XML Professional Publisher:

Fonts

for use with XPP 9.7
March 2024

Contents

Chapter 13 The Typesetter Font Map Spec (TSF)

Chapter 14 Encoding Tables

Chapter 15 The Kerning Pairs Spec (KP)

Figures

Tables

About This Manual

This manual is divided into two parts:

- **Part I: Installing Fonts—**How to install new fonts for use with XPP and use the font utilities.
- **Part II: Managing Fonts**
  —An introduction to fonts, characters and keyboards, the Xyvision Standard Format and Xyvision Character Set, display fonts and output device fonts
  —How to manage font libraries, set up font specs for composition and output, generate Font Access Tables (FASTs) and access the FASTs.
  —Setting up specs to accommodate details in font appearance such as kerning, ligatures, and accent placement.

Fonts does not address:

- Loading fonts on your output devices. Refer to the documentation provided by the output device manufacturer for information on loading fonts onto your particular output device.

Where Do I Start?

This manual is for those who have experience with fonts. RWS recommends that you use this manual to accompany XPP font training.

If you need to...	*Then read...*
Install a new font (including CID font) or use the font utilities: Font Copy and Build Font FAST.	Chapter 1
Learn about Xyvision Standard Format and output device fonts. Get an overview of the steps involved in setting up fonts for the Xyvision application.	Chapter 2
Learn about the Xyvision Character Set.	Chapter 3
Learn about keyboard maps.	Chapter 4
Create or manage font libraries and font specs. Learn general information about editing font specs.	Chapter 5
Edit the font width specs: PTS, PSF, PSN, FGS, FGX.	Chapters 6-10
Run GenFAST or the Font Width utility. Learn about Font Access Table (FAST) specs.	Chapter 11
Learn about the Font Variant Spec.	Chapter 12
Learn about the Typesetter Font Map Spec.	Chapter 13
Learn about encoding tables.	Chapter 14
Learn about the Kerning Pairs Spec.	Chapter 15
Learn about accent and ligature control.	Chapter 16
Learn how to adjust the placement of accents over PostScript small caps fonts.	Chapter 17

Conventions Used in This Manual

This manual uses a set of symbolic, typographic, and terminology conventions to categorize specific information. Take a few moments to become familiar with these conventions before you use this manual.

Convention	*Description*
Bold	Bold type, used in procedures, indicates the object of the action. It may be a menu option, a push button, or a field, and so forth. For example, "select **Open**" means select the menu option called Open. Position **cursor** means to move the cursor to a specific location. Enter appropriate **information** means that you enter information that is appropriate for your site or for the specific job. Bold may be used elsewhere in the manual to denote emphasis.
Key	Capital first letter and the word "key" indicates a key on the keyboard. Capital first letter and the words "Softkey menu" indicate the menu pad to the right of the XyView. Unless otherwise indicated, this manual assumes that you press the Enter key at the end of a command line.
Key+Key	This sequence indicates a Shortcut Key combination. Hold down the first key while also pressing the second key, that is **ALT+F4** means to hold down the *Alt* key while also pressing the *F4* key.
`Message`	A monospaced typeface indicates an application's response to your action. For example, "the message `Enter Value` appears" means that the application displays the words "Enter Value."
Italics	In running text, an italic typeface indicates a new term; for example, "The replacement string of characters is the *output string*." In a system message, a field entry, or an argument to a command, an italic typeface indicates a variable. For example, *filename* is a variable in the message "Can't open *filename*." Italics are also used for the names of programs, such as *Perl*.
" "	Quotation marks indicate that you should enter exactly what the instructions tell you to enter. For example, type "**yes**" means to type the letter **y**, the letter **e**, and the letter **s**.
[ ]	Reverse-video square brackets represent tags in standard XPP. Tags are general-purpose commands defined in the Item Format Spec and embedded in a document. They generally format logical components of text, such as chapter openings, headers, or lists. For example, the tag for beginning a chapter might be **[chap]**.

Convention	*Description*
⟨ ⟩	Reverse-video angle brackets represent XPP-supplied macros (XyMacros) and user-defined macros. XyMacros are commands embedded in text to set or change formatting or typographic style. For example, the XyMacro to end a page is **⟨ep⟩**. Reverse-video angle brackets also represent tags when you use XPP in either XML or SGML mode. Note that when in XML or SGML mode, XPP does not use the conventional reverse-video square brackets.
⟨?xxx⟩	Reverse-video angle brackets with a beginning question mark represent macros when using XPP in SGML mode.
⟨?xxx?⟩	Reverse-video angle brackets with a beginning and ending question mark represent macros when using XPP in XML mode.

When entering values for some arguments in macros and for some fields in specs, you must *qualify* the value by specifying a *unit* of measure. The available unit qualifiers are:

- q — Points
- p — Picas
- c — Ciceros
- d — Didots
- i — Inches
- m — Millimeters
- k — Kyus
- n — Microns (XPP units)
- z —Centimeters

For example, 6q means 6 points, 4p means 4 picas.

Notes:

- *You can also use pc, pt, in, mm, and cm in fields where the system allows the standard ISO unit abbreviations.*
- *For fields in specs, no more than two decimal places can be entered for Points, Inches, Didots, Centimeters, Millimeters, or Kyus units. If a more precise value is needed, use the equivalent value in Microns (XPP units).*

For More Information

This manual is part of a comprehensive document set. Other documents describe topics such as:

- Basic skills needed to be productive on the XML Professional Publisher (XPP).
- Tables, XyMacros, and system administration.
- Optional applications, such as Classification Marking, Electronic Notes, Loose-leaf, CITI, and Math.

Refer to *XML Professional Publisher: XPP Document List* for a complete list of documents and their descriptions.

Part I

Installing Fonts

Chapter 1

Installing Fonts

You can install PostScript Type 1, CID, or OpenType fonts on the XPP server. This chapter contains the following installation information:

- Understanding the installation process
- Using Font Copy
- Using Build FAST
- Troubleshooting fonts

The remaining chapters in this manual describe the various font specs and the process for managing your fonts.

Understanding the Installation Process

XPP delivers software with the 163 Noto OpenType fonts and two XPP *Pi* fonts already set up for use on your system. If you need additional fonts, you will have to obtain, install and prepare them for use by your XPP software.

Installing new fonts involves three steps:

1. Obtain the fonts from a font vendor.
2. Put the fonts in the correct directories on your XPP server using Font Copy, an XPP utility.
3. Create the corresponding font width and setup specs in the XPP application using Build FAST, an XPP utility.

Note: *If you want to install a Pi font, RWS strongly advises that you have a good understanding of the XPP font environment. If you do not have this foundation, RWS recommends that you read the remaining chapters of this manual.*

Obtain the Proper Fonts and Font Files

XPP supports fonts and font files in these formats:

- Type 1 base font
- AFM file
- Type 1 CID font
- CMap file
- OpenType/PostScript base font
- Open Type/PostScript CID font
- OpenType/TrueType base font

Note: *Fonts that fail with checksum errors, ttftotype42 conversion errors, or ttftotype42: ″no such table″ errors are rejected as not valid for use with XPP.For more information about TrueType font conversions, see ″TrueType Fonts″ in the XML Professional Publisher: Managing XPP manual.*

Type 1 Base Fonts

Type 1 base fonts are traditional PostScript fonts that allow access to no more than 256 characters at a time.

Type 1 base fonts are available in one of these two file types that contain the character definitions:

- PFA—Printer Font ASCII (*.pfa*)
- PFB—Printer Font Binary (*.pfb*)

XPP uses the PFA format. Since most Type 1 base fonts are sold in PFB format, the Font Copy utility converts PFB files to PFA as part of the copy activity.

Type 1 base font packages also include AFM (Adobe Font Metrics) files, which contain width, kerning and other information that the Build FAST utility uses to create the XPP specs.

XPP requires both the font file (*.pfa* or *.pfb*) and the Metrics file (*.afm*) for Type 1 base fonts.

XPP accesses characters in a Type 1 base font by use of an encoding table. Each font has a default encoding. To access additional characters, you must use an encoding file. Refer to “Chapter 14 Encoding Tables” for additional information.

Type 1 CID Fonts

A Type 1 CID font is a Type 1 font in which the characters are accessed in the PostScript code with CIDs (character identifiers), rather than by character names as in a Type 1 base font.

Type 1 CID fonts are available in a binary file format that contains the character definitions. This is not the same format as a PFB file.

Type 1 CID font packages also include AFM files. The Build FAST utility uses these files to create the XPP specs.

XPP requires both the font file (which typically has no file extension, but may have a .cid or other file extension) and the Metrics file (*.afm*) for Type 1 CID fonts.

XPP accesses characters in a Type 1 CID font by using an (external) CMap file. Since most Type 1 CID fonts are based on standard character collections, common CMap files may be used with many different Type 1 CID fonts. Thus, CMap files for Type 1 CID fonts are stored in a common CMap directory, as explained on page 1-6. These CMap files typically have no file extension, but may have a .cmap or other file extension.

OpenType Base and CID Fonts

An OpenType font is a font recorded in a platform-independent format. It can contain definitions for many thousands of characters within a single font. It is much different internally from a Type 1 base or Type 1 CID font. An OpenType font can contain either PostScript (*.otf*) or TrueType (*.ttf*) character definitions, both of which are supported by XPP. (The PostScript form is called Compact Font Format, or CFF.)

An OpenType/PostScript CID font is a font in which the characters are accessed in the PostScript code with CIDs (character identifiers), rather than by character names as in an OpenType/PostScript base font.

To install an OpenType base or CID font, you need only the font file (*.otf* or *.ttf*). Font Copy extracts mapping data and metrics data from the font to create a Unicode CMap file and a Metrics file for the font and stores the CMap file (with the extension *.cmap*) and the AFM file (with the extension *.afm*) in the same directory as the font itself. Build FAST temporarily extracts metrics information while creating XPP specs if an OpenType/PostScript font file (*.otf*) is selected instead of a Metrics file (*.afm*).

For an OpenType/PostScript CID font (when using *psfmtdrv*, but not when using *divpdf*), Build FAST can also specify to use an external CMap file that has been imported into the `XYV_EXECS/psres/fonts/CMap` directory rather than the default extracted CMap file created in the same directory as the font itself.

XPP accesses characters in an OpenType font by using a CMap file. This provides direct access to most or all of the characters in the font. For information about OpenType Fonts with unmapped glyphs, see Appendix B: Status, Tips, and Troubleshooting.

Putting Fonts in the Correct Directories

XPP installs the fonts it copies into a **XYV_EXECS/psres/fonts/*fontname*.font** directory to ensure there is no conflict between fonts that XPP uses and fonts you may have already installed on your system for use with other applications. XPP provides the Font Copy utility to place the font files you want to install in the appropriate directories.

Font Directories

The following table identifies the location and describes what font information is in that location.

Table 1-1 *XPP Font Locations*

Location	***Description***
XYV_EXECS/psres	This directory contains the *PSres.upr* file, which is an index to the PostScript fonts that XPP installs. Font Copy asks to update the *PSres.upr* file each time you install a new font. XPP uses the information in the *PSres.upr* file to locate the fonts.
XYV_EXECS/psres/encodings	The XYV_EXECS/procs/sc/pdfsetup.pl Perl script uses the XYV_EXECS/bin/pdfencode.pl Perl script to convert Type 1 base font PostScript-type encoding files used by *psfmtdrv* in the XYV_EXECS/sys/od/ps_dlf/encodings subdirectory into this subdirectory for Type 1 base font PDF-type encoding files needed by *didvpdf*.
XYV_EXECS/psres/fonts	This is the directory path for user-installed fonts and AFM and CMap files.
XYV_EXECS/psres/fonts/ *fontname*.font/	Font Copy creates this subdirectory and names it for a specific font according to the actual name of the font. For example, for a font called NotoSerif-Regular, the directory name is: NotoSerif-Regular.font.

Table 1-1 *XPP Font Locations (Continued)*

Location	***Description***
XYV_EXECS/psres/fonts/ *fontname*.font/*fontname.xxx*	This is the name of the actual font file. ● For Type 1 base font files, *.xxx* is *.pfa* and is the PostScript code for the font. ● Type 1 CID fonts are not *.pfa* files and do not typically have a file extension, but may have a .cid or other file extension. ● For OpenType font files, *.xxx* is *.otf* or *.ttf*. ● OpenType fonts also have a CMap file generated by the Font Copy utility. For these, *.xxx* is *.cmap*. Some OpenType/TrueType fonts (*.ttf*) can also have a Type 42 file generated by the Font Copy utility, where *.xxx* is *.t42*. ● For AFM files, *.xxx* is *.afm*. This file may be optional. It is only necessary when building font width specs; if font width specs are already available and you are not using the Direct to PDF (*divpdf*) program, you do not need this file. Otherwise, this file is required for Type 1 base and CID fonts. For OpenType fonts, Build FAST extracts this information from the *.otf* or *.ttf* file. ● With all fonts, the name of the generated font directory *fontname*.font and copied or generated font file *fontname*.xxx is the same as the internal font name, even if the name of the original font file is different.
XYV_EXECS/psres/fonts/ CMap	Font Copy creates this subdirectory for (external) CMap files that are used with Type 1 CID fonts, and also those that can be used with OpenType/PostScript CID fonts (but only when using *psfmtdrv* and not when using *divpdf*).
XYV_EXECS/psres/fonts/ Noto_Fonts_License.txt	Legally required license/usage file included in distribution of Noto fonts.

Use Build FAST, explained on page 1-10, to create or update the font specs for newly acquired fonts.

Using Font Copy

Font Copy, an XPP utility, copies source fonts into the XPP directory structure in the following steps:

1. Allows the user to select the font files to copy.
2. Performs the copy and generates necessary font files.
3. Creates or updates the *PSres.upr* file, the file that lists all the font files and their locations.

Using Font Copy makes the font available for use with XPP.

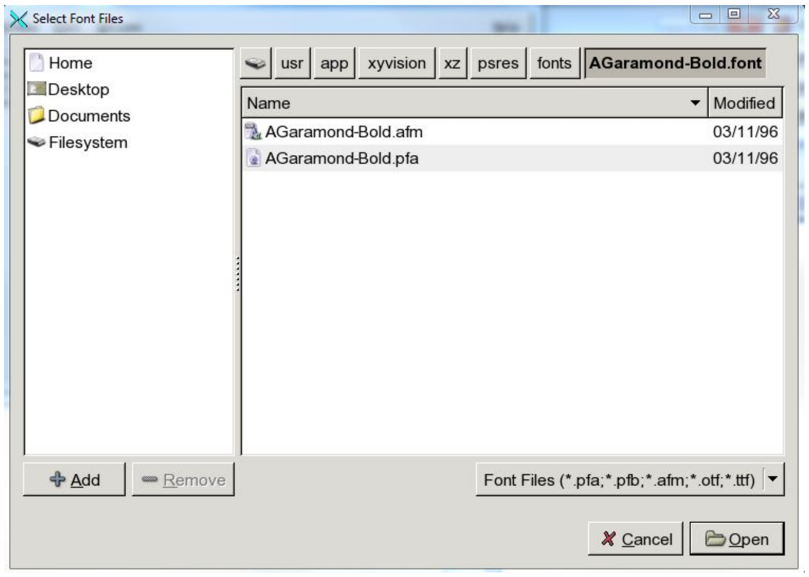
Running Font Copy

To run *Font Copy* from the PathFinder toolbar:

1. Select **Tools > Font Copy**
   PathFinder displays the Select Font Files list box.

   For Type 1 CID Fonts and external CMap files, set *Font Files* to *All Files* since these files may not have a file extension.

2. Select the **directory** containing the files you want to copy and click the **Open** button.
   XPP displays the following font files for a PostScript Type 1 base font.



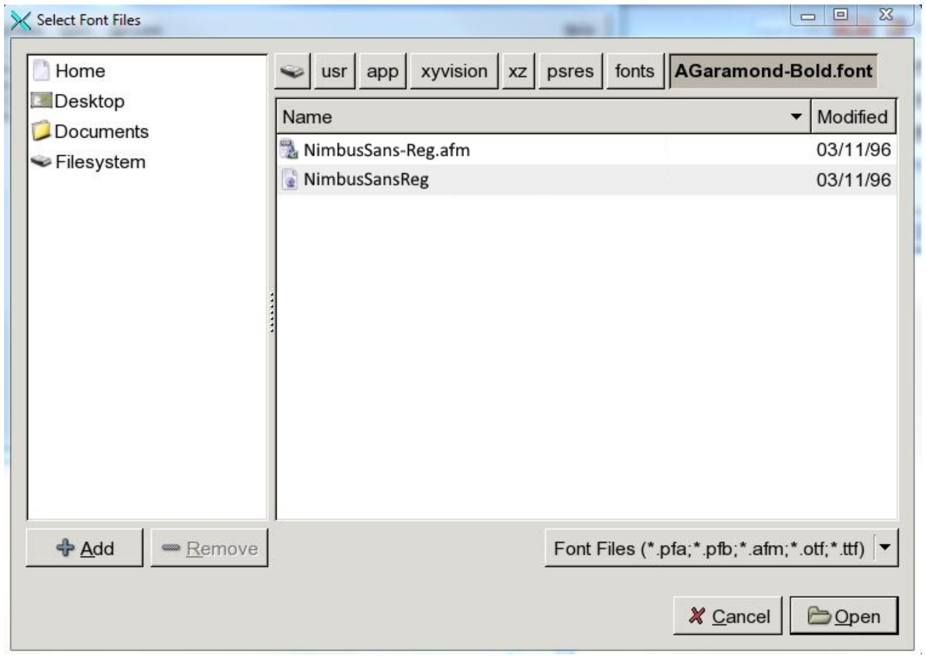
Select Font Files

usr / app / xyvision / xz / psres / fonts / AGaramond-Bold.font

Name	Modified
AGaramond-Bold.afm	03/11/96
AGaramond-Bold.pfa	03/11/96

Font Files (*.pfa;*.pfb;*.afm;*.otf;*.ttf)

   In this example, AGaramond-Bold is the font specified.
   There may be two or more files for each PostScript Type 1 base font, but XPP uses the following:
   - *font*.afm
   - *font*.pfa **or** *font*.pfb



In this example, NimbusSans is a Type 1 CID font. Type 1 CID fonts also have two or three files:

- *filename*.afm
- *filename*
- *cmapfilename* (optional external CMap)

Note that there is no *.pfb* file for a Type 1 CID font and *Font Files* is set to ‘All Files’. For an OpenType font, there is only one file: *fontname*.otf or *fontname*.ttf.

3. Select the **font files** for the fonts you want to copy and click the **Open** button to start the copy.
   XPP closes the window. Font Copy displays a message box, stating “Please wait...”
   Font Copy copies or generates the font files into the XYV_EXECS/psres directory tree. If the font is OpenType, it also generates a CMap file and an AFM file in the same directory as the font file.

4. When the copy is complete, Font Copy asks if you want to update the font search path.
   Click **Yes**.
   When the update is complete, Font Copy displays a pop-up window that says, “Font Copy completed.”

5. Click the **OK** button in the pop-up window.
   Font Copy closes the window.

You may need to add the font to the appropriate download table. If you are using enable font download (**–efd**), add the font to the appropriate font download table (XYV_EXECS/xz/sys/od/ps_dlf/tables). Refer to ″Font Download Tables for PostScript Fonts″ on page 2-10 for complete information.

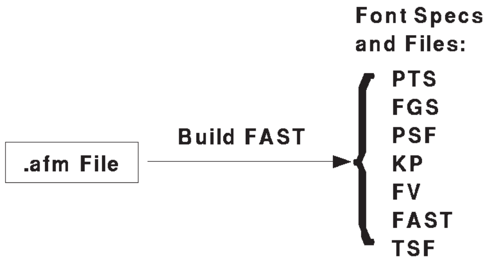
Note: *When a TrueType font has no glyph names, Font Copy will generate a Type 42 (*fontname*.t42) file and save that file along with the* fontname*.ttf file.*

Using Build FAST

Use Build FAST to automatically create all necessary XPP supporting font files for fonts.

Build FAST uses the *.afm* or *.otf* files located in the appropriate XYV_EXECS/psres/fonts subdirectory to do the following:

- Create the Phototypesetter Spec (PTS), FAST Generation Spec (FGS), and Pseudofont Spec (PSF).
- Generate a machine-readable Font Access Table (FAST) file.
- Update the Font Variant (FV) Spec you specify.
- Update the Typesetter Font Map (TSF) Spec.
- Optionally create the Kerning Pairs (KP) Spec and generate a corresponding machine-readable Kerning Pairs (KP) data file.



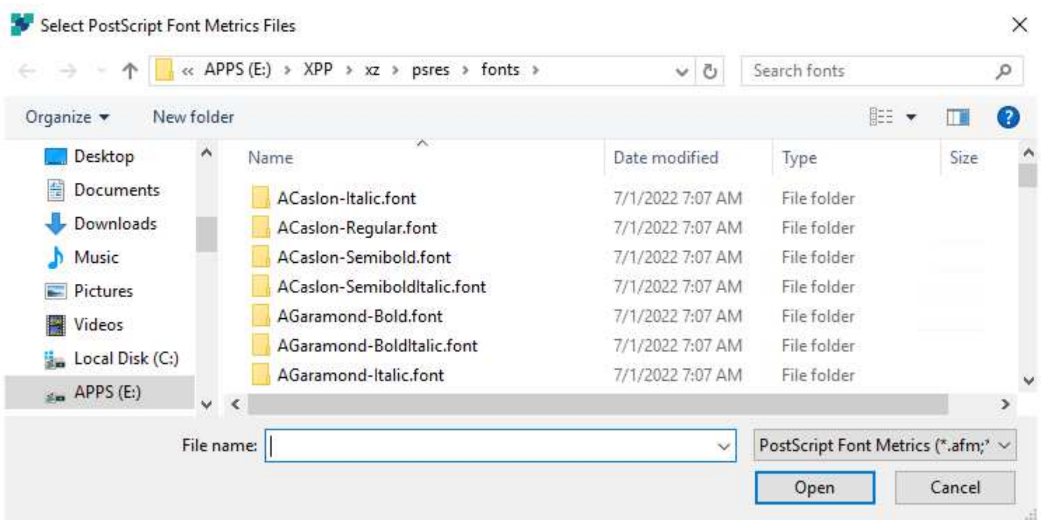
Note: *XPP does not support Type 3 fonts.*

Running Build FAST

To run Build FAST from the PathFinder toolbar:

1. Select **Tools > Build FAST**

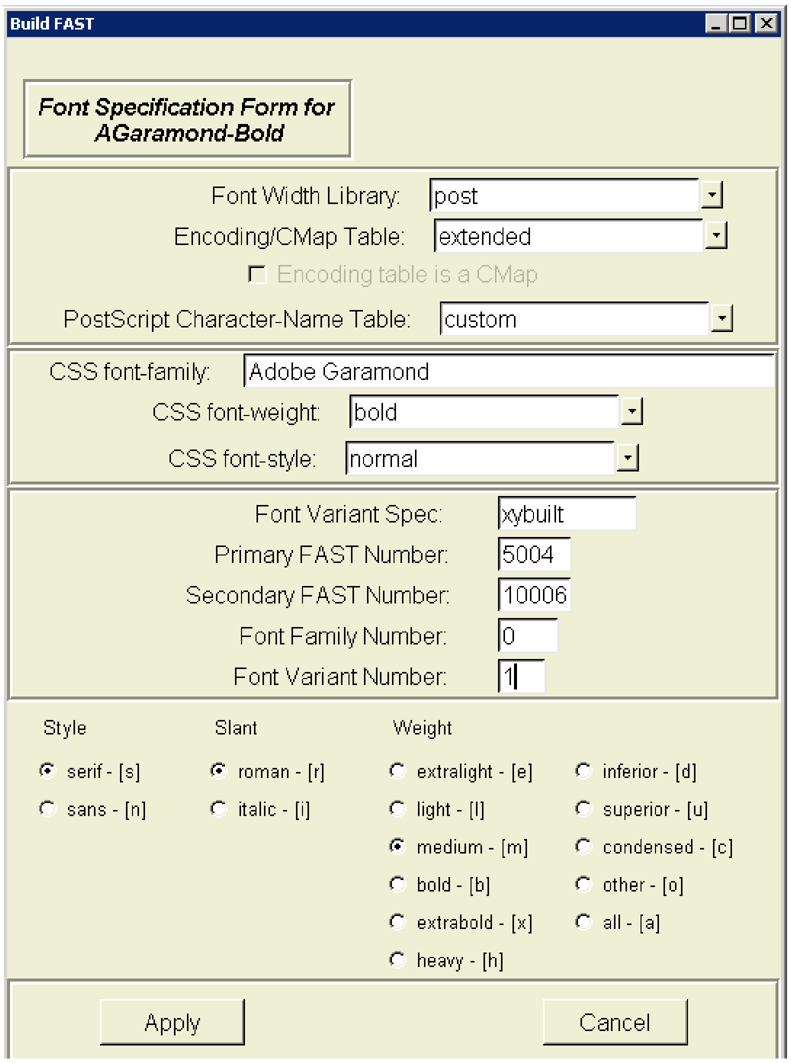
   XPP displays the Select PostScript Font Metrics Files list box, displaying the font folders in your XYV_EXECS/psres/fonts directory.



Name	Date modified	Type	Size
ACaslon-Italic.font	7/1/2022 7:07 AM	File folder	
ACaslon-Regular.font	7/1/2022 7:07 AM	File folder	
ACaslon-Semibold.font	7/1/2022 7:07 AM	File folder	
ACaslon-SemiboldItalic.font	7/1/2022 7:07 AM	File folder	
AGaramond-Bold.font	7/1/2022 7:07 AM	File folder	
AGaramond-BoldItalic.font	7/1/2022 7:07 AM	File folder	
AGaramond-Italic.font	7/1/2022 7:07 AM	File folder	

2. Select the **folder** of the font for which you are building the FAST and click the **Open** button.
   XPP displays the files associated with the font you selected.

3. Select the **.afm** or **.otf** file and click the **Open** button.
   XPP displays the Build FAST dialog box.



Font Specification Form for AGaramond-Bold

Font Width Library:	post
Encoding/CMap Table:	extended
☐ Encoding table is a CMap	
PostScript Character-Name Table:	custom
CSS font-family:	Adobe Garamond
CSS font-weight:	bold
CSS font-style:	normal
Font Variant Spec:	xybuilt
Primary FAST Number:	5004
Secondary FAST Number:	10006
Font Family Number:	0
Font Variant Number:	1

Style	Slant	Weight	
● serif - [s]	● roman - [r]	○ extralight - [e]	○ inferior - [d]
○ sans - [n]	○ italic - [i]	○ light - [l]	○ superior - [u]
		● medium - [m]	○ condensed - [c]
		○ bold - [b]	○ other - [o]
		○ extrabold - [x]	○ all - [a]
		○ heavy - [h]	

The dialog box displays a title box, showing the font for which you are specifying a FAST. In the illustration, the title shows the FAST will be built for the Postscript Type 1 base font named *AGaramond-Bold*.

4. Set the **Font Width Library** field to the font library of your choice. The default is **noto** the first time you use Build FAST. Subsequently, Build FAST defaults to the last **Font Width Library** that was specified the last time Build FAST was run. The Font Width Library name is a maximum of eight alphanumeric characters of your choice (not including the preceding *L*).

   If you enter a library that does not exist, the program creates it.

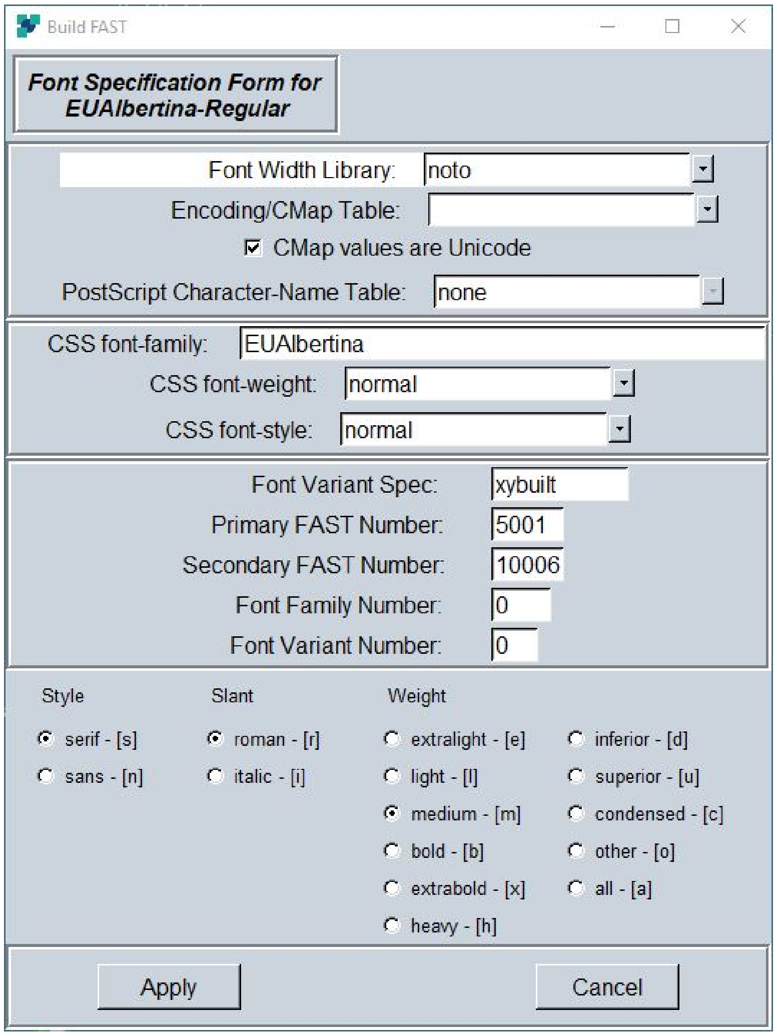
5. Set the **Encoding/CMap Table** name field.
   An encoding table maps PostScript character names to font glyphs in a

Type 1 base (non-CID) font. A CMap maps ID numbers or PostScript character names to glyphs in a Type 1 CID or OpenType font.

- **PostScript Type 1 Base Font**

  If you are using a PostScript Type 1 **base** font, you can use the drop-down menu to view the list of available encoding tables located in XYV_EXECS/sys/od/ps_dlf encodings.
  - The default entry the first time you use Build FAST is **extended**. Subsequently, Build FAST defaults to the last encoding table that was specified the last time for a Type 1 base font.
  - For ***Pi*** and **non-standard** text fonts, enter **none** for the encoding table name. Refer to ″Non-Standard Font Encodngs″ on page 14-3 for more information.
  - To create custom tables, refer to Chapter 14 “Encoding Tables”.
  - The *Encoding table is a CMap* field is disabled when using a PostScript Type 1 base font. Instead, the dialog enables the *PostScript Character-Name Table* field.

- **Type 1 CID Font**



Build FAST

Font Specification Form for EUAlbertina-Regular

Font Width Library:	noto
Encoding/CMap Table:	
☑ CMap values are Unicode	
PostScript Character-Name Table:	none
CSS font-family:	EUAlbertina
CSS font-weight:	normal
CSS font-style:	normal
Font Variant Spec:	xybuilt
Primary FAST Number:	5001
Secondary FAST Number:	10006
Font Family Number:	0
Font Variant Number:	0

Style	Slant	Weight	
● serif - [s]	● roman - [r]	○ extralight - [e]	○ inferior - [d]
○ sans - [n]	○ italic - [i]	○ light - [l]	○ superior - [u]
		● medium - [m]	○ condensed - [c]
		○ bold - [b]	○ other - [o]
		○ extrabold - [x]	○ all - [a]
		○ heavy - [h]	

In the illustration, the title shows the FAST will be built for the Type 1 CID font named *EUAlbertina-Regular*.

If you are using a Type 1 CID font, the *Encoding/CMap Table* field drop-down menu contains the list of CMaps located in XYV_EXECS/psres/fonts/CMap.

a. Select the appropriate **CMap** file name.

   Note: If you are installing a **CID** font, XPP enables a *CMap values are Unicode* field instead of the *Encoding table is a CMap* field.

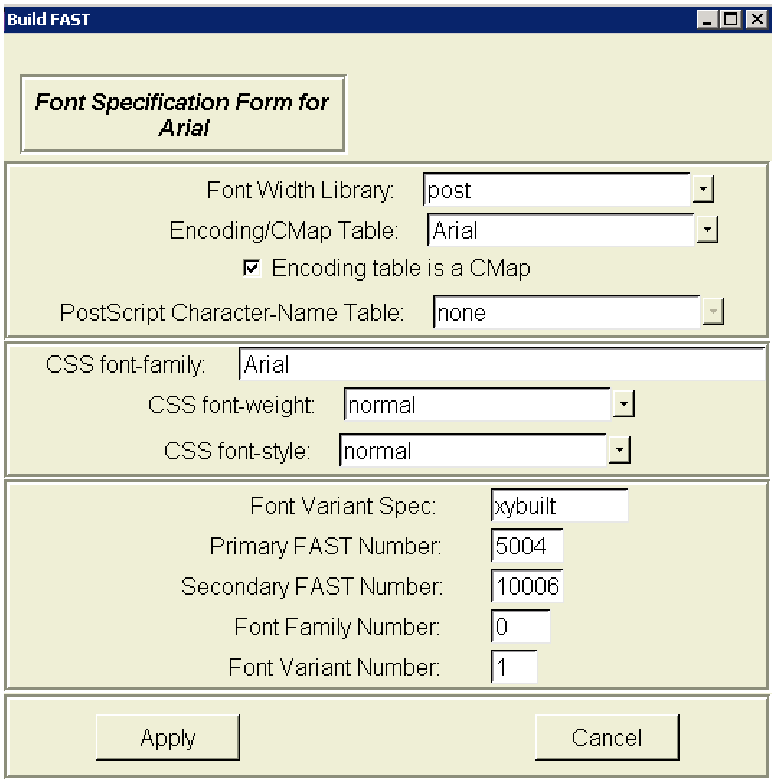
b. Click the button in front of the *CMap values are Unicode* field if you are installing CID fonts and the CMap encoding is a subset of the Unicode character set.

   The button changes color to indicate that the field is active. Build FAST uses the Unicode number as the XCS number and inserts it in the computer generated PTS table.

   Note that if you do not check the box in front of *CMap values are*

Unicode field, indicating that the CMap is not a Unicode CMap, and the CMap value is not a valid XCS number, Build FAST generates the message, “Assigning character # to 999”, rather than leaving an invalid XCS number assigned to that font character that would cause the Build FAST process to abort when creating the PTS Spec and FAST.

- **OpenType font**



Font Width Library:	post
Encoding/CMap Table:	Arial
☑ Encoding table is a CMap	
PostScript Character-Name Table:	none
CSS font-family:	Arial
CSS font-weight:	normal
CSS font-style:	normal
Font Variant Spec:	xybuilt
Primary FAST Number:	5004
Secondary FAST Number:	10006
Font Family Number:	0
Font Variant Number:	1

In the illustration, the title shows the FAST will be built for the non-CID OpenType font named *Arial*.

An OpenType PostScript font may be either

- a base font
- a CID font

OpenType Base Font

If it is an OpenType base font, its characters are always accessed with the Unicode CMap that is located in the same directory as the font file, and which has the same name as the font. This CMap is generated when you import the font with the Font Copy utility.

With an OpenType base font, the CMap name is set to this value (and the drop-down menu is disabled since it is the only valid value). Also, the Encoding table is a CMap field is disabled (set to selected) when using an Open Type base font.

