Standard Type Services Framework – Unicode-based Framework for Rendering Typographically Sophisticated Text

Alexander Gelfenbain
Sun Microsystems, Inc.
adg@sun.com

Abstract. The Standard Type Services Framework (STSF) is a new architecture for rendering typographically sophisticated text and managing fonts Designed primarily for UNIX® and UNIX®-like systems it is highly portable. It provides a rich set of APIs available directly to software developers or through the X11 XST extension.

STSF is a Sun Mictrosystems, Inc. sponsored open source project hosted at http://stsf.sourceforge.net

1 Introduction

Text output is one of the most fundamental tasks of any graphical user interface system. Implementing a fast and efficient text rendering system that can withstand the challenge of usage in multi-language environments and also provide satisfactory APIs to the wide range of developers is an incredibly difficult task. While some developers may write simple "Hello, world" graphical user interface (GUI) programs, others need to implement word processors and desktop publishing packages. The X Window System[1] did not provide a satisfactory text imaging API for this spectrum of requirements.

Early computers did not have graphical user interfaces and they did not require sophisticated text output. As processing power increased and high quality printing became available to consumers, they started demanding WYSIWYG applications.

In the early 1970s Dr. Peter Karow wrote his IKARUS software for creating digital typefaces. This was the beginning of the digital type revolution – the most fundamental change in typography since 1450 when Johannes Gutenberg started using movable type. Digital type made desktop printing and publishing possible. It also made possible mass production of high quality digital type for personal computers and printers.

More and more personal computer applications developers wanted to take advantage of emerging typographic technologies for sophisticated text rendering. As a result, the first desktop platform to provide an extensive text rendering API – the Apple® Macintosh® – instantly became an unchallenged leader in user-friendly internationalized software and in the professional desktop publishing market.

With STSF we are planning to provide a set of APIs for Solaris™ Operating Environment, UNIX®, UNIX®-like systems (Linux,) and the X Window System that match or exceed those available for Microsoft® Windows® or Apple® Mac® OS.

2 Functional Requirements for Modern Text Output Systems

A text output system is typically a component or a layer of some graphical user interface environment. GUI environments become more sophisticated as the processing power of computers increases and the requirements they impose on the functionality of their text rendering system become more extensive. A text output system itself consists of several different components – font management, text processing, and rendering.

The font management subsystem is responsible for making fonts available to users and to applications. It is in charge of installing, removing, and enumerating fonts.

Text processing is the process that turns attributed logical text (a combination of text with some formatting properties, like the font that a text segment should be rendered with) from a sequence of characters to a set of positioned glyphs. A character is a logical unit of a written language. A glyph is a visual shape that represents characters graphically on an output device. Text processing may involve several steps – pre-processing, mapping characters to glyphs and post-processing. A single character may be replaced by several glyphs, and several characters may be replaced by a single glyph. The order of glyphs in the target glyph array may be different from the character order in the source buffer.

The Unicode bi-directional algorithm is an example of pre-processing. Mapping characters to glyphs is performed using a set of font-specific rules. Kerning and ligature substitution are examples of post-processing.

Rendering is the process of mapping outline shapes of scalable fonts or pixels of bitmap fonts to a digital output device.

GUI environments of two major desktop platforms[2] [3] and Sun Java™ 2 Platform, Standard Edition[4] include sophisticated text rendering frameworks. The STSF team analyzed features and architecture of these systems and came up with the list of features that it wanted STSF to support:

- Support for sophisticated rendering technologies
  Anti-aliasing and sub-pixel rendering

- Support for modern font technologies
  TrueType and OpenType

- Support for rich graphics environment
  Fractional metrics, affine transformations, alpha-blending

- Support for internationalization
  Unicode support, support for complex scripts

- User-friendly font managing and naming
  Font grouping, user-friendly font names, ease of font installation and removal

- Programmer-friendly API
  The API should be easy to learn and easy to use, yet powerful and rich

3 STSF Concepts

The STSF font management component is based on two complementary objects – fonts and scalers. Fonts contain scalable glyphs, scalers can convert glyphs to arrays of pixels on output devices. Both objects – fonts and scalers can be loaded and unloaded dynamically, without forcing STSF applications to restart.

