```
//-----------------------------------------------------------------------------------
//  Filename: gfxFontManager.h
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
//  API for Reading and Rendering Fonts
//  Version 1.0
//  Copyright 2016, MERL, 201 Broadway, Cambridge, MA, 02139
//  All rights reserved
//  Ronald Perry
//-----------------------------------------------------------------------------------


//-----------------------------------------------------------------------------------
//  To avoid multiple inclusion of header files
//-----------------------------------------------------------------------------------
#ifndef _GFX_FONTMGR_
#define _GFX_FONTMGR_


//-----------------------------------------------------------------------------------
//  Required include files for this header file
//-----------------------------------------------------------------------------------
#include "Nitro.h"


//-----------------------------------------------------------------------------------
//  To make functions accessible from C++ code
//-----------------------------------------------------------------------------------
#ifdef __cplusplus
extern "C" {
#endif


//-----------------------------------------------------------------------------------
//  Create a thread-safe instantiation of a font manager. A font manager instance
//  enables high quality rendering of fonts in a variety of formats such as TrueType
//  and OpenType. This function returns an opaque pointer to a font manager instance
//  upon success; a NULL pointer is returned if the request cannot be satisfied.
//-----------------------------------------------------------------------------------
void *gfxCreateFontMgr (void);


//-----------------------------------------------------------------------------------
//  Destroy the specified font manager instance
//-----------------------------------------------------------------------------------
void gfxDestroyFontMgr (void *fontMgr);


//-----------------------------------------------------------------------------------
//  Load the font specified by filename into the given slot of the specified font
//  manager instance. 16 fonts can be simultaneously loaded into a font manager
//  instance. Font slots range from 0 to 15. It is up to the application to manage
//  these slots by loading and unloading as needed. If you load a font into an already
//  occupied slot, the font in that slot will be automatically unloaded before
//  proceeding with the requested load. This function returns 0 upon success; a
//  non-zero value is returned if the request cannot be satisfied.
//-----------------------------------------------------------------------------------
int gfxLoadFont (void *fontMgr, char *filename, int slot);


//-----------------------------------------------------------------------------------
//  Unload the font in the given slot of the specified font manager instance
//-----------------------------------------------------------------------------------
void gfxUnloadFont (void *fontMgr, int slot);


//-----------------------------------------------------------------------------------
```

```
//  For the loaded font in the specified slot of the given font manager instance, get
//  the following attributes:
//
//  numGlyphs: Total number of glyphs in the loaded font
//  unitsPerEM: Number of font units per EM square for the loaded font
//  isFixedPitch: True if the loaded font is mono-spaced (i.e., fixed width)
//  underlinePosition: Underline position (in font units)
//  underlineThickness: Underline thickness (in font units)
//  ascender: Distance from baseline to highest ascender (in font units)
//  descender: Distance from baseline to lowest descender (in font units)
//  lineGap: Typographic line gap (in font units)
//  caretSlopeRise: The rise value of the cursor's slope (slope = rise / run)
//  caretSlopeRun: The run value of the cursor's slope. For a vertical caret,
//  caretSlopeRise is set to 1 and caretSlopeRun is set to 0.
//  caretOffset: The cursor's offset for slanted fonts (in font units). This value
//  is set to 0 for non-slanted fonts.
//  height: The vertical distance between two consecutive baselines, expressed in
//  font units. It is always positive.
//  bbox[4]: The font bounding box. Coordinates are expressed in font units. The box
//  is large enough to contain any glyph from the font. Thus, "bbox.yMax" can be seen
//  as the "maximum ascender" and "bbox.yMin" as the "minimum descender". xMin, yMin,
//  xMax, and yMax are contained in bbox[0], bbox[1], bbox[2], and bbox[3].