OpenType CID Font

If this is an OpenType CID font, its characters may be accessed either with the font-specific Unicode CMap generated by Font Copy, which is the first entry in the drop-down menu, or by an appropriate CMap located in the XYV_EXECS/psres/fonts/CMap directory (but only when using *psfmtdrv* and not when using *divpdf*).

With an OpenType CID font, do the following:

- Select the desired **CMap** name.

  If you are selecting the font-specific Unicode CMap or any other Unicode-based CMap, be sure the radio button in front of the *CMap values are Unicode* is selected.

  If you are selecting a non-Unicode CMap, be sure the button is clear.

6. If you are importing a Type 1 (non-Unicode) base font, the dialog activates the *PostScript Character-Name Table* field. Choose one of the following options:

   - **custom** table: If you are adding a new Type 1 base font for existing jobs that were based on XCS values and were converted to Unicode for XPP 8.x based on the XCS Spec, this choice matches how Unicode values were assigned for the fonts in the legacy data brought forward and this new Type 1 base font being added to legacy data.
   - **unicode** table: If you are adding Type 1 base fonts, but are importing Unicde characters as-is, this option overrides the Unicode values in the original XCS Spec that were preserved to maintain legacy data.

7. For PostScript Type 1 base and CID fonts, use the *Style, Slant,* and *Weight* fields to indicate what type of glyphs the font contains. Set these fields to the values that apply to the glyphs in the font.

   It may be that a *Pi* font contains more than one variation of glyphs, that is, bold, italic, and medium variations, and you want to extract only the glyphs of a particular variant for a FAST. In that case, set these fields to the values that best apply to the majority of glyphs in the font relative to the main variant or secondary FAST to which this font would be added. Otherwise, if you want to extract all variations of glyphs in a *Pi* font for a FAST, then select the ″a″ value.

   For OpenType fonts, Build FAST automatically applies ″a″ to specify all style codes.

   You can always change the *Style Code* values later in the generated font specs and re-genfast a FAST.

8. For the remaining fields, use the following table as a guide:

Field:	*Description*
CSS font-family *CSS font-weight* *CSS font-style*	These fields are used if your DIV is in css-xml mode; they correspond to the font-family, font-style, and font-weight properties in your .css style sheet definitions. Build FAST does its best to set these field values based on values in the AFM file. Change any values that are not correct for the font. Refer to the *Styling Content with CSS* publication for more information on using fonts with CSS.
Font Variant Spec	The default is **xybuilt**. The name is a maximum of eight alphanumeric characters. You can change this field to specify any FV Spec. If you specify an FV Spec that does not exist, Build FAST creates it for you. ***Note:*** The FV Spec that you name/create lives in the font library, so you still need to update the FV in a style library.
Primary FAST Number	This is the font number that identifies the font. Numbering starts at 501 by default the first time you use Build FAST. You may use your own numbering convention. Valid values are 0-32767. Whether you use the default convention or your own, Build FAST uses the same value that was used last time unless the Font Variant Number that was used last time was 255, in which case Build FAST will increment the Font Family Number by one.
Secondary FAST Number	This is the number of the Secondary FAST. Valid values are 0-32767.
Font Family Number	This is the number of the font family. Valid entries are 0-2047. Build FAST will use the same value that was used last time, unless the *Font Variant Number* that as used last time was 255, in which case Build FAST will increment the *Font Family Number* by one.
Font Variant Number	This is the number of the font variant. Valid entries are 0-255. Build FAST will increment the number by one on each subsequent use. If the value used last time was 255, Build FAST will increment the last *Font Family Number* by one and set the *Font Variant Number* to zero.

If you enter an invalid character or value in any field, including too many characters in a library or spec name, a beep sounds, and the field rejects the last character you entered.

9. When you have completed the form, click the **Apply** button.

   If you leave any field blank, a beep sounds and a Warning box displays the following message, “Please enter all values.”

Click the **OK** button in the Warning box.
XPP places the cursor in the first blank field that requires a value.

If Build FAST finds the specified font name in the TSF file with a different FAST number and/or another font is already assigned to the specified FAST number, the utility displays a pop-up message window. For example, if you specify font NotoSansMono-Regular and FAST 30, the utility displays the following warning messages:

WARNING: NotoSansMono-Regular already exists as font: 0
WARNING: NotoSerif-Regular already exists as font: 30
Do you want to continue?

10. Click **no** to abort the process.
    —or—
    Click **yes** to continue.

    Build FAST closes the dialog box and displays the Font Build window, displaying the program's progress about the specs Build FAST created.

11. Click the **OK** button.
    Build FAST displays a message box that states "Completed."

12. Click the **OK** button.
    Build FAST closes the message window.

Warning/Error Messages from Build FAST

When you click the *Apply*, you may get an error message. The following table displays the possible error messages you might see:

Error Message	*Description*
WARNING: Unencoded character *charactername*, using 0	See ”Modifying the Encoding Table” for instructions.
WARNING: Unmapped character *CHAR CODE*, using *XCS number*	This means that the font is a CID font and the *CMap values are Unicode* field was unchecked and a CHAR CODE in the font either did not match an XCS number in the XCS Spec or the matched XCS character did not have a Unicode value assigned to it.

Error Message	***Description***
INFO: There are *count* undefined characters in PTS ′*name*′: these appear as Unicode # ′x0′	This means that when GenFAST was run by Build FAST on the PTS Spec, there were *count* rules in the spec that had a zero UNICODE NUMBER value. When Build FAST creates a PTS Spec for a font, it will add rules to he PTS Spec for unencoded glyphs, with zero for both the CHAR CODE and UNICODE NUMBER fields but with the actual glyph width and description.
WARNING: line 10 field ′fontname′ (*fontname*) too long - limit is 18	This means that the Font Family name exceeds the 18-character limit for the Font Family. The offending name is truncated and is noticeable in the Status Window display. To see what the utility has done and edit it if you want, edit the FGS Spec fields.
WARNING: line 10 field ′fontvar′ (*variantname*) too long - limit is 12	This means that the Font Variant name exceeds the 12-character limit for the Font Variant. The offending name is truncated and is noticeable in the Status Window display. To see what the utility has done and edit it if you want, edit the FGS Spec fields.
WARNING: No (*character name*) PSN	Enter the character name(s) in *psn_custom* or *psn_unicode* in *Lsyslib*, with an appropriate Unicode number and rerun Build FAST.
WARNING: Font contains (*number*) kerning pairs; KP table can only hold first 6,553,500; rest will be ignored	The KP table cannot hold more than 6,553,500 kerning pairs.
WARNING: CMap code (*number*) collides with xyps, 0xf800-f8ff; characters in this range will be ignored.	XPP reserves the code range f800-f8ff for its own use, to define xyps, as allowed by the Unicode specification. Fonts must not define characters in this range.

After Running Build FAST

When Build FAST has completed its process, there are some different follow-up steps you may need to perform, depending on whether you ran Build FAST for text fonts or *Pi* fonts.

Follow-up for Text Fonts

You may need to:

- Update the Font Variant Spec in the style library that is specified in the Job Ticket.

  When you run Build FAST, the copy of the Font Variant Spec that Build FAST updates is located in your font library. The system, however, uses the Font Variant Spec in the style library that is specified in the Job Ticket. Therefore, you may want to save the rule from the FV Spec in the font library and restore it to the FV Spec in the style library.

- Add the font to the appropriate download table.

  If you are using enable font download (**–efd**), add the font to the appropriate font download table (in the XYV_EXECS/sys/od/ps_dlf/ tables directory).

Follow-up for Pi Fonts

Setting up a *Pi* font requires a good understanding of the XPP font environment. If you do not have this foundation, RWS strongly recommends that you read the PostScript Name Spec, FAST Generation Spec, Creating and Viewing FAST Specs, and Encoding Tables chapters in this manual.

You may need to complete the following steps to properly install a *Pi* font:

1. Edit the PTS Spec that Build FAST created if you want to use style codes for proper character access.
2. Add a rule for this *Pi* font PTS to the FGS Spec(s) for your *Pi* (secondary) FASTs so you can access it without changing font families.

   The FGS Specs typically used for generating *Pi* (secondary) FASTs are fgs_10001 through 10009. Each of these specs generally relates to specific style codes, for example 10002 for sans bold.
3. Run *GenFAST* against the number of the FGS Spec you have edited. This utility includes the new *Pi* characters in the FAST.
4. Run a font width test against the *Pi* FAST.
5. Test the font by using it in a division.
   To determine which key sequence or character entity accesses your new *Pi* characters, check the XCS Spec in the *syslib* library, which lists the ASCII sequence for each character.
6. If you are using enable font download (**–efd**), add the font to the appropriate font download table (XYV_EXECS/sys/od/ps_dlf/tables dirctory).

Resolving Unidentified PSN Names During Build FAST Processing

The warning ″Unidentified PSN″ occurs primarily in *Pi* fonts if the system does not recognize the PostScript ″name(s)″ used to identify characters in the *fontname.afm* file. Often the ″name″ is a number.

The *syslib* library contains the following PSN Specs:

- **psn_custom**
- **psn_unicode**
- **psn_ps2xcs**

Build FAST checks the character names in the *.afm* file against two of these specs for matching names (always psn_ps2xcs and then either psn_custom or psn_unicode). If some or all do not match, Build FAST reports a warning that the character name(s) is not recognized. This is important because this is how XPP assigns XCS numbers (a requirement) to character names. See step 6 on page 1-16 for an explanation of the difference between the psn_custom and the psn_unicode Specs.

The warning does not necessarily prevent you from loading your *Pi* font, but it does prevent you from functionally using the characters in XPP. There are two choices available for you to continue.

Choice 1

Choice 1 involves editing psn_custom and/or psn_unicode and re-running Build FAST.

Note: **Do not** *edit psn_ps2xcs. It contains 3720 character names that have become standard usage by many font vendors. If you edit this spec, you run the risk of inadvertently deleting a standard character name and causing extra work in future font installations.*

RWS recommends adding these characters to psn_custom and/or psn_unicode as the preferred way to resolve the warnings for the following reasons:

1. If the *Pi* font has a character(s) that could occur in other fonts, you want to add the character to the PSN Spec. Having the character in *_psn_custom.sde* and/or *_psn_unicode.sde* allows XPP to read the character name the next time you build a font containing this character.

2. If you must use custom XCS numbers, this allows you to keep track of used numbers.

Modifying the PSN Spec

If your print log indicates that the PostScript Name Spec does not recognize the names of your *Pi* font characters, do the following:

1. Identify the **custom slots** in the Xyvision Character Set (XCS) and determine the **number(s)** to which you can assign the character(s) in your *Pi* font.
2. Access the **PostScript Name Spec** from the PathFinder Tree View, using the following sequence:

   STYLE LIBRARIES > Lsyslib > PostScript Names

   PathFinder displays the available file names in the List View.
3. Select **custom** or **unicode** from the List View.
   XPP displays the *_psn_custom.sde* or *_psn_unicode.sde* Spec, containing one or more rules with the following fields:

Field	*Description*
PS Name	Specifies the name of a PostScript character.
XCS number	Specifies a Xyvision Character Set number to associate with the PostScript character.
Description	An optional identifying description for the character.

4. Add a **rule** for each character in your *Pi* font by specifying the following:
   - A name for the character.
   - The XCS number for the position you want assigned to it.
   - An optional description for the character.
5. Run Build FAST again (refer to page 1-10).
   If, after completing these steps and running Build FAST a second time, you still have warning/error messages in the Build FAST log, consult the Troubleshooting section on page 1-23.

Choice 2

Choice 2 involves modifying the PTS Spec

1. Modify the PTS Spec and assign XCS numbers to those characters.
2. Run GenFAST against the corresponding FGS Spec.

Note: *With either Choice 1 or 2, be aware that while the system doesn't recognize the character name in the .afm file, the character itself may exist in the XCS Spec under another name or description. Carefully check the XCS Spec. If the character exists, you can map it to that XCS number in psn_custom and/or psn_unicode, or the PTS Spec. If not, select an unassigned XCS number in the customer definable range.*

Troubleshooting Fonts

Occasionally you may experience some difficulty installing fonts. This section addresses some of the common situations that RWS has encountered.

The following procedures **require** a solid understanding of the XPP font environment and some familiarity with operating system utilities and commands. If you do not have this foundation, RWS strongly advises that you read the remaining chapters of this manual before attempting any of these procedures.

Troubleshooting Tips

Use the following sequence of steps to isolate font problems:

1. Verify that the font file is the correct format - PostScript Type 1 base, Type 1 CID, OpenType/PostScript, OpenType/TrueType, OpenType/PostScript CID, and external CMap (refer to page 1-2).
2. Verify that it is in the correct location with a correct index entry in the *PSres.upr* file (refer to page 1-5).
3. Run a font width test against the FAST in question (refer to page 11-15).
4. Determine if the problem is output only. If so, check the font download table.
5. Determine if the problem is with the XyView display. If so, check the font specs in XPP and the associated font library.

Verifying that You Have a Valid Font File

If you have trouble with a particular font, first verify that it is a valid font file. You can do this by running the XPP **xyprfont.pl** utility on the font file from a command window. Xyprfont.pl is located in XYV_EXECS/bin. You can run **xyprfont.pl** on PostScript Type 1 base, Type 1 CID, OpenType/PostScript, OpenType/TrueType, and Open Type/Postscript CID fonts.

Xyprfont.pl is a Perl program that produces a PostScript file that shows all the characters in a font. The output file bears the PostScript name of the font and the extension ".ps". If certain options are used, they are reflected in the file name, too.

Each page prints a range of 256 character codes; code points on the page that are not mapped in the font appear as gray cells.

For OpenType fonts, characters in the font that are not mapped in the font's

Unicode cmap table appear on pages following the mapped characters.

PostScript Type 1 Base Font

To run **xyprfont.pl** on a PostScript Type 1 base font, enter the following syntax on the command line:

```
xyprfont.pl FontFile [-fn FontName][-enc EncodingFile]
```

Where:

FontFile is the name of the PostScript font file.

–fn FontName (optional) is the PostScript font name. Use this option if the program cannot find the PostScript name within the file.

–enc EncodingFile (optional) is the name of a PostScript encoding file; this file must be in the current directory or in:

XYV_EXECS/sys/od/ps_dlf/encodings

Using this option creates a PostScript file with a name using the following format: *FontName[_EncodingFile].ps*

For example, the following command:

```
xyprfont.pl Times.pfa
```

executes the *xyprfont* utility on the **Times.pfa** font file in the current directory, using the default encoding, and produces the file Times.ps.

In rare cases, where the program cannot find the PostScript name within the file, you may need to use the following syntax:

```
xyprfont.pl xxx.pfa -fn Times
```

where xxx.pfa is the file name but “Times” is the actual PostScript font name.

Type 1 CID Font

To run **xyprfont.pl** on a Type 1 CID font, use the following syntax on the command line:

```
xyprfont.pl FontFile [-cmap [CMapName]][-from Start]
[-to End]
```

Where:

FontFile is the name of the font file.

–cmap (optional) Character Map maps from the character codes in the document to the glyphs in the font.

CMapName (optional) is the name of the CMap file. This entry must be in the current directory or in XYV_EXECS/psres/fonts/CMap.

If you do not provide a CMap, the program uses Identity-H CMap (built into the program), which generates all possible

65K code positions (up to 256 pages) unless limited by **–from** and **–to** options.

–from Start says to start processing with the page that contains this character (e.g.: in hex you might enter 3a00). If you don't provide the character, processing starts with the first character in the CMap file.

–to End says to end the processing with the page that contains this character (e.g.: in hex, you might enter 3bff). If you don't provide this character, processing continues through the last character in the CMap file.

For example, the following command:

```
xyprfont.pl Orient-Bold –cmap EUC-H –from 2121 –to 277e
```

produces Orient-Bold_EUC-H_2100-27ff.ps in the current directory.

OpenType Font

You can run *xyprfont.pl* on an OpenType/TrueType, OpenType/PostScript, or OpenType/PostScript CID font. You can access characters with either a CMap or an encoding file (for non-CID fonts).

CMap Access

To run *xyprfont.pl* on an OpenType font using CMap access, use the following syntax on the command line:

```
xyprfont.pl FontFile [-cmap [CMapName]] [-from
Start] [-to End] [-no_unmap] [-tone]
```

Where the following is true:

FontFile is the name of the font file.

–cmap (optional) Character Map maps from the character codes in the document to the glyphs in the font.

CMapName (optional) is the name of the CMap file. If you do not enter a CMapName, the program extracts the mapping from the Windows Unicode cmap table in the font.

–from Start says to start character output with the page that contains the character Start (in hex; e.g., '-from a7e' starts with page 0a00-0aff); if there is no Start page, start with first page in the CMap.

–to End says to end character output with the page that contains the character End (in hex; e.g., '–to 3b60' ends with page 3b00-3bff); if there is no end page, end with the last page in the CMap.

–no_unmap (optional) Used to suppress unmapped glyphs.

–tone (optional) converts OpenType/PostScript fonts to Type 1 so that they can be printed on older PostScript devices.

Encoding File Access

To run *xyprfont.pl* on an OpenType font, using encoding file access (non-CID font only), use the following syntax on the command line:

```
xyprfont.pl FontFile –enc [EncodingFile]
```

Where the following is true:

FontFile is the name of the font file.

EncodingFile (optional) is the name of a PostScript encoding file.

This file must be in the current directory or in XYV_EXECS/sys/od/ps_dlf/encodings

Use the encoding (8-bit) access method to address character positions 0-255. If an EncodingFile is provided, use that encoding; otherwise, use the font's default encoding.

If the **–enc** option is not present, *xyprfont.pl* uses the CMap access method.

Unmapped Glyphs

OpenType fonts often contain additional glyphs that are not included in the default Unicode CMap. Unmapped glyphs are printed following the mapped characters only when the -from, -to, -cmap and -enc options are not used. To suppress unmapped glyphs even when those options are not used, supply the **-no_unmap** option.

For information about mapping unmapped glyphs, see "Mapping Unmapped Open Type PostScript Font Glyphs" on page B-8.

Viewing Output File

If you generate an output file, you can look at the contents with a PostScript viewer or print it. If the font is a valid font, the output contains a grid of the characters. If the contents of the output file are in NotoSansMono typeface, then the font is not valid (unless you are printing the NotoSansMono font).

If the file does not contain a valid font, the program displays an error message on the screen during processing. If this occurs, go back to your font vendor to exchange it or convert it using a font conversion program.

Notes:

- *You can also use* `XYV_EXECS/gs/gsx FontFile.ps` *to view the xyprfont.pl output.*
- *If you are running xyprfont.pl in the XYV_EXECS/psres/fonts directory tree, be sure to remove the .ps file when you finish testing. Otherwise, the next time you run makepsres, you will receive a warning about an invalid font file.*

Modifying the Encoding Table

At times, a font being installed contains characters that are not included in the encoding table. You may need to modify the encoding table. Refer to the Encoding Tables chapter on page 14-1 for complete information.

Verifying the Directory Structure

A font can appear in XPP in NotoSansMono or another font or not match the actual font you thought you set up. This problem can be caused by an invalid font file or a configuration problem.

Consult the following list of scenarios to see whether any match the problem you have encountered.

Note: *If you run* makepsres *to solve your problem, make sure that you run the XPP-supplied version in XYV_EXECS/bin, and not any version that may be a part of your operating system.*

- If you used Font Copy to copy fonts into place and did not respond with *Yes* at the prompt to *update the font search path*, some or all of your fonts may not be accessible. The preferred way to recover is to go back into Font Copy and rerun the utility, selecting *yes* to update the font search path.

  An alternative method at the command prompt level is to change to the XYV_EXECS/psres directory and run XYV_EXECS/bin/makepsres. Note that the command **must** be issued from within the XYV_EXECS/psres directory.

- If you manually moved fonts into the XYV_EXECS/psres/fonts directory, you have to update the *PSres.upr* index file.

  Follow the instructions outlined above for running *makepsres*.

  This is not recommended with OpenType fonts because the FontCopy utility also creates an AFM and a CMap file for the font.

- If you issue XYV_EXECS/bin/makepsres from a directory other than XYV_EXECS/psres, you create a *PSres.upr* file in the directory from where you issued the command. As a result, you may not be able to access fonts from XPP.

  To recover, remove any PSres.upr files that are in an incorrect location. In addition, go to XYV_EXECS/psres and rename or remove any PSres.upr files that are there. Then, rebuild psres by following the instructions outlined above for running *makepsres*.

- If the variable PSRESOURCEPATH has been customized previously (for Adobe applications) in your home directory, it may need to be modified.

- If you cannot access a font, verify that the entries in the encoding *Type* and *Encoding* table name fields of the TSF Spec for that font are correct. For encoding Type: standard, the correct entry is usually **extended** for text fonts or **none** for Pi fonts.

  Sometimes, the encoding field contains a typographic error and/or extraneous information from the old encoding table file. For example, if the correct entry is **none**, using **none1** does not work.

- Verify that the contents of the PostScript font *Name* field in the TSF Spec is an exact match with the PostScript font name found in the font itself. Open the TSF Spec using PathFinder (STYLE LIBRARIES > Lsyslib (or the font library that Build FAST created/updated) > Typesetter Font Maps > system). Check the rule associated with the font in question, verifying spelling, capitalization, and punctuation.

Part II

Understanding XPP Fonts and Font Specs

Chapter 2

Introduction to Fonts

This chapter contains information on the following topics:

- Understanding Xyvision Standard Format
- PostScript fonts
- Font download tables for PostScript fonts
- Overview of the specs necessary for display and output
- Unicode capabilities

Understanding Xyvision Standard Format

The following section:

- Defines Xyvision Standard Format
- Defines Xyvision Character Set
- Explains the conversion process

Xyvision Standard Format (XSF)

Xyvision Standard Format (XSF) is a system of using numbers to represent characters. These numbers are called Xyvision Character Set (XCS) numbers.

The Xyvision Standard Format files contain XPP specific codes and characters, in addition to ASCII characters.

Because output devices, such as PostScript printers, can print more characters than the standard ASCII character set, XPP uses an expanded character set called the Xyvision Character Set (XCS) to support these output devices.

Xyvision Character Set (XCS)

The Xyvision Character Set (XCS), consisting of more than 32,000 characters, is defined in the XCS Spec in the *syslib* library.

The XCS Spec defines standard alphanumeric characters, accents, foreign characters, math and chemical symbols, custom characters, such as company logos, and format codes, such as tags, macros, text stream elements, entities, and Unicode® values.

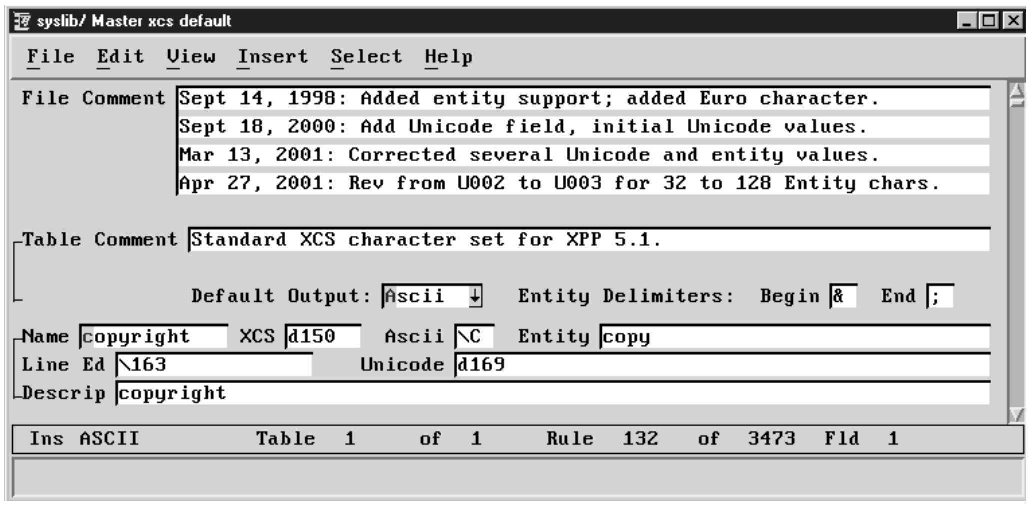
For each character in the spec, XPP assigns an XCS number, a name, and a unique ASCII escape sequence, consisting of one, two, or three characters. When you convert a text file to or from Xyvision Standard Format, the conversion program refers to the XCS Spec.

To examine the XCS Spec from PathFinder:

1. Navigate to **STYLE LIBRARIES > Lsyslib > Xyvision Character Set**.
   PathFinder displays the available libraries in the List View.
2. Double-click the **default** Spec.
   XPP opens the spec in the SDeditor.

The following figure is an example of the copyright rule in the XCS Spec:

***Note:** "default" is the only valid name for the XCS Spec. Though PathFinder*



```
File Comment  Sept 14, 1998: Added entity support; added Euro character.
              Sept 18, 2000: Add Unicode field, initial Unicode values.
              Mar 13, 2001: Corrected several Unicode and entity values.
              Apr 27, 2001: Rev from U002 to U003 for 32 to 128 Entity chars.

Table Comment  Standard XCS character set for XPP 5.1.

               Default Output: Ascii    Entity Delimiters:  Begin &   End ;

Name copyright    XCS d150    Ascii \C    Entity copy
Line Ed \163              Unicode d169
Descrip copyright

Ins ASCII        Table  1    of  1     Rule  132   of  3473  Fld  1
```

may contain other specs with different names (these may be back-up copies), XPP does not recognize them and does not open them.

The Conversion Process

When you enter characters in a division, the system uses the XCS numbers that correspond to these characters to access information, such as:

- Character widths for composition
- Character glyphs for screen display and output

Text in an XPP division can include any of the defined characters in the XCS Spec and is always stored in Xyvision Standard Format.

PostScript Fonts

PostScript fonts are character glyphs that are loaded into various locations on your computer system or on an output device. RWS supplies 163 Noto OpenType fonts and two XPP pi fonts that are ready to use as soon as you install XPP.

Standard 35 PostScript Type 1 Base Fonts

Many PostScript devices include 35 standard fonts—33 text fonts and two *Pi* (non-alphanumeric character) fonts. These 35 fonts are usually built into an output device. Each text font contains the same set of characters. XPP does not deliver these fonts nor any font specs for these fonts.

Some readers and devices can render only characters from those fonts or from a different set of standard fonts; all other fonts need to be rendered by substitution of characters from these fonts, unless you embed your fonts directly into your outputs. Readers and devices will use the embedded fonts to render their characters correctly.

XPP identifies fonts by numbers; PostScript interpreters identify fonts by names. The following table lists the standard 35 PostScript fonts by name.

Table 2-1 *Standard 35 PostScript Fonts*

PostScript Font Name	*PostScript Font Name*
AvantGarde-Book	Helvetica-Narrow-Bold
AvantGarde-BookOblique	Helvetica-Narrow-BoldOblique
AvantGarde-Demi	NewCenturySchoolbook-Roman
AvantGarde-DemiOblique	NewCenturySchoolbook-Italic
Bookman-Light	NewCenturySchoolbook-Bold
Bookman-LightItalic	NewCenturySchoolbook- BoldItalic
Bookman-Demi	Palatino-Roman
Bookman-DemiItalic	Palatino-Italic
Courier	Palatino-Bold
Courier-Oblique	Palatino-BoldItalic
Courier-Bold	Symbol *(a Pi)*
Courier-BoldOblique	Times-Roman

Table 2-1 *Standard 35 PostScript Fonts (Continued)*

PostScript Font Name	*PostScript Font Name*
Helvetica	Times-Italic
Helvetica-Oblique	Times-Bold
Helvetica-Bold	Times-BoldItalic
Helvetica-BoldOblique	ZapfChancery-MediumItalic
Helvetica-Narrow	ZapfDingbats (*a Pi*)
Helvetica-Narrow-Oblique	

XPP-delivered Noto and Pi OpenType Fonts

The following 163 Noto OpenType fonts and two XPP *Pi* fonts are delivered by XPP:

NotoSansArabic-Black
NotoSansArabic-Bold
NotoSansArabic-ExtraBold
NotoSansArabic-ExtraLight
NotoSansArabic-Light
NotoSansArabic-Medium
NotoSansArabic-Regular
NotoSansArabic-SemiBold
NotoSansArabic-Thin
NotoSans-Black
NotoSans-BlackItalic
NotoSans-Bold
NotoSans-BoldItalic
NotoSansCJKhk-Black
NotoSansCJKhk-Bold
NotoSansCJKhk-DemiLight
NotoSansCJKhk-Light
NotoSansCJKhk-Medium
NotoSansCJKhk-Regular
NotoSansCJKhk-Thin
NotoSansCJKjp-Black
NotoSansCJKjp-Bold
NotoSansCJKjp-DemiLight
NotoSansCJKjp-Light
NotoSansCJKjp-Medium
NotoSansCJKjp-Regular
NotoSansCJKjp-Thin
NotoSansCJKkr-Black
NotoSansCJKkr-Bold
NotoSansCJKkr-DemiLight
NotoSansCJKkr-Light
NotoSansCJKkr-Medium
NotoSansCJKkr-Regular
NotoSansCJKkr-Thin
NotoSansCJKsc-Black
NotoSansCJKsc-Bold
NotoSansCJKsc-DemiLight
NotoSansCJKsc-Light
NotoSansCJKsc-Medium
NotoSansCJKsc-Regular
NotoSansCJKsc-Thin
NotoSansCJKtc-Black
NotoSansCJKtc-Bold
NotoSansCJKtc-DemiLight
NotoSansCJKtc-Light
NotoSansCJKtc-Medium
NotoSansCJKtc-Regular
NotoSansCJKtc-Thin
NotoSans-Condensed
NotoSans-CondensedBlack
NotoSans-CondensedBlackItalic
NotoSans-CondensedBold
NotoSans-CondensedBoldItalic
NotoSans-CondensedExtraBold
NotoSans-CondensedExtraBoldItalic
NotoSans-CondensedExtraLight
NotoSans-CondensedExtraLightItalic
NotoSans-CondensedItalic
NotoSans-CondensedLight
NotoSans-CondensedLightItalic
NotoSans-CondensedMedium
NotoSans-CondensedMediumItalic
NotoSans-CondensedSemiBold
NotoSans-CondensedSemiBoldItalic
NotoSans-CondensedThin
NotoSans-CondensedThinItalic
NotoSans-ExtraBold
NotoSans-ExtraBoldItalic
NotoSans-ExtraLight
NotoSans-ExtraLightItalic
NotoSansHebrew-Black
NotoSansHebrew-Bold
NotoSansHebrew-ExtraBold
NotoSansHebrew-ExtraLight
NotoSansHebrew-Light
NotoSansHebrew-Medium
NotoSansHebrew-Regular
NotoSansHebrew-SemiBold
NotoSansHebrew-Thin
NotoSans-Italic NotoSans-Light
NotoSans-LightItalic
NotoSansMath-Regular (Pi)
NotoSans-Medium
NotoSans-MediumItalic
NotoSansMono-Black
NotoSansMono-Bold
NotoSansMono-Condensed
NotoSansMono-CondensedBlack
NotoSansMono-CondensedBold
NotoSansMono-CondensedExtraBold

NotoSansMono-CondensedExtraLight
NotoSansMono-CondensedLight
NotoSansMono-CondensedMedium
NotoSansMono-CondensedSemiBold
NotoSansMono-CondensedThin
NotoSansMono-ExtraBold
NotoSansMono-ExtraLight
NotoSansMono-Light
NotoSansMono-Medium
NotoSansMono-Regular
NotoSansMono-SemiBold
NotoSansMono-Thin
NotoSans-Regular
NotoSans-SemiBold
NotoSans-SemiBoldItalic
NotoSansSymbols2-Regular (Pi)
NotoSansSymbols-Black (Pi)
NotoSansSymbols-Bold (Pi)
NotoSansSymbols-ExtraBold (Pi)
NotoSansSymbols-ExtraLight (Pi)
NotoSansSymbols-Light (Pi)
NotoSansSymbols-Medium (Pi)
NotoSansSymbols-Regular (Pi)
NotoSansSymbols-SemiBold (Pi)
NotoSansSymbols-Thin (Pi)
NotoSansThai-Black
NotoSansThai-Bold
NotoSansThai-ExtraBold
NotoSansThai-ExtraLight
NotoSansThai-Light
NotoSansThai-Medium
NotoSansThai-Regular
NotoSansThai-SemiBold
NotoSansThai-Thin
NotoSans-Thin
NotoSans-ThinItalic
NotoSerif-Black
NotoSerif-BlackItalic
NotoSerif-Bold
NotoSerif-BoldItalic
NotoSerifCJKhk-Black
NotoSerifCJKhk-Bold
NotoSerifCJKhk-ExtraLight
NotoSerifCJKhk-Light
NotoSerifCJKhk-Medium
NotoSerifCJKhk-Regular
NotoSerifCJKhk-SemiBold
NotoSerifCJKjp-Black
NotoSerifCJKjp-Bold
NotoSerifCJKjp-ExtraLight
NotoSerifCJKjp-Light
NotoSerifCJKjp-Medium
NotoSerifCJKjp-Regular
NotoSerifCJKjp-SemiBold
NotoSerifCJKkr-Black
NotoSerifCJKkr-Bold
NotoSerifCJKkr-ExtraLight
NotoSerifCJKkr-Light
NotoSerifCJKkr-Medium
NotoSerifCJKkr-Regular
NotoSerifCJKkr-SemiBold
NotoSerifCJKsc-Black
NotoSerifCJKsc-Bold
NotoSerifCJKsc-ExtraLight
NotoSerifCJKsc-Light
NotoSerifCJKsc-Medium
NotoSerifCJKsc-Regular
NotoSerifCJKsc-SemiBold
NotoSerifCJKtc-Black
NotoSerifCJKtc-Bold
NotoSerifCJKtc-ExtraLight
NotoSerifCJKtc-Light
NotoSerifCJKtc-Medium
NotoSerifCJKtc-Regular
NotoSerifCJKtc-SemiBold
NotoSerif-Condensed
NotoSerif-CondensedBlack
NotoSerif-CondensedBlackItalic
NotoSerif-CondensedBold
NotoSerif-CondensedBoldItalic
NotoSerif-CondensedExtraBold
NotoSerif-CondensedExtraBoldItalic
NotoSerif-CondensedExtraLight
NotoSerif-CondensedExtraLightItalic
NotoSerif-CondensedItalic
NotoSerif-CondensedLight
NotoSerif-CondensedLightItalic
NotoSerif-CondensedMedium
NotoSerif-CondensedMediumItalic
NotoSerif-CondensedSemiBold
NotoSerif-CondensedSemiBoldItalic
NotoSerif-CondensedThin
NotoSerif-CondensedThinItalic
NotoSerif-ExtraBold
NotoSerif-ExtraBoldItalic
NotoSerif-ExtraLight
NotoSerif-ExtraLightItalic

NotoSerif-Italic	NotoSerif-SemiBold
NotoSerif-Light	NotoSerif-SemiBoldItalic
NotoSerif-LightItalic	NotoSerif-Thin
NotoSerif-Medium	NotoSerif-ThinItalic
NotoSerif-MediumItalic	XPPOne (Pi)
NotoSerif-Regular	XPPTwo (Pi)

To access these Noto fonts, XPP delivers a set of font specs to the **noto** font library.

Embedding Fonts

PostScript fonts may reside on each output device. In addition, for screen display, they must reside in the proper format on the XPP system. In some cases, it may be desirable to embed PostScript fonts from XPP to the output device with the printed pages. Note that printer throughput is reduced using this method because the fonts are downloaded with each print task; they are not loaded into the printer's memory. Note that font embedding is required when your PostScript printer does not have the PostScript fonts stored locally.