The STSF text processing is based on the concept of styled text and "smart" font processing. Displayable text is attributed with styles that specify, among other things, a font. Smart font technology puts some part of text processing intelligence into fonts. Two competing smart font technologies – OpenType and TrueType GX differ in the amount of text processing functionality they delegate to fonts. TrueType GX fonts include all rules for language-specific text pre- and post-processing in the fonts. OpenType requires language-specific text pre-processing to be implemented outside of the fonts. Font-specific post-processing is defined in OpenType tables. STSF supports both models of smart fonts.

While STSF takes full advantage of smart fonts, it does not require them. Without smart fonts it defaults to built-in rules for text processing.

The STSF rendering component supports two types of rendering – rasterization and vector rendering. Rasterization can be optimized for a variety of output devices with unique pixel properties like LCD screens or analog TV screens.

4 STSF Programming Model

STSF offers a rich object oriented ANSI C language API to application developers. There are two different implementations of the API – the ST API available for applications linked with the STSF Client Library directly, and the XST API available for applications accessing STSF through the XST X11 extension. The XST API includes all functionality of the ST API with the addition of some convenience functions for X11 clients related to drawing and translating data from one encoding to another.

The STSF API defines a set of objects implemented as pointers to opaque structs in ST or XIDs in XST and methods that operate on these objects. These objects include:

- *STFont (XSTFont)*: a font object.
- *STFontFamily (XSTFontFamily)*: a font family object.
- *STScaler (XSTScaler)*: a font engine object.
- *STTypeEnv (XSTTypeEnv)*: a session object.
- *STText (XSTText)*: a logical text object.
- *STLine (XSTLine)*: a visual line of text object.
- *STStyle (XSTStyle)*: a style object.

- *STDevice*: an ST-only output device object.
  Encapsulated in X11 Graphics Context by XST.
- *STGraphics:* an ST-only output graphics context object.
  Encapsulated in X11 X Graphics Context by XST.

4.1 STSF Font Objects

Font objects represent STSF Font Server logical fonts. All fonts are presented by STSF to applications as abstract objects that have one or more names tagged with encoding and meaning. A font can contain its font family name, a typeface name, and even a name and a URL of its designer or the type foundry. All these names can be translated in a number of different languages by a font vendor and included in the font. STSF does not impose a limitation on the number of names a font can have, and it provides a way to access these names in Unicode encoding regardless of the way they are stored in a font.

STSF does not create logical fonts for bitmap font files. Bitmap font files are grouped together with an appropriate scalable font file and are regarded as instances of that scalable font rendered at specific point size and resolution. If no appropriate scalable font file can be found, STSF creates a virtual font that can be scaled only to those sizes that those bitmap font files represent.

STSF supports two "smart font" technologies – TrueType GX and OpenType. It enumerates all GX and OpenType features available in a font so that an application can query if a particular font supports a specific feature. TrueType GX fonts are mostly available for the Macintosh platform. OpenType fonts are supported by both Apple and Microsoft, and more fonts are commercially available.

Additional font properties that applications can query include metrics information, both font-wide and for individual glyphs.

STSF supports persistent fonts that are available on the host system and client (session) fonts that an STSF application can instantiate for the duration of its execution.

A user application can search for fonts by any of their names, returning sets of matching fonts, or it can enumerate all available fonts.

4.2 STSF Font Family Objects

Font families are logical groups of fonts with similar design. For example, "Regular," "Italic," "Bold," and "Bold Italic" are typefaces in the "Times New Roman®" font family. STSF combines fonts into font families and instantiates font family objects for simplified access to individual fonts.

An application that needs to present a font selection menu to a user enumerates all available font families and creates menu items for each of them. Sub-menus may be created for typeface names.

An application can also search for a font family by its name. It can map any given font to the font family object that contains the font and find out what other fonts are available in that font family. Sometimes when no typeface for a specific font style – for example, Italic – exists in a font family, algorithmic styling can be used to imitate it. However, the results are typically not as good as a typeface designed by a professional type designer.

4.3 STSF Font Engine Objects

The STSF Font Server can parse the font files and extract metric and other information from them, but it does not scale the fonts. Drop-in font engines, or "scalers" in STSF terminology, are responsible for that.

This design allows greater flexibility of STSF deployment by allowing system vendors to license font rendering engines independently of STSF. For open source platforms like Linux/XFree86, a FreeType-based font engine is provided. Commercial UNIX® vendors have an option of licensing proprietary font engines and plugging them in without a need to recompile either their X Servers or STSF itself.

