct _POLYGONHEADER {
        ULONG       cb;
        ULONG       iType;          /*  Must be FD_POLYGON_TYPE */
    } POLYGONHEADER;
```

cb The size of this particular polygon record in bytes. This is used to determine how many primitives make up the polygon. All fields of the header and the following primitives are included in this size.

iType This must be FD_POLYGON_TYPE to identify this as a POLYGON record.

The POLYGONHEADER is followed by a list of polygon primitive records. There are two types of primitives allowed, lines and cubic splines. Line primitives are defined as follows:

```
typedef struct _PRIMLINE /* lnp */
{
        ULONG       iType;          // Must be PRIM_LINE.
        POINTFX     pte;            // Starting vertex of line.
} PRIMLINE;
```

Likewise, spline primitives are defined as:

```
typedef struct _PRIMSPLINE /* splnp */
    ULONG     iType;          // Must be PRIM_SPLINE.
    POINTFX  pte[3];          // Starting vertex of spline, control points.
} PRIMSPLINE;
```

For both primitive types, the ending vertex is omitted. It can be found as the starting vertex of the following primitive. The ending vertex of the last primitive can be found as the starting vertex of the first primitive. I.e. every polygon record describes a closed figure.

All points in the polygon are assumed to be in Device Coordinates and relative to the character origin. The Graphics Engine will offset the outline correctly before inserting it into a path.

Any number of POLYGON records may be put in the buffer, one for each disconnected section of the character outline. Typically the Graphics Engine will call first with CHARATTR.cbLen = 0 to find the total size of the data, then will call this function again using the length returned on the first call in CHARATTR.cbLen.

The Graphics Engine will fill the given polygons using the PM filling rule specified by the Application (WINDING or ALTERNATE rules). The outlines returned should be such that the internal WINDING count is 1. Note that the PM filling rules determine that the border curves of the polygon are always filled.
The Graphics Engine will not call this function for the font context of a bitmap font.

(iQuery & **FD_QUERY_CHARIMAGE**) == TRUE

Given the description of the device, cell size, transforms, and selected font in hfc, the Font Driver is asked to draw a single character image into the given bitmap buffer.

If CHARATTR.cbLen is zero the Font Driver should only return the size of the image that would have been returned.

If additionally (iQuery & **FD_QUERY_BITMAPMETRICS**) == TRUE the graphics engine has precomputed bounds for the size of the bitmap which will be returned. This improves performance by avoiding requiring the font driver to rasterize everything twice, first to find the bitmap size, then to return the bitmap. The engine uses the following algorithm for precomputing the size of the bitmap:.in +.2i For a given font deduce 4 points:

P0 = (-lMaxDescender, A)
P1 = (lMaxAscender , A)
P2 = (-lMaxDescender, B)
P3 = (lMaxAscender , B)

where:

lMaxDescender and lMaxAscender come from the IFIMETRICS
A = min(all font characters, a_space) and may be negative
B = max(all font characters, a_space + b_space)

(A and B are in effect pseudo a and (a + b) spaces.)

Thus, the parallelogram P0, P1, P3, P2 is a bounding box for the whole font (ignoring rounding), given that the characters are positioned at the 'origin'.

Now transform these 4 points to pixel coordinates, getting P0', P1', P2', P3'. The character origin is still at the origin.

Now compute:

Xmin = min(P0'.x, P1'.x, P2'.x, P3'.x)

and similarly Xmax, Ymin and Ymax.

Defining:

Ascent = ceil(BITMAPMETRICS.pfxOrigin.y)
Descent = ceil(BITMAPMETRICS.sizlExtent.cy - BITMAPMETRICS.pfxOrigin.y)

then the following is assumed:

Ascent max(ceil(Ymax),0) + 1
Descent max(ceil(-Ymin),0) + 1
BITMAPMETRICS.sizlExtent.cx ceil(Xmax)-floor(Xmin)+2
BITMAPMETRICS.sizlExtent.cy Ascent + Descent

The important thing here is that the top of the bitmap is rounded up on average 1.5 pixels and similarly for the bottom. Similarly for the width of the bitmap.

Returns:

#	Number of bytes in buffer
0	~~Codepoint~~ *Glyph index* not in font
-1	Error

FdQueryFullFaces

LONG FdQueryFullFaces(hff, pBuffer, cBufLen, cFontCount, cStart)

```
HFF         hff          // Font file handle.
PFFDESCS2   pBuf         // Buffer to hold family and face names.
PULONG      cBufLen      // Length of the buffer.
PULONG      cFontCount   // Number of fonts wanted/returned.