When embedding fonts:

- The font must be OpenType, PostScript Type 1 base, or Type 1 CID.
- Store the fonts in the XYV_EXECS/psres/fonts area.
- When using *psfmtdrv*, list the fonts in your enable font download table.
- When using *divpdf*, fonts are subset embedded by default. You may use the *-nosubset* or *-noembed* options to change the default behavior.

PSRESOURCEPATH Environment Variable

The PSRESOURCEPATH environment variable specifies the path to the psres (PostScript resources) directory. The psres directory can be a link elsewhere. This path is important in order to display, download, and output PostScript or PDF.

On Unix systems, you define the PSRESOURCEPATH environment variable in the `/etc/xyvision/xyv.cshrc` and/or `/etc/xyvision/xyv.profile` file. The following is an example for the `/etc/xyvision/xyv.profile` file:

```
PSRESOURCEPATH="$XYV_EXECS/psres::"
export PSRESOURCEPATH
```

On Windows, you define the PSRESOURCEPATH environment variable in the computer settings. For example, on Windows Server 2022:

1. Select **Control Panel > System > Advanced system settings > Environment Variables**.
2. Select **New** under **System variables** to add the environment variable definition.

Font Download Tables for PostScript Fonts

A font download table is a simple text file, and may be created and/or modified using any ASCII or text editor.

You need font download tables to enable/disable font embedding when you generate a PostScript file in XPP or output to a PS device or when you create PDF output files via PS-to-PDF workflow (psfmtdrv -distill).

Note: *When you activate a font download table, only those fonts needed to output the current job are downloaded.*

When outputting, you use the enable font download (**–efd**) option to activate font download tables. Font download tables must be located in the following directory:

XYV_EXECS/sys/od/ps_dlf/tables

Font download tables must be named **table**_*number*,

where *number* is an integer greater than 1. For example, *table_2*, *table_3*, etc.

Format of an Enable Font Download Table

In a font download table, each font appears on a separate line in the following format:

enable-number   font-name

where:

Entry	***Description***
enable-number	Specifies whether or not to download this font. Enter **1** to enable downloading of this font; enter **0 if you do not want to download a particular font.**
font-name	Specifies the official PostScript name of the font to download.

Font Download Table (table_1)

XPP delivers **table_1** to the directory:

XYV_EXECS/sys/od/ps_dlf/tables

The following example displays the enable font download table, **table_1** :

```
1 NotoSans-Black
1 NotoSans-BlackItalic
1 NotoSans-Bold
1 NotoSans-BoldItalic
1 NotoSans-Condensed
1 NotoSans-CondensedBlack
...
1 NotoSerif-Thin
1 NotoSerif-ThinItalic
1 XPPOne
1 XPPTwo
```

Creating Custom Font Download Tables

Because XPP delivers table_1 with each release, you need to copy this table and rename it, using a different number (**table_***number*), and edit it for your site. You may have multiple font download tables.

Adding Fonts to Your Font Download Table

1. Edit the appropriate font control table (**table_***number*). Be sure the lines in the table begin with a **1** (one) if you want them to be downloaded and are in the following format:

   1 *PostScriptfontname*

   For example:

   1 NewBaskerville–Roman

2. Store and exit the table.

Activating Your Enable Font Download Table

You can activate your font download table from the Print Dialog.

To activate your font download table:

1. Select the **Print** tab and select *Printer, PS to file,* or *PS to PDF file*.
2. Select the **PS/PDF** tab.
   XPP displays the PostScript options.
3. In the PostScript content area, check the **box** in front of *Download fonts*.
   XPP enables the *Control table number* field.
4. Enter the **number** for the download table you want to use.
5. When you have finished selecting your print options, press the **Run** button.
   XPP displays the progress in the right-hand pane.

Specs Needed for Display and Output

The following table lists the specs XPP needs to display and output characters. These specs are described in subsequent chapters.

Table 2-2 *Specs Needed to Display/Output Characters*

Spec	*Mnemonic*	*The system uses it to determine*
Xyvision Character Set	XCS	the ASCII escape sequences, character entities, and Unicode assigned to the characters' XSF codes.
Keyboard Map	KB	XSF codes mapped to the keys pressed.
Font Access Table (FAST)	FX	details on characters specified by the XSF codes (result of processing PTS, PSF, FGS, and FGX specs).
Font Variant	FV	the *Primary FAST*, *Secondary FAST*, and *Default FAST* mapped to the font family and variant.
PostScript Name	PSN	PostScript characters mapped to the standard XCS character set.
Typesetter Font Map	TSF	PostScript font names mapped to font numbers and for CSS mode mapping of CSS font properties to PostScript font names.
Kerning Pairs *(optional)*	KP	spacing between specified characters.
Ligature/Accent Replacement *(optional)*	RP	whether input characters are to be replaced by specified accents or ligatures.
Job Ticket	none	the name of the font library and font variant spec(s)[1].
Division Ticket *(optional)*	none	the name of an alternate font variant spec (if any).
Item Format	IF	the font family and variant that are currently active[2] This Spec is used only in non-CSS-XML modes.
CSS	CSS	the font-family, font-style and font-weight properties that are currently active.[2] This Spec is used only in CSS-XML mode.

.

[1]Font Variant Specs can reside at the library level or at the job level.

[2]The system also checks any Font Family (*ff*) and Font Variant (*fv*) XyMacros for the currently active font family and variant.

How the System Uses the Specs

When you press a key on the keyboard, the system follows a series of steps, checking the various specs for information. Then it displays the character onscreen.

The following figure gives an overview of the process involved in displaying a character onscreen and outputting. For detailed information on the specs, refer to the individual chapter.

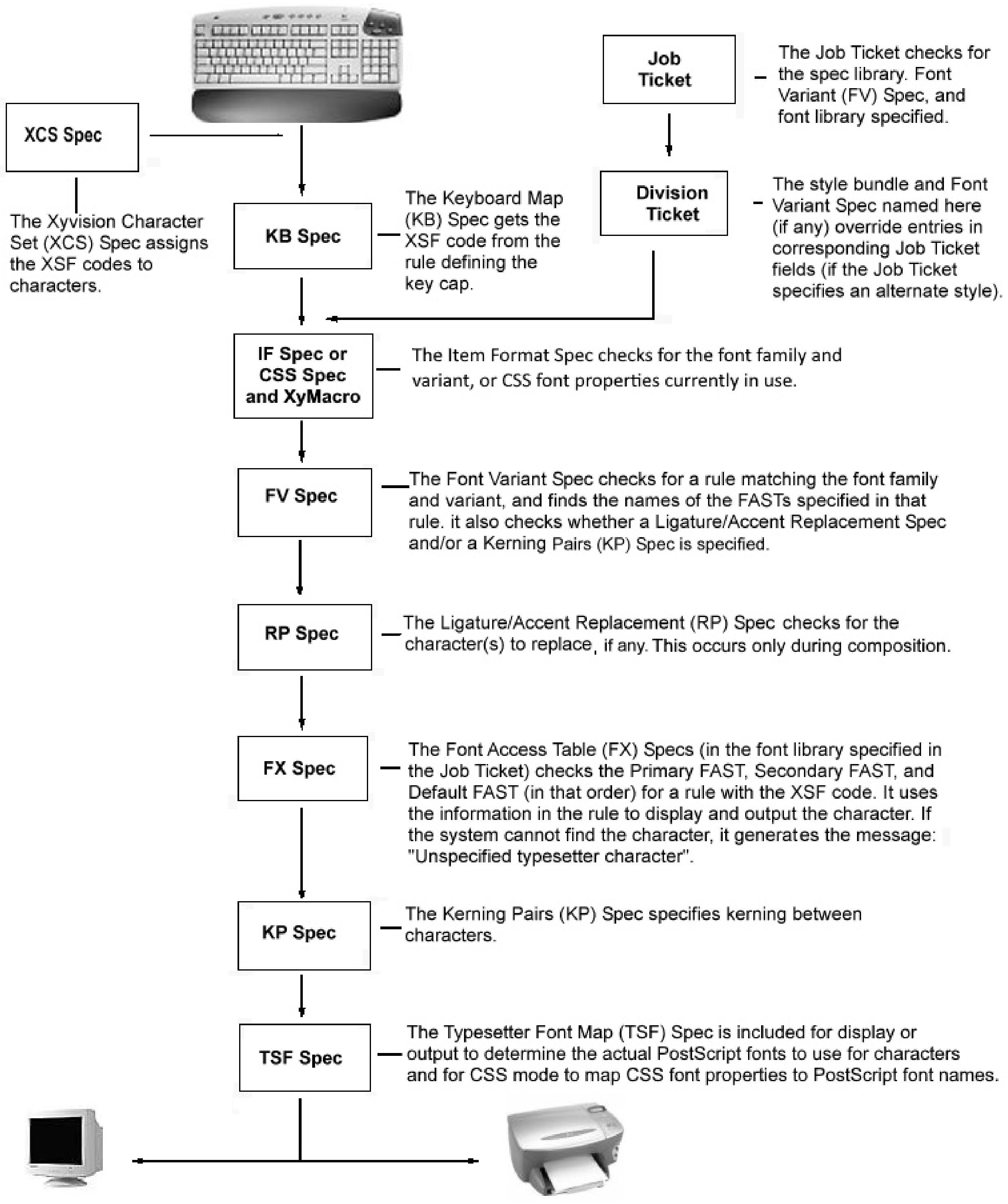

XCS Spec
The Xyvision Character Set (XCS) Spec assigns the XSF codes to characters.
KB Spec
The Keyboard Map (KB) Spec gets the XSF code from the rule defining the key cap.
Job Ticket
The Job Ticket checks for the spec library. Font Variant (FV) Spec, and font library specified.
Division Ticket
The style bundle and Font Variant Spec named here (if any) override entries in corresponding Job Ticket fields (if the Job Ticket specifies an alternate style).
IF Spec or CSS Spec and XyMacro
The Item Format Spec checks for the font family and variant, or CSS font properties currently in use.
FV Spec
The Font Variant Spec checks for a rule matching the font family and variant, and finds the names of the FASTs specified in that rule. it also checks whether a Ligature/Accent Replacement Spec and/or a Kerning Pairs (KP) Spec is specified.
RP Spec
The Ligature/Accent Replacement (RP) Spec checks for the character(s) to replace, if any. This occurs only during composition.
FX Spec
The Font Access Table (FX) Specs (in the font library specified in the Job Ticket) checks the Primary FAST, Secondary FAST, and Default FAST (in that order) for a rule with the XSF code. It uses the information in the rule to display and output the character. If the system cannot find the character, it generates the message: "Unspecified typesetter character".
KP Spec
The Kerning Pairs (KP) Spec specifies kerning between characters.
TSF Spec
The Typesetter Font Map (TSF) Spec is included for display or output to determine the actual PostScript fonts to use for characters and for CSS mode to map CSS font properties to PostScript font names.

Figure 2-2 *Displaying and Outputting Fonts*

When Do I Need to Edit Font Specs?

You need to set up font specs on your system if you:

- Acquire additional fonts.
- Want to customize characters on output.

XPP provides two utilities to facilitate adding fonts to your composition system: Font Copy and Build FAST.

In many cases, there is very little additional spec editing that you need to do. However, if you do make changes to a spec, RWS strongly suggests that you create a backup copy first.

The use of symbols or other special characters requires some specific setup in order to optimize the composition capabilities of XPP. Both situations are addressed later in this manual.

How Do I Create New Font Specs?

You can create new specs for FAST Generation (FGS), FAST Generation Exceptions (FGX), Font Variant (FV), Kerning Pairs (KP), PseudoFont (PSF), PostScript Name (PSN), Phototypesetter (PTS), and Typesetter Font Map (TSF).

XPP delivers the Xyvision Character Set and XPP only recognizes the spec named *default*. XPP also delivers the Ligature/Accent Replacement Spec (RP) and XPP only recognizes the one delivered to Lsyslib. Do not attempt to create, copy, or rename this spec.

To create a new Keyboard Map Spec (KB), copy an existing spec. Refer to “Modifying a Keyboard” on page 4-13 for complete information.

The Build FAST utility generates the Phototypesetter Spec (PTS) and corresponding FAST Generation Spec (FGS), and Kerning Pairs Spec (KP) (if the source font contains kerning pair data). Refer to “Creating Font Specs with Build FAST” on page 1-10 for complete information.

To create new specs for FGX, PSF, and PSN Specs:

1. Double-click **STYLE LIBRARIES** from the PathFinder Tree View.
   PathFinder displays the available libraries.
2. Right-click **L*fontlib*** (e.g., Lnoto).
   PathFinder displays a pop-up menu.
3. Select **New > Type of spec you want to create** (e.g., FAST Generation Exceptions).

PathFinder creates a FAST Generation Exceptions style file (if one has not already been created) and displays a *new* spec in the List View.

Unicode Capabilities

XPP has Unicode capabilities. In the case of fonts, this means that XPP can use the W3C (World Wide Web Consortium) character tables to determine the case of a character. For example, if you are using a language, such as Greek, composition allows you to use smallcaps, all upper case, all lower case, or mixed case transformations of the text.

Chapter 3

The Xyvision Character Set Spec (XCS)

This chapter contains the following information on the Xyvision Character Set (XCS) Spec:

- Understanding the XCS Spec
- Learning about the files generated from the XCS Spec
- Obtaining updates to the Standard XCS Spec
- Modifying the XCS Spec
- Understanding the structure of the XCS Spec

Understanding the XCS Spec

The XCS Spec contains the definitions of more than 3,000 characters (including some XPP-specific characters). For each character, XPP has assigned a unique Xyvision Character Set (XCS) number and many characters have Unicode values and character entity strings assigned to them. XPP used to identify and access all characters by these XCS numbers, but now it uses Unicode values and named character entity or numeric character reference strings to identify and access all characters.

If a character in the XCS spec has no Unicode value assigned to it, the implied (custom) Unicode value assignment is taken as the assigned XCS number added to 0xF0000 (which results in a Unicode value in the Supplementary Private Use Area-A). Not all characters you enter into a division document will have a corresponding entry in the XCS spec.

For a named character entity string to be recognized by core XPP, that entity needs to be assigned to a character in the XCS spec.

In the XCS Spec, each XCS number is associated with the following:

- A name.
- A unique ASCII character sequence. The system uses the ASCII sequence when converting files to and from Xyvision Standard Format (only when using Classic mode).
- An XML/SGML character entity string (optional). Required if a character is going to be accessed using a named character entity string. In some cases, more than one entity string can be entered for a character.
- A Unicode value (optional). In some cases, more than one Unicode value can be entered for a character.
- Characters for the system to use to display XPP-specific characters (e.g., a pgraf character in the Line Edit window and in spec fields).
- A description (optional) of the character.

When you enter a character in a division, a Unicode number is generated. Ordinarily, you would not be aware of the underlying Unicode number; instead, you see the “glyph” or visual representation of the character.

The currently active font family and font variant are mapped to a particular FAST (Font Access Table), which determines the glyph that the system selects and places in the division. The font family and variant mappings are controlled by the active rule in the Font Variant Spec (as specified in the Family and Variant fields of the tag or by the Font Family (*ff*) and Font Variant (*fv*) XyMacros, or the active CSS font-family, font-style, and font-weight properties).

When to View the XCS Spec

There are three common reasons to view or edit the XCS Spec:

- To determine the Unicode number that is assigned to a character.
- To determine the ASCII escape sequence (when using Classic mode) or named character entity of a particular character. This information helps you find out how to enter the named character when doing a transformation.
- To modify or assign new named character entity strings.

Accessing the XCS Spec from PathFinder

To access the XCS Spec from the PathFinder:

1. Navigate to **STYLE LIBRARIES > Lsyslib > Xyvision Character Set**. PathFinder displays the *default* character set file in the List View.
2. Right-click the *default* **icon** in the List View and select **Edit** from the pop-up menu.
   —or—
   Double-click the *default* **icon**.
   XPP displays the XCS Spec.

Caution

If you need to modify the spec to implement conversion to and from XML/SGML, change only the Default Output, Entity Delimiters, and Entity fields as needed.

Modifying any other fields in the spec can cause problems in displaying text.

Accessing the XCS Spec from the Operating System

To access the XCS Spec from your UNIX or Windows file system:

1. Access the XYV_STYLES/Lsyslib directory.
2. Use either of the following options:

   In Windows Explorer, double-click **_xcs_default.sde**

   —or—

   From the command line, type the following command at the operating system prompt:

   sdedit xcs default

Note: *To view rather than edit the spec, use the* **–r** *switch after the Sdedit program name.*

Learning About the Files Generated from the XCS Spec

This section provides background information and is included only to help you understand how the XCS Spec works. You should rarely, if ever, need to view or edit these files.

The following table lists the files generated from the XCS Spec, the XCS Spec fields used to generate each file, and how the system uses each file. These files are in XYV_STYLES/Lsyslib.

Table 3-1 *Files Generated from the XCS Spec*

File	*File Name*	*XCS Spec field*	*The file is used to...*
ASCII to XSF	_a2x_default.p	Ascii	Perform ToXSF and Importxsf (Classic mode only)[1]
Unicode to XSF	_u2x_default.p	Unicode	Perform ToXSF and Importxsf (SGML and XML/CSS modes)
XSF to ASCII	_x2a_default.p	Ascii, Entity, Unicode	Perform FromXSF, Xsfchange, Showxsf, etc.[2]
XSF to Terminal	_x2t_default.p	Line Ed	Display XPP-specific characters in the Line Edit window and in spec fields
ASCII to XSF	_a2x_default.m	Ascii	Provide information only[3].

[1]ToXSF is the program for converting text from XyASCII to Xyvision Standard Format.

[2]FromXSF is the program for converting text from Xyvision Standard Format to XyASCII.

[3]This file contains a list of the available XyASCII escape sequences, by keyboard (for Classic mode only). The system does not use this file; it is only for informational purposes. You can print it for reference.

Warning

The files generated from the XCS Spec are critical to XPP operation. Do not copy, edit, or delete any of these files. If you should need to check or verify the contents of any of these files, use the ***showxcs*** program.

ASCII to XSF Files

When using Classic mode, the ToXSF and Importxsf programs convert text from XyASCII format to Xyvision Standard Format. XyASCII text files contain ASCII escape sequences for characters that cannot be represented by single ASCII characters (e.g., special characters such as math symbols). The ToXSF program replaces the ASCII escape sequences with the corresponding XPP codes and Unicode characters. Refer to the *XML Professional Publisher: Transformaing Data* for information on the ToXSF program. When using SGML or XML/CSS modes, the ToXSF and Importxsf programs import UTF-8 characters and convert named character entities and numeric character references to Xyvision Standard Format.

XyASCII escape sequences

A XyASCII escape sequence can be a string of one, two, or three characters (for Classic mode only), a named character entity string, or a numeric character reference string (representing a Unicode value):

- A one-character string consists of the character itself. For example, a capital “A” is a valid ASCII escape sequence.
- A two-character string consists of a backslash (\) followed by a single character (Classic mode only). Characters with this type of XyASCII escape sequence appear on the Default keyboard (keyboard d).
- A three-character string consists of a vertical bar (|) followed by a two-character string (Classic Mode only). The first character in the string represents the keyboard on which the character appears (e.g., c for the Chemical Arrows keyboard). The second character represents the key cap that you press to access the character.
- A named character entity string consists of an ASCII string beginning with an ampersand (&) and ending with a semicolon (;)—but, you do not enter the & and ; in the *Entity* field.
- A Unicode value can be expressed as a decimal (d32 through d1114109) or a hex (x20 through x10FFFD) in the *Unicode* field. Each Unicode value can map to one and only one XCS character. However, several different Unicode values can map to the same XCS character.

In Classic mode, if the system finds a backslash or a vertical bar when converting an ASCII text file to Xyvision Standard Format, it interprets the subsequent characters as the XyASCII string. Next, it reads the _a2x_default.p file and finds the matching XCS number and (first) Unicode value (which may be calculated from the XCS number if a Unicode value is absent). In SGML and XML/CSS modes, it reads the _a2x_default.p file to look up named character entity strings to get the Unicode value. Finally, the system uses the Unicode value to access the character glyph for screen display and printer output. During composition it uses the Unicode value to access the necessary character information (e.g., the character width).

Note: *You do not ever actually see the XCS numbers in a document or a file.*

XSF to ASCII File

The system uses the XSF to ASCII file _x2a_default.p when running the programs which convert XSF files from Xyvision Standard Format to XyASCII (for example, FromXSF). When processing the XSF file, the system reads the _x2a_default.p file and replaces the Unicode value with the corresponding XyASCII sequence (Classic mode only), named character entity string, or UTF-8 character.

Refer to the *XML Professional Publisher: Transformaing Data* manual and to the online help files for information on the programs that convert XSF to XyASCII format.

XSF to Terminal File

System fonts cannot by default represent XPP-specific characters in the Line Edit Window or in spec fields (e.g. the pgraf character). In these cases, the system uses the information in the XSF to Terminal file _x2t_default.p to determine what character to use from the XPP-installed system font.. Do not copy, edit, or delete this file.

This file contains information such as the contents of the *Line Ed* field from the XCS Spec. In Classic mode, for XPP-specific characters, the system searches the file and displays the *Line Ed* string defined for that character.

Obtaining Updates to the Standard XCS Spec

When you upgrade software, the program delivers the latest XCS Spec to XYV_STYLES/xylibrary/Lsyslib.

Copy the latest XCS Spec to the directory where the system can use it, that is, copy the XCS Spec from **XYV_STYLES/xylibrary/Lsyslib** to **XYV_STYLES/Lsyslib**.

Caution

When you copy the latest XCS Spec, you overwrite the XCS Spec in the syslib library. Save any edited rules according to the following instructions before copying the spec. As an alternative, you can check for custom changes to the spec using the **upgradexcs.pl** or **update_xcs.pl** utilities to help update your XCS spec. For more information, see desciptions of these utilities in *XML Professional Publisher: Command Line Utilities*.

Copying an XCS Spec

To copy the latest version of the XCS Spec:

1. If you edited the XCS Spec, save the edited rules *before* copying the latest XCS Spec.

   To save these rules:

   - Access the Spec
   - Select the rules
   - Save the rules
     - If the rules are contiguous, you can save several rules in one buffer (i.e., use the *Save* and *Restore* options on the Softkey menu).
     - If you selected rules that are not contiguous, you can use the Copy Append function from the Edit menu on the Sdedit Tool bar (i.e., save the selected rules to a cut buffer).

2. Copy the file **_xcs_default.sde** from *XYV_STYLES/xylibrary/Lsyslib/* to *XYV_STYLES/Lsyslib/*

3. If you saved edited XCS Spec rules to save buffers, restore them to the Spec in XYV_STYLES/Lsyslib/.

   a. Access the XCS Spec in XYV_STYLES/Lsyslib.

   b. Use the Restore function on the Softkey menu. (Menu > Select > Restore) to restore the edited versions you saved in the save buffers.
   If you used the Copy Append function from the Edit menu on the Sdedit Tool bar, paste the rules into the XCS Spec using the Edit menu on the Sdedit Tool bar.

The spec now contains both the edited versions of those rules *and* the newly delivered versions of those rules.

c. Select and delete the newly delivered versions of the rules. The XCS Spec now contains the latest updated information plus any edits you may have made.

4. Run GenXCS to access the information in the latest XCS Spec. (Refer to page 3-11 for details.)

Updating the XCS Spec

To update the current XCS spec after an upgrade:

- If there have been few (or no) XCS spec customizations, first run the **upgradexcs.pl** utility at a command prompt.
- If there have been extensive XCS spec customizations, run the **update_xcs.pl** utility.

  For more information about these utilities, refer to *XML Professional Publisher: Command Line Utilities*.

Note: *You can access full details about the syntax and available options of each utility by entering one of the following at a command prompt:*

perldoc %XYV_EXECS%\procs\sc\*utility* **(Windows)**

perldoc $XYV_EXECS/procs/sc/*utility* **(Linux)**

where *utility* is **upgradexcs.pl** or **update_xcs.pl**.

Modifying the XCS Spec

You seldom need to edit the XCS Spec because XPP has already pre-assigned XyASCII escape sequences (for Classic mode) to all XCS numbers in the XCS Spec, even to custom XCS numbers (1281 - 2559), and the most commonly used Unicode values and named character entity strings have been assigned as well.

Note: *Multiples of 256 are invalid XCS numbers. The Sdeditor does not allow you to enter an invalid XCS code in the XCS field and displays an "Invalid XSF code" message.*

However, you may need to modify the XCS Spec if you need to perform one of the following tasks:

- Change or add a named character entity string or Unicode value.
- Add a new character.

Caution Edits to the XCS Spec may affect previously completed work.

First, refer to *The Xyvision Character Set* to see if the character you want is already in the XCS Spec and is therefore, assigned to a keyboard and key cap.

If you want to re-map a character to a different key cap or different software keyboard, edit a Keyboard Map Spec. Refer to the "The Keyboard Map Spec (KB)" on page 4-1.

Editing or Adding a Named Character Entity String

To add a named character entity string in the XCS Spec:

1. Access the XCS Spec.
2. Locate the rule for the XCS/Unicode character for which you want the XML/SGML named character entity to be recognized.
3. Enter the entity name in the *Entity* field. You may enter multiple entity names by using a space as a delimiter, but the entire *Entity* field may not exceed 92 characters.

   Do not enter the entity delimiter characters; those are defined in the header section of the spec in the *Begin* and *End* fields, usually ampersand (&) for the *Begin* field and semi-colon (;) for the *End* field.
4. Indicate in the *File Comment* or *Table Comment* field which rule(s) has been edited.

5. Run the Generate XCS (GenXCS) program to process and activate your edits.

Editing or Adding a Unicode Value

To add a Unicode value to the XCS Spec:

1. Access the XCS Spec.
2. Locate the rule for the XCS character for which you want the Unicode value to be recognized.
3. Enter the Unicode value in the *Unicode* field. You may enter multiple Unicode values by using a space as a delimiter, but the entire *Unicode* field may not exceed 92 characters.
4. Indicate in the *File Comment* or *Table Comment* field which rule(s) has been edited.
5. Run the Generate XCS (GenXCS) program to process and activate your edits.

Assigning a Custom Character

To add a new character:

1. Optionally, edit the *Name, Description, Entity,* and *Unicode* fields in an XCS Spec rule in the XCS number range 1281 - 2559 (excluding multiples of 256, which are invalid) for the new character.
2. If you edited the XCS file, run the Generate XCS (GenXCS) program to process and activate your edits.
3. Add the new Unicode value to all necessary PTS, PSF, and/or FGX Specs, then run GenFAST as you would for any other font spec change.
4. If you want to remap the new character to a different keycap, edit a Keyboard Map Spec. See ″The Keyboard Map Spec (KB)″ on page 4-1.
5. Indicate in the *File Comment* or *Table Comment* field which rule(s) has been edited.

Editing the Name and Description Fields

To edit the *Name* or *Description* field in the XCS Spec:

1. Access the XCS Spec.

2. Locate the desired rule in the XCS number range reserved for custom characters — 1281 through 2559 (excluding multiples of 256).
3. Edit the *Name* and *Description* fields for the character you are adding, and optionally the *Entity* and *Unicode* fields. You do not need to edit the *Line Ed* field. **Do not edit the** *XCS No., or Ascii* **fields.** Refer to the section "XCS Rule Fields" for information on the valid entries for the rule fields.
4. Indicate in the *File Comment* or *Table Comment* field which rule(s) has been edited.
5. Store the XCS Spec.
6. Run the Generate XCS (GenXCS) program to process and activate your edits.

Caution Do not modify any rules except those for XCS numbers 1281 - 2559 (excluding multiples of 256). If you edit fields in other rules, you may not be able to access those characters.

Running GenXCS

For the system to access the edits you made to an XCS Spec, you must run GenXCS (Generate XCS).

To run GenXCS:

1. In PathFinder, right-click the **default** XCS Spec.
2. Select **Tools > Generate XCS** from the pop-up menu.
3. Click the **OK** button when GenXCS displays a message box indicating that the GenXCS process is complete.

When you run GenXCS, the system creates five files, as listed in the table on page 3-4, These files contain information the system needs to run ToXSF, FromXSF, etc. and to display characters in the Line Edit window and spec fields. Consult this table to familiarize yourself with the files.

Note: *You can also run GenXCS from the operating system command line. For information, consult the XML Professional Publisher: Command Line Utilities manual or you can enter* **xyhelp genxcs** *at the operating system prompt to access the online help file.*

Locating Unused XyASCII Sequences

XPP delivers an _a2x_default.m file to XYV_STYLES/Lsyslib along with the delivered XCS files. This file contains a map of all the unused XyASCII sequences (used for Classic mode only). Once you have modified the XCS file, you can generate a new _a2x_default.m file.

To locate unused XyASCII sequences:

- Run **genxcs –m** from the command line.
  This command generates a new _a2x_default.m file.

You can select an unused XyASCII sequence and add it to any new characters you have added/defined in the spec.

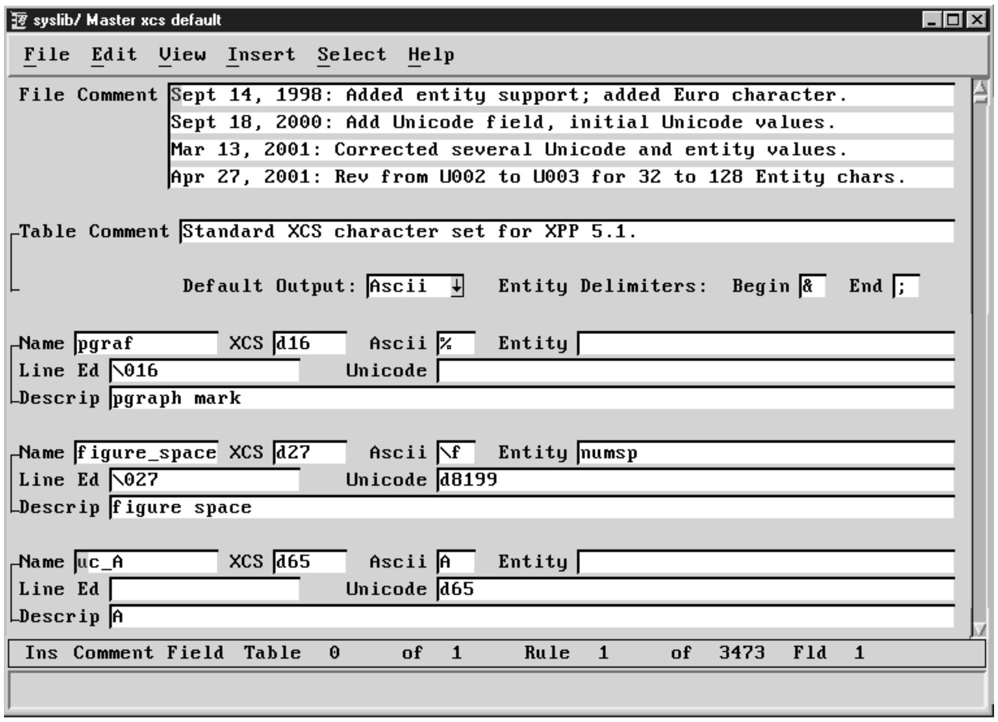
Understanding the Structure of the XCS Spec

The XCS Spec default name is *_xcs_default.sde* and must be located in the *syslib* library. This is the only XCS Spec name that XPP recognizes, though you may have copies with different names.

The XCS Spec consists of the following sections:

- Header — contains comment, default output, and entity delimiter fields.
- Rules — contains rules defining characters.

The following figure shows the structure of the XCS Spec and sample rules which define various types of characters. For example, the rules show XPP-specific characters (e.g., the pgraf), space characters (e.g., the figure space), and uppercase alphabetic characters (e.g., A).



```
File Comment  Sept 14, 1998: Added entity support; added Euro character.
              Sept 18, 2000: Add Unicode field, initial Unicode values.
              Mar 13, 2001: Corrected several Unicode and entity values.
              Apr 27, 2001: Rev from U002 to U003 for 32 to 128 Entity chars.

Table Comment Standard XCS character set for XPP 5.1.

              Default Output: Ascii     Entity Delimiters:  Begin &   End ;

Name    pgraf         XCS d16     Ascii %     Entity
Line Ed \016                      Unicode
Descrip pgraph mark

Name    figure_space  XCS d27     Ascii \f    Entity numsp
Line Ed \027                      Unicode d8199
Descrip figure space

Name    uc_A          XCS d65     Ascii A     Entity
Line Ed                           Unicode d65
Descrip A

Ins Comment Field  Table  0    of  1    Rule  1    of  3473  Fld  1
```

XCS Spec Fields

Following are descriptions of the fields in the XCS Spec and the values you can enter in them.

Header Fields

The header of the XCS Spec contains the following fields:

Table 3-2 *XCS Spec Header Fields*

Field Name	*Valid Entry*	*Description*
File comment	Ascii text	Generally used to store comments about the edit history of the file.
Table Comment	Ascii text	Generally used to describe the contents of the body rules.
Default Output	When exporting from XPP, use ...	
	Ascii	To output XyASCII escape sequences and ignore the *Entity* field. This is the default.
	Entity	To output XML/SGML named character entities when they are defined.
	Unidec	To produce the first value of the Unicode field, if present, as a decimal numeric character reference. Outputs a decimal character in the format: &#8211;.
	Unihex	To produce the first value of the Unicode field, if present, as a hex numeric character reference. Outputs a hex character in the format: &#x2811;.
	Note: This field value is often overridden by options used during export.	
Entity Delimiters:		
Begin	Ascii character	Enter the character that marks the beginning of an XML/SGML entity. There is no default for this field. The ampersand (&) character is the most commonly used in XML/SGML.
End	Ascii character	Enter the character that marks the end of an XML/SGML entity. There is no default for this field. The semi-colon (;) character is the most commonly used in XML/SGML.

Rule Fields

The XCS Spec contains a rule for each character. RWS recommends that if you modify the _xcs_default.sde Spec at all, only modify the *Name, Entity, Unicode,* and *Description* fields.

Table 3-3 *XCS Spec Rule Fields*

Field Name	*Valid Entry*	*Description*
Name	Briefly describes the character defined in this rule. Rules for XCS numbers 1281 - 2559 (excluding multiples of 256) contain the entry **custom**. You may want to delete this entry and enter the name of the character. GenFAST puts this name into the Character Comment field of the FAST Spec (truncating it at a space, if there is one).	
	string	A unique name consisting of up to 12 alphanumeric characters. Do not insert spaces between characters; enter an underbar instead of a space.
XCS No.	The Xyvision Standard Format (XSF) code assigned to the character defined in this rule, preceded by a letter denoting the notation. This field can contain XCS numbers in the decimal ranges d23 through d6143 (octal o27 through o13777 or hexadecimal x17 through x17FF). Typically, decimal notation is used. Do not modify the XCS No. field. To add a character to the XCS Spec, locate a rule in the XCS number range 1281 - 2559 (excluding multiples of 256). Each XCS number can appear only once in the XCS Spec. Ranges of XCS numbers (in decimal) are reserved for certain types of characters:	
	XSF Code Range	**Reserved for...**
	1 - 1280	XPP-defined characters. This range includes characters for system use (d1 - d22)
	1281- 2559	User-defined characters
	2561- 6143	XPP-defined characters
	Add user-defined characters, such as company logos, to the rules for XCS numbers 1281 through 2559 (excluding multiples of 256). XPP does not routinely deliver screen glyphs for these XCS numbers. Unless you have acquired a PostScript font glyph for these characters, XPP displays reverse-video question marks onscreen and a blank space on output. XPP uses the specified character widths during composition.	