A font engine is identified by its tag and version number. Sun Microsystems, Inc. maintains the registry of tags that it assigns to rendering engine vendors.

Applications can enumerate available font engines or search for a specific engine. They can query a list of features that a font engine supports and request text be rendered by a specific one. A list of features of font engines includes:

- Native font hints
- Autohinting
- Anti-aliased rendering
- Subpixel rendering (LCD optimization)
- Fractional metrics

4.4 STSF Session Objects

The STSF session object keeps track of all objects that the STSF Font Server manages – fonts, font families and font engines. When this object gets created, the STSF Font Server allocates some resources for the new client and keeps track of them. A good example is application "private" fonts that exist only during the lifetime of the STSF session object.

The session object also keeps track of a global font fallback policy. If some characters are missing in a font that an application program wants its text to be rendered by, a font fallback mechanism is used to find another font that contain this characters. Since potentially a system can contain many thousands of different fonts, searching all of them to find an appropriate font can take quite a long time. A font fallback policy controls this process. It can be set to disable font fallbacks entirely or to limit a set of fonts scanned for an appropriate substitution font.

4.5 STSF Text Objects

The styled text model that STSF uses for text output requires separation of two objects – logical text and its formatting attributes. Logical text is represented by STSF text objects.

A text object stores a reference to its text and provides a convenient placeholder for metrics information that can be used when text is displayed. Global font fallbacks specified in the session object can be overridden for a specific text. Typically text objects represent a paragraph of displayable text, which can be a couple of sentences. Sometimes a separate text object is created for a single word, or even a single character. Creating a single text object for the entire multi-page text document is also possible.

Text objects provide context for language-specific processing like a bi-directional algorithm. The native encoding of STSF is Unicode represented as UTF-16BE. The XST component of STSF includes a set of convenience functions that convert between X11 client encodings and Unicode.

4.6 STSF Style Objects

Style objects represent formatting attributes applied to logically contiguous segments of text objects. These attributes include:

- Font
- Font size
- Baselines
- Font features
- Font engine
- Effects like underline or strikethrough applied to the style

4.7 STSF Line Objects

An STSF line object is an atomic unit of displayable and measurable text. All lines are derived from text objects and are internally represented as arrays of positioned glyphs that map somehow to characters of their source text. STSF is very flexible in allowing its client to use any line breaking mechanism. It does not break text into lines itself; it merely provides several methods to calculate line metrics. The clients then can decide how to break text into lines according to their metrics or linguistic rules.

Lines have two sets of metrics – design metrics and imposed metrics. When STSF performs a line layout process, it determines the optimal metrics for the line – unconstrained or “design” line metrics. An application writer can then impose different metrics on the line by telling STSF how wide it wants this line to appear on the screen; STSF will repeat the line layout process adjusting glyph positions according to constrains imposed on it.

Lines manage their selected regions and can render them highlighted with different

colors. They also manage carets and provide convenience functions that map logical caret positions to output device coordinates. For bi-directional languages, STSF supports split carets – a set of two carets displayed at text direction boundaries. A caret can be moved both directions. Both logical (move the caret to the next or the previous character) and visual (move the caret one position left or right) caret movements are supported.

Lines also perform hit-testing – converting output device coordinates to the logical character offset in the text object buffer. Hit-testing is performed when an application needs to map a mouse click event to a logical character position to initiate selection or to set the current caret position.

4.8 ST Device Objects

Device objects are an abstraction for output devices and they interface ST with the graphics environment that ST is part of. XST creates an ST Device Object internally and never exposes it to the clients. A different graphics environment will have to instantiate a device object to allow ST render to it.

There are two types of device objects defined by STSF – raster devices and vector devices. As the name implies, raster devices are discrete pixel-oriented devices with some finite pixel density. The process of rendering on raster devices – rasterization – involves conversion from scalable outlines to areas of pixels rendered with different colors. Vector devices represent the analogue class of output devices with no discrete pixels.

STSF does not provide any constructors or destructors for device objects. It only specifies a set of methods that a device needs to implement. For example, raster devices define a method that copies an array of pixels from the internal STSF buffer to the device. One of the steps that happens during STSF integration with the rendering system is implementation of the appropriate STSF device object. XST implements one inside the X11 server.

All devices include an affine transformation matrix that is applied during the drawing process and define a set of convenience functions to manipulate the matrix.