ULONG       cStart       // Index of the font to start with.
```

Purpose
This call returns a list of all typefaces as an array of FFDESCS2 in pBuf. The Font driver must provide as many items as buffer has room for. If cBufLen is 0 then only the size of the required buffer is returned. The size of the buffer needed to return all of the faces will always be returned in cBufLen. Succeeding FFDESCS2 structures in the return buffer will each be aligned on a four byte boundary. The number of fonts returned will be placed in cFontCount. The FFDESCS2 structure is defined as follows:

```
typedef struct _FFDESCS2 {
        ULONG   cbLength;
        ULONG   cbFacenameOffset;
        UCHAR   abFamilyName~~[4]~~[1];
} FFDESCS2;
```

where:

cbLength	is the length of this instance of the FFDESCS2 structure. This length is always rounded up to a multiple of four bytes.
cbFacenameOffset	is the byte offset from the start of the structure to the first character of the Facename. The Facename is a null terminated ASCII string. This length is always rounded up to a multiple of four bytes.
abFamilyName	is a null terminated ASCII string containing the Family name of the font.

The IFIMETRICS structure

This is an explanation of the IFIMETRICS fields.
IFIMETRICS is a parallel structure with FONTMETRICS as returned to applications in the GpiQueryFonts() API call (see OS/2 toolkit documentation). FONTMETRICS fields are derived from IFIMETRICS in an obvious way, except where described below.

~~ULONG cb size of IFIMETRICS structure.~~

UCHAR szFamilyname[~~FACESIZE~~ *32*] Specifies the family name of the font. Examples of common family names are Courier, Helvetica, and Times New Roman.

UCHAR szFacename[~~FACESIZE~~ *32*] Specified the typeface of the font. Examples of common typeface names are Courier, Helvetica Bold.

UCHAR szGlyphlistName[~~FACESIZE~~ *32*] Name of the glyphlist. ~~The only recognized name at present is PM331 - indicating PM's 331 glyphs as supported in OS/2 release 1.3.~~ *"PM383", "PMJPN", "PMKOR", "PMCHT", "PMPRC", "UNICODE" and "" (Null - passthru encoding) are supported by OS/2 2.X and WPOS OS/2 1.0.*

USHORT idRegistry IBM registration number. If this font has been registered with IBM then the number assigned can be placed in this field. Otherwise use 0.

LONG lCapEmHeight *Unit: Notional Coordinates. Specifies the height of the upper case M.. It is also called the EM square.*
For outline Fonts, this field is effectively a duplicate of lEmSquareSizeY, and OS/2 ignores it when computing FONTMETRICS and substitutes lEmSquareSizeY.
Font Drivers should set it to -1L.

LONG lXHeight Unit: Notional Coordinates. Specifies the average height of lowercase characters, measured from the baseline to the top of the character.

LONG lMaxAscender Unit: Notional Coordinates. Specifies the maximum height of any character in the font, measured from the baseline to the top of the tallest character. The max ascender may go beyond the top of the EM square.

LONG lMaxDescender Unit: Notional Coordinates. Specifies the maximum depth of any character in the font, measured from the baseline to the bottom of the lowest character. The max Descender may go beyond the bottom of the EM square. This number is normally positive, indicating the descenders go below the baseline.

LONG lLowerCaseAscent Specifies the maximum height of any lowercase character in the font, measured from the baseline to the top of the ascender of the tallest lowercase character.

LONG lLowerCaseDescent Unit: Notional Coordinates. Specifies the maximum depth of any lowercase character in a font, measured from the baseline to the bottom of the descender on the lowest lowercase character. This number is normally positive, indicating the descenders go below the baseline.

LONG lInternalLeading Unit: Notional Coordinates. Specifies the amount of space to be subtracted from ~~MaxAscender~~ *lMaxAscender* to give a font design dependent, but Glyphset independent measure of the distance above the baseline that characters extend. It approximates the visual 'top' to a row of characters without actually looking at the characters in the row.

The recommended use of this field by applications is to use it to position the first line of a block of text by subtracting it from ~~MaxAscender~~ *lMaxAscender* and positioning the baseline that distance below whatever is above the text. For compatibility with early releases of some applications, OS/2 currently ignores this field when computing FONTMETRICS and substitutes:

lMaxBaselineExt - lEmHeight

Hence Font Drivers should set this field to -1L.

LONG lExternalLeading Unit: Notional Coordinates. Specifies the amount of guaranteed white space advised by the font designer to appear between adjacent rows of text.