Table 3-3 *XCS Spec Rule Fields (Continued)*

Field Name	*Valid Entry* *Description*
Ascii	The ASCII character (e.g., a, 1, #, and so on) or the XyASCII escape sequence (used in Classic mode only) representing the character defined in this rule. The system uses the information in this field when converting files to and from Xyvision Standard Format (i.e., when running the ToXSF or FromXSF programs) when the *Default Output* field is set to *Ascii*. If the system does not have a screen glyph of a character and the *Line Ed* field does not contain an entry, the system displays the contents of this field to represent the character in the Line Edit window, in Text Mode, and in spec fields. In some cases, when both the *Ascii and Line Ed fields are blank or an entered Unicode value does not have an XCS spec entry or does not have a glyph in the specified system fonts, XPP may display a character as a small box with the hex Unicode value displayed within it.* *XPP has assigned an ASCII character or XyASCII escape sequence to each XCS number in the user-definable range 1281- 2559 (excluding multiples of 256).* **Do not change these.** To add a character, choose a rule in the user-definable range. If the XCS Spec contains rules with duplicate XyASCII sequences, GenXCS does not run successfully.To locate an unused XyASCII sequence, refer to page 3-12. A XyASCII escape sequence can be a one-character, two-character, or three-character string. If you want to key in some characters using a key sequence different from the one listed in the XCS Spec, you do not need to change their XyASCII escape sequences in the XCS Spec—instead, modify a keyboard spec. Refer to the *"The Keyboard Map Spec (KB)"* on page 4-1.
Entity	Enter the name(s) of an XML/SGML named character entity. You may enter one or more entity names delimited by spaces. The maximum *Entity* field length is 92 characters. This number includes all names and delimiting spaces. The maximum number of characters for each entity name is 32. Enter only the entity name(s) itself without the delimiters defined in the header fields *Begin* and *End*. This is an optional field, that is, you only use it when you have an XML/SGML named character entity that you want recognized as a single XCS/Unicode character. When transforming for output in Classic mode, XPP only uses a value in this field when the *Default Output* field is set to *Entity* or you use the –ent switch when running from the command line. Otherwise the value is taken from the *Ascii* field.
Line Ed	This field is used to specify how to display XPP-specific characters (e.g. a pgraf) in the Line Editor and spec fields. Do not edit this field.

Table 3-3 *XCS Spec Rule Fields (Continued)*

Field Name	*Valid Entry*	*Description*
Unicode	Use this field to specify Unicode values for the characters in the XCS Spec. You can enter a maximum of 92 characters which includes zero, one, or multiple entries that are space separated. Unicode values can be expressed as either a decimal value (from d32-d1114109) or a hexadecimal value (from x20-x10fffd). Each Unicode value can map to one and only one XCS character. However, several different Unicode values can map to the same XCS character.	
Description	Enter a brief comment describing the character defined in this rule.	
	string	A comment up to 80 characters long, including uppercase and lowercase characters, spaces, symbols (such as $, &, /) and the integers 0-9).

Print XCS Layout

To print the XCS layout from PathFinder:

1. Navigate to **STYLE LIBRARIES > Lsyslib > Xyvision Character Set**.
   PathFinder displays the XCS *default* Spec in the List View.

2. Right-click **default**.
   PathFinder displays a pop-up menu.

3. Selec **Tools > Print Character Layout**.
   XPP displays the Print Keyboards utility window, Task Select.

4. In the Keyboard/Character Font Maps section, select the following sequence: **Generate Layouts > OK > Character Set > OK.**
   The Print Keyboards utility displays the following messages:
   Processing Keyboard ...
   Processing Xyvision Character Set layout.
   The utility displays a second message box stating that the process is complete and provides the location of the newly created division.
   The message box also asks if you want to examine the division?
   - Click **Cancel** to close the window and stop the process.
   - Click **No** and the Print Utility gives you an opportunity to select another task.
   - Click **Yes** and XPP opens the division in the XyView.

   When you close the division, the Print Keyboards utility offers you the opportunity to select another task.

For additional information on the Print Keyboards utility, refer to ″Print a Keyboard Mapping″ on page 4-14.

Note: *XPP prints only characters for which a PostScript font is available on your system.*

Chapter 4

The Keyboard Map Spec (KB)

This chapter contains the following information on the Keyboard Map (KB) Spec:

- About Keyboards and key caps
- Understanding the Keyboard (KB) Spec
- Modifying KB Specs
- Updating KB Specs

About Keyboards and Key Caps

To input characters in a division, you typically access them using the keyboard. KB Specs map Unicode numbers or character strings to keys.

Using XPP, you can access *Standard* and *alternate* keyboards. These are software keyboards. The *Standard* keyboard maps characters to the keys on an actual U.S. keyboard and the *alternate* keyboards allow you to access characters not printed on the keys.

The *Standard keyboard* consists of characters such as the lowercase and uppercase English alphabet and the Arabic numerals 0 through 9. This keyboard is active when you open a division.

There are more characters available than there are key caps on a keyboard. Additional characters are mapped to key caps on *alternate keyboards* — keyboards other than the Standard keyboard. Alternate keyboards typically contain logically grouped *Pi* (non-alphanumeric) characters, for example, the Arrows keyboard (keyboard a) and the Bullets/Circles keyboard (keyboard b).

The name that displays in the *Keybd* field of the XyView Status area while the keyboard is active comes from the name field of the KB Spec. Refer to the section "Structure of the KB Spec" on page 4-7 for information on the valid entries for this field.

Note: *Although XPP maps characters to software keyboards, you may not be able to access a given character unless you have installed the appropriate PostScript font on your XPP server.*

Standard and Alternate Keyboards

Characters are mapped to keyboards named by a single alphabetic or numeric character (a-z, 0-9) or by 2-leter names indicating uppercase alphabetic characters (e.g., file kb_uh is keyboard H). XPP delivers the Standard keyboard (keyboard 0), keyboard 1, and many alternate keyboard specs, as shown in Table 4-1.

Many of the keys on keyboards h, i, j, k, l, o, p, H, I, J, and K are assigned to custom Unicode numbers, and therefore do not have characters mapped to them. You can edit the KB Specs for these keyboards to map custom characters or strings to the key caps.

Keyboards 3 through 9 are reserved for customer use. RWS reserves the right to deliver all other keyboards.

The following table identifies the XPP-delivered keyboards and provides sample characters for each keyboard.

Table 4-1 *XPP-delivered Keyboards*

Keyboard ID (filename)	*Keyboard Name*	*Sample Characters*
0[1]	Standard2	a B 2 5 # & / @
1[2]	Pi	§ ƀ ® ‗ Ā ℠[5]
2	4.2 math macros	**⟪sd⟫ ⟪ed⟫ ⟪intg⟫**
a	Arrows	↗ ↻ ↰ ↨[5] ↓
A (ua)	Accents	ˆ ˘ ¸ ¯ ` ˚ /[5]
b	Bullets/Circles	Ⓢ ○ ●[5]
B (ub)	Boxes/Squares	■ 🅂[5]
c	Chemical Arrows	↔ ⇇ ⇊ ⤢ ← ⇌ ⇄ ⇅[5]
C (uc)	Chemical bonds	⑊ ═ ⬭ ↻ ⇒ ⫴ ⇚ ⇔[5]
d	Default	> ﬄ × ¶ § ‹ œ ÷[5]
D (ud)	Accents/Chars 2	ý Ý[5]
e	Math Option	ψ ξ ϕ ς s $ Q + − x[5]
E (ue)	Accents/Chars 3	ÿ Ÿ š Š ž Ž[5]
f	Accents/Chars 1	ð ï á æ é ç ú[5]
F (uf)	Math Segments	⎩ ⎨ ⎝ ∏[55]
g	Greek Alphabet	α ζ τ δ ϕ π ι λ[5]
G (ug)	Greek Text	NA[3]
L (ul)	Lesser/Greater	⋜ ⌟ ⊓[5]
m	Equals/Similars	≅ ∴[5]
M (um)	Math Symbols	∑ *h* ∞ ⌜ ∅ ∱[5]
n	Subsets/Angles	⫫ ∠[55]
N (un)	Math/Multi-line	⌜ ∫ Γ[5]
O (uo)	Ornament Keycap	NA[3]
P (up[2])	Punct/Symbol	§ ƀ ® ‗ Ā ℠[5]
q	Dingbats 200	NA[3]
Q (uq)	Dingbats 300	NA[3]

Table 4-1 *XPP-delivered Keyboards (Continued)*

Keyboard ID (filename)	*Keyboard Name*	*Sample Characters*
r	Rules - Borders	NA[3]
R (ur)	Cyrillic-Text	NA[3]
s	Shapes	♦ ⩚ ♥ ⚠ ⧓ ∇[5]
S (us)	Symbols	m̸h ☎ ↯ ₽ :: m̳[5]
t	Greek Piece Accents (part 1)	NA[3]
T (ut)	Greek Piece Accents (part 2)	NA[3]
u	XML/SGML Entities	NA[4]
U (uu)	XML/SGML Entities	NA[4]
w	Hebrew Text	NA[3]
x	JIS Symbols	Japanese characters [5]
y	Hiragana	Japanese characters
Y (uy)[5]	Katakana	Japanese characters[5]
z	Zodiac/Weather	⏝ ⏝ ⊖ ♏[5]
Z (uz)	Dingbats 100	① ② ③ ④[5]

[1]Do not edit the Standard keyboard (keyboard 0).

[2]XPP delivers the same characters on the Punct/Symbol keyboard P and keyboard 1. Keyboard 1 is also mapped to the [ALT KYBD] key and the PI key for easy access.

[3]NA = Not Available. Characters on this keyboard are not part of the standard font delivery. You can access them only if you have purchased the appropriate fonts.

[4]Added for keyboard access to XML/SGML entities.

[5]Some characters on this keyboard are not part of the standard font delivery. You can access them only if you have purchased the appropriate fonts.

Displaying Characters

In the Line Edit window, the system displays the contents of the *Ascii* field (i.e., the XyASCII escape sequence for the character) in the XCS Spec if a Unicode character has a XyASCII escape sequence defined in the XCS Spec and the Unicode value is in the Private Use Area range or the specified system fonts do not contain a glyph for the character.

Generally, the system displays characters with one- or two-character escape sequences. Characters from the default keyboard (keyboard D) have two-character XyASCII escape sequences. The system has screen glyphs (from the XYmono character set) to represent most of these characters in the Line Edit window.

To display three-character escape sequences, the system displays the vertical bar (|) as four dots (::). The remaining two characters of the escape sequence follow the four dots.

The first character after the four dots represents the keyboard on which the character appears. The second character represents the legend (i.e., key cap) to which the character is mapped. For example, ::B5 = |B5 and means the character is on keyboard B (Boxes/Squares keyboard), key cap number 5.

An accent indicator symbol, ↟, in the Line Editor indicates a 2-piece accented character: first the accent, then the base. For example, ↟`'e` indicates a lowercase acute accent over a lowercase e, whereas ↟::`AAE` indicates an uppercase acute accent (::AA) over an uppercase E.

Two-piece accented characters may be keyed in the XyView using the F2 (Accent) key. If accented characters exist in the imported XyASCII file, they are preceded by |$$ which indicates to float (center) the accent over the base. The accent indicator symbol, ↟, is not present with one-piece accented PostScript characters, nor is it present if non-spacing marks are enabled, in which case, the base character is just followed by its Unicode non-spacing accent character(s).

For information on entering a two-piece accented character, refer to Chapter 15 in the *XML Professional Publisher: User Guide*.

Any up and down arrows in the Line Editor indicate that composition has replaced the input characters with a modified representation, usually due to RP Spec entries. ″The Ligature/Accent Replacement Spec (RP)″, on page 16-1, explains how to display information in the Line Edit window to show ligature and accent replacement.

Difficulty Displaying Characters

If a PostScript character is not available, composition displays a reverse-video question mark on the screen in the page display. There are two instances in which this may happen:

- You enter a character that is mapped to a custom Unicode number (xF0501-xF0856), but the font you are using does not contain the character.
- You enter a character mapped to a standard Unicode number, but that character is not included in any active FAST. Composition reports an ″Unspecified typesetter character″ error message.

- You can right-click the character in the Xyview to see the unspecified typesetter character’s Unicode, entity, escape sequence, and other character values.

If you press a key to which no character is mapped, the system beeps and does not enter any character.

The Keyboard Spec (KB)

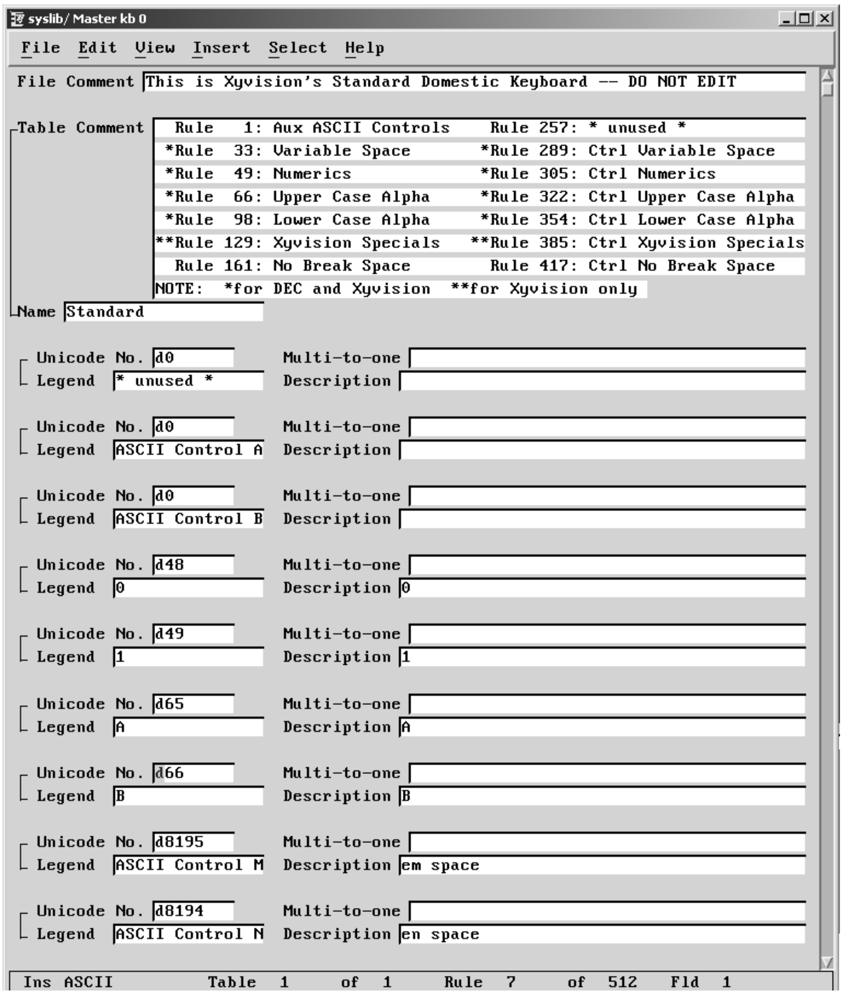
The following section describes the structure of the KB Spec and the organization of its rules.

Structure of the KB Spec

The KB Spec consists of the following sections:

- Header — contains comment and name fields.
- Rules — contain rules mapping characters to legends (i.e., key caps).

The following figure shows the structure of the KB Spec and sample rules. The spec shown is for the Standard keyboard (0). The figure shows rules from various parts of the spec (i.e., the rules in the figure are not contiguous).



```
File Comment  This is Xyvision's Standard Domestic Keyboard -- DO NOT EDIT

Table Comment   Rule   1: Aux ASCII Controls     Rule 257: * unused *
               *Rule  33: Variable Space        *Rule 289: Ctrl Variable Space
               *Rule  49: Numerics              *Rule 305: Ctrl Numerics
               *Rule  66: Upper Case Alpha      *Rule 322: Ctrl Upper Case Alpha
               *Rule  98: Lower Case Alpha      *Rule 354: Ctrl Lower Case Alpha
              **Rule 129: Xyvision Specials    **Rule 385: Ctrl Xyvision Specials
                Rule 161: No Break Space         Rule 417: Ctrl No Break Space
              NOTE:  *for DEC and Xyvision  **for Xyvision only
Name Standard

  Unicode No. d0               Multi-to-one
  Legend  * unused *           Description

  Unicode No. d0               Multi-to-one
  Legend  ASCII Control A      Description

  Unicode No. d0               Multi-to-one
  Legend  ASCII Control B      Description

  Unicode No. d48              Multi-to-one
  Legend  0                    Description 0

  Unicode No. d49              Multi-to-one
  Legend  1                    Description 1

  Unicode No. d65              Multi-to-one
  Legend  A                    Description A

  Unicode No. d66              Multi-to-one
  Legend  B                    Description B

  Unicode No. d8195            Multi-to-one
  Legend  ASCII Control M      Description em space

  Unicode No. d8194            Multi-to-one
  Legend  ASCII Control N      Description en space

 Ins ASCII        Table  1   of  1    Rule  7   of  512   Fld  1
```

Figure 4-1 *Keyboard Map Spec*

Header Fields

The header includes two comment fields and the Name field.

Table 4-2 *KB Spec Header Fields*

Field Name	*Valid Entry*	*Description*
Name	Enter the name of the keyboard defined by this spec. This name appears in the *Keybd* field of the XyView Status area when you access this keyboard. RWS recommends entering a name indicating the type of characters on the keyboard. For example, if the keyboard contains symbols commonly used by the pharmacology department, enter **Pharmacology** in this field. When you access this keyboard, Pharmacology appears in the *Keybd* field of the Status window.	
	string	A string, up to 20 characters long, consisting of uppercase and lowercase characters, spaces, symbols (such as $, &, /), the integers 0-9.

Rule Fields

Each rule maps a character to a U.S keyboard legend (i.e., keycap). The characters printed on key caps may vary according to keyboard. To accommodate these differences, the character printed on the key is referred to as the *legend*. Legend names may imply either unshifted or shifted key caps, though not apparent from the name. For example, the legend $ is a shifted keycap on most keyboards ($ is a number 4 + shift).

Table 4-3 *KB Spec Rule Fields*

Field Name	*Valid Entry*	*Description*
Unicode No.	The Unicode number assigned to the character defined in this rule, preceded by a letter denoting the notation (e.g., d for decimal). This field defaults to d0 if there is no character assigned to that position, although octal or hexadecimal notation could also be used.	
	d*integer*	A positive decimal integer in the ranges d32 through d63487, and d63744 through d1114109.
	o*integer*	A positive base 8 (octal) integer in the ranges o40 through o173777, and o174400 through o4177775.
	x*integer*	A positive base 16 (hexadecimal) integer in the ranges x20 through xF7FF, and xF900 through x10FFFD.

Table 4-3 *KB Spec Rule Fields (Continued)*

Field Name	*Valid Entry*	*Description*
Multi-to-one	Enter the character string you want to access using this key cap. The character string can consist of text, XyMacros, tags, and so on. For the system to use the entry in this field, the Unicode number must be zero. To enter a character from an alternate keyboard, use the corresponding XyASCII escape sequence. When you select this key cap from this keyboard while editing a division, the system inserts the character string. If you use a set of XyMacros frequently, you may want to map them to key caps on an alternate keyboard for easy access.	
	string	A string, up to 512 characters long, consisting of uppercase and lowercase characters, spaces, symbols (such as $, &, /), the integers 0-9, XyMacros and tags.
Legend	This field identifies the symbol printed on the U.S. keyboard key cap defined in this rule. XPP delivers the KB Spec with the Legend field of each rule already filled in. You cannot edit this field, nor can you search on it or even place your cursor on it. The KB Spec contains many rules that are not currently used (that is, Unicode numbers are not mapped to the legends).	
Description	Enter a brief comment describing the character defined in this rule.	
	string	A comment up to 80 characters long, including uppercase and lowercase characters, spaces, symbols (such as $, &, /) and the integers 0-9.

Organization of KB Spec Rules

The rules in the KB Spec correspond to the International Organization of Standards (ISO) Latin standard character set. Rules exist for every possible key cap on a U.S. Latin keyboard; however, in XPP, not all rules are filled in.

The rules in each keyboard spec are grouped according to the type of legend: unshifted, shifted, control+, and so forth. Each keyboard spec has the same set of rules; what makes each spec unique is the keyboard letter with which it is associated. For example, the “7” legend represents a different character on different keyboards:

—c keyboard = short right arrow
—b keyboarad = solid bullet #7
—G keyboard = Greek alternate lowercase kappa

The following table shows the rule numbers and the legends or group of legends that are mapped to those rules. This information also appears in the Table Comment field of the KB Spec.

Table 4-4 *KB Spec Rules*

Rule #	*Legend or group of legends*
1-32	ASCII control characters (unused eaxcept for fixed spaces on kb_0)
33	variable space (non-printing character)
34-48	ASCII characters ! ″ # $ % & ′ ( ) * + , - . /
49-58	ASCII characters 0 through 9
59-65	ASCII characters : ; < = > ? @
66-91	ASCII characters A through Z
92-97	ASCII characters [\ ] ^ _ ‘
98-123	ASCII characters a through z
124-127	ASCII characters { ǀ } ~
128	unused
129-137	Unique RWS keys (unused in current XPP revision)
138-160	unused
161	no break space (Mapped to the unbreakable spaceband *usb* XyMacro on kb_0 only.)
162-256	International keys ¡ £ ¥ , etc.
257-288	unused
289	[Control] variable space (Mapped to the unbreakable spaceband *usb* XyMacro on kb_0 only.)
290-304	[Control] ASCII characters ! ″ # $ % & ′ ( ) * + , - . /
305-314	[Control] ASCII characters 0 through 9
315-321	[Control] ASCII characters : ; < = > ? @
322-347	[Control] ASCII characters A through Z
348-353	[Control] ASCII characters [\ ] ^ _ ‘
354-379	[Control] ASCII characters a through z
380-383	[Control] ASCII characters { ǀ } ~
384	unused
385-393	[Control] unique RWS keys (unused in current XPP revision)
394-416	unused

Table 4-4 *KB Spec Rules (Continued)*

Rule #	*Legend or group of legends*
417	[Control] No break space
418-512	[Control]International keys ¡ £ ¥ , etc.

When editing a spec, you can go to a particular rule in the KB Spec by selecting *Search, Find Rule #,* then specifying the number of the rule to which you want to go.

Modifying KB Specs

All XCS numbers that are in the XCS Spec already have XyASCII escape sequences and therefore have keyboard mapping pre-defined. This is true even for the XCS numbers that are reserved for customer use (XCS numbers 1281-2134, which correspond to Unicode numbers xF0501-xF0856).

When Do I Need to Modify KB Specs?

Using KB Specs, you can set up alternate keyboards and map the following to the key caps:

- Unicode numbers that you want to remap so they can be entered using a different keystroke from a more accessible keyboard.
  For example, you may want to use the star set in a dark circle as a bullet. This symbol is on keyboard Z. Moving it to keyboard P (Punctuation/Symbols) makes it easier to access in the XyView.
- Character strings. The string can consist of tags and XyMacros as well as characters.
  For example, you may want to be able enter your company slogan, completely formatted, with one keystroke.

You can either customize existing KB Specs or create new KB Specs. Creating a new KB Spec is useful if the characters you *commonly* enter in divisions appear on various keyboards. You can create one keyboard and edit it to include those characters. RWS recommends that you customize keyboard 1, which is mapped to the [PI#/PI] and [Alt Keyboard] keys, and is therefore easy to access. XPP delivers kb_1 as a duplicate of keyboard P (kb_up), the Punctuation/Symbol Keyboard.

Note: *Edits to KB Specs affect subsequently entered text, not text entered* ***before*** *the edits. For example, if you change the character mapped to a particular key cap, the new character does not appear in previously entered text; it appears in text entered after the change.*

Modifying an Existing KB Spec

To modify an existing KB Spec from PathFinder:

1. In the Tree View, expand **Lsyslib** and select **Keyboard Maps** from the list of available style specs.
2. In the List View, right-click the KB Spec you want to modify and select **Edit** from the pop-up menu.

Caution Do not modify KB_0, the Standard keyboard!

Having a hard copy of the KB Spec is often useful when entering characters. However, because many rules in the KB Spec are not used, it is preferable to print selected portions of the spec, rather than printing the entire spec. (With the spec open in the XyView, select the portion of the spec you want to print, and select **File > Print > Selected** from the Menu bar.)

Creating a New KB Spec

You may need to create a new KB Spec.

To create a new Keyboard Spec:

1. Navigate to **STYLE LIBRARIES > Lsyslib > Keyboard Maps** in the PathFinder Tree View.
   PathFinder displays the available Keyboard Maps in the List View.

2. Right-click **template** in the List view and select **Copy** from the pop-up menu.
   Note whether there are other copies of *template* already in Lsyslib so you know which copy is yours. Any existing copies appear as *templateCOPY#* where # is an integer.

3. Right-click **Lsyslib** in the Tree View and select **Paste** from the pop-up menu.
   PathFinder pastes a copy of the Keyboard Template to Lsyslib and places the List View cursor on it.

4. Right-click your copy of **template** in the List View and select **Rename**. The new name can be up to two characters, consisting of lowercase letters or a number.

   RWS reserves the right to deliver KB Specs 0, 1, 2, a-z, and ua-uz. Therefore, RWS recommends that you copy your customized KB Specs to the range 3-9. For example, kb_7.

Mapping a Character or a String to a Key Cap

You can map characters and character strings to the Unshift, Shift, Control, and Control Shift positions of any key cap on any keyboard. For example, on keyboard 0 (the standard keyboard) the 《**usb**》 XyMacro for an unbreakable spaceband is mapped to the Ctrl Var Space (Var Space = space bar) legend. Thus, when you press Ctrl + spacebar, you get a 《**usb**》 XyMacro.

To map a character or a string of characters to a key cap:

1. Access an existing or new KB Spec, as previously described.

2. Go to the rule for the key cap on which you want to put the character or string of characters. The *Legend* field contains a description of the key cap.

The rules in the KB Spec are grouped by type of key cap. Press [Search], then *Find Rule #* to specify a particular rule.

a. If you are mapping a single character to a key cap, enter the Unicode number for the character you want to map to this key cap and enter a description in the *Description* field.

b. If you are mapping a string of characters to a key cap, enter the string in the *Multi-to-one* field and edit the *Description* field. The Unicode number must be 0 for the *Multi-to-one* field to be used.
   For example, if you wanted to map your company slogan to the alternate keyboard, choose an open position on that keyboard and enter, in the *Multi-to One* field, something like the following:
   ❮ff;2❯❮fv;3❯❮size;13❯Grammy Lu
   Creations**❮ff;6❯❮fv;2❯❮size;11❯**—sewn with love just for you!
   In your division, access the proper keyboard, enter the appropriate keystroke (the keycap to which you mapped your slogan) and you should see
   Grammy Lu Creations*—sewn with love just for you!*

3. Repeat these steps to map additional characters to this keyboard.

4. Store the KB Spec.

You can now enter the character or string of characters by accessing the keyboard and pressing the appropriate key. Refer to Chapter 15 in the *XML Professional Publisher: User Guide* for information on entering characters from alternate keyboards.

Print a Keyboard Mapping

The *Print Keyboard* utility enables you to see your complete character set and all the keyboard mappings. It creates a one page diagram of a keyboard. This diagram is in a division that you can print for easy reference.

The first time you use the utility, you will be prompted to enter values for the fonts used to output titles and labels on your keyboard map printout. You need to specify these using the 5-digit FAST number. The choices you make for printing Keyboards and XCS specs, such as title and label fonts, extraction FASTs, etc., are stored in `xz/procs/util` files `kblib`, `kbsetup`, `xcslib`, and `xcssetup`, and retained for the next time the utility is executed.

The Print Keyboard utility outputs character number data in Unicode hex.

You will also be prompted for extraction FASTs.

To print a keyboard:

1. Right-click the **KB Spec** you want to print and select **Tools > Print**

Keyboard from the pop-up menu.
The Print Keyboard utility displays a Task Select list box with the following options:

Option	*Description*
View Defaults	Displays a text box containing the following information: ● FAST Library for KB layout ● FAST for Labels ● PI FAST for Labels ● FAST for Extractions ● PI FAST for Extractions Displays a message: "Setup and Font Variant Files have been updated." Repeats the same information for the FAST Library for XCS Layouts. Labels refer to the text, such as headings or character labels, that make the keyboard diagram understandable. Extractions refer to the font used to display the non-alphanumeric characters.
Change Defaults	Allows you to change any of the default values for FASTs or *Pi* FASTs for labels or the FASTs or *Pi* FASTS for extractions for the KB layouts or the XCS layouts. Refer to the figure at the end of this procedure.
Generate Layouts	Generates the one page diagram of the selected keyboard and places it in a division that you can view in the XyView.
Exit	Allows you to close the Task Select window and exit the Print Keyboard utility.

2. Select **Generate Layouts** and click the **OK** button.
The Keyboard/Character Font Maps area displays another set of options:

Options	*Description*
One or Multiple Keyboards	Displays a list box containing all the available keyboards. You can select one, or using the Control Key, you can select multiple keyboards.
All Keyboards	Processes all keyboards in the list box.
Character Set	Processes the Xyvision Character Set.
Previous Menu	Returns you to the menu that allows you to view or change defaults, or generate layouts.
Exit	Closes the Task Select list box and exits the Print Keyboard utility.

3. Select **one** of the options listed above and click the **OK** button.
   The Print Keyboard utility displays a message box detailing the progress, "Processing Keyboard ... layout." Then, it displays another message box, "Keyboard layouts completed.", and provides the exact path to the division.

   The divisions that are written out with the characters and keyboard mappings are in the first Document Root Path in:

   `/CLS_xpputils/GRP_system/JOB_keyboards/DIV_`*`keyboardname`*

   Note: *If you select "One or Multiple Keyboards", the keyboard utility displays a Keyboard Select list box, allowing you to select one or many keyboards.*

4. Click the **OK** button in the message box.
   The Print Keyboard utility closes the message box and displays the Task Select list box, allowing you to make another task selection, or exit the utility.

The following figure is a sample keyboard displaying the changes to the default values:

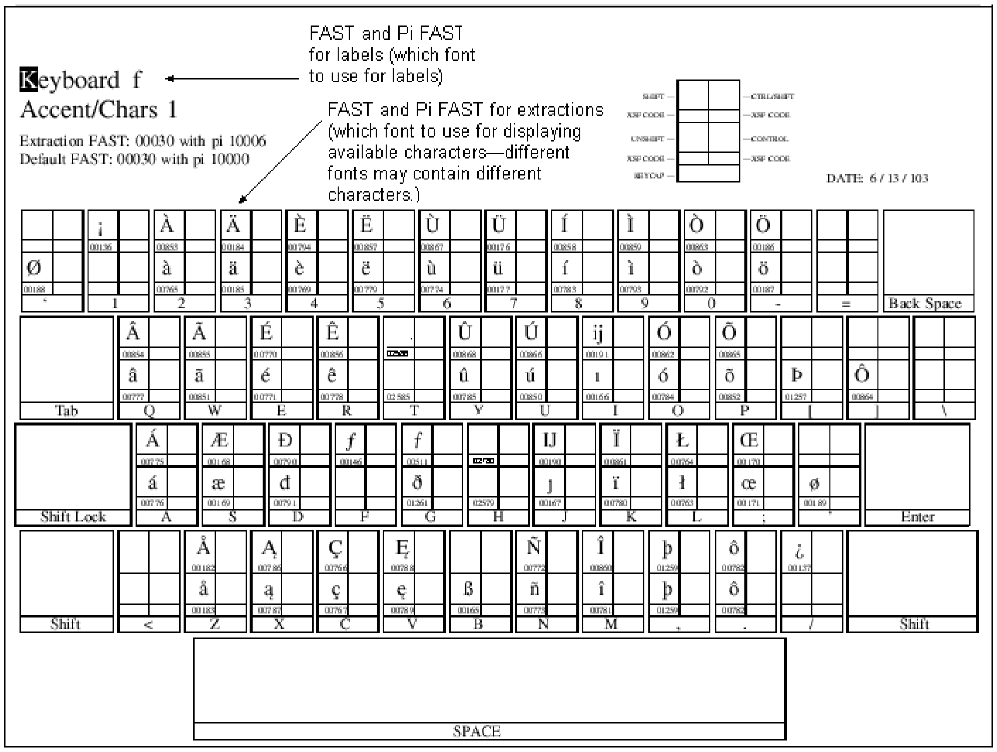

Keyboard f
Accent/Chars 1
Extraction FAST: 00030 with pi 10006
Default FAST: 00030 with pi 10000
FAST and Pi FAST for labels (which font to use for labels)
FAST and Pi FAST for extractions (which font to use for displaying available characters—different fonts may contain different characters.)
DATE: 6 / 13 / 103
Back Space
Tab
Shift Lock
Enter
Shift
SPACE

Figure 4-2 *Sample Keyboard Displaying Default Values*

Updating KB Specs

When you upgrade software, XPP delivers the latest KB Specs to XYV_STYLES/xylibrary/Lsyslib, and does not overwrite the existing KB Specs (except keyboard 0, which is always overwritten). This allows you to continue to use your custom KB Specs, if you prefer.

Copying New Keyboard Specs

You can copy all new KB Specs in XYV_STYLES/xylibrary/Lsyslib directly to XYV_STYLES/Lsyslib.

To copy KB Specs globally:

- Using Windows Explorer, copy all files named **_kb_*.sde** in %XYV_STYLES%/xylibrary/Lsyslib to %XYV_STYLES%/Lsyslib.
- Using a DOS window, navigate to the %XYV_STYLES%/Lsyslib directory and enter the following command:
  ```
  copy %XYV_STYLES%\xylibrary\Lsyslib\_kb_*.sde .
  ```
- In UNIX, navigate to the XYV_STYLES/Lsyslib directory, and enter the following command:
  ```
  cp XYV_STYLES/xylibrary/Lsyslib/_kb_*.sde .
  ```

Note: *In the above instructions, the asterisk (*) in the filename represents one or more characters from the sets a-z, ua-uz, and 0-9.*

Copying Individual KB Specs

If you want to copy individual KB Specs through PathFinder, you follow the general procedure for copy specs. Refer to page 5-10 for information on copying specs.

Chapter 5

Font Libraries and Specs

This chapter contains information on the following topics:

- Font libraries
- Font specs

Managing Font Libraries

There are two basic types of libraries in XPP: style libraries and font libraries. Style libraries are those libraries where style bundles, translation tables, loose-leaf specs, CITI specs, and the like are stored. A font library contains only font-related specs and machine-readable files.

A font library typically consists of three parts, all bearing the same root name (e.g. noto) but having a different prefix: K, L, or X (as in Knoto, Lnoto, and Xnoto). These library "parts" fall into two categories—source and destination libraries.

***Tip:** RWS recommends keeping Lfontlib and Lstylelib separate, using base names that clearly indicate the type of library.*

Source and Destination Font Libraries

A *source* font library contains the editable font specs used to generate the Font Access Tables (FASTs) and Kerning Pairs machine-readable files.

A source font library contains:

- Files necessary to create a FAST:
  a. Phototypesetter (PTS) Specs and FAST Generation (FGS) Specs (required)
  b. Pseudofont (PSF) Specs and FAST Generation Exception (FGX) Specs (optional)
- Other specs not used for producing FASTs:
  a. Typesetter Font Map (TSF) Spec (required)
  b. Kerning Pairs (KP) Specs (optional)
  c. Font Variant (FV) Specs (optional)

Destination font libraries contain the machine-readable font files, called FASTs, and the machine-readable Kerning Pairs (KP) data files needed for composition, screen display, and output. Machine-readable files in the destination libraries are not editable. FASTs are located in X*fontlib* libraries, while Kerning Pairs data files are in K*fontlib* libraries.