//-------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------
typedef struct {
    int numGlyphs;
    int unitsPerEM;
    int isFixedPitch;
    int underlinePosition;
    int underlineThickness;
    int ascender;
    int descender;
    int lineGap;
    int caretSlopeRise;
    int caretSlopeRun;
    int caretOffset;
    int height;
    int bbox[4];
}   GFXFontData;
//-------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------
void gfxGetFontData (void *fontMgr, int slot, GFXFontData *fontData);


//-------------------------------------------------------------------------------------
//  Set the hint level for the specified font manager instance
//-------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------
#define GFX_HINTING_NONE     0  // No hinting
#define GFX_HINTING_AUTO_FT  1  // FreeType auto-hinting (Latin and CJK)
#define GFX_HINTING_AUTO_MAZ 2  // MAZ auto-hinting (CJK only)
#define GFX_HINTING_FILE     3  // File-based traditional hinting
//-------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------
void gfxSetHintLevel (void *fontMgr, int hintLevel);


//-------------------------------------------------------------------------------------
//  Set the renderer for the specified font manager instance
//-------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------
#define GFX_RENDERER_FREETYPE 0 // FreeType renderer
#define GFX_RENDERER_NITRO    1 // Nitro renderer
//-------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------
void gfxSetRenderer (void *fontMgr, int renderer);
```

```
//-----------------------------------------------------------------------------------
//  Set the rendering mode for the specified font manager instance
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
#define GFX_PIXEL_RENDER_MODE       0   // Traditional pixel rendering
#define GFX_SUBPIXEL_RENDER_MODE    1   // LCD optimized subpixel rendering
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
void gfxSetRenderingMode (void *fontMgr, int renderMode);


//-----------------------------------------------------------------------------------
//  Set the kerning Boolean for the specified font manager instance. If enableKerning
//  is true, kerning will be applied when creating a render string via the function
//  gfxCreateRenderString(). If enableKerning is false, kerning will be disabled when
//  a render string is created. Applications that intend to use render strings as a
//  basis for performing their own typesetting (via single glyph rendering) should
//  disable kerning so that advance width data for individual glyphs can be correctly
//  derived from render string attributes.
//-----------------------------------------------------------------------------------
void gfxSetKerning (void *fontMgr, int enableKerning);


//-----------------------------------------------------------------------------------
//  Create and return a CSM table with the specified cutoff values for the given font
//  manager instance. A NULL is returned if the request cannot be satisfied. The
//  cutoff values are represented as signed 16.16 fixed point numbers. For an in depth
//  treatment of CSM, see the Nitro.h include file.
//-----------------------------------------------------------------------------------
void *gfxCreateCSMTable (void *fontMgr, NTO_I1616 outsideCutoff, NTO_I1616
insideCutoff);


//-----------------------------------------------------------------------------------
//  Destroy the specified CSM table for the given font manager instance
//-----------------------------------------------------------------------------------
void gfxDestroyCSMTable (void *fontMgr, void *csmTable);


//-----------------------------------------------------------------------------------
//  Create a render string using the loaded font in the given slot of the specified
//  font manager instance. s is a GFXString containing the unicode characters to
//  render. pixelSize is the size of each rendered glyph in pixels (not points).
//  rgb[3] specifies a foreground color for each glyph; individual color channels
//  (e.g., rgb[0] is the red channel) must range from 0 to 255. If LCD optimized
//  subpixel rendering is active (see above), the specified foreground color is
//  ignored -- all glyphs are rendered as black. If the current renderer for the
//  specified font manager instance requires a CSM table, cmsTable must be supplied;
//  otherwise, cmsTable is ignored. Once a render string is created, it can be used
//  to render at any (x,y) location on the display. This function returns an opaque
//  pointer to a render string upon success; a NULL pointer is returned if the
//  request cannot be satisfied.