The recommended use of this field by applications is to add it to ~~MaxBaseline-Extent~~ *lMaxBaselineExtent* to obtain the vertical (line - to - line) escapement. Note however that many applications ignore it and add a constant percentage of the point size.

PM's built-in fonts have 0 in this field.

LONG lAveCharWidth Unit: Notional Coordinates. Specifies the average character width for characters in the font. The average character width is determined by multiplying the width of each lowercase character by a predetermined constant, adding the results, and then dividing by 1000. For Roman character set, letters and their predetermined constants are listed as follows:

Letter	Pre-Assigned Factor	Letter	Pre-Assigned Factor
a	64	b	14
c	27	d	35
e	100	f	20
g	14	h	42
l	63	j	3
k	6	l	35
m	20	n	56
o	56	p	17
q	4	r	49
s	56	t	71
u	31	v	10
w	18	x	3
y	18	z	2
space	166		

For FIXED PITCH fonts this value must be the same as the (A width + B width + C width) (escapement) of each character.

LONG lMaxCharInc Unit: Notional Coordinates. Specifies the maximum increment between characters in the font.

For FIXED PITCH fonts this value must be the same as the (A width + B width + C width) (escapement) of each character.

LONG lEmInc Unit: Notional Coordinates. Specifies the width of an uppercase M in the font.

For IFI outline fonts, Font Drivers should set this field to -1L, since OS/2 ignores it when computing FONTMETRICS and substitutes ~~sXDeviceRes~~ *lEmSquareSizeX . For bitmap fonts, Font Drivers should set this field to the width of an uppercase M in the font as defined.*

LONG lMaxBaselineExt Unit: Notional Coordinates. Specifies the sum of the maximum ascender and maximum descender values.

FIXED fxCharSlope Specify the angle (in degrees and minutes) between a vertical line and the upright strokes in characters in the font. The first nine bits of this value contain the degrees, the next six bits contain the minutes, and the last bit is reserved. The slope of characters in a normal font is zero; the slope of italic characters is nonzero.

FIXED fxInlineDir Specifies an angle (in degree and minutes, increasing clockwise) from the x-axis that the system uses when it draws a text string. The system draws each consecutive character from the text string in the inline direction. The inline-direction angle for a Swiss font is zero; the inline direction for a Hebrew font is 180.

FIXED fxCharRot Specifies the angle (in degrees and minutes) between baseline of characters in the font and the x-axis. This is the angle assigned by the font designer.

USHORT usWeightClass Specifies the thickness of the strokes that form the characters in the font. This field can be one of the following values:

1	Ultra-light
2	Extra-light
3	Light
4	Semi-light
5	Medium (normal)
6	Semi-bold
7	Bold
8	Extra-bold
9	Ultra-bold

USHORT usWidthClass Specifies the relative-aspect ratio of characters in the font in relation to the normal-aspect ratio for a font of this type. The following are the possible values:

Value	Description	Normal aspect ratio
1	Ultra-condensed	50%
2	Extra-condensed	62.5%
3	Condensed	75%
4	Semi-condensed	87.5%
5	Normal	100%
6	Semi-expanded	112.5%
7	Expanded	125%

8	Extra-expanded	150%
9	Ultra-expanded	200%

LONG lEmSquareSizeX Unit: Notional Coordinates. Specifies the width of cell box. It is also called, the EM square width. *For bit-map fonts this is the resolution in the X direction of the intended target device, measured in Pels per inch.*

LONG lEmSquareSizeY Unit: Notional Coordinates. Specifies the height of cell box. It is also called, the EM square height. *For bit-map fonts this is the resolution in the Y direction of the intended target device, measured in Pels per inch.*
For bit-map fonts, if the lEmSquareSizeX and lEmSquareSizeY fields are ZERO, the graphics engine will substitute the proper values to the lEmSquare-SizeX, lEmSquareSizeY, usNominalPointSize, usMinimumPointSize and usMaximumPointSize fields which are calculated from the display device driver resolution.

GL**YPH giFirstChar** Specifies the glyph index for the first chracter in the font.

GLYPH giLastChar Specifies the glyph index for the last character in the font.

GLYPH giDefaultChar Specifiies the glyph index for the default chracter in the font. The default character is the character the system uses when an application specifies a glyph index that is out of the range of a font's code page.

GLYPH giBreakChar Spcifies the glyph index for the space chracter in the font.

USHORT usNominalPointSize Specifies the height of the font (in decipoints--each decipoint is 1/720th of an inch). The nominal point size is the point size the font was designed to be drawn.
For compatibility with early releases of some applications, Font Drivers should set this field to 120 (12 point).