A FAST contains character width and placement information that the system needs for composition and output. GenFAST processes the source font specs (PTS, FGS, PSF, FGX), and places the resulting FASTs in a destination library. Similarly, when you store a Kerning Pairs Spec, the system creates the destination Kerning Pairs library and places the machine-readable Kerning Pairs data file there. Machine-readable FASTs and Kerning Pairs data files vastly enhance XPP composition speed.

Naming Font Libraries

XPP delivers a font library called *noto*. This library contains the font-width specs for the delivered 163 Noto fonts.

Because RWS also provides the PostScript fonts for this set, these fonts can be used immediately upon installation of your XPP system. Optionally, XPP has a font library available called *lnpost*, which contains font-width specs for 1200 fonts. This is **not**, however, part of a standard delivery, and does not include the corresponding PostScript fonts.

You may create your own font libraries, or use those provided with XPP. RWS recommends that if you create a unique font library, you copy the *noto* library (K, L, and X) as a base, and add to it or modify it to meet your needs. This ensures at least minimal access to the delivered set of 163 fonts.

A valid library name consists of up to eight characters (not including the L, K, or X prefix), consisting of uppercase and lowercase characters, symbols (such as $ and &), and the digits 0-9. A library name must contain at least one alphabetic character.

Do not end font library names with numbers unless you are using linked numbered libraries, that is, you intend to use genfast to string multiple L*fontlib#* libraries together into a single X*fontlib*. Genfast will drop any numerical characters in the name of the generated output X*fontlib* FAST library (even if there is only one library named L*fontlib#*).

When you rename a library in PathFinder, XPP automatically prepends K, L, or X to the name. When creating a new font library, you can only create the L version; the K*fontlib* version is built as a post-process invoked when you store out of a KP Spec; the X*fontlib* version is generated when you initiate GenFAST. Likewise, if you are creating a new library by using the Build FAST utility, all three font library versions are created automatically.

Caution Do not delete or rename the *noto* font library.

Do I Need More Than One Font Library?

Ordinarily, there is no need for multiple font libraries. However, you may need multiple font libraries if any of the following situations are true:

- You are working with dissimilar sets of fonts, either for different publications or for different customers (in the case of commercial typesetters), and you want to organize them in unique font libraries.
- You are customizing characters within a font for a given document, but not for others.
- You are upgrading from an earlier XPP release and don't want to mix *post*, *post*uni and *noto* font libraries moving forward.
- You want to stop using Type 1 fonts and upgrade to a more advanced

font technology to take advantage of OpenType fonts and their greater character coverage, language support, etc.

Font Libraries in PathFinder

Font libraries appear as L*fontlib*, X*fontlib*, or K*fontlib* in the Tree View display of STYLE LIBRARIES.

File System Location

XPP libraries are always located in a directory called sd_liz. The path to this directory may vary, although typically it is located in an “XPP” (or similarly named) folder, along with xz, the default jobs, and other delivered folders. The path prefix may be determined or changed by doing one of the following:

- Using the Document Paths feature on the XyAdmin Tool (refer to the *XML Professional Publisher: Managing XPP* for information about the XyAdmin Tool.)
- Using the XYV_STYLES environment variable from the command line:
  - Linux: `printenv | grep XYV_STYLES` or `echo $XYV_STYLES`
  - Windows: `set XYV_STYLES`
- Using the following sequence in Windows®:

  Control Panel > System > Advanced system settings > Environment Variables...

  Select the XYV_STYLES entry in the System Variables list to display the directory path for sd_liz in the *value* field.

Accessing Font Libraries

You access font libraries from the PathFinder Tree View under *STYLE LIBRARIES*.

You can create a new library from the Menu bar.

- Select **Tools > Create Library**.
  PathFinder displays L*new* in alphabetical order in the STYLE LIBRARIES Tree View.

You can duplicate, delete, or rename a library.

1. Right-click the **library name**.
2. Select the appropriate **task** from the pop-up menu.

You can create new font specs.

1. Right-click **L*fontlib***.
2. Select **New** from the pop-up menu.
3. Select the specific **type** of spec from the submenu.

Caution

Do not delete or rename the *std-dict*, *std-fmt*, *std-tran*, *syslib*, or *noto* libraries. These are RWS-supplied libraries and must remain in the system and retain their original names.

Managing Font Specs

All of the XPP font specs have individual purposes as well as being dependent on each other to set quality type. The following table shows the font spec names, their associated mnemonics, and a brief description of each spec. The specs are listed in the order in which they are described in this manual.

Table 5-1 *Font Spec Names and Descriptions*

Spec	*Mnemonic*	*Description*
Xyvision Character Set	XCS	Defines XCS numbers, named character entity names, XyASCII escape sequences, and Unicode values for characters.
Keyboard	KB	Assigns characters to positions based on their XyASCII escape sequences.
Phototypesetter	PTS	Contains information for every character in a particular font.
Pseudofont (optional)	PSF	Modifies output of one or more characters. Also for combining existing characters to create a new character.
FAST Generation	FGS	Groups characters from one or more fonts into a FAST (Font Access Table).
FAST Generation Exception (optional)	FGX	Includes characters that would otherwise not be in the FAST.
PostScript Name	PSN	Maps PostScript character names to Unicode numbers.
Viewable FAST	VFX	Viewable version of a FAST.
Font Variant	FV	Maps font families and variants to FASTs.
Typesetter Font Map	TSF	Maps the font FAST number in the PTS Spec to the PostScript font name and assigns an encoding table name or CMAP file name as well as CSS font property names.
Kerning Pair (optional)	KP	Defines kerning between specified characters.
Ligature/Accent Replacement	RP	Replaces specified input characters with ligatures and/or accents.

Naming Font Specs

How you name specs depends on whether you are working through the command line or through PathFinder.

- Through the Command Line—Use the spec name, such as FGS, FV, PSN in lowercase letters, preceded and followed by an underbar, then by a name consisting of no more than eight (8) alphanumeric characters, and the suffix .sde, for example, _fgs_00001.sde.
- Through PathFinder—It is only necessary to use the name of no more than eight (8) alphanumeric characters, for example, 00001. PathFinder does not display the prefix, suffix, or the underbar.

The following table displays the naming conventions for the font specs and files, and provides examples.

Table 5-2 *Naming Conventions for Font Specs and Files*

PathFinder Spec Categories	*Spec Mnemonic*	*Parameters*	*Example*
Xyvision Character Set	XCS	The only valid name is “default”	_xcs_default.sde
Keyboard Maps	KB	Up to two alphabetic or numeric characters	_kb_2.sde _kb_a.sde _kb_ua.sde
Phototypesetter Specs	PTS	Up to 7 alphanumeric characters	_pts_00003.sde
Pseudofonts	PSF	Up to 7 alphanumeric characters; do not use the same name as a PTS Spec (or, append “ps” to the end)	_psf_math.sde _psf_00003ps.sde
FAST Generation	FGS	Five-digit number between 00001 and 65535	_fgs_00236.sde
FAST Generation Exceptions	FGX	The same name as the corresponding FGS Spec	_fgx_00236.sde
PostScript Names	PSN	The delivered names are “ps2xcs” “custom” “unicode”	_psn_ps2xcs.sde _psn_custom.sde _psn_unicode.sde
FASTs	FX	Machine-readable file. The same name as the corresponding FGS spec (system-assigned)	fx_00236 Located only in X*fontlib*.

Table 5-2 *Naming Conventions for Font Specs and Files (Continued)*

PathFinder Spec Categories	***Spec Mnemonic***	***Parameters***	***Example***
Font Variants	FV	Up to 7 alphanumeric characters	_fv_doc.sde
Typesetter Font Maps	TSF	The only valid name is "system"	_tsf_system.sde
Kerning Pairs	KP	Up to 8 alphanumeric characters	_kp_00001.sde _kp_timesrom.sde Located only in L*fontlib*.
Kerning Data	KP	Machine-readable file. Up to 8 alphanumeric characters	_kp_00630.*x* Located only in K*fontlib*.
Ligature/Accent Replacement	RP	The only valid name is "sys"	_rp_sys.sde

Accessing Font Specs

Access font specs in PathFinder just as you would any other spec:

1. Navigate to **STYLE LIBRARIES > L*fontlib*** and select the desired spec type. Pathfinder displays the the specs in the List View.
2. From the List View, right-click the specific **spec** and select the **task** you want to perform from the pop-up menu.
   —or—
   Double-click the **spec**.
   XPP opens the spec in the Sdeditor in edit mode.

Note: *Refer to Table 5-3 to determine which spec category you should select for the particular kind of spec you want to access.*

When to Edit Font Specs

The following table lists the specs and when you should edit them. Run GenFAST after editing a PTS, PSF, FGS, or FGX Spec for the changes to take effect.

Table 5-3 *When to Edit the Font Specs*

Spec	***Mnemonic***	***When to Edit***
Xyvision Character Spec	XCS	Does not usually need to be edited
Keyboard Spec	KB	Does not usually need to be edited
Phototypesetter	PTS	To set up or add fonts
Pseudofont (optional)	PSF	To modify the output of characters
FAST Generation	FGS	To add characters from one or more fonts to a FAST
FAST Generation Exception (optional)	FGX	To add an exception character(s) to a FAST
Font Variant	FV	To set up or add fonts
Typesetter Font Map (not always necessary)	TSF	To map font numbers to additional information or adjust the values for CSS-related fields
Kerning Pairs (optional)	KP	To specify spacing between characters
Ligature/Accent Replacement (optional)	RP	Does not usually need to be edited

NOTE: FAST specs (FX, VFX, and VPX) are not edited. See Chapter 9, ″Creating & Viewing FAST Specs″.

Copying Font Specs through PathFinder

When you copy a font spec you are, in effect, creating a new one, but a new one identical to the one you just copied.

Copying a Spec to the Same Library

To copy a spec to the same library:

1. Navigate to **STYLE LIBRARIES > L*fontlib*** and select the desired spec type.

   Pathfinder displays the the specs in the List View.

2. Right-click the **spec** you want to copy and select **Copy**.

   XPP copies the selected spec and places it in the buffer.

3. Right-click **L*fontib*** in the Tree View and select **Paste**.

   XPP does the following:

   - Adds the copied spec to the appropriate spec type in the ListView at the appropriate alphabetical location.
   - Gives it the name *XXXCopy1*, where XXX is the name of the original spec.
   - Positions the List View cursor on the newly copied spec.

4. Right-click the **name** of the newly copied spec to change its name. XPP highlights the spec name and displays a pop-up menu.

5. Select **Rename** from the pop-up menu.

   XPP places a frame around the file name and prepares it for editing.

6. Type a new **file name** following the naming conventions for a new spec.

7. Press the **Enter** key when you are finished renaming the file.

   XPP places the new file name in the appropriate alphabetical order.

Copying a Spec to a New Library

Copying a Spec to a new library follows the same procedure as copying a spec to the same library. The only difference is in step 3 in the previous section. In this case, you right-click the new L*fontlib* to which you want to paste the spec. If the spec type container does not exist in that library, PathFinder creates it before pasting the spec. Since this is the only spec with that name, there is no copy1 affixed to the spec name. If there is already a spec by the same name in the new location, PathFinder informs you and asks if you wish to overwrite the file.

Field Notation

Some of the fields in the PTS, PSF, and FGX Specs accept decimal, hexadecimal, or octal values. When entering values in these fields, the value is preceded by a letter denoting the notation, for example, *d* for decimal, *x* for hexadecimal, or *o* for octal. Do not place a spaceband between the notation and the number.

Changing Field Notation

You can change the display in the fields to decimal, hexadecimal, or octal notation.

To change the notation in which a field is displayed while editing or viewing a spec:

1. Place the cursor in the field and press **Menu > Display** on the Softkey menu (or use the Display key).
   The system displays the Display Control menu with the *See Octal*, *See Decimal*, and *See Hex* options.
2. Select the **notation** in which you want to display the field.
   All fields of that type change to the new notation; fields of other types do not change.

Using Field Notation

Below is some helpful information to keep in mind when using decimal, hexadecimal, and octal display:

- The default notation for the spec fields is decimal. For example, if you are editing a new spec and have not changed the notation, the default values are in decimal. If you enter a value without a preceding notation, the system assumes that the field is in decimal notation and inserts a preceding d.
- The spec fields retain the notation from the last time you edited *and* stored the spec. For example, if the fields were in octal display the last time you stored the spec, they will be in octal display the next time you open the spec.
- You can enter values in different notations within a table. For example, in the PTS spec, the *PTS Code* field in one rule can be in decimal notation, and the *PTS Code* in another rule can be in hexadecimal notation.

  Once you change the notation for that field, all fields of that type will be in the specified notation.
- You can mix notations within a rule. For example, in the PTS Spec, the *PTS Code* and *Unicode Number* fields can have different notations.
- If you change the notation display and enter a value without a

preceding d, x, or o, the system assumes that a field is in the notation you specified. When you exit the field, the system automatically inserts the appropriate preceding notation.

- You cannot enter a hexadecimal value without the appropriate preceding notation. For example, you cannot exit the field if you enter 2D0 without a preceding x.

Header and Rule Fields

The header fields in the PTS, PSF, and FGS Specs contain global entries which apply to the majority of the characters described in the specs. Some header fields have corresponding rule fields that you can use to override the entries in the header fields.

Editing Spec Fields

Below is helpful information for editing the fields in a font spec:

- Fields that accept values in decimal, octal, and hexadecimal default to d0, o0, or x0 depending on the notation currently in use.
- In fields that accept hyphens, the hyphen indicates that the entry in the corresponding header field will be used.
- Rule field entries are only in effect for that rule. Subsequent rules do not inherit field entries from previous rules.

Overriding PTS, PSF, and FGS Header Fields

The following table shows the header fields that can be overridden and the corresponding rule fields you can use to override the header field.

Table 5-4 *Overriding Spec Header Fields*

Spec	*Override this header field ...*	*with this rule field ...*
PTS	Typesetter Information:	
	Style Code	CHARACTER OVERRIDES STYLE CODE
	Slant	CHARACTER OVERRIDES SLNT
	Slant	FONT SLNT (in FGX Spec)
PSF	Style code	Style Code
FGS	Style Code Exact Match	Style Code Overrides

Chapter 6

The Phototypesetter Spec (PTS)

This chapter contains the following information:

- Understanding the PTS Spec
- Setting up PTS Specs

Understanding the PTS Spec

A Phototypesetter (PTS) Spec maps each glyph in a font to a Unicode number and provides information on the glyph width, style, and so on.

The GenFAST program uses this information, along with information from other specs, to create a Font Access Table Spec (FAST) consisting of glyphs from one or more fonts.

The Build FAST utility creates the PTS Spec from information in the font's AFM (Adobe Font Metrics) file, which is provided separately by the font vendor for Type 1 base and Type 1 CID fonts. This information includes the character code and character name or character ID (CID). XPP uses the character names or CIDs in the AFM file to map to the appropriate Unicode numbers.

For OpenType fonts, when Font Copy has been performed on the font it creates the separate AFM file in the `XYV_EXECS/psres/fonts/`*`fontname`*`.font` folder. When using Build FAST, the user can select this generated AFM file or for OTF fonts, Build FAST can extract the metric data directly from the OTF file if it is selected instead of the AFM file.

Accessing PTS Specs

You access PTS Specs from PathFinder, using the following sequence:

STYLE LIBRARIES > L*fontlib* library > Phototypesetter Spec.

Delivered PTS Specs

XPP delivers PTS Specs for the delivered Google Noto fonts to the **Lnoto font library.**

AFM Files

Adobe Font Metrics (AFM) files are separately provided by the PostScript font vendors for Type 1 base and Type 1 CID fonts — one AFM file for each PostScript font. Font Copy generates the AFM files for OpenType fonts. AFM filenames for Type 1 and OpenType fonts generally reflect the PostScript font name and have a ".afm" filename extension. AFM filenames for Type 1 CID fonts have various filenames and generally do not have any filename extension or might have a ".cid" filename extension.

AFM files are ASCII files and are therefore easy to view and print. They contain several types of information that is useful for setting up font specs:

- The unique PostScript font name
- The number of glyphs in the font
- The access (encoding) code, width, and name or character ID (CID) for each glyph
- Optionally, kerning pair definitions (See ″The Kerning Pairs Spec (KP),″ for more information.)

The unique PostScript font name is located near the top of file on the ″FontName″ line (not the ″FullName″ or ″Family Name″ lines).

For example: `FontName NotoSerif-Regular`

The ″StartCharMetrics″ line specifies the number of glyphs in the font.

For example: `StartCharMetrics 3255`

The following figure shows a single line of glyph information from an OpenType AFM file and explains how to interpret the information. The glyph width is based on 1,000 units per em.

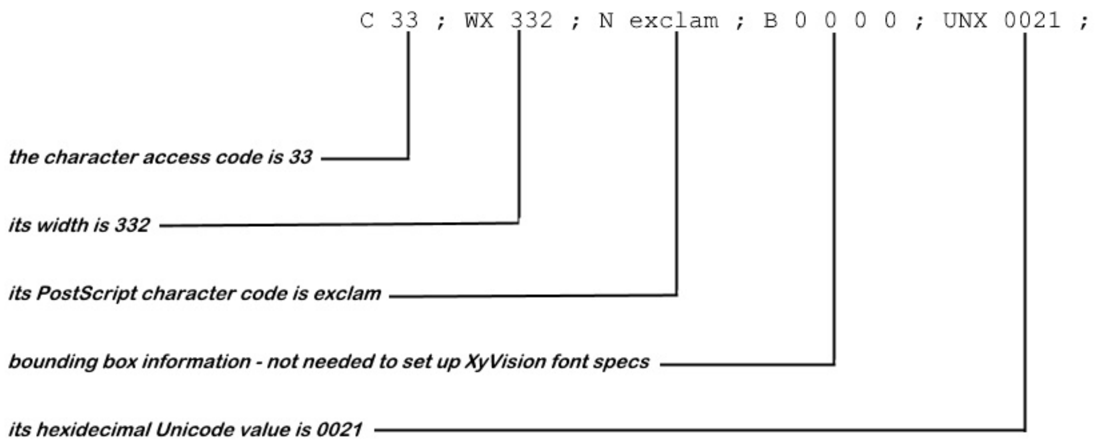

```
C 33 ; WX 332 ; N exclam ; B 0 0 0 0 ; UNX 0021 ;
```
the character access code is 33
its width is 332
its PostScript character code is exclam
bounding box information - not needed to set up XyVision font specs
its hexidecimal Unicode value is 0021

Figure 6-1 *AFM File Glyph Information*

Kerning Data

The AFM file contains kerning data for each glyph, such as the following for the NotoSerif-Regular font:

StartKernData
StartKernPairs 51456
KPX U+0022 U+0041 -80
KPX U+0022 U+0063 -20
KPX U+0022 U+0064 -20

```
KPX U+0022 U+0065 -20
KPX U+0022 U+0067 -40
KPX U+0022 U+006f -20
...
```

OTF Files

For OpenType CFF fonts (.otf), metric information can be extracted directly from the OTF file to a temporary file during Build FAST, then deleted, rather than using the AFM file during Build FAST.

The Relationship Between PTS Spec and PSN Spec for Type 1 Fonts

When using Build FAST to create the PTS Spec for a Type 1 font, it will also use information located in the PSN (PostScript Name) Specs. The PSN Specs are not used for OpenType Unicode fonts. Two PSN Specs are delivered with the system. These PSN Specs map standard glyph names to Unicode numbers. If a glyph name is not recognized, you may customize a PSN Spec, adding glyph names and assigning them to Unicode numbers.

Naming a PTS Spec

A PTS Spec name consists of ***_pts_*** followed by up to eight alphanumeric characters, for example, _pts_00001.sde. Build FAST automatically names the PTS Spec after the FAST number specified in the Build FAST dialog box. Thus, if the FAST is 5001, the PTS Spec will be pts_05001.

Note: *When running Build FAST, you can think of PTS and FAST names as basically the same. But, when you run GenFAST to create the actual FAST, it is the FGS name that ultimately determines a FAST's name. For text fonts, all three will usually be the same, but Pi FASTs usually combine glyphs from numerous PTS Specs, and the FAST name might not correspond to any of the PTS names.*

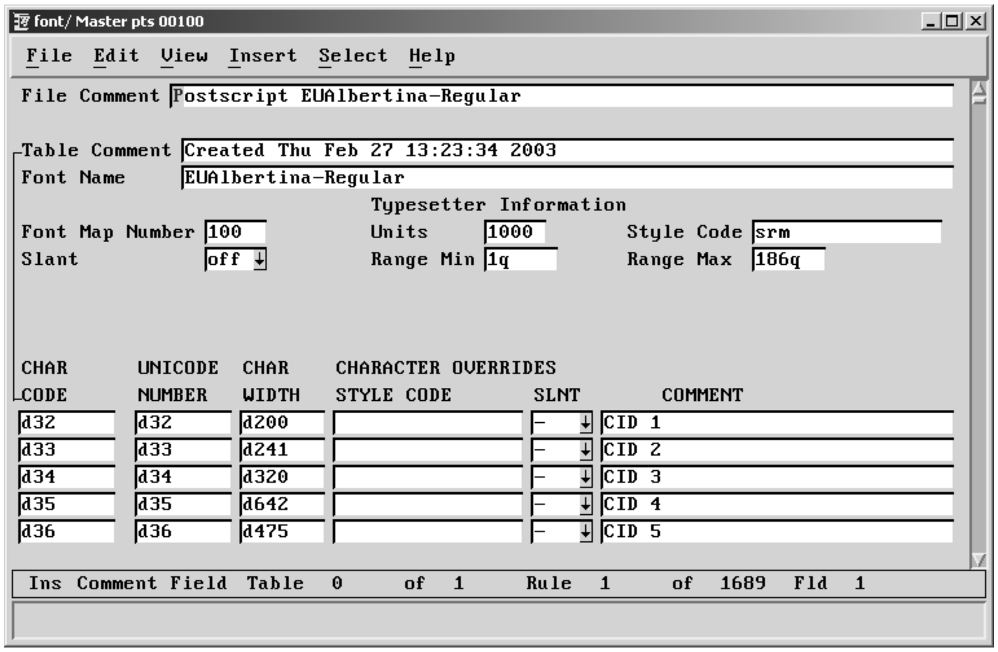
For a FAST, a PTS Spec cannot have the exact name as a PSF Spec (Pseudofonts Spec). If the PTS and PSF Spec names are identical, GenFAST does not know which type of spec to use — PTS or PSF. A PSF Spec typically has the same name as a PTS Spec with *ps* appended to the end, for example, 05001ps. For additional information on the PSF Spec, refer to page 7-1.

Structure of a PTS Spec

A PTS Spec consists of a *File Comment* field and one table with the following sections:

- Header — contains a *Table Comment* field and *Typesetter Information* section containing default information defining the majority of the glyphs in the font.
- Rules — contain information about each glyph in the font. Some information in the rules overrides information in the header fields. There is one rule for each glyph in the font.

The following figure shows the structure of the PTS Spec.



File Comment: Postscript EUAlbertina-Regular

Table Comment: Created Thu Feb 27 13:23:34 2003
Font Name: EUAlbertina-Regular

Typesetter Information

Font Map Number: 100
Slant: off
Units: 1000
Range Min: 1q
Style Code: srm
Range Max: 186q

CHAR CODE	UNICODE NUMBER	CHAR WIDTH	CHARACTER OVERRIDES STYLE CODE	SLNT	COMMENT
d32	d32	d200		-	CID 1
d33	d33	d241		-	CID 2
d34	d34	d320		-	CID 3
d35	d35	d642		-	CID 4
d36	d36	d475		-	CID 5

```
Ins Comment Field  Table  0     of  1     Rule  1     of  1689  Fld  1
```

Figure 6-2 *Phototypesetter Spec*

Header Fields

The entries in the header fields are global entries for the spec; you can override entries in these fields if there is a corresponding rule field.

Font Name

The name of the font.

Entry	*Description*
string	As many as 70 alphanumeric characters, including uppercase and lowercase characters, symbols (such as $, &, and /) and the integers 0-9.

Font Map Number

Uses number of FAST from Build FAST utility.

PostScript fonts do not usually have numbers already assigned to them, except for the set of Noto fonts delivered with XPP and the delivered **noto** font library. Use your own numbering scheme. To avoid conflicting with numbers that XPP has already assigned to PostScript fonts, use numbers above 5000 if using the delivered **noto** font library. For instance, start with _pts_05001.sde and work up.

The *Font Map Number* is the same number as the one in the *Font Map No.* field of the TSF Spec. For information on the TSF Spec, refer to “The Typesetter Font Map Spec (TSF)” on page 13-1.

The Font Map Number assigns a number to the font’s *PostScript name* because the composition component of XPP can only read numbers, whereas the display and output components need to read a PostScript name.

Note: *There is no relationship between the Font number and the Font Family/ Variant numbers. Within a division, fonts are accessed by family/variant number. These, in turn, are mapped to the Font Number (FAST), which is then mapped to a PostScript font name via the TSF Spec.*

Entry	***Description***
integer	A positive integer in the range of 0 through 32767. This field entry defaults to 0.

Slant

This field controls the amount of electronic slant applied to glyphs and is useful if an italic version is not available . For example, if you purchased roman, italic, and bold versions of a font but not bold italic, you could slant the bold font to simulate bold italic.

Electronic slant is usually available on PostScript output devices.

You can slant roman glyphs; you can also give italic glyphs more slant. Slanting the glyphs using this field slants all the glyphs, including bullets, periods, and so on.

Enter the amount that you want to slant the glyphs. Do not enter a unit qualifier; the unit is degrees. Override this field with the *PTS Slant* rule field.

The entries 9 and 14 are commonly used degrees of slant; if you are unsure of the amount of slant you want, try these entries first.

Entry	*Description*
integer	An integer in the range of –45 through 45. The unit is degrees. Negative values slant glyphs to the left; positive values slant glyphs to the right.
off	Specifies no slant (default). This value is available with *Prev Choice, Next Choice*.
0	Specifies no slant. An entry of 0 changes to off when you exit the field.
9	Specifies 9 degrees of slant. This value is available with *Prev Choice, Next Choice*.
14	Specifies 14 degrees of slant. This value is available with *Prev Choice, Next Choice*.

Units

PostScript width values are specified relative to 1000 units per em.

Entry	*Description*
integer	An integer in the range of 1 through 65535. This field entry defaults to 1000.

Range Min

The smallest point size for outputting any glyph in this font.

Entry	*Description*
number	A positive numeric value, followed by a valid unit qualifier, in the range of 0 through 186.1q or the equivalent in p, i, m, n, c, d, k units. This field entry defaults to 1q, however, the minimum point size the system can output is 4.5q. (Technically, you can specify a smaller point size, but some things having to do with pickup placement and/or leading may not work properly.)

Range Max

The largest point size for outputting any glyph in this font.

Entry	*Description*
number	A positive numeric value, followed by a valid unit qualifier, in the range of 0 through 186.1q or the equivalent in p, i, m, n, c, d, k units. The maximum point size of 186.1 points is the result of 16-bit values, which are used in XPP, and is the maximum point size the system can output. This field entry defaults to 186.1q.

Style Code

This field is relevant for *Pi* fonts. It describes the majority of the glyphs in the font. Consider the following questions when assigning style codes:

- Is this a serif or a sans serif font?
- Is this a roman or an italic font?
- What is the weight of the glyphs in the font?

Assign a style code that describes the majority of the glyphs in the font, then, if necessary, enter exception style codes in the *PTS Character Style Code* rule fields.

You may need to change a style code if either one of the following situations is true:

- You are using a font that contains the same characters in multiple variations, for example, a bold glyph and a medium glyph.
- You are using XPP to set math, particularly in pre-7.x revisions, whereby GenFAST generated special FASTs. For example, in a math equation, you might enter a standard alphanumeric characters, but want to output a lower case italic glyph without having to change the font variant. This type of math font environment is still valid in XPP.

Other than these two situations, there should be little need to change a style code when using text fonts.

There is a relationship between the style code entered in the PTS Spec and the style code specified in the FGS Spec; this is explained in more detail in the chapter about the FGS Spec.

Use only the valid entries described below; spacebands or underbars are not valid characters. The system does not check for invalid codes when you exit this field, however, you cannot successfully run GenFAST if this field contains an invalid code.

Use the *PTS Character Style Code* rule field to override this field for exceptions to this style code.

Entry	*Description*
no entry	If this field is blank (default), the GenFAST program uses the rule field entry. If both this field and the rule field are blank, GenFAST does not pick up any glyphs from this spec.
s	serif
n	sans serif
i	italic (*slanted; oblique*)

Entry	*Description*
r	roman (*upright*)
e	extra light; describes the glyph weight
l	light; describes the glyph weight
m	medium; describes the glyph weight
b	bold; describes the glyph weight
h	heavy; describes the glyph weight
x	extra bold; describes the glyph weight
d	inferior; describes a typeface used to set inferior text
u	superior; describes a typeface used to set superior text
c	small caps; describes small cap glyph
o	other; a characteristic not represented by any other style code in this list. This code is undefined until you determine the meaning. For example, you have a font with a particular characteristic (e.g., an outline font). You could set a standard in your shop that style code o stands for that characteristic.
a[1]	all attributes; the same as individually entering all the attribute codes listed

[1] This style code is often used for *Pi* glyphs that can be used with any style text font.

For example, if you have a *Pi* font containing serif medium glyphs with a few serif bold glyphs, enter **snirelmbxh** in the *Style Code* field. In the rules in the PTS Spec for the serif bold glyphs, enter **snirbxh** in the *Style Code* field of those rules. In the rules in the PTS Spec for the serif medium glyphs, enter **snirmel** in the *Style Code* field of those rules.

Rule Fields

The PTS Spec contains a rule for each glyph in the font. Fields that accept values in decimal, octal, and hexadecimal default to d0, o0, or x0 depending on the notation currently in use.

A hyphen in the *Slant* rule fields indicates that the entry in the corresponding header field is being used.

Char Code

The character/access code for the glyph preceded by a letter denoting the notation. For non-CID fonts, use the .afm file ″C″ value as the entry for the *Char Code* field.

Entry	*Description*
d*integer*	A positive decimal integer in the range of d0 through d32767.

Entry	*Description*
o*integer*	A positive base 8 (octal) integer in the range of o0 through o77777.
x*integer*	A positive base 16 (hexadecimal) integer in the range of x0 through x7FFF.
0	Use this entry for spacebands, em spaces, en spaces, and so on. For more information on these characters, refer to the description of the *Character Width* field.

Unicode Number

Enter the Unicode number assigned to this glyph.

When using the Build FAST utility for Type 1 fonts, XPP uses the PostScript Name (PSN) Specs to match the glyph name to the appropriate Unicode number.

Unicode numbers xF0501 through xF0856 are reserved for user-defined characters, such as company logos and pseudo characters.

Be careful when entering the same Unicode number more than once—it may not give you the results you expected. GenFAST puts only one of the glyphs with the same Unicode number into the FAST. For more information on how GenFAST processes Unicode numbers, refer to “Creating & Viewing FAST Specs” on page 11-1.

Character Width

This is the width of the glyph based on 1,000 units per em. Do not enter 0 in this field; glyphs must have a width of at least one unit.

The font vendor provides the glyph width information, usually in the .afm file. The Build FAST utility automatically populates the PTS Spec with the following entries for Type 1 fonts.

Char.	*Unicode No.*	*Char Code*	*Width*
unit space	x200A	0	units per em divided by 100, rounded to the nearest integer. (For example, 10 for 1000-unit PostScript devices.)
figure space	x2007	0	width of the number 2
thin space	x2009	0	width of a period
en space	x2002	0	width of an en dash
em space	x2003	0	width of an em dash
apostrophe	x27	d39	width of a close quote

Style Code

Enter a style code to give this glyph a different style from that defined in the header field. Rules defining *Pi* glyphs typically have an entry in this field.

Leave this field blank if the style in the *Style Code* header field adequately describes this glyph. For text fonts, you would typically leave this field blank and define the style code in the header field.

You may have several variants of one *Pi* character available in one *Pi* font (e.g., serif, sans serif, bold, medium). Create one rule for each variant the font has for the *Pi* character (each rule has the same Unicode number). Enter the appropriate PTS codes, glyph widths, and style codes for each of the variants. The results depend on how you set up the FGS Specs for running GenFAST. For more information, refer to “Setting up FGS Specs” on page 9-4.

The valid entries are the same as for the *Style Code* field on page 6-8 in the *PTS Spec Typesetter Information Header Fields* section.

Slant

Enter a value to slant the glyph defined in this rule by a different amount than defined in the *Slant* field in the Typesetter Information section of the header. This field overrides the *Slant* field in the Typesetter Information section of the header.

The valid entries are the same as for the *Slant* field on page 6-6 in the *PTS Spec Typesetter Information Fields* section.

Comment

A comment that describes the glyph accessed by this rule. For Type 1 fonts, this information may be derived from the PSN Spec.

Chapter 7

The Pseudofont Spec (PSF)

This chapter contains the following information on the Pseudofont (PSF) Spec:

- Understanding the PSF Spec
- Setting up PSF Specs
- Examples of defining pseudo characters

Understanding the PSF Spec

A Pseudofont (PSF) Spec allows you to send commands directly to the output device to modify glyph placement and/or appearance. Modifying glyphs on output also affects screen display.

For example, in large point sizes, the open parenthesis looks too wide. You can define the open parenthesis in the PSF Spec and modify its width.

Using a PSF Spec, you can:

- Modify the height, width, or placement (both horizontal and vertical) of a glyph.
- Issue typesetter direct commands that affect the output of glyphs (e.g., reverse video).
- Combine glyphs from one or more PostScript fonts to form a single character.

Once you have included a PSF Spec in a Font Access Table (FAST), you can use those characters for display and output as you can use any other characters.

Accessing PSF Specs

You access PSF Specs from PathFinder, using the following sequences:

STYLE Libraries* > *Lfontlib* > *Pseudofonts

Setting Up PSF Specs

Before editing a PSF Spec, gather the following information:

- PTS codes, Unicode numbers, and glyph widths of the glyphs you want to use or modify.
- Unicode numbers — decide which custom or standard Unicode numbers you want to assign to the pseudo characters.

Naming a PSF Spec

If you are creating a PSF Spec, assign a name to it that consists of the _psf_ prefix followed by a name of up to eight alphanumeric characters. Do not give PSF and PTS Specs identical names; otherwise, GenFAST does not know which type of spec to use — PTS or PSF.

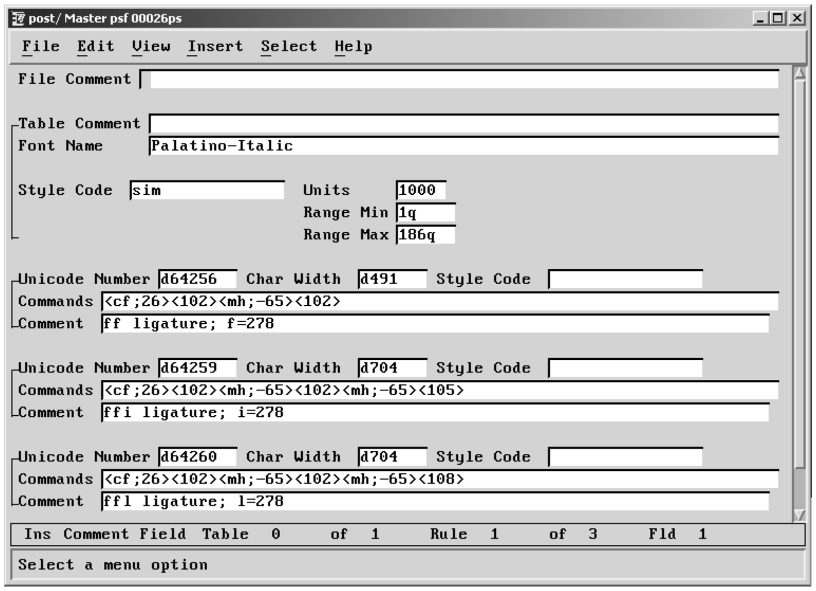
PTS and PSF Specs are typically named similarly, but PSF Spec names are typically followed by **ps** — for example, _psf_05001ps.sde. This distinguishes it from _pts_05001.sde when running GenFAST.