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
typedef struct {        // For representing Unicode strings:
    int numChars;       // Number of character codes in charCodes[]
    int *charCodes;     // Unicode character codes packed in slots 0..numChars-1
}   GFXString;          // GFXString type definition
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
void *gfxCreateRenderString (void *fontMgr, int slot, GFXString *s, int pixelSize,
char rgb[3], void *csmTable);


//-----------------------------------------------------------------------------------
//  Draw the specified render string renderString associated with the given font
//  manager instance beginning at the starting pen position (x,y). Drawing occurs in
//  the current OpenGL draw buffer (e.g., GL_BACK or GL_FRONT) using the current
//  OpenGL transformation state. If useTextureMaps is true, texture maps are created,
//  stored, and used to draw the render string. Texture mapping can provide a
//  significant performance improvement if the render string is intended to be drawn
//  more than once. It also provides a means for performing various effects such as
//  rotation, reflection, and shearing. If singleGlyph is zero, every glyph of the
//  render string is drawn; if singleGlyph is greater than 0, only the glyph at index
//  (singleGlyph - 1) is drawn.
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
//  Applications can use render strings to perform their own typesetting using the
//  following steps:
//
//  1. Turn off kerning via gfxSetKerning().
//  2. Create a render string S via gfxCreateRenderString() at the desired size with
//     all of the necessary character codes.
//  3. Use gfxGetRenderStringAttrs() to derive the advance width for each glyph in S.
//     The advance width for a glyph at index i in S is simply the difference in the
//     position (i.e., (penX, penY)) between the glyph at i+1 and the glyph at i.
//  4. Under control of a typesetting algorithm that uses the advance width data
//     determined in step 3, use the single glyph drawing feature of
//     gfxDrawRenderString() to draw a desired glyph in S at the computed typeset
//     position.
//  5. Space bands are not included in render strings; they simply adjust the position
//     for each glyph immediately following their occurrence. To determine the default
//     advance width for a space band (note: typesetting algorithms typically adjust
//     the advance width for space bands and therefore this may be unnecessary), you
//     can perform the following steps: create a render string for a GFXString
//     consisting of a space band and a single character (e.g., " M"), use the pen
//     position for the first glyph in the render string (which is "M" since space
//     bands are not included in render strings) to derive the default advance width
//     for the space band -- it equals the (penX, penY) position for the first glyph.
//
//  See gfxGetAdvanceWidths() for an alternative to the steps outlined above - it's
//  simpler and more direct, it doesn't require precomputed render strings, it
//  directly handles space bands, and it supports the computation of advance widths
//  with and without kerning.
//-----------------------------------------------------------------------------------
void gfxDrawRenderString (void *fontMgr, void *renderString, int x, int y,
int useTextureMaps, int singleGlyph);


//-----------------------------------------------------------------------------------
//  Return the number of glyphs in the specified render string renderString
//  associated with the given font manager instance.
//-----------------------------------------------------------------------------------
int gfxGetRenderStringLen (void *fontMgr, void *renderString);


//-----------------------------------------------------------------------------------
//  Return the size in bytes of the specified render string renderString associated
//  with the given font manager instance.
//-----------------------------------------------------------------------------------
int gfxGetRenderStringSize (void *fontMgr, void *renderString);


//-----------------------------------------------------------------------------------
//  Get the attributes of the i-th glyph in the specified render string renderString
//  associated with the given font manager instance. Attributes include:
//
//      penX: Base x coordinate for drawing pixmap
//      penY: Base y coordinate for drawing pixmap
//      xOffset: Add to penX to determine drawing (e.g., BLT) position for pixmap
//      yOffset: Add to penY to determine drawing (e.g., BLT) position for pixmap
//      w: Width of pixmap in pixels
//      h: Height of pixmap in pixels
//      texmap: OpenGL texture map ID or 0 if not available
//      pixmap: OpenGL style row major sequential 8-bit RGBA components
//-----------------------------------------------------------------------------
void gfxGetRenderStringAttrs (void *fontMgr, void *renderString, int i, int *penX,
int *penY, int *xOffset, int *yOffset, int *w, int *h, unsigned int *texmap,
char **pixmap);


//-----------------------------------------------------------------------------
//  Destroy the specified render string renderString associated with the given font
//  manager instance
//-----------------------------------------------------------------------------
void gfxDestroyRenderString (void *fontMgr, void *renderString);


//-----------------------------------------------------------------------------
//  Using the loaded font in the given slot of the specified font manager instance,
//  determine the advance width for each character of the specified string s at the
//  given pixelSize. Both the x and y components of each advance width value are
//  represented as signed 16.16 fixed point numbers. Upon return, xAdvance[] and
//  yAdvance[] will contain the results. xAdvance[] and yAdvance[] are pre-allocated
//  arrays provided by the application and must be sufficiently large enough to hold
//  the advance width for each element of s.