USHORT usMinimumPointSize Specifies the minimum height of the font (in decipoints). A font should not be reduced to a size smaller than the minimum point size.
Font Drivers should set this field to 10 (1 point).

USHORT usMaximumPointSize Specifies the maximum height of the font (in decipoints). Some applications may use this field to limit the size of characters in this font.

USHORT fsType A collection of flags.

Set IFIMETRICS_FIXED to indicate this is a fixed pitch font.

Set IFIMETRICS_LICENSED to indicate this font is subject of a licensing agreement

Set IFIMETRICS_KERNING to indicate this font has kerning data.

Set IFIMETRICS_DBCS to indicate the font has the DBCS character set.

Set IFIMETRICS_MBCS to indicate that the font has the MBCS character set.

~~Set IFIMETRICS_ATOM_NAMES to indicate that the atom name fields (atFamily-Name and atFacename) of the IFIMETRICS structure are valid.~~

Set IFIMETRICS_FAMILY_TRUNC to indicate that the szFamilyname field is truncated, i.e. that the font family name is longer than 31 characters.

Set IFIMETRICS_FACE_TRUNC to indicate that the szFacename field is truncated, i.e. that the font face name is longer than 31 characters.

Set IFIMETRICS_UNICODE to indicate that the font has the Unicode character set.

Other flag bits are reserved and must be set to zero.

Set IFIMETRICS_NO_CACHE to indicate that the glyph image in this font should not be cached by the graphics engine.

MBCS character set consists of DBCS (Double Byte Character Set) and SBCS (Single Byte Character Set) portion. The definition of the FIXED PITCH MBCS character set in Asian countries is that the fixed width of the SBCS character set is half of the fixed width of the DBCS character set. It is recommended that the font driver return the metrics fields which are related to the character width such as lAveCharWidth, lMaxCharInc and lEmInc based on the calculation from the SBCS character set. This is a recommendation for the font drivers to be developed for Asian countries.

USHORT fsDefn

Set IFIMETRICS_OUTLINE to indicate outline font.

If IFIMETRICS_OUTLINE is not set, the font is a bitmap font.

Other flag bits are reserved and must be set to zero.

Set IFIMETRICS_UDC to indicate user defined font. The user defined font may be updated dynamically.

Set IFIMETRICS_ANTI_ALIAS to indicate anti alias font.

USHORT fsSelection A collection of flags.

Set IFIMETRICS_ITALIC to indicate italic font.

Other flag bits are reserved and must be set to zero.

USHORT fsCapabilities Set to 0.

LONG lSubscriptXSize Unit: Notional Coordinates. Specifies the horizontal size for subscripts in the font.

LONG lSubscriptYSize Unit: Notional Coordinates. Specifies the vertical size for subscripts in the font.

LONG lSubscriptXOffset Unit: Notional Coordinates. Specifies the horizontal offset from the left edge of the character cell.

LONG lSubscriptYOffset Specifies the vertical offset from the character cell baseline. This number is normally positive, indicating the baseline for subscripts is below the baseline for main text.

LONG lSuperscriptXSize Unit: Notional Coordinates. Specifies the horizontal size for superscripts in the font.

LONG lSuperscriptYSize Unit: Notional Coordinates. Specifies the vertical size for superscripts in the font.

LONG lSuperscriptXOffset Unit: Notional Coordinates. Specifies the horizontal offset from the left edge of the character cell.

LONG lSuperscriptYOffset Unit: Notional Coordinates. Specifies the vertical offset from the character cell baseline.

LONG lUnderscoreSize Unit: Notional Coordinates. Specifies the width of the underscore.

LONG lUnderscorePosition Unit: Notional Coordinates. Specifies the distance from the baseline to the underscore line. Note that positive values mean BELOW the baseline.

LONG lStrikeoutSize Unit: Notional Coordinates. Specifies the width of the overstrike.

LONG lStrikeoutPosition Unit: Notional Coordinates. Specifies the position of the overstrike in relation to the baseline.

SHORT cKerningPairs Specifies the number of kerning pairs in the kerning-pair table for the font. Note that OS/2 only returns the kerning pairs for characters in the current codepage when responding to a GpiQueryKernPairs from the application. OS/2 handles the sorting of kerning pairs.

ULONG ulFontClass IBM font classification. This should be set as described in the separate document IBMCLASS.DOC *which is included in the IFI font driver toolkit.*

~~USHORT atFamilyname The atom identifying the font family name in the system atom.~~

~~USHORT atFacename The atom identifying the font face name in the system atom table.~~