Structure of a PSF Spec

A PSF Spec consists of a *File Comment* field and one table with the following sections:

- Header — contains a *Table Comment* field and a *Typesetter Information* section containing default information that defines the majority of the pseudo characters in the file.
- Rules — contain information about each pseudo character that is being used on output. There is one rule for each pseudo character.

The following figure shows the structure of the PSF Spec.



```
File Comment

Table Comment
Font Name      Palatino-Italic

Style Code  sim              Units      1000
                             Range Min  1q
                             Range Max  186q

Unicode Number d64256  Char Width d491  Style Code
Commands  <cf;26><102><mh;-65><102>
Comment   ff ligature; f=278

Unicode Number d64259  Char Width d704  Style Code
Commands  <cf;26><102><mh;-65><102><mh;-65><105>
Comment   ffi ligature; i=278

Unicode Number d64260  Char Width d704  Style Code
Commands  <cf;26><102><mh;-65><102><mh;-65><108>
Comment   ffl ligature; l=278

Ins Comment Field  Table  0    of  1    Rule  1    of  3    Fld  1
Select a menu option
```

Figure 7-1 *Pseudofont Spec*

Header Fields

The entries in the header fields are global entries for the spec; you can override entries in these fields if there is a corresponding rule field.

These fields control the appearance of typeset pseudo characters.

Font Name

Enter a name that describes the pseudo characters in the pseudofont, for example, pseudomath; or, enter just the name of the font for which you are creating pseudo characters, for example, Garamond Bold.

Entry	*Description*
string	As many as 70 alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, and /), and the integers 0-9.

Style Code

This field is relevant for *Pi* fonts. It describes the majority of the pseudo characters in the font. Consider the following questions when assigning style codes:

- Is this for a serif or a sans serif FAST?
- Is this for a roman or an italic FAST?
- What is the weight of the pseudo characters or the target FAST?

Assign a style code that describes the majority of the glyphs in the font, then, if necessary, enter exception style codes in the *PSF Character Style Code* rule fields.

Use only the valid entries described below; spacebands or underbars are not valid characters. The system does not check for invalid codes when you exit this field, however, you cannot successfully run GenFAST if this field contains an invalid code.

Use the *PSF Character Style Code* rule field to override this field for exceptions to this style code.

Entry	*Description*
no entry	If this field is blank (default), the GenFAST program uses the rule field entry. If both this field and the rule field are blank, GenFAST does not pick up any pseudo characters from this spec.
s	serif
n	sans serif
i	italic (*slanted; oblique*)
r	roman (*upright*)
e	extra light; describes the pseudo character weight
l	light; describes the pseudo character weight
m	medium; describes the pseudo character weight
b	bold; describes the pseudo character weight
h	heavy; describes the pseudo character weight
x	extra bold; describes the pseudo character weight
d	inferior; describes a typeface used to set inferior text
u	superior; describes a typeface used to set superior text
c	small caps; describes small cap character
o	other; a characteristic not represented by any other style code in this list. This code is undefined until you determine the meaning. For example, you have a font with a particular characteristic (e.g., an outline font). You could set a standard in your shop that style code o stands for that characteristic.
a[1]	all attributes; the same as individually entering all the attribute codes listed

[1] This style code is often used for *Pi* glyphs that can be used with any style text font.

For example, if you have a *Pi* font containing serif medium glyphs with a few serif bold glyphs, do the following:

- Enter **snirelmbxh** in the *Style Code* field.
- In the rules in the PSF Spec for the serif bold pseudo characters, enter **snirbxh** in the *Style Code* field.
- In the rules in the PSF Spec for the serif medium pseudo characters, enter **snirmel** in the *Style Code* field.

Units

PostScript width values are specified relative to 1000 unit per em.

Entry	***Description***
integer	An integer in the range of 1 through 65535. This field entry defaults to 1000.

Range Min

The smallest point size for outputting any pseudo character in this font.

Entry	***Description***
number	A positive numeric value, followed by a valid unit qualifier, in the range of 0 through 186.1q or the equivalent in p, i, m, n, c, d, k units. The minimum point size of 4.5q. (Technically, you can specify a smaller point size, but some things having to do with pickup placement and/or leading may not work properly.) This field entry defaults to 1q.

Range Max

The largest point size for outputting any pseudo character in this font.

Entry	***Description***
number	A positive numeric value, followed by a valid unit qualifier, in the range of 0 through 186.1q or the equivalent in p, i, m, n, c, d, k units. This field entry defaults to 186.1q. The maximum point size the system can output is 186.1q, which is a result of using 16-bit programming in XPP.

Rule Fields

The PSF Spec contains a rule for each pseudo character that is being used on output. The rule fields contain information specific to the particular pseudo character. Fields that accept values in decimal, octal, and hexadecimal default to d0, o0, or x0 depending on the notation currently in use.

Unicode Number

The Unicode number may be the same as in the PTS Spec you are using, or different if you are combining multiple glyphs. See examples below for details.

Character Width

The pseudo character width follows a formula:

```
CharWidth = original width of glyph
               ± <mh> commands
               × <sw> commands
```

See Commands section.

Style Code

The valid entries are the same as those described on page 6-8 for Style Code under PTS Spec Header Fields for the Phototypesetter Spec.

Commands

The command must include the character access code as well as all commands to modify it. The commands and PTS codes are defined within angle brackets. Use the less than (<) and greater than (>) keycaps to access these angle brackets. These commands use *relative* moves and point sizes. This enables them to work well at various point sizes.

Note: *The glyphs you want to modify in the Pseudofont Spec need to be expressed by their encoding values, therefore, the PTS code you enter in the* **Commands** *field is the value in the* **Char Code** *field of the PTS spec and not the Unicode number.*

The font vendor does not supply information about the amount of space around the glyphs. When modifying a glyph, it may take several attempts to place the glyph as desired.

Use these commands in this field only. They are not XyMacros; do not use them in divisions.

Entry	*Description*
string	Command(s) and PTS code(s) defining the pseudo character; as long as 7½ lines (512 characters). Open angle brackets (〈), close angle brackets (〉), and semicolons are counted as characters. Refer to the following table for descriptions of the valid commands.

Comment

Describes the character.

Pseudofont Commands

The following table explains commands that are only used with the Pseudofont Spec.

Table 7-1 *Pseudofont Commands*

Command	*Description*
⟨mb;10⟩[1]	Moves the baseline by a percentage of the current font height (including any prior font height changes made in the same pseudofont character definition). Enter the percentage as the argument. Positive arguments move the baseline up; negative arguments move the baseline down. Reset the baseline with an ⟨mb⟩ command at the end of the command string, with the opposite value for the argument (as long as no <sh> commands have been entered between the two <mb> commands). For example, if you entered ⟨mb;-10⟩, insert ⟨mb;10⟩ at the end of the command to reset the baseline.
⟨mh;6⟩	Moves horizontally by a number of relative units. Enter the number of relative units as the argument; the units are based on the entry in the *Units* field in the *Header* section (as well as the current font width). Do not enter a qualifier. Positive units move right; negative units move left.
⟨mv;2⟩[1]	Moves vertically by a number of relative units. Enter the number of relative units as the argument; the units are based on the entry in the *Units* field in the *Header* section (as well as the current font height). Do not enter a qualifier. Positive units move the baseline down; negative units move it up. Reset the baseline with an ⟨mv⟩ command, at the end of the command string, with the opposite value for the argument (as long as no <sh> commands have been entered between the two <mv> commands). For example, if you entered ⟨mv;-2⟩, insert ⟨mv;2⟩ at the end of the command to reset the baseline.
⟨sh;10⟩[1]	Sets the font height to a percentage of the current font height (including any prior font height changes made in the same pseudofont character definition). Enter the percentage as the argument.
⟨sw;10⟩[1]	Sets the font width to a percentage of the current font width (including any prior font width changes made in the same pseudofont character definition). Enter the percentage as the argument.

Table 7-1 *Pseudofont Commands (Continued)*

Command	*Description*
⟨cf;30⟩	Changes to the specified font number. The ⟨cf⟩ command must appear before any <PTS code> entry that you want to ensure comes from a particular font. In most cases, the Commands field will contain a ⟨cf⟩ command. Refer to the section "Examples of Defining Pseudo Characters" on page 7-11 for more information. Enter the font number as the argument. The font number is the entry in the *Font Map Number* field in the Header section of the PTS Spec. It is *not* the font family or font variant number. After the pseudo character is output, XPP returns to the previous font; you do not have to reset the font number.
⟨td;1⟩	Typesetter direct command. Using a ⟨td⟩, you can send commands directly to the typesetter. The argument for this command must be in the "native language" of the typesetter. When the formatter finds a ⟨td⟩ it does not process the information, it passes the typesetter direct command to the typesetter for processing. The ⟨td⟩ command is ignored by the Direct-to-PDF (*divpdf*) program. If you use the ⟨td⟩ command to change fonts, the typesetter stays in the "new" font after outputting the pseudo character. It does not return to the previous font. If you want to change back to the previous font, enter the appropriate ⟨td⟩ command. Typesetter direct commands can be a three-digit decimal code, a three-digit octal code preceded by a backslash (\ ), or a two-digit hexadecimal code preceded by a caret (^). You can enter multiple typesetter direct commands within one ⟨td⟩ command by separating them with semicolons (;). For information on valid commands for your typesetter, consult the typesetter manufacturer.
⟨*PTS code*⟩	The PTS code, in decimal notation, for the component glyphs is the value in the *Char Code* field of the PTS spec and not the Unicode number. If the original PTS code is in octal (for example, for a Linotron typesetter), the value falls into one of three ranges — unshift, shift, or super shift. The following table describes converting the octal codes to decimal. If the PTS code is decimal 40, 41, or 92, these positions have special meanings in PostScript. Therefore, these characters must be escaped by preceding them with a backslash (PTS code 92). For details, see Example 2 later in this chapter.

[1]This command affects subsequent text unless you cancel it by entering another command.

The following table explains the conversion of PTS codes from octal to decimal.

Table 7-2 *Converting Octal PTS Codes to Decimal*

If the value is in the octal range	***Then ...***
0-77 (unshift)	convert to decimal
100-177 (shift)	subtract 100; convert the remaining number to decimal[1]
200-277 (super shift)	subtract 200; convert the remaining number to decimal[2]

[1]Precede the converted decimal PTS code with the shift command ⟨27⟩; follow the PTS code with the unshift command ⟨31⟩.

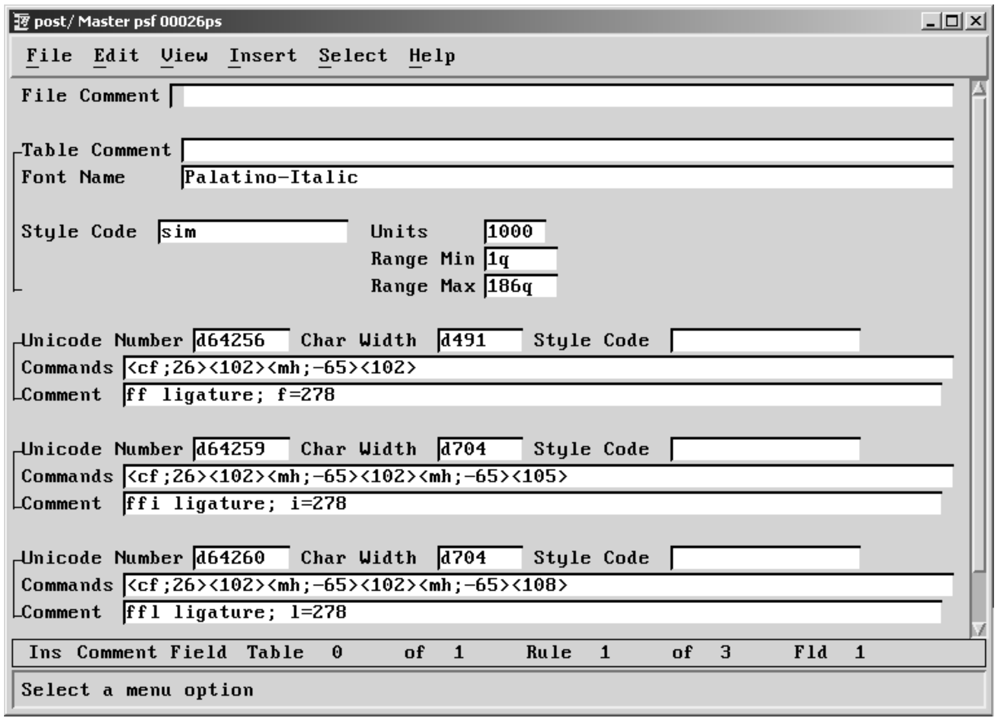
[2]Precede the converted decimal PTS code with the super-shift command ⟨41⟩; you do not have to follow the PTS code with an unshift command.

Examples of Defining Pseudo Characters

This section consists of two examples of customizing pseudo characters.

Example 1

In this example, the first rule creates an ff ligature. Some fonts may not include this ligature. (A ligature is a character or type combining two or more letters, such as fi or ffl.)



```
File Comment  

Table Comment 
Font Name      Palatino-Italic

Style Code   sim              Units      1000
                              Range Min  1q
                              Range Max  186q

Unicode Number d64256  Char Width d491  Style Code
Commands  <cf;26><102><mh;-65><102>
Comment   ff ligature; f=278

Unicode Number d64259  Char Width d704  Style Code
Commands  <cf;26><102><mh;-65><102><mh;-65><105>
Comment   ffi ligature; i=278

Unicode Number d64260  Char Width d704  Style Code
Commands  <cf;26><102><mh;-65><102><mh;-65><108>
Comment   ffl ligature; l=278

 Ins Comment Field  Table  0   of  1   Rule  1   of  3   Fld  1
 Select a menu option
```

Figure 7-2 *Examle PSF Spec*

Creating the pseudo character involves the following steps:

1. Obtain information from the PTS Spec.
2. Calculate the custom pseudo character width.
3. Write the the command field entry.
4. Assign a Unicode number.

Obtaining information from the PTS Spec

To create this ligature, look in the PTS Spec to obtain the following information for the glyph ″f″

- PTS code - 102
- Width - 278
- Font number - 26 (from the PTS Spec Typesetter Information)

Calculating the custom pseudo character width

There is no specified amount of space around the glyphs. When creating a custom pseudo character, it may take several attempts to place the component glyphs as desired.

To create the ligature, space was removed using the 〈mh〉 command. The amount of space to remove was determined through trial and error. If 〈mh;-90〉 made the two “f” glyphs collide when they were output, then lesser amounts were tried until finally the 〈mh;-65〉 was decided upon.

The pseudo character width of the ff ligature was calculated by adding the original width of each “f” for a total width of 556, then subtracting the space removed between the two glyphs, in this example -65, for a final width of 491. The value d491 is entered in the Char Width field of the PSF Spec.

(278 + 278) – 65 = 491

Writing the Commands field entry

In this example, the Font Map Number is 26 and the PTS code for “f” is decimal 102. For the *Commands* field entry, you need to do the following:

- Change to font 26.
- Specify the first “f” by its PTS code.
- Reduce space by moving back with a negative <mh> command.
- Specify the second “f” by its PTS code.

Below is the entry for the *Commands* field:

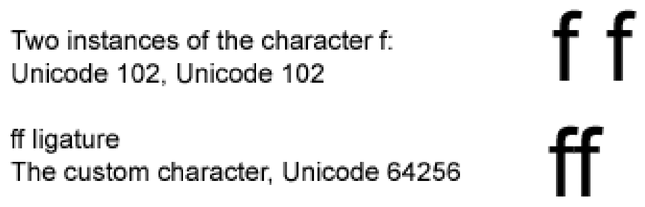
〈cf;26〉〈102〉〈mh;-65〉〈102〉

Assigning a Unicode Number

Assign a Unicode number to the new custom character. The Unicode number for this ligature is 64256 (or xFB00); enter d64256 in the *Unicode Number* field to display and output the ligature.

After you set up a PSF Spec, you must add it to the appropriate FGS

Spec(s), then you must run GenFAST before the changes take effect. For more information, refer to PostScript Name on page 8-1 and FAST Generation Exception on page 10-1.



Two instances of the character f:
Unicode 102, Unicode 102

ff ligature
The custom character, Unicode 64256

Example 2

There appears to be extra white space around the open and close parentheses in the PostScript NotoSerif-Regular font. You want to modify these glyphs to eliminate this extra space.

The line below shows the placement of the parentheses and the amount of space between the glyphs when they are set without any horizontal moves.

C (4+2)

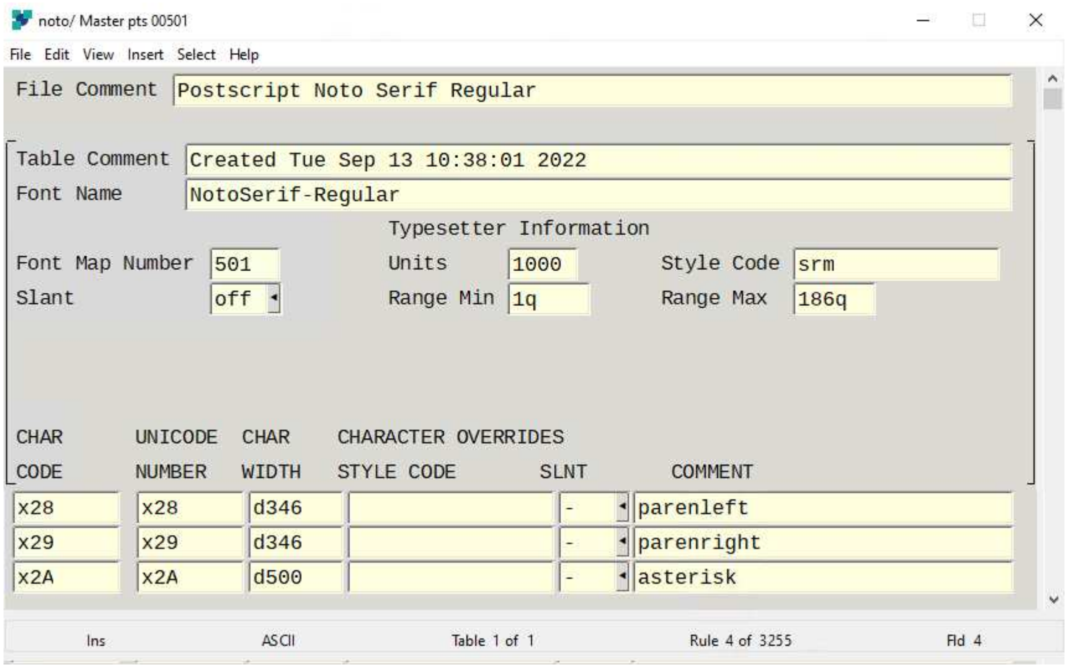
You want to modify the open and close parentheses to reduce the amount of space around them. This involves the following steps:

1. Obtain information from the PTS Spec.
2. Calculate the glyph widths.
3. Write the *Commands* field.
4. Assign Unicode numbers.

Obtaining information from the PTS Spec

To locate the values needed to customize the parentheses, look in the PTS Spec for NotoSerif-Regular (pts_00501 in the *noto* library) as follows.

1. In PathFinder, navigate to `STYLE LIBRARIES > Lnoto > Phototypesetter Specs`.
   The pts specs appear in the List Views.
2. Right-click `00501` and select `Edit`.
   XPP opens the `_pts_00501.sde` file in the Sdeditor.



File Comment: Postscript Noto Serif Regular

Table Comment: Created Tue Sep 13 10:38:01 2022

Font Name: NotoSerif-Regular

Typesetter Information

Font Map Number: 501
Units: 1000
Style Code: srm
Slant: off
Range Min: 1q
Range Max: 186q

CHAR CODE	UNICODE NUMBER	CHAR WIDTH	CHARACTER OVERRIDES STYLE CODE	SLNT	COMMENT
x28	x28	d346		-	parenleft
x29	x29	d346		-	parenright
x2A	x2A	d500		-	asterisk

Ins ASCII Table 1 of 1 Rule 4 of 3255 Fld 4

Figure 7-3 *PTS Spec: pts_00501.sde*

Gather the following information:

- PTS codes - x28 (d40) for open parenthesis, x29 (d41) for close parenthesis
- Unicode numbers - 40 (x28) for open parenthesis, 41 (x29) for close parenthesis
- widths - d346 for both glyphs
- font number - 501 (from the *Font Map Number* field of the PTS Spec)

Calculating the pseudo character widths

The original width of each glyph is 346 units. After experimenting with various values, you decide to remove 30 units of space before and after each glyph:

346 – (30 + 30) = 286

The entry in the *Character Width* fields for these pseudo characters is 286.

Note: *The font vendor does not supply information about the amount of space around the characters. When modifying a glyph, it may take several attempts to place it as desired.*

Writing the Commands field entries

In this example, the font number is 501, the PTS code for the open parenthesis is 40, and the PTS code for the close parenthesis is 41. These two character codes have special meanings in PostScript. To use them literally, they must be preceded by a backslash, which is PTS code 92. Only one other character code requires this treatment: the backslash itself (in position 92).

Therefore, the command line must do the following:

- Change to font 501.
- Reduce space before the glyph with an <mh> command.
- Enter the literal glyph by first specifying the PTS code for a backslash, then specifying the PTS code for the parenthesis.
- Reduce space after the character with another <mh> command.

Below are the two *Commands* field entries.

⟨cf;501⟩⟨mh;-30⟩⟨092⟩⟨040⟩⟨mh;-30⟩	for the open parenthesis
⟨cf;501⟩⟨mh;-30⟩⟨092⟩⟨041⟩⟨mh;-30⟩	for the close parenthesis

Note: *These entries are used only for setting the parentheses as modified pseudo characters. They do not include information on the glyphs before and after the parentheses. This was just an example; these situations are probably better handled with kerning pairs.*

Assigning Unicode Numbers

Use the same Unicode numbers that were assigned to these parentheses in the PTS Spec (40 and 41).

After you set up a PSF Spec, you must add the information to the appropriate FGS Spec(s) (in this case, _fgs_00501.sde); then, you must run GenFAST before the changes take effect. Refer to PostScript Name on page 8-1 and FAST Generation Exception on page 10-1 for more information. In this situation, where pseudo characters in the PSF Spec have the same Unicode numbers as found in the PTS Spec, list the PSF Spec before the PTS Spec in the FGS Spec.

The following example shows the placement of the characters after being modified. The extra space between the characters is reduced.

C (4+2)

Note: *If you need to adjust the placement of accents over PostScript small caps, see " Accents Over Small Caps" on page 17-1.*

Chapter 8

The PostScript Name Spec (PSN)

This chapter describes the PSN Spec, its contents, and its workings.

The PSN Spec

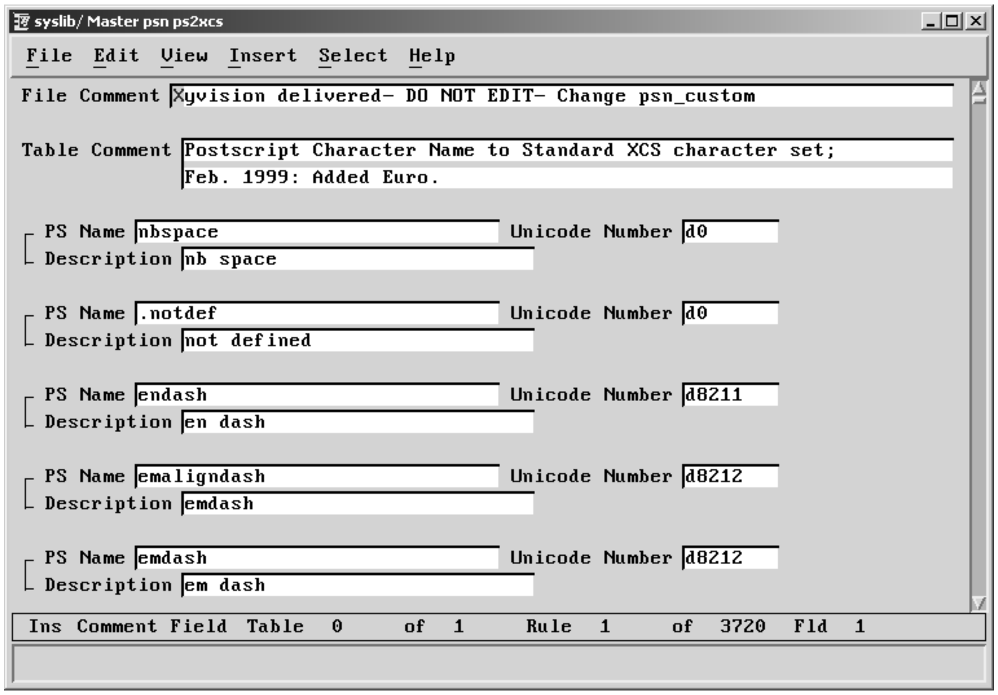
The PostScript Name (PSN) Spec maps PostScript character (or glyph) names to the Unicode numbers. Build FAST uses this information to generate FASTs only for PostScript Type 1 base fonts (not OpenType or CID fonts). You may map multiple PostScript names to the same Unicode number.

There are two PSN Specs:

- psn_ps2xcs
- psn_custom

psn_2xcs Spec

The psn_2xcs Spec contains many of the character names you need for most applications. This is a default spec and should not be edited. If XPP cannot locate a character name in *psn_custom*, it searches this spec.



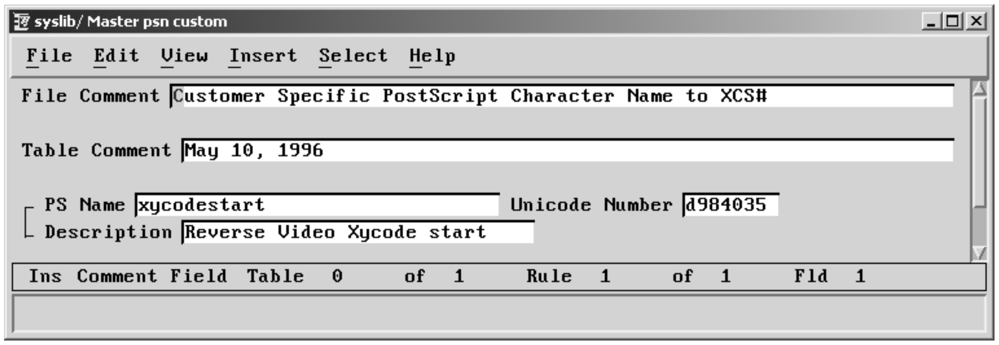
```
File Comment  Xyvision delivered- DO NOT EDIT- Change psn_custom

Table Comment Postscript Character Name to Standard XCS character set;
              Feb. 1999: Added Euro.

PS Name     nbspace              Unicode Number d0
Description nb space

PS Name     .notdef              Unicode Number d0
Description not defined

PS Name     endash               Unicode Number d8211
Description en dash

PS Name     emaligndash          Unicode Number d8212
Description emdash

PS Name     emdash               Unicode Number d8212
Description em dash

Ins Comment Field  Table  0   of  1   Rule  1   of  3720  Fld  1
```

Figure 8-1 *Top of the PostScript to Unicode, psn_ps2xcs Spec*

The psn_ps2xcs Spec contains the same fields as the psn_custom Spec. They are discussed on page 8-4.

psn_custom Spec

Build FAST searches the psn_custom Spec first. You can edit this spec and specify character names that are specific to your own site, which may include logos or other specialty characters.



```
File Comment  Customer Specific PostScript Character Name to XCS#

Table Comment  May 10, 1996

  PS Name  xycodestart                 Unicode Number  d984035
  Description  Reverse Video Xycode start
 Ins Comment Field  Table  0    of  1    Rule  1    of  1    Fld  1
```

Figure 8-2 *The PostScript to Unicode psn_custom Spec*

You would typically modify the psn_custom Spec when, in the process of loading a new font, you encounter character names in the PostScript .afm file that XPP does not recognize.

If you know what the character is, you may attempt to map it to a Unicode number that represents the same character, only under a different name.

If you do not know what glyph the character represents, or if there is no apparent existing Unicode mapping for the character, you may assign it to the Unicode private area.

Occasionally, with *Pi* fonts, characters with unintelligible names (e.g., L52368 or thang1) are characters for which XPP does not have a Unicode number (e.g., the character, horsetrail, in the Carta font). To use these characters successfully, you must map them in the psn_custom Spec.

The psn_custom Spec contains the same fields as the psn_ps2xcs Spec. They are discussed on page 8-4.

The Build FAST Process

The .afm file assigns a character code to a character/glyph. The .afm file also assigns a character name to a character.

Build FAST does the following:

- Searches the *PS Name* field in the PSN Spec for that character name, for example, **endash**.

  It searches the psn_custom Spec first. If unable to locate the character name there, it searches the psn_ps2xcs Spec.
- Copies the Unicode number and the entry in the *Description* that are associated with that character name.

  For example, the Unicode number for character name **endash** is d8211 (x2013) (which may or may not be the same as the character code in the .afm) and the entry in the *Description* field is **en dash**.
- Enters the information in the appropriate fields in the PTS Spec.

For OpenType fonts, Build FAST does not refer to the PSN Spec for information. All OpenType font information is contained in the *.otf* or *.ttf* file and the *.afm* file generated from them by Font Copy.

Accessing the PSN Spec

You access the PSN Spec (psn_ps2xcs and psn_custom) from PathFinder using the following sequence:

STYLE LIBRARIES > Lsyslib > PostScript Names

The fields are the same in both the psn_ps2xcs Spec and the psn_custom Spec.

The PSN Spec Fields

Each spec consists of one table, with one rule for each PostScript character name. The fields in the PSN Spec are:

File Comment

User-provided information about this spec, often includes information concerning the edit history of the file.

Table Comment

Generally describes the rules of the table. Maximum 7.5 lines.

PS Name

PostScript character (or glyph) name. Maximum 31 characters.

Unicode Number

Enter the Unicode number assigned to that character. Maximum eight alphanumeric characters. It must be a valid Unicode number for XPP.

Description

Description of the character. Maximum 30 characters.

Chapter 9

The FAST Generation Spec (FGS)

This chapter contains the following information on the FAST (Font Access Tables) Generation (FGS) Spec:

- Understanding the FGS Spec
- Setting up FGS Specs
- Examples of Style Code Overrides Field Entries

Understanding the FGS Spec

To create FASTs for text fonts, you set up an FGS Spec for each font that calls in usually one PTS Spec. You may also call in a PSF Spec if you defined any pseudo characters. By default, the Build FAST utility creates a PSF Spec for three ligatures, and includes this in the FGS Spec (you may enable or disable use of these ligatures through the FV Spec).

To create FASTs for *Pi* fonts, you typically set up FGS Specs that call in several PTS Specs. This gives you the flexibility of accessing characters from several fonts without having to manually change font families while editing a division.

You can reference as many as three FASTs in one rule of the Font Variant Spec. Typically, you would reference a text FAST as the primary FAST and a *Pi* FAST as the secondary FAST.

If all of the *Pi* characters are in one FAST, you can access them without changing fonts. If these characters were in separate FASTs, each tied to its own Font Variant, to access them, you would have to change fonts while editing a division.

Delivered FGS SpecsXPP delivers FGS Specs to the *noto* library, which access the standard 163 Noto OpenType fonts.

Accessing the FGS Spec

Access the FGS Spec using the following sequence:

STYLE LIBRARIES > L*library* (e.g., **Lnoto**) **> FAST Generation**

When to Edit FGS Specs

You would edit an FGS Spec when you want to create or modify a FAST. In the case of secondary FASTs, you may want to add more *Pi* fonts to the FAST. In the case of a primary FAST, which usually represents a single text font, you may want to add pseudofont characters (e.g., you have modified a copyright character so that it always outputs in superscript).

GenFAST

GenFAST is a program that you run against an FGS Spec in order to build the FAST. If you have modified a PTS or PSF Spec that is referenced in the FGS Spec, or the FGS Spec itself, you need to run GenFAST.

Running GenFAST

To run GenFAST:

1. From the PathFinder Tree View, select the following sequence:
   STYLE LIBRARIES > L*library* (e.g., **Lnoto**) **> FAST Generation**
   PathFinder displays the FGS Specs in the List View.
2. Right-click the **FGS Spec number** in the List View.
   PathFinder displays a pop-up menu.
3. Select the following sequence from the Pop-up menus:
   Tools > Generate FAST .
   XPP displays a message box when the process is complete.

Setting Up FGS Specs

The Build FAST utility creates the FGS Spec. In order to edit an FGS Spec, gather the following information.

- A list of the PTS (and, optionally, PSF) Specs that you want to include in a FAST.
- Style codes — A list of the codes for the character styles you want included in the FAST.
- *Pi* font numbering scheme — refer to Table 9-1 for an explanation of FGS Spec names for *Pi* fonts.
- Font name — the name of the font, for example, Times, Helvetica, and so on.
- Variant name — the name of the variant, for example, Bold, Medium, Italic, and so on.

Naming an FGS Spec

FGS Specs for text fonts are usually named for the FAST number specified in the Build FAST utility. An FGS Spec name consists of the _fgs_ prefix followed by a five-digit number between 00001 and 65535.

Note: *Always include leading zeros.*

The name of the FGS Spec becomes the FAST number, when you run GenFAST on it. FGS Spec names should be unique.

The FGS Spec for text fonts is typically referenced as the primary FAST number in the FV Spec. The FGS Spec for *Pi* fonts is typically referenced as the secondary FAST number in the FV Spec.

The following table explains the recommended FGS Spec names for the NotoSansSymbols fonts Secondary FASTs.

Table 9-1 *Recommended FGS Spec Naming Conventions for NotoSansSymbols Fonts Secondary FASTs*

Digit	*1 to 4*	*5*
Entries:	1000[1]	Font Weight 1 = Regular 2 = Bold 3 = Medium 4 = SemiBold 5 = ExtraBold 6 = Black 7 = ExtraLight 8 = Light 9 = Thin

[1]The 10000 range is used for the *Pi* FAST numbers to avoid conflicts with lower FAST numbers typically used for text fonts.

XPP delivers 9 *Pi* font FGS Specs for the NotoSansSymbols family to the **noto** library:

Regular fgs_10001	Black fgs_10006
Bold fgs_10002	ExtraLight fgs_10007
Medium fgs_10003	Light fgs_10008
SemiBold fgs_10004	Thin fgs_10009
ExtraBold fgs_10005	

The following table explains the recommended FGS Spec names for the deprecated Type 1 *Pi* fonts.

Table 9-2 *Recommended FGS Spec Naming Conventions for Deprecated Type 1 Fonts Secondary FASTs*

Digit 1 and 2	***3***	***4***	***5***
Entries: 10[1]	Typeface	Variant	Range[2]
	0 = serif 1 = sans serif	0 = medium 1 = bold 2 = italic 3 = bold italic 4 = lite 5 = lite italic 6 = extra lite 7 = extra lite italic 8 = extra bold 9 = extra bold italic	6 = non-ranging or range 1 7 = range 2 8 = range 3 9 = range 4

[1]The historic naming conventions that XPP uses for secondary FASTs are derived from the use of typesetting devices, most of which have been replaced by PostScript output devices. The "10" entries were above the range of font numbers on most typesetters.