4.9 STSF Graphics Objects and the Color Model

Graphics objects encapsulate ST rendering contexts. XST extends the X Graphics Context structures to store ST graphics objects internally.

Graphics objects control the process of rendering by specifying the output mode, a set of output colors, and the output device.

STSF defines five colors – normal (unselected) text color, foreground and background colors for highlighted text, and colors for the underline and the strikethrough effects.

STSF uses RGBA (24 bits for RGB and 8 bits for an alpha channel) colors. The process of alpha-compositing is internal to STSF Device Objects.

4.10 STSF Coordinate Model

STSF uses three different coordinate systems.

- User space - an abstract coordinate space that most STSF functions use. The unit of user space is one typographic point defined to be 1/72 of an inch.

- Device space – a device coordinate system of a specific raster or vector output device that ST renders text onto. Units are device-specific.

- Font design space an abstract coordinate system that represents metric information of unscaled fonts in abstract font units.

STSF represents units of all coordinate systems as floating-point numbers.

Translation from user space to device space is controlled by a 3x3 transformation matrix that defines an affine transformation. Therefore conversion from user space to device space might include any affine transformation such as scaling, rotation, translation, shearing, or any combination thereof.

In the simplest case translation from user to device space might be described by the identity matrix, but it is not always the case and ST applications should make no assumption about it. The device object constructor sets the appropriate initial transformation matrix for the device that can be queried by an application. If an ST application wants to apply an additional affine transformation, it should save the initial matrix and concatenate the matrix of the affine transformation with the initial device matrix.

Font design space units are abstract units used for measuring unscaled glyphs that can be converted to units of either design or user space by applying simple formulae.

4.11 Character encodings

The native encoding that STSF uses is UTF-16BE and all ST functions expect their text arguments to be UTF-16BE encoded. XST allows X11 applications to use any encoding that the host operating environment supports. It converts text to UTF-16BE before sending it to the X11 server via the XST X11 extension protocol.

5 STSF Implementation Architecture

Several independent components, APIs and SPIs (service provider interfaces) constitute the Standard Type Services Framework.

STSF Font Server is a UNIX daemon process that defines objects for fonts and scalers that it loads from the file system. These objects are then made available to STSF applications.

STSF Font Server has unlimited and exclusive access to binary font file data. It recognizes fonts and extract a lot of information from them including font names, information about characters, glyphs, and their design metrics. Fonts can be dynamically added to and removed from the system – STSF Font Server detects both events and makes appropriate changes to its internal state.

Among other font formats, STSF Font Server supports TrueType font files and TrueType Collections, OpenType in both TTF and OTF formats, Type 1 in both ASCII and binary formats and several bitmap font file formats.

To scale the glyphs and to make them available for applications, STSF Font Server loads drop-in **STSF Font Scalers.**

STSF Font Scalers are shared libraries built on top font rasterization engines like FreeType that can be dynamically loaded and unloaded. The SPI between them and the STSF Font Server is straightforward in order to facilitate rapid creation of font scalers by third parties. Sun developed several scalers based on open source FreeType versions 1 and 2 libraries for the open source version of STSF and several scalers based on commercial font engines for Solaris OS.

STSF Client Library is a shared library that STSF applications link with to access STSF functionality. It defines a set of user objects; some of them are mapped to STSF Font Server objects, while other are implemented inside STSF Client Library.
X11 Server is linked with STSF. STSF Client Library defines the STSF programming model. Several STSF Client Libraries can connect to a single STSF Font Server simultaneously, one for each STSF client. All elements of STSF programming environment are managed by STSF Client Library.

STSF X11 Protocol Extension – XST exports STSF functionality to X11 clients via an X11 extension. In the XST environment, the STSF Client Library is linked with the X11 Server. For each library object, the X11 server allocates a handle and makes it available to X11 clients.

The **XST Library** is a client-side X Window System library that maps between the XST protocol and user callable functions.

6 References

1 Robert W. Scheifler and James Gettys. X Window System. Digital Press, third edition 1992

2 Microsoft typography ttp://www.microsoft.com/typography/default.asp

3 Apple Type Services for Unicode Imaging
http://developer.apple.com/techpubs/macosx/Carbon/text/ATSUI/atsui.html

[4] Java Developer Connection
http://developer.java.sun.com/developer/infodocs/#docs