//
//  This function can be used by applications to perform their own typesetting on
//  glyphs rendered by this system. Note that kerning is applied to the specified
//  input string if kerning is enabled; disable kerning if this is not the desired
//  behavior (i.e., if you want unmodified advance widths). To determine the kerning
//  deltas for a given input string, simply call this function once with kerning
//  enabled and once with kerning disabled and compute the differences.
//-----------------------------------------------------------------------------
void gfxGetAdvanceWidths (void *fontMgr, int slot, GFXString *s, int pixelSize,
NTO_I1616 xAdvance[], NTO_I1616 yAdvance[]);


//-----------------------------------------------------------------------------
//  Using the loaded font in the given slot of the specified font manager instance,
//  create and return a GFXOutline for the given character code at the specified
//  pixel size. A NULL pointer is returned if the request cannot be satisfied.
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
typedef struct {
    int         sizeInBytes;    // Total size in bytes of this structure and its parts
    int         slot;           // Input slot
    int         charCode;       // Input character code
    int         pixelSize;      // Input pixel size
    int         width;          // Width in pixels for bitmap when rendered
    int         height;         // Height in pixels for bitmap when rendered
    int         xOffset;        // Add to pen x to position bitmap properly on display
    int         yOffset;        // Add to pen y to position bitmap properly on display
    NTO_I1616   xAdvance;       // x component of advance width
    NTO_I1616   yAdvance;       // y component of advance width
    NTOPath     path;           // Output path; pen cmds immediately follow GFXOutline
}   GFXOutline;                 // Outline allocated as a single contiguous block
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
GFXOutline *gfxCreateOutline (void *fontMgr, int slot, int charCode, int pixelSize);


//-----------------------------------------------------------------------------
//  Using the same conventions as gfxDrawRenderString(), draw the specified outline
//  associated with the given font manager instance at the pen position (x,y). rgb[]
//  and csmTable define the color and CSM table to use during drawing as described in
//  gfxCreateRenderString().
//-----------------------------------------------------------------------------
void gfxDrawOutline (void *fontMgr, GFXOutline *outline, int x, int y, char rgb[3],
void *csmTable);
```

```
//-----------------------------------------------------------------------------------
//  Identical to gfxDrawOutline() with one exception - the rendered outline is not
//  drawn to the display and instead the resulting 8-bit per channel RGBA rendered
//  image is returned to the caller. A NULL pointer is returned if the request cannot
//  be satisfied. To release the memory used to represent the returned image, use
//  the C standard library function free().
//-----------------------------------------------------------------------------------
char *gfxGetOutlineBitmap (void *fontMgr, GFXOutline *outline, char rgb[3],
void *csmTable);


//-----------------------------------------------------------------------------------
//  Destroy the specified outline associated with the given font manager instance
//-----------------------------------------------------------------------------------
void gfxDestroyOutline (void *fontMgr, GFXOutline *outline);


//-----------------------------------------------------------------------------------
//  Return the Nitro renderer instance associated with the specified font manager
//  instance
//-----------------------------------------------------------------------------------
void *gfxGetNitroInstance (void *fontMgr);


//-----------------------------------------------------------------------------------
//  End of C++ wrapper
//-----------------------------------------------------------------------------------
#ifdef __cplusplus
}
#endif


//-------------------------------------------------------------------------------
//  End of _GFX_FONTMGR_
//-------------------------------------------------------------------------------
#endif
```