[2]Older typesetters required multiple fonts to output the same character at different point size ranges.

Eight Type 1 *Pi* font FGS Specs were delivered to the deprecated **post** and **postuni** libraries:

fgs_10006	fgs_10106
fgs_10016	fgs_10116
fgs_10026	fgs_10126
fgs_10036	fgs_10136

The following table shows the recommended names for FGS Specs for the deprecated Type 1 *Pi* fonts by the style of the font. The name was selected according to the style of the *Pi* font being specified. Style codes are described later in this chapter.

Table 9-3 *Recommended FGS Spec Names for Deprecated Type 1 Pi Fonts*

Serif Pi Font Style	***FGS Spec Name***	***Sans Serif Pi Font Style***	***FGS Spec Name***
srm	10006	nrm	10106
srb	10016	nrb	10116
sim	10026	nim	10126

Table 9-3 *Recommended FGS Spec Names for Deprecated Type 1 Pi Fonts (Continued)*

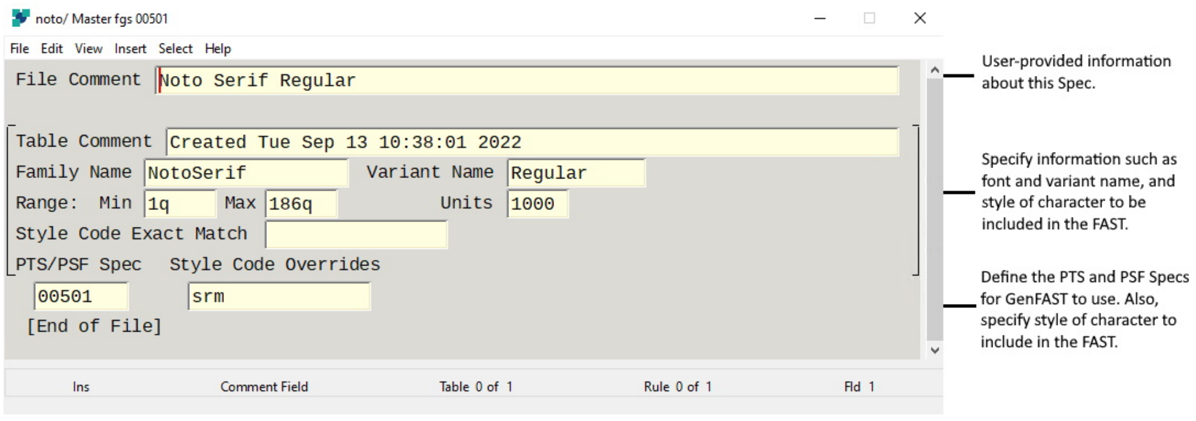
Serif Pi Font Style	*FGS Spec Name*	*Sans Serif Pi Font Style*	*FGS Spec Name*
sib	10036	nib	10136
srl	10046	nrl	10146
sil	10056	nil	10156
sre	10066	nre	10166
sie	10076	nie	10176
srx	10086	nrx	10186
six	10096	nix	10196

Structure of an FGS Spec

An FGS Spec consists of a *File Comment* field and one table with the following sections:

- Header — contains comment fields plus default information for the whole file
- Rules — contain the name of the PTS or PSF Spec you are adding to the FAST and a field for specifying style codes

The following figure shows the structure of the FGS Spec.



```
File Comment  Noto Serif Regular

Table Comment  Created Tue Sep 13 10:38:01 2022
Family Name  NotoSerif           Variant Name  Regular
Range:  Min  1q      Max  186q          Units  1000
Style Code Exact Match
PTS/PSF Spec   Style Code Overrides
  00501          srm
 [End of File]
```

User-provided information about this Spec.

Specify information such as font and variant name, and style of character to be included in the FAST.

Define the PTS and PSF Specs for GenFAST to use. Also, specify style of character to include in the FAST.

Figure 9-1 *FAST Generation Spec*

Header Fields

This section contains descriptions of the header fields and their valid entries.

Family Name

The name of the font family (e.g., NotoSerif, NotoSans, and so on), or indicate *Pi* fonts (e.g., Pi Characters.)

Do not enter font variant information in this field; enter font variant information in the *Variant Name* field.

Entry	*Description*
string	As many as 18 alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, and /), and the integers 0-9.

Variant Name

The variant name for this font family (e.g., Bold, Medium, and so on). If you have applied pseudo-slant to the glyphs in this font, enter a name that indicates the pseudo-slant. Do not enter font family information in this field; font family information goes in the *Family Name* field.

Entry	*Description*
string	As many as 12 alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, /), and the integers 0-9.

Range: Min/Max

The smallest point size or the largest point size for outputting any character in this FAST. The entry in this field must be greater than or equal to the entry in the *Range: Min* field or *Range: Max* field in all the PTS/PSF Specs listed in the FGS Spec.

Entry	*Description*
number	A positive numeric value, followed by a valid unit qualifier, in the range of 0 through 186.1q or the equivalent in p, i, m, n, c, d, k units. This field entry defaults to 1q for the min, but XPP can only output a minimum of 4.5q, and 186.1q for the max.(Technically, you can specify a smaller point size, but some things having to do with pickup placement and/or leading may not work properly.)

Units

This field controls the base units (units per em) the system uses to display all character widths when you view a FAST.

Entry	*Description*
integer	An integer in the range of 1 through 65535. This field entry defaults to 1000.

Style Code Exact Match

Using this field, you can specify glyphs you want GenFAST to pick up by their style codes. GenFAST picks up only the glyphs in the PTS and PSF Specs that *exactly* match the style codes in this field. It ignores glyphs that do not have all the attributes listed in this field; it also ignores glyphs that have attributes in addition to those listed in this field. Therefore, this field is frequently left blank.

GenFAST uses the *Exact Match* field if the *Style Code Overrides* rule field is blank. GenFAST uses the *Style Code Overrides* field if it contains an entry whether the *Exact Match* field is blank or not. If the *Exact Match* field *and* the *Style Code Overrides* rule field for all the rules are blank and there is no FAST Generation Exception Spec, GenFAST does not produce a FAST. Therefore, do not leave all *Style Code Overrides* fields blank. Typically, you would leave the *Exact Match* field blank and use the *Style Code Overrides* field to select characters.

Follow these guidelines when entering style codes in this field:

- Use *only* valid codes.
- Enter the codes in any order.
- Do not use spacebands and underbars; they are not valid entries. The system does not check for invalid codes when exiting the field, however, you cannot successfully run GenFAST if the field contains an invalid code.
- If the field contains the value ***a*** (all attributes) GenFAST picks up glyphs with an a or all style codes individually entered in the *Style Code* field of the PTS or PSF Spec.
- Use only lowercase letters for entries.

Entry	*Description*
no entry	If this field is blank (default), the *Style Code Overrides* field must contain an entry.
s	serif.
n	sans serif.
i	italic (*slanted; oblique*).

Entry	*Description*
r	roman (*upright*).
e	extra light; describes the glyph weight.
l	light; describes the glyph weight.
m	medium; describes the glyph weight.
b	bold; describes the glyph weight.
h	heavy; describes the glyph weight.
x	extra bold; describes the glyph weight.
d	inferior; describes a typeface used to set inferior text.
u	superior; describes a typeface used to set superior.
c	small caps; describes small cap glyph.
o	other; a characteristic not represented by any other style code in this list. This code is undefined until you determine the meaning. For example, you could set style code o to stand for an outline font.
a[1]	all attributes; the same as individually entering all the attribute codes listed.

[1]This style code is often used for *Pi* fonts that can be used with any style text font.

Rule Fields

This section contains descriptions of the rule fields and their valid entries.

PTS/PSF Spec

The name of the PTS or PSF Specs that contain the glyphs to include in the FAST. Enter the PTS/PSF Specs in the order of preference.

If the same Unicode Number is in multiple PTS/PSF Specs, the FAST contains the first occurrence it finds.

Entry	*Description*
string	The name of an existing PTS or PSF Spec can be as many as eight alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, and /), and the integers 0-9.

Style Code Overrides

Using this field, you can specify glyphs you want GenFAST to pick up, even though they do not match the style codes in the *Exact Match* field.

Determine the font characteristics from the PTS/PSF you want to include in the FAST and the font characteristics you want to exclude from the FAST.

Enter the style codes for these characteristics in the format (without any space before or after the hyphen):
`codes you want - codes you do not want`

Of the codes specified before the hyphen, GenFAST picks up the glyphs having at least these style codes; they may have additional style codes. For the codes specified after the hyphen, GenFAST ignores glyphs having any of these style codes even if they have any of the style codes specified before the hyphen.

The field does not have to contain style codes that both include and exclude glyphs from the FAST. If you enter only codes to include a glyph in a FAST, do not enter the hyphen. Likewise, you can enter only codes that will exclude a glyph—enter a hyphen followed by the codes; do not enter any codes before the hyphen.

If you enter a value of **a** (for all attributes) before the hyphen, GenFAST picks up all glyphs in the specified PTS/PSF Spec. If you enter a value of **a** after the hyphen, GenFAST picks up glyphs in the PTS/PSF with no style codes specified.

GenFAST uses the *Style Code Overrides* field if it contains an entry whether the *Exact Match* field is blank or not. If the *Style Code Overrides* field is blank, the system uses the entry in the *Exact Match* field. If the *Style Code Overrides* fields for all the rules *and* the *Exact Match* header field are blank, and there is no FAST Generation Exception Spec, GenFAST does not produce a FAST.

Entry	*Description*
style code(s)	Up to 15 style codes and the optional hyphen (-). Entries that are valid for the *Exact Match* field are also valid for this field. Refer to the description of the *Exact Match* field entries.
no entry	If this field is blank (default), GenFAST uses the Exact Match header field entry. If both this field and the Exact Match field are blank, and there is no FAST Generation Exception Spec, GenFAST does not produce a FAST.

Examples of Style Code Overrides Field Entries

This section consists of two examples of how the system uses the entries in the *Style Code Overrides* field.

Example 1

The *Style Code Overrides* field contains **srm**.
GenFAST does the following:

- Searches the PTS or PSF Spec listed in the FGS Spec.
- Picks up glyphs having the style codes **srm** (either in the header or the rule field).

If the header or the rule field contains other style codes in addition to **srm** (e.g., **usrm**), GenFAST still picks up the glyphs.

Example2

The *Style Code Overrides* field contains **srm-u**.
GenFAST does the following:

- Searches the PTS or PSF Specs listed in the FGS Spec.
- Picks up glyphs having the style codes **srm** but **not** the style code **u** (either in the header or the rule field):
  - If the PTS/PSF rule field contains **usrm** or **ucsrm**, GenFAST does **not** pick up the glyph, even though it matches the style codes **srm**.
  - If the *Style Code* header field in the PTS/PSF Spec contains **usrm** or **ucsrm**, GenFAST does **not** pick up any glyphs in that PTS/PSF because of the **u**.

Chapter 10

The FAST Generation Exception Spec (FGX)

This chapter contains the following information on the FAST Generation Exception (FGX) Spec:

- Understanding the FGX Spec
- Setting up FGX Specs

Understanding the FGX Spec

Using a FAST Generation Exception (FGX) Spec, you can include glyphs in a Font Access Table (FAST) Spec that are not in any of the Phototypesetter (PTS) or Pseudofont (PSF) Specs specified in the FAST Generation (FGS) Spec. These glyphs are additions to the glyphs in the FGS Spec that has the same name. FGX Specs are optional.

When you run GenFAST against a specified FGS Spec, it also runs against the FGX Spec of the same name—if one exists.

You can also specify glyphs that are exceptions to the glyphs in the specified PTS/PSF Spec. By using the same Unicode number, but different glyph information in the fields, you can include an exception glyph.

For example, you are using Century Schoolbook, but you want Century Old Style numbers. Using the FGX Spec, you can overwrite the Century Schoolbook numbers in the PTS with the Century Old Style numbers in the FGX, and thus in the FAST.

In another case, GenFAST included a trademark symbol with Unicode number x2122 from a PTS Spec. You prefer the trademark symbol from another font, so you include a rule in the FGX Spec for the trademark symbol you like. When processing the FGX Spec, GenFAST finds the trademark symbol with Unicode number x2122 and overwrites the trademark symbol from the PTS Spec.

Note: *Since GenFAST reads the FGX Spec last, glyphs in the FGX Spec overwrite existing glyphs in the FAST with the same Unicode numbers.*

Alternatively, you could achieve the same result by creating another PTS Spec with the additional glyphs, and listing that PTS Spec earlier in the FGS Spec. In this case, the order in which the PTS Specs are listed makes a difference.

Accessing the FAST Generation Exception Spec

You access the FAST Generation Exception (FGX) Spec using the following sequence in the PathFinder Tree View:

STYLE LIBRARIES > L*fontlib* (e.g., **Lnoto**) **> FAST Generation Exceptions**

Setting Up FGX Specs

Before editing an FGX Spec, get a printout of the PTS Spec(s) containing information on the exception glyph(s) you want to include. You can refer to it for information such as character codes, Unicode numbers, and glyph widths.

Naming an FGX Spec

When you create an FGX Spec, you assign it a name consisting of the _fgx_ prefix followed by a five-digit name between 00001 and 65535. Note that leading zeros must be included.

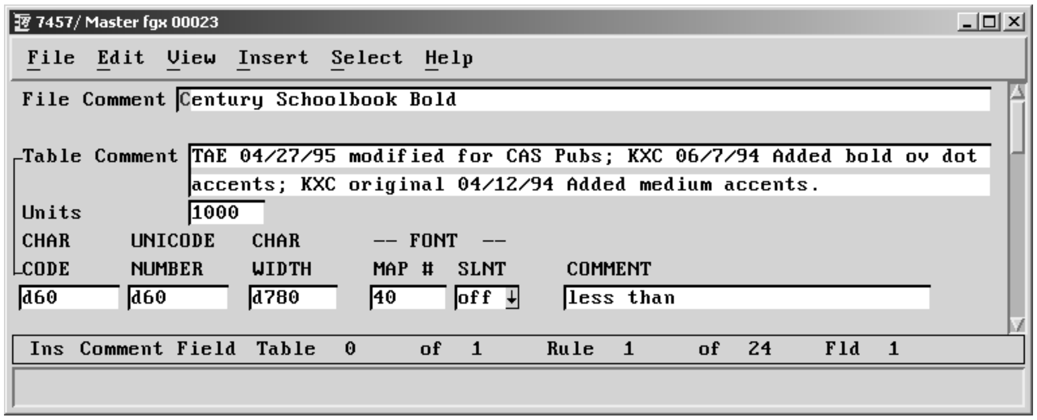
The FGX Spec *must* have the same name as the FGS Spec, for example, _fgx_00015.sde and _fgs_00015.sde. GenFAST looks in the specified font library and processes only the FGX Spec with the same name as the specified FGS Spec.

Structure of an FGX Spec

An FGX Spec consists of a *File Comment* field and one table with the following sections:

- Header — contains *Table Comment* and *Units* fields.
- Rules — contain information about each exception glyph in the FAST. There is one rule for each exception glyph.

The following figure shows the structure of the FGX Spec.



```
File Comment  Century Schoolbook Bold

Table Comment  TAE 04/27/95 modified for CAS Pubs; KXC 06/7/94 Added bold ov dot
               accents; KXC original 04/12/94 Added medium accents.
Units          1000
CHAR     UNICODE   CHAR      -- FONT --
CODE     NUMBER    WIDTH     MAP #  SLNT    COMMENT
d60      d60       d780      40     off     less than

 Ins Comment Field  Table  0    of  1    Rule  1    of  24    Fld  1
```

Figure 10-1 *FAST Generation Exception Spec*

Note: *The Comment field has been reduced to produce this image. The field can contain 30 characters on one line.*

Header Fields

This section contains descriptions and valid entries for the header fields.

Units

PostScript width values are specified relative to 1000 units per em. This value is the same as the value in the *Units* field of the PTS Spec.

Entry	*Description*
integer	An integer in the range of 1 through 65535. This field entry defaults to 1000.

Rule Fields

The FGX Spec contains a rule for each exception glyph in the font. Do not leave any field blank. Fields that accept values in decimal, octal, and hexadecimal default to d0, o0, or x0 depending on the current notation.

Char Code

Refer to the Char Code description on page 6-9 for valid entries.

Unicode Number

Enter the Unicode number assigned to this glyph.

Character Width

Refer to the Character Width description on page 6-10 for valid entries.

Font Map

Refer to the Font Map Number description on page 6-6 for valid entries.

Font Slant

Refer to the Font Slant description on page 6-6 for valid entries.

Comment

A comment pertaining to the character accessed by this rule.

Entry	*Description*
string	A comment as long as 30 alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, and /), and the integers 0-9.

Chapter 11

Creating and Viewing FASTs

This chapter contains information on the following topics:

- The FAST generation process (GenFAST)
- How GenFAST reads the font specs
- Running GenFAST
- Font Access Tables (FASTs)
- Listing all FASTs in a font library
- Verifying correct widths in FASTs

The FAST Generation Process (GenFAST)

The GenFAST program creates a FAST by combining information from the PTS and PSF (if any) Specs referenced in the specified FGS Spec as well as any FGX Spec that matches the name of the FGS Spec. GenFAST reads the specs, selects the Unicode characters, and puts the information in the FAST in order by Unicode number.

The following figure shows the GenFAST process of reading the font specs for information and building the FAST. For a detailed description of how GenFAST reads each of the specs, refer to the section “How GenFAST Reads the Font Specs” on page 11-3.

When Do I Run GenFAST?

Run GenFAST after editing any one of the following specs:

- An FGS Spec
- Any PTS Spec or PSF Spec listed in an FGS Spec
- An FGX Spec that matches the name of the FGS Spec

Note: *To add a new supported font, use the Build FAST utility, which automatically creates all the necessary font files and runs GenFAST for you. For more information about Build FAST, refer to page 1-10.*

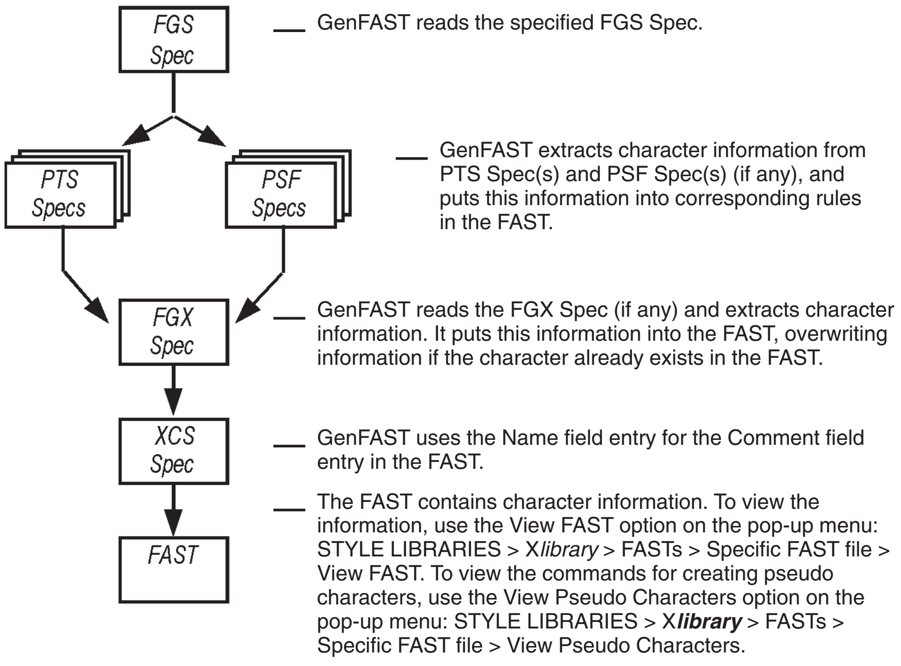

FGS Spec
PTS Specs
PSF Specs
FGX Spec
XCS Spec
FAST
GenFAST reads the specified FGS Spec.
GenFAST extracts character information from PTS Spec(s) and PSF Spec(s) (if any), and puts this information into corresponding rules in the FAST.
GenFAST reads the FGX Spec (if any) and extracts character information. It puts this information into the FAST, overwriting information if the character already exists in the FAST.
GenFAST uses the Name field entry for the Comment field entry in the FAST.
The FAST contains character information. To view the information, use the View FAST option on the pop-up menu: STYLE LIBRARIES > X*library* > FASTs > Specific FAST file > View FAST. To view the commands for creating pseudo characters, use the View Pseudo Characters option on the pop-up menu: STYLE LIBRARIES > X***library*** > FASTs > Specific FAST file > View Pseudo Characters.

Figure 11-1 *The GenFAST Process*

How GenFAST Reads the Font Specs

GenFAST processes the entries in the FGS, PTS, PSF, and FGX Specs in a particular order. This section explains how GenFAST processes the PTS and PSF Specs and their entries so you can determine the order in which to list them in the FGS Spec.

The following figure shows the order in which GenFAST reads the font specs.

The following sections describe the phases shown in the figure above.

Phase 1

GenFAST performs the following steps during Phase 1:

1. GenFAST reads the FGS Spec from top to bottom, reading the first rule in the spec and processing the referenced PTS or PSF Spec.

   Note: *The FGS Spec can reference PTS and PSF Specs and GenFAST can reference FGX Specs that live in related numbered libraries (e.g., noto, noto1, noto2). GenFAST also looks in the related numbered font libraries for the specs.*

2. GenFAST processes the PTS/PSF Spec from top to bottom and adds

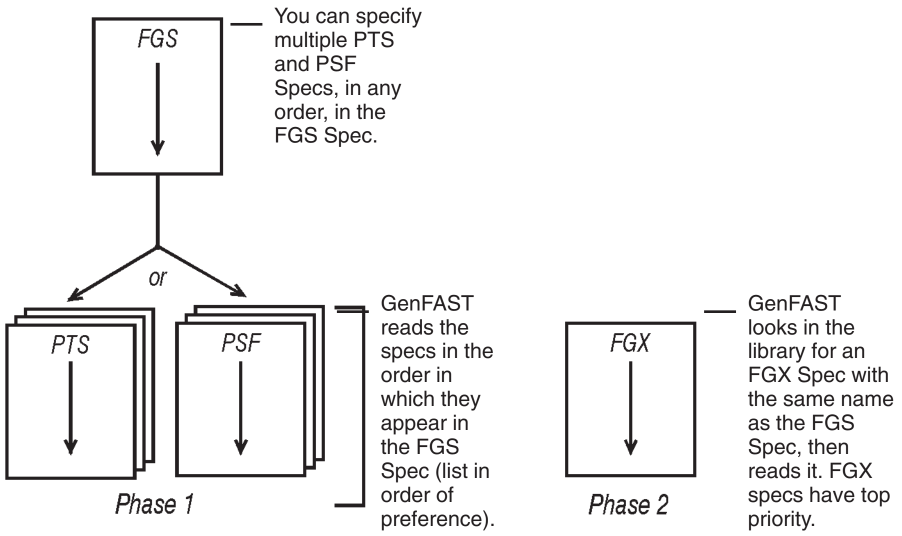

FGS
You can specify multiple PTS and PSF Specs, in any order, in the FGS Spec.
or
PTS
PSF
GenFAST reads the specs in the order in which they appear in the FGS Spec (list in order of preference).
Phase 1
FGX
GenFAST looks in the library for an FGX Spec with the same name as the FGS Spec, then reads it. FGX specs have top priority.
Phase 2

Figure 11-2 *How GenFAST Reads the Font Specs*

Unicode characters to the FAST, in order by Unicode number, to the FAST if one of the following criteria is met:

- The style of the glyph defined by this Unicode number matches the style specified in the *Style Code Overrides* field in the FGS Spec.
- If the *Style Code Overrides* field is blank, the style of the glyph defined by this Unicode number matches the style specified in the *Exact Match* field.

If both the *Style Code Overrides* and *Exact Match* fields are blank, GenFAST does not add any of the Unicode characters in the specified PTS/PSF to the FAST.

GenFAST adds Unicode characters that match the specified style code(s) to the FAST.

If GenFAST finds a duplicate Unicode number that also matches the specified style code(s) in the *same* PTS or PSF Spec, it overwrites the existing information in the FAST with the new information. If GenFAST finds a duplicate Unicode number that does not match the specified style code(s), it ignores it.

Be careful when repeating Unicode numbers in a PTS/PSF Spec: you may not get the glyph you expected in the FAST. You can have two rules specifying the same Unicode number, but with different style codes. For example, you may have a serif and a sans serif copyright glyph in one font. In the PTS Spec, they have the same Unicode number, but different style codes.

3. GenFAST then goes back to the FGS Spec, reads the next rule, and processes the referenced PTS or PSF Spec. If GenFAST finds a Unicode

character that is already in the FAST (from a *previous* PTS/PSF Spec), it ignores it. Therefore, it is important to list the PTS and PSF Specs in order of preference.

4. GenFAST continues reading the FGS Spec, processing the specs, until it has processed the last FGS Spec rule. It then goes on to Phase 2.

Phase 2

GenFAST performs the following steps during Phase 2:

1. GenFAST looks in the specified font library for an FGX Spec with the same name as the FGS Spec. If the library contains an FGX Spec, GenFAST processes it. If the library does not contain an FGX Spec, the GenFAST process is complete.

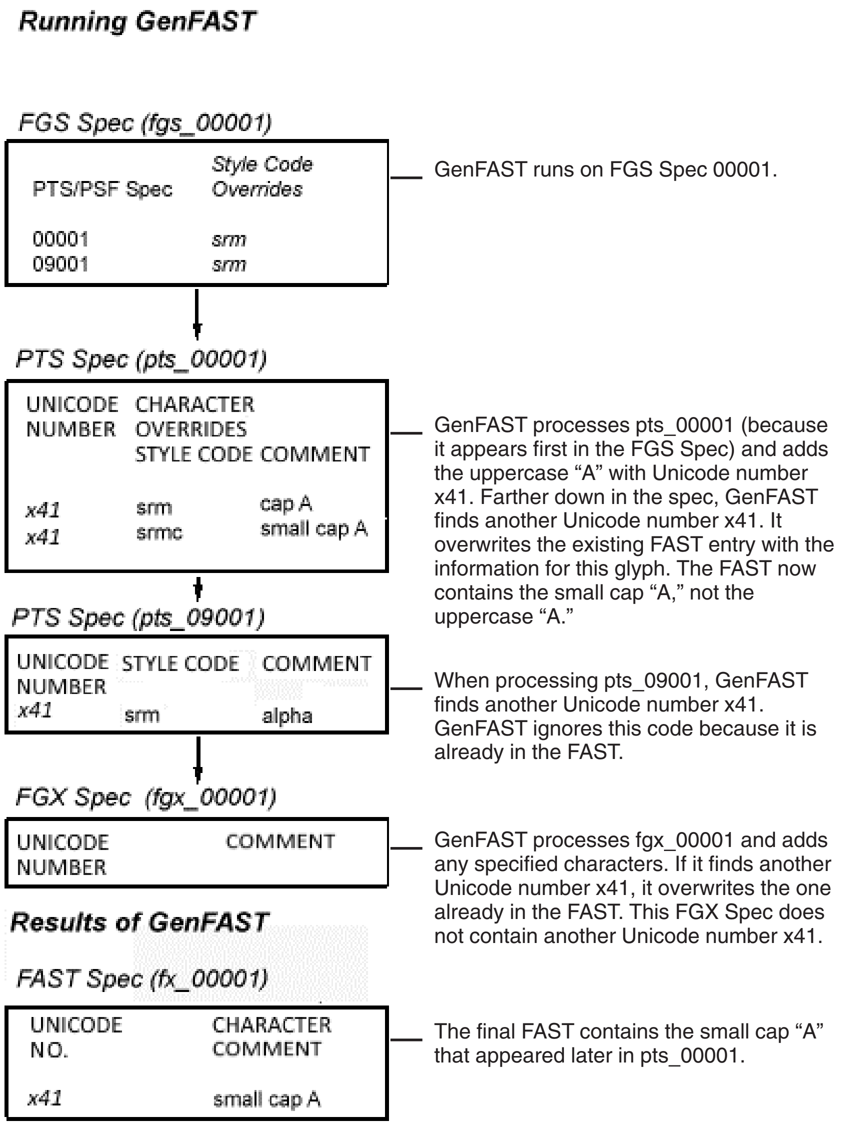
2. GenFAST processes the FGX Spec from top to bottom, adding or replacing the specified Unicode characters in the FAST.

   GenFAST adds the FGX Unicode characters to the FAST independent of style codes, that is, there are no style codes to match. If GenFAST finds a Unicode number that was already selected from a PTS or PSF, it overwrites it. Exception Unicode characters defined in the FGX Spec supersede the Unicode characters specified in the PTS/PSF Specs.

Example of GenFAST

GenFAST is run on the fgs_00001 Spec containing rules specifying PTS Specs 00001 and 09001, both with **srm** in the *Style Code* field in the Typesetter Information section.

The following figure shows how GenFAST processes the specs and adds Unicode characters to the FAST.



FGS Spec (fgs_00001)

PTS/PSF Spec	*Style Code Overrides*
00001	*srm*
09001	*srm*

GenFAST runs on FGS Spec 00001.

PTS Spec (pts_00001)

UNICODE NUMBER	CHARACTER OVERRIDES STYLE CODE	COMMENT
x41	srm	cap A
x41	srmc	small cap A

GenFAST processes pts_00001 (because it appears first in the FGS Spec) and adds the uppercase “A” with Unicode number x41. Farther down in the spec, GenFAST finds another Unicode number x41. It overwrites the existing FAST entry with the information for this glyph. The FAST now contains the small cap “A,” not the uppercase “A.”

PTS Spec (pts_09001)

UNICODE NUMBER	STYLE CODE	COMMENT
x41	srm	alpha

When processing pts_09001, GenFAST finds another Unicode number x41. GenFAST ignores this code because it is already in the FAST.

FGX Spec (fgx_00001)

UNICODE NUMBER	COMMENT

GenFAST processes fgx_00001 and adds any specified characters. If it finds another Unicode number x41, it overwrites the one already in the FAST. This FGX Spec does not contain another Unicode number x41.

FAST Spec (fx_00001)

UNICODE NO.	CHARACTER COMMENT
x41	small cap A

The final FAST contains the small cap “A” that appeared later in pts_00001.

Figure 11-3 *GenFAST Example*

Running GenFAST

This section describes the tasks to complete and the steps to perform before running GenFAST. Read both sections before attempting to run GenFAST.

Note: *The Build FAST utility sets up all the necessary specs and runs GenFAST for you automatically. If you are installing new fonts, use the Build FAST utility (refer to page 1-10 for information).*

Before running GenFAST, ensure that you have set up the specs needed to generate a FAST:

- PTS Spec(s)
- FGS Spec(s)
- PSF Spec(s) (optional — to manipulate the output of a glyph or multiple glyphs)
- FGX Spec(s) (optional — to specifically include glyphs that otherwise would not be in the FAST)

To run GenFAST from PathFinder:

1. Select **STYLE LIBRARIES > L***library* (e.g., **Lnoto**) **> FAST Generation** in the Tree View.
   PathFinder displays the FGS Specs in the List View.
2. Right-click the FGS Spec you want to process.
   PathFinder displays a pop-up menu.
3. Select **Tools > Generate FAST**.

Caution When you start GenFAST, the program immediately overwrites any existing FAST with the same number in the font library.

GenFAST reads the FGS Spec and creates the FAST using the information in the font specs in the specified source library. It then processes the FGX Spec (if any).

The resulting FAST goes in a destination library (an "X" library) named for the source library (e.g., Xnoto). If you are using numbered source libraries (e.g., noto, noto1, noto2, etc.), GenFAST places the FAST in the destination library named for the main source library (e.g., noto). A FAST has the same name as the FGS Spec from which it was created. Refer to "Source and Destination Libraries" on page 5-2 for information on the full path names of source and destination libraries.

For example, you can run GenFAST in *Lnoto1* using the fgs_00100 Spec. GenFAST stores the resulting FAST, named fx_00100, in *Xnoto*, the destination library.

Font Access Tables (FASTs)

A FAST is a machine-readable file that contains the glyph width information needed for composition.

You cannot directly edit a FAST. You can create a view-only version of the FAST. The view-only version of a FAST is known as the VFX Spec. A FAST also contains the commands from PSF Specs (if any) that were used to modify glyphs; however, you cannot see them when you view the VFX Spec. To view those commands, you must generate and view the VPX (View Pseudo FAST) Spec.

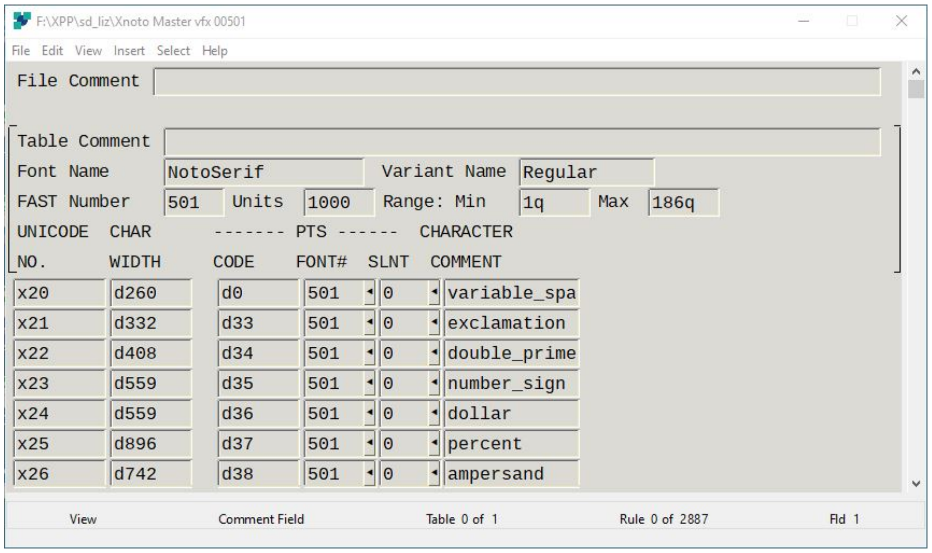
Viewing Characters in a FAST

To view the characters in a FAST from the PathFinder:

1. Navigate to **STYLE LIBRARIES > X***library* (e.g., **Xnoto**) **> FASTs**
   PathFinder displays the FASTs in the List View that are available for that font library.

2. Right-click a specific **FAST**.
   PathFinder displays a pop-up menu.

3. Select **View FAST.**
   PathFinder generates and displays the vfx *FAST name* in the Sdeditor.



F:\XPP\sd_liz\Xnoto Master vfx 00501

File Comment

Table Comment
Font Name NotoSerif Variant Name Regular
FAST Number 501 Units 1000 Range: Min 1q Max 186q

UNICODE NO.	CHAR WIDTH	CODE	PTS FONT#	SLNT	CHARACTER COMMENT
x20	d260	d0	501	0	variable_spa
x21	d332	d33	501	0	exclamation
x22	d408	d34	501	0	double_prime
x23	d559	d35	501	0	number_sign
x24	d559	d36	501	0	dollar
x25	d896	d37	501	0	percent
x26	d742	d38	501	0	ampersand

View   Comment Field   Table 0 of 1   Rule 0 of 2887   Fld 1

Figure 11-4 *Sample of FAST (VFX Spec) 00501—NotoSerif-Regular*

Viewing Pseudo Characters in a FAST

If you modify any glyphs using a PSF Spec, GenFAST puts the modifications into the FAST. You can view the relevant Unicode characters with their modifications only by generating and viewing a View Pseudo

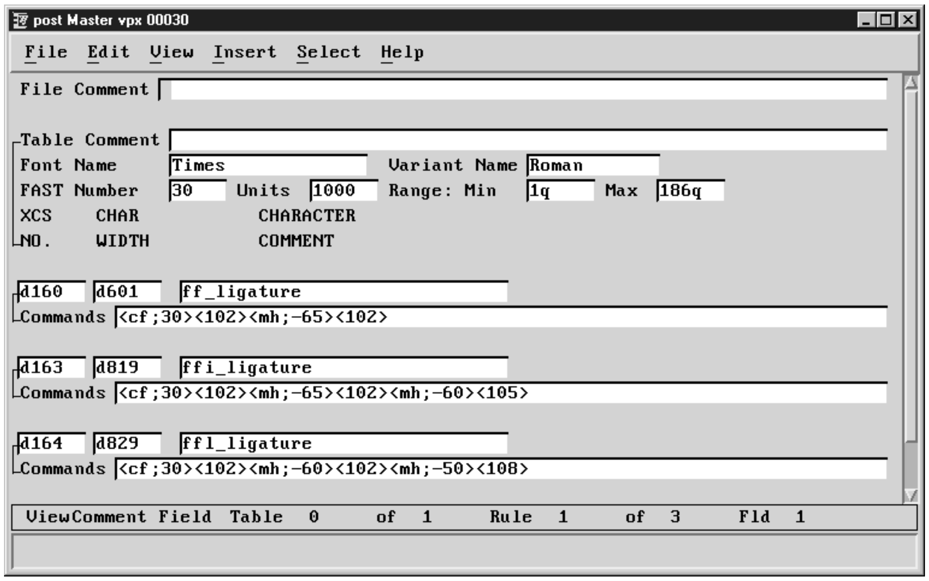
FAST (VPX) Spec. You cannot see the modifications when viewing a View FAST (VFX) Spec. Like the VFX (View FAST) Spec, you cannot edit this View Pseudo FAST (VPX) Spec.

When you generate the VPX Spec from PathFinder, it does not remain on the system; it only exists while you are viewing or printing it. You can create a permanent VPX Spec by running the Viewpx program from the command line on a FAST. For information on the Viewpx program, refer to the *XML Professional Publisher: Command Line Utilities* or type the following at the command line:

xyhelp viewpx

To view the pseudo characters in a FAST from PathFinder:

1. Follow steps 1-2 for viewing the regular characters in a FAST on the previous page.
2. Select **View Pseudo Characters** from the pop-up menu. PathFinder displays the vpx *FAST name* in the Sdeditor.



```
File Comment
Table Comment
Font Name     Times          Variant Name Roman
FAST Number   30   Units 1000  Range: Min  1q   Max 186q
XCS    CHAR          CHARACTER
NO.    WIDTH         COMMENT

d160   d601   ff_ligature
Commands <cf;30><102><mh;-65><102>

d163   d819   ffi_ligature
Commands <cf;30><102><mh;-65><102><mh;-60><105>

d164   d829   ffl_ligature
Commands <cf;30><102><mh;-60><102><mh;-50><108>

ViewComment Field  Table  0   of  1   Rule  1   of  3   Fld  1
```

Figure 11-5 *Sample of View Pseudo FAST (VPX Spec) 00030—Times Roman*

Listing All FASTs in a Font Library

The *font descriptor file* contains a list of the FASTs, by name and number, that are available in a specified font library. The file is located in the destination (X) library. If the font_desc file does not already exist, *listfx* creates it.

Enter the following commands at the command line to create or update the font descriptor file in a specific library and view its contents:

1. Change to the **X***library* directory where you want to create the file.
2. Enter the following command:

   listfx –desc

   This command creates or updates a file named *font***_***desc*.

To view the contents of the file, open it with an editor such as Wordpad.

The following example shows the font descriptor file for the *noto* library. The pathname of the file is:

XYV_STYLES/Xnoto/font_desc

Table 11-1 *Font Description File for the Xnoto Library*

NotoSerif	Regular	00501
NotoSerif	Bold	00502
NotoSerif	Italic	00503
NotoSerif	BoldItalic	00504
NotoSerif	Medium	00505
NotoSerif	SemiBold	00506
NotoSerif	MediumItalic	00507
NotoSerif	SemiBoldItal	00508
NotoSerif	ExtraBold	00509
NotoSerif	ExtraBoldIta	00510
NotoSerif	Black	00511
NotoSerif	BlackItalic	00512
NotoSerif	ExtraLight	00513
NotoSerif	ExtraLightIt	00514
NotoSerif	Light	00515
NotoSerif	LightItalic	00516
NotoSerif	Thin	00517
NotoSerif	ThinItalic	00518
NotoSansMono	Regular	00521
NotoSansMono	Bold	00522
NotoSansMono	Medium	00523
NotoSansMono	SemiBold	00524
NotoSansMono	ExtraBold	00525
NotoSansMono	Black	00526
NotoSansMono	ExtraLight	00527
NotoSansMono	Light	00528
NotoSansMono	Thin	00529
NotoSans	Regular	00551
NotoSans	Bold	00552
NotoSans	Italic	00553

Table 11-1 *Font Description File for the Xnoto Library (Continued)*

NotoSans	BoldItalic	00554
NotoSans	Medium	00555
NotoSans	SemiBold	00556
NotoSans	MediumItalic	00557
NotoSans	SemiBoldItal	00558
NotoSans	ExtraBold	00559
NotoSans	ExtraBoldIta	00560
NotoSans	Black	00561
NotoSans	BlackItalic	00562
NotoSans	ExtraLight	00563
NotoSans	ExtraLightIt	00564
NotoSans	Light	00565
NotoSans	LightItalic	00566
NotoSans	Thin	00567
NotoSans	ThinItalic	00568
NotoSansArabic	Regular	00570
NotoSansArabic	Bold	00571
NotoSansArabic	Medium	00572
NotoSansArabic	SemiBold	00573
NotoSansArabic	ExtraBold	00574
NotoSansArabic	Black	00575
NotoSansArabic	ExtraLight	00576
NotoSansArabic	Light	00577
NotoSansArabic	Thin	00578
NotoSansHebrew	Regular	00580
NotoSansHebrew	Bold	00581
NotoSansHebrew	Medium	00582
NotoSansHebrew	SemiBold	00583
NotoSansHebrew	ExtraBold	00584
NotoSansHebrew	Black	00585
NotoSansHebrew	ExtraLight	00586
NotoSansHebrew	Light	00587
NotoSansHebrew	Thin	00588
NotoSansThai	Regular	00590
NotoSansThai	Bold	00591
NotoSansThai	Medium	00592
NotoSansThai	SemiBold	00593
NotoSansThai	ExtraBold	00594
NotoSansThai	Black	00595
NotoSansThai	ExtraLight	00596
NotoSansThai	Light	00597
NotoSansThai	Thin	00598
NotoSansCJKhk	Regular	00600
NotoSansCJKhk	Bold	00601
NotoSansCJKhk	Medium	00602
NotoSansCJKhk	DemiLight	00603
NotoSansCJKhk	Black	00606
NotoSansCJKhk	Light	00607
NotoSansCJKhk	Thin	00608
NotoSansCJKjp	Regular	00610
NotoSansCJKjp	Bold	00611
NotoSansCJKjp	Medium	00612
NotoSansCJKjp	DemiLight	00613
NotoSansCJKjp	Black	00616
NotoSansCJKjp	Light	00617
NotoSansCJKjp	Thin	00618
NotoSansCJKkr	Regular	00620
NotoSansCJKkr	Bold	00621
NotoSansCJKkr	Medium	00622

Table 11-1 *Font Description File for the Xnoto Library (Continued)*

NotoSansCJKkr	DemiLight	00623
NotoSansCJKkr	Black	00626
NotoSansCJKkr	Light	00627
NotoSansCJKkr	Thin	00628
NotoSansCJKsc	Regular	00630
NotoSansCJKsc	Bold	00631
NotoSansCJKsc	Medium	00632
NotoSansCJKsc	DemiLight	00633
NotoSansCJKsc	Black	00636
NotoSansCJKsc	Light	00637
NotoSansCJKsc	Thin	00638
NotoSansCJKtc	Regular	00640
NotoSansCJKtc	Bold	00641
NotoSansCJKtc	Medium	00642
NotoSansCJKtc	DemiLight	00643
NotoSansCJKtc	Black	00646
NotoSansCJKtc	Light	00647
NotoSansCJKtc	Thin	00648
NotoSerif	Condensed	00701
NotoSerif	CondensedBol	00702
NotoSerif	CondensedIta	00703
NotoSerif	CondensedBol	00704
NotoSerif	CondensedMed	00705
NotoSerif	CondensedMed	00706
NotoSerif	CondensedSem	00707
NotoSerif	CondensedSem	00708
NotoSerif	CondensedExt	00709
NotoSerif	CondensedExt	00710
NotoSerif	CondensedBla	00711
NotoSerif	CondensedBla	00712
NotoSerif	CondensedExt	00713
NotoSerif	CondensedExt	00714
NotoSerif	CondensedLig	00715
NotoSerif	CondensedLig	00716
NotoSerif	CondensedThi	00717
NotoSerif	CondensedThi	00718
NotoSansMono	Condensed	00721
NotoSansMono	CondensedBol	00722
NotoSansMono	CondensedMed	00723
NotoSansMono	CondensedSem	00724
NotoSansMono	CondensedExt	00725
NotoSansMono	CondensedBla	00726
NotoSansMono	CondensedExt	00727
NotoSansMono	CondensedLig	00728
NotoSansMono	CondensedThi	00729
NotoSans	Condensed	00751
NotoSans	CondensedBol	00752
NotoSans	CondensedIta	00753
NotoSans	CondensedBol	00754
NotoSans	CondensedMed	00755
NotoSans	CondensedMed	00756
NotoSans	CondensedSem	00757
NotoSans	CondensedSem	00758
NotoSans	CondensedExt	00759
NotoSans	CondensedExt	00760
NotoSans	CondensedBla	00761
NotoSans	CondensedBla	00762
NotoSans	CondensedExt	00763
NotoSans	CondensedExt	00764

Table 11-1 *Font Description File for the Xnoto Library (Continued)*

NotoSans	CondensedLig	00765
NotoSans	CondensedLig	00766
NotoSans	CondensedThi	00767
NotoSans	CondensedThi	00768
NotoSansSymbols	Regular	10001
NotoSansSymbols	Bold	10002
NotoSansSymbols	Medium	10003
NotoSansSymbols	SemiBold	10004
NotoSansSymbols	ExtraBold	10005
NotoSansSymbols	Black	10006
NotoSansSymbols	ExtraLight	10007
NotoSansSymbols	Light	10008
NotoSansSymbols	Thin	10009
PiCharacters	XPPOne&Two	30020
XPPOne	Roman	30021
XPPTwo	Roman	30022

Note: *The font descriptor file lists FAST numbers, not font numbers.*

Verifying Correct Widths in FASTs

XPP delivers a font width test utility, *fwtest.pl*, that tests FASTs for correct widths. This utility creates and composes a division for each FAST. Each division contains all glyphs in the FAST in order, according to the Unicode number. You can determine whether any font width errors exist by viewing the division.

Before Running the Font Width Test Utility

If you need a list of all the FASTs that are available for a specific font library, print the *font_desc* file. Information for running and locating the file is on page 11-11.

Running the Font Width Test Utility on a Single FAST

To run the font width utility on a single FAST:

1. Navigate to **STYLE LIBRARIES > X***library* (e.g., **Xnoto**) **> FASTs**
   PathFinder displays the FASTs in the List View that are available for that font library.

2. Right-click a specific **FAST**.
   PathFinder displays a pop-up menu.

3. Select **Width Test**.
   The Font Width utility displays a Font Width Test pop-up list box with the specific FAST already selected.

4. Click the **Cancel** button to cancel the test.
   —or—
   Click the **OK** button to continue the font width test.
   The Font Width utility displays the following message:
   Automatically overwrite Division(s)?

5. Click the **Cancel** button to cancel the test.
   —or—
   Click the **Yes** button to continue the font width test.
   —or—
   Click the **No** button to continue the font width test. If the output division already exists PathFinder will display a prompt asking to replace the output division or not.
   The font width utility displays the following messages:

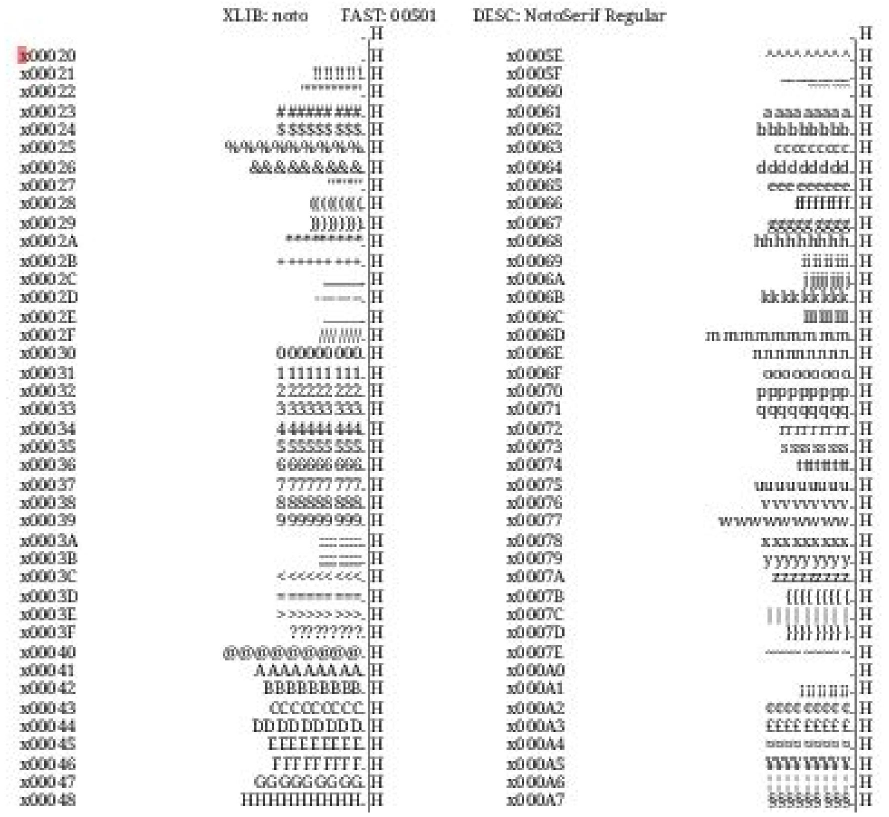
   ```
   Please wait...processing fontname.
   Width test completed
   ```

   followed by the location of the division. (Each division is listed as

DIV_fontnumber under the first document root handle, then CLS_xpputils/GRP_fwtest/JOB_*libraryname*.)

```
Do you wish to examine?
```

6. Click the **No** button.
   The information remains in that location for you to examine at another time.
   —or—
   Click the **Yes** button.
   XPP displays the division in the XyView.



```
                 XLIB: noto     FAST: 00501     DESC: NotoSerif Regular
                                .|H                                        .|H
x00020                          .|H      x0005E              ^^^^ ^^^^ ^.|H
x00021                 !!!!!!!!!.|H      x0005F              ________ _.|H
x00022                """"""""".|H      x00060              `````````.|H
x00023             ######## ###.|H      x00061             a aaa aaaa a.|H
x00024             $ $$$$$ $$$.|H      x00062             bbbbbbbbbb.|H
x00025          %%%%%%%%%%.|H      x00063             cccccccc.|H
x00026           &&&&&&&&&.|H      x00064             dddddddddd.|H
x00027                 '''''''''.|H      x00065             eee eeeeee.|H
x00028                ((((((((((.|H      x00066                ffffffff.|H
x00029                )))))))))).|H      x00067             gggggg gggg.|H
x0002A             **********.|H      x00068             hhhhhhhhhh.|H
x0002B             + +++++ +++.|H      x00069                iiiiiiii.|H
x0002C                 ,,,,,,,,,.|H      x0006A                jjjjjjjj.|H
x0002D                 ---------.|H      x0006B             kkkkkkkkkk.|H
x0002E                 .........|H      x0006C                llllllll.|H
x0002F                ////////.|H      x0006D          m mmmmmmmm mm.|H
x00030             0 000000 000.|H      x0006E             nnnnnnnnnn.|H
x00031             1 111111 111.|H      x0006F             ooooooooo o.|H
x00032             2 222222 222.|H      x00070             pppppppppp.|H
x00033             3 333333 333.|H      x00071             qqqqqqqqq.|H
x00034             4 444444 444.|H      x00072                rrrrrrrr.|H
x00035             5 555555 555.|H      x00073               s sss ss sss.|H
x00036             6 666666 666.|H      x00074                tttttttt.|H
x00037             7 777777 777.|H      x00075             uuuuuuuuuu.|H
x00038             8 888888 888.|H      x00076             vvvvvvvvv.|H
x00039             9 999999 999.|H      x00077          wwwwwwwwww.|H
x0003A                :::: ::::.|H      x00078             x xx xxx xxx.|H
x0003B                ;;;; ;;;;.|H      x00079             yyyyyyyyy.|H
x0003C             < <<<<<< <<<.|H      x0007A             zzzzzzzzz.|H
x0003D             = ====== ===.|H      x0007B                {{{{ {{{{ {.|H
x0003E             > >>>>>> >>>.|H      x0007C             ||||||||||.|H
x0003F                ?????????.|H      x0007D                }}}} }}}} }.|H
x00040          @@@@@@@@@@.|H      x0007E             ~~~~ ~~~~ ~.|H
x00041            AAAAAAAAAA.|H      x000A0                        .|H
x00042             BBBBBBBBBB.|H      x000A1                ¡¡¡¡¡¡¡¡.|H
x00043             CCCCCCCCCC.|H      x000A2             ¢¢¢¢ ¢¢¢¢ ¢.|H
x00044           DDDDDDDDDD.|H      x000A3             ££££ ££££ £.|H
x00045             EEEEEEEEEE.|H      x000A4             ¤¤¤¤ ¤¤¤¤ ¤.|H
x00046             FFFFFFFFFF.|H      x000A5             ¥¥¥¥ ¥¥¥¥ ¥.|H
x00047           GGGGGGGGGG.|H      x000A6             ¦¦¦¦¦¦¦¦¦.|H
x00048          HHHHHHHHHH.|H      x000A7             §§§§§§ §§§.|H
```

Figure 11-6 *Sample section of the font width test on NotoSerif-Regular (00501)*

Running the Font Width Test on Multiple FASTs

You can run the Font Width utility on multiple FASTs using the command line or using PathFinder.

Using the Command Line

From the command line, the *fwtest.pl* script uses the name of the X*library* as the argument.

To process multiple FASTs from the command line:

1. Enter the following command at the prompt:
   - UNIX:

     $XYV_EXECS/procs/sc/fwtest.pl $XYV_STYLES/X*library*
   - Windows:

     perl   %XYV_EXECS%\procs\sc\fwtest.pl

     perl   %XYV_STYLES%\X*library*

   The Font Width utility displays a Font Width Test pop-up list box containing the FAST numbers for that library.
2. Select the **FASTs** on which you want to run the font width test and click the **OK** button.

   To select continguous numbers, select the first FAST, press the Shift key, select the last FAST.

   To select non-contiguous FASTs, select the first FAST, press and hold the Control key, select the remaining FASTs.

   The Font Width utility displays a message box asking if you want to overwrite the existing division(s).
3. Click the **yes** button.

   The Font Width utility displays a processing message box.
   The Font Width Utility displays a message box containing the location of the divisions.

   The location is always CLS_xpputils/GRP_fwtest/JOB_*libraryname (without the X prefix).*

Using PathFinder

The process of running the Font Width utility on multiple FASTs is similar to running the Font Width utility on a single FAST. Refer to page 11-15.

To run the Font Width utility on multiple FASTs from PathFinder:

1. Follow steps 1-3 on page 11-15.
   The Font Width utility displays a Font Width Test pop-up list box containing the FAST numbers for that library with the specific FAST already selected.

2. Select the other **FASTs** on which you want to run the font width test and click the **OK** button.

   To select contingous numbers, select the first FAST, press the Shift key, select the last FAST.

   To select non-contiguous FASTs, select the first FAST, press and hold the Control key, select the remaining FASTs.

   The Font Width utility displays a message box asking if you want to overwrite the existing division(s).

3. Click the **yes** button.

   The Font Width utility displays a processing message box.
   The Font Width Utility displays a message box containing the location of the divisions.

   The location is always CLS_xpputils/GRP_fwtest/JOB_*libraryname (without the X prefix).*

Correcting Width Errors

Once you have run the font width test utility, you can determine if there are any glyphs with width errors by looking at the period and the letter H at the end of each line. Refer to Figure 11-6. They should line up with the reference periods and H's at the top and bottom of each page. The vertical lines are for reference only, to make it easier to spot any errors.

If the periods and Capital H's are offset to the left, the width in the PTS Spec is too high. Conversely, if the periods and capital H's are offset to the right the width in the PTS Spec is too low.

To determine which PTS Spec to edit, do the following:

- *Text FASTs*: The PTS Spec usually has the same number as the FAST. If the number is different follow the procedure for *Pi* FASTs.
- *Pi FASTs*: Since *Pi* FASTs usually contain glyphs from several fonts, do the following:

1. View the **FAST** using the following sequence:
   STYLE LIBRARIES > X*library* > FASTs > FAST *number* > View FAST
   Viewing the FAST allows you to determine from which font a particular glyph is coming; therefore, which PTS Spec to edit.
2. Once the FAST is open, search for the Unicode number that corresponds to the glyph. (The Unicode number is the first column in the FAST.) Note the number in the PTS *Font #* field; this is the PTS Spec you need to edit. (The *PTS Font #* field contains an entry greater than 32767 for modified glyphs, e.g., pseudofont characters).

To correct the error:

1. Measure the **amount of offset** in the division.
2. Locate the **PTS** Spec.
   (STYLE LIBRARIES > L*library* > Phototypesetter Specs > *pts_font* Spec)
3. Search for the **Unicode Number** you need to edit.
4. Change the **value** in the *Char Width* field, using the following conversion:
   All PostScript devices: 1 point = 10 PTS units
5. Run GenFAST on all related FASTS.
   (STYLE LIBRARIES > L*library* > FAST Generation > FAST *number* > Tools > Generate FAST)
6. Recompose the fwtest division. (You do not have to recreate the division unless you add new glyphs to the FAST.)

Chapter 12

The Font Variant Spec (FV)

This chapter contains the following information on the Font Variant (FV) Spec:

- Understanding the FV Spec
- Setting up an FV Spec

Understanding the FV Spec

The Font Variant (FV) Spec maps the fonts specified in the Item Format (IF) Spec tags or CSS Spec font properties or Font Family (*ff*) and Font Variant (*fv*) XyMacros to the Font Access Tables (FASTs).

Using the rules in the FV Spec, you do the following:

- Define font families.
  For example, you may define font family 0 as NotoSerif, font family 1 as NotoSans, and so on.
- Define the variants for each family.
  For example, you may have sixteen rules for the NotoSerif font, each defining a different variant — Regular, Bold, Italic, Bold Italic, Medium, SemiBold, Medium Italic, SemiBold Italic, ExtraBold, ExtraBold Italic, Black, Black Italic, ExtraLight, ExtraLight Italic, Light, and Light Italic.
- Specify the point size range for each family and variant.
- Specify the Primary FAST associated with each of these font families and variants.
- Specify the Secondary FAST and Default FAST associated with each of these font families and variants (optional).
- Specify a Kerning Pair (KP) file (optional).
- Set up kerning tracks (optional).
- Specify percentages for the height of small caps and ascenders.
- Specify entries in the Ligature/Accent Replacement (RP) Spec (optional).

Once you have set up a Font Variant Spec, you can specify the font families and variants in the *Family* and *Variant* fields in the Item Format Spec or use them as arguments in XyMacros. For CSS-XML mode, instead of the Item Format Spec fields you determine the font families and variants with font properties in the CSS Spec.

For anything other than CSS-XML mode, the following table shows the relationship between the Item Format Spec, Font Variant Spec and FASTs, and explains how the system processes each of them during composition.

Table 12-1 *How Composition Obtains Character Info (Non-CSS-XML Mode)*

《text》 Sample Text	The system processes the text as character codes. For example, the *CHAR CODE* for the letter T is 84.
↓	
Font Family Font Variant *IF Spec*	It checks the Item Format Spec for the **《text》** tag and gets the Font Family and Font Variant specified there (or from the current *ff* and *fv* XyMacros or other XyMacros that change the Family and Variant). For example, the **《text》** tag specifies Family 0, Variant 0.
↓	
Font Family Font Variant *FV Spec*	The system checks the Font Variant Spec specified in the Job Ticket for a rule matching the Family and Variant for the current point size. For example, it finds the rule with Font Family 0 and Font Variant 0. It then checks the following fields in that rule: ● *Small Caps FAST* ● *Primary FAST* ● *Secondary FAST* ● *Default FAST*
↓	
Small Caps FAST *FX Spec*	If the character is in **case mode 1** (small caps—**《cm;1》** XyMacro), composition uses the FAST specified in the *Small Caps FAST* field (if any). If the *Small Caps FAST* field contains **primary**, it uses the FAST specified in the *Primary FAST* field. If the the *Small Caps FAST* field contains **algorithm**, composition calculates the size of small capital letters based on the entry in the *Normal Small Caps Size %* or *Smallcap Size %* fields. (The later overrides the former if numeric values are present.)
↓	
Primary FAST *FX Spec*	Composition searches the Primary FAST for a rule with the character code. For example, if 11 is the value in the *Primary FAST* field, composition searches FAST Spec 11 for *CHAR CODE* 84. If it finds the rule, it uses the character information; otherwise, it goes to the *Secondary FAST* field to determine the Secondary FAST number (if any).
↓	
Secondary FAST *FX Spec*	It searches the Secondary FAST (if any) for a rule with the character code. For example, if 10006 is the value in the *Secondary FAST* field, composition searches FAST Spec 10006 for *CHAR CODE* 84. If it finds the rule, it uses the character information; otherwise, it goes to the *Default FAST* field to determine the Default FAST number (if any).
↓	
Default FAST *FX Spec*	It searches the Default FAST Spec (if any) for a rule with the character code. For example, if 10001 is the value in the *Default FAST* field, composition searches FAST Spec 10001 for *CHAR CODE* 84. If it finds the rule, it uses the character information; otherwise, it outputs the following message:
↓	
"Unspecified Typesetter Character"	XPP displays this message if it cannot find *CHAR CODE* 84. Use the *Find Errors* option to obtain more information. Use the *View Log* option to obtain the character code for the character.

For CSS-XML mode, the following figure shows the relationship between the CSS Spec, Typesetter Font Map (TSF) Spec, Font Variant Spec, and FASTs and explains how the system processes each of them during composition.

Table 12-2 *How Composition Obtains Character Info (CSS-XML Mode)*

《text》 Sample Text	The system processes the text as character codes. For example, the *CHAR CODE* for the letter T is 84.
↓	
CSS Spec font-family font-weight font-style	It checks the CSS Spec for the **《text》** element's font-family, font-weight and font-style properties. It attempts to match these values in the CSS fields in the TSF Spec. Any *ff* or *fv* override XyMacros or other XyMacros that change the Family and Variant, will directly determine the FV Spec that gets used.
↓	
TSF Spec font-family font-weight font-style	The system checks the TSF Spec for the rule with the best match for the combination of the font-family, font-weight, and font-style properties and then uses the Font Map Number in that rule as the Primary FAST to lookup in the Font Variant Spec.
↓	
FV Spec Primary FAST -or- Font Family Font Variant	The system checks the FV Spec for the first rule matching that Primary FAST to determine the FF and FV values to use. For example, it finds the rule with Font Family 0 and Font Variant 0. If any override XyMacros that change the FF and FV are used in the CSS Spec, they will directly determine the FV Spec rule that is used. It then checks the following fields in that rule: ● *Small Caps FAST* ● *Primary FAST* ● *Secondary FAST* ● *Default FAST*
↓	
Small Caps FAST *FX Spec*	If the character is in **case mode 1** (small caps—**《cm;1》** XyMacro), composition uses the FAST specified in the *Small Caps FAST* field (if any). If the *Small Caps FAST* field contains **primary**, it uses the FAST specified in the *Primary FAST* field. If the the *Small Caps FAST* field contains **algorithm**, composition calculates the size of small capital letters based on the entry in the *Normal Small Caps Size %* or *Smallcap Size %* fields. (The later overrides the former if numeric values are present.)
↓	
Primary FAST *FX Spec*	Composition searches the Primary FAST for a rule with the character code. For example, if 11 is the value in the *Primary FAST* field, composition searches FAST Spec 11 for *CHAR CODE* 84. If it finds the rule, it uses the character information; otherwise, it goes to the *Secondary FAST* field to determine the Secondary FAST number (if any).
↓	
Secondary FAST *FX Spec*	It searches the Secondary FAST (if any) for a rule with the character code. For example, if 10006 is the value in the *Secondary FAST* field, composition searches FAST Spec 10006 for *CHAR CODE* 84. If it finds the rule, it uses the character information; otherwise, it goes to the *Default FAST* field to determine the Default FAST number (if any).
↓	
Default FAST *FX Spec*	It searches the Default FAST Spec (if any) for a rule with the character code. For example, if 10001 is the value in the *Default FAST* field, composition searches FAST Spec 10001 for *CHAR CODE* 84. If it finds the rule, it uses the character information; otherwise, it outputs the following message:
↓	
"Unspecified Typesetter Character"	XPP displays this message if it cannot find *CHAR CODE* 84. Use the *Find Errors* option to obtain more information. Use the *View Log* option to obtain the character code for the character.

***Note:** When you run GenFAST, the utility creates a FAST, an FX file. You cannot view an FX file directly; you can create and view a VFX Spec and a VPX Spec from a FAST in the* `Xlibrary`.

Specifying an FV Spec

Specify the FV Spec in the Job Ticket or Division Ticket. The FV Spec may exist either in the *Spec Library* (also called the Style Library) named in the Job Ticket or at the job level.

When to Edit an FV Spec

Typically, Build FAST sets up an FV Spec in a font L*library* when you set up fonts on your system. You must move or copy this spec, or the most recently added rule(s) from this spec, to the job level FV Spec or to the library level FV Spec in the library that is specified as the *Spec Library*. Once you set up a Font Variant Spec, you do not need to edit it again unless one of the following conditions exits:

- You install additional fonts.
  In this case, run Build FAST. The utility modifies the FV Spec in the font L*library* to include the new FASTs. Copy those new rules to the active FV Spec for the affected jobs.
- You are using different FV Specs for different jobs.
  In some jobs, font family 0 might be NotoSerif and in other jobs, it might be NotoSansArabic. Edit the FV Specs to specify the desired family as font family 0, font family 1, and so on.

  Set up multiple FV Specs only if you have several hundred fonts on your system. In that case, you can group fonts together that are used for certain types of jobs.

***Note:** Every FV Spec must contain a rule defining font family 0 and font variant 0 in a range that includes a size of 13 points to display the line “Uncomposed text in Xyvision Standard Format” in an uncomposed page or division. Without this rule, you cannot edit or view a division.*

If you try to edit a division, the system displays the message “Variant 0, family 0, size 13; Missing FAST or FFVAR entry. Font family 0, variant 0 must be defined in your Font Variant Spec.”

Setting Up an FV Spec

When you add fonts to your system using Build FAST, the utility sets up the FV Spec automatically in the source font library (L*library*). Since XPP looks for the FV Spec in the job or *Spec Library* (or Style), you need to copy the rules from the FV Spec that was generated by Build FAST in the source font library to the active FV Spec.

If you need to edit the FV Spec manually, gather the following information:

- Units — the number of units assigned per em space (usually 1000 for PostScript devices).
- FAST numbers — the numbers of the FASTs on your system that you want to use.
- Smallcaps FAST numbers — the numbers of the FASTs (if any) containing the small capital letters for the font family/variant range (optional).
- Font family numbers — decide which numbers you want to assign to font families.
- Font variant numbers — decide which numbers you want to assign to font variants.
- Point size range — the minimum and maximum point sizes of your output device. (XPP supports a maximum of 186.1 points for PostScript output.)
- Kerning Pair (KP) file name — the name of any KP file you may want to use (optional).
- Characters or accents to replace — any character combinations you want replaced with ligatures or any accents you want replaced. You specify Accent and ligature replacement in the Ligature/Accent Replacement (RP) Spec (optional).

Naming an FV Spec

The Build FAST utility names the FV Spec in the source font library using the _fv_ prefix followed by the name you enter in the *Font Variant Spec Name* field of the Build FAST dialog box. The default name in this field is **xybuilt**.

When creating an FV Spec in the job or Spec Library, you assign it a name consisting of the _fv_ prefix followed by a maximum of eight alphanumeric characters, for example, _fv_doc.sde or _fv_noto.sde. The FV Specs can exist at both the Spec Library and job levels. Composition does not recognize the FV Spec in the source font library (unless that library is also the designated Spec Library).

Accessing an FV Spec

Access FV Specs from PathFinder using the following sequence:

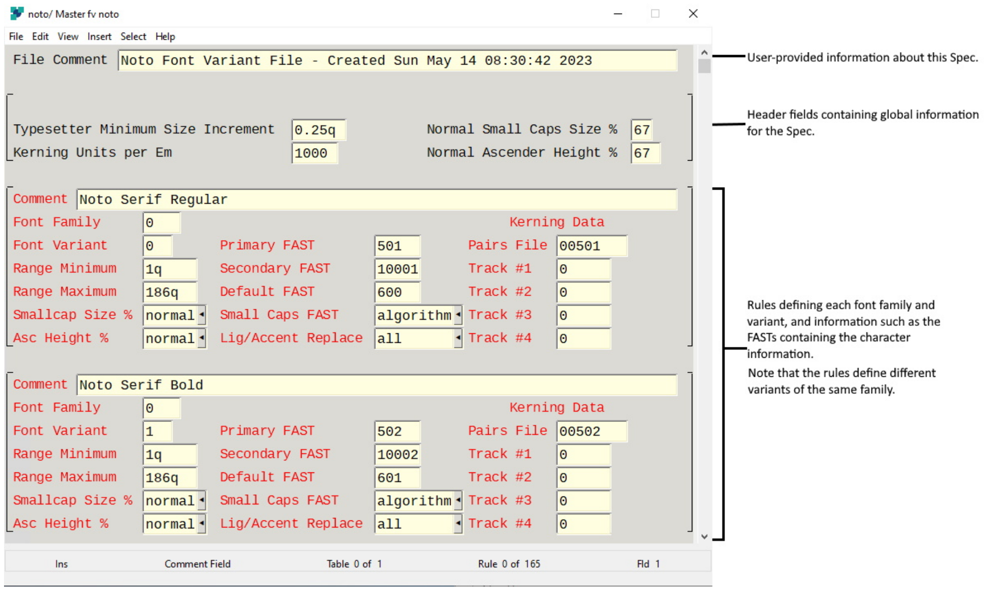
STYLE LIBRARIES > L*library* > Font Variants > Specific Font Variant Spec

Structure of an FV Spec

An FV Spec consists of a *File Comment* and one table with the following sections:

- Header — contains global entries for the spec
- Rules — contain information about each family and variant defined

The following figure shows the structure of the FV Spec.



File Comment	Noto Font Variant File - Created Sun May 14 08:30:42 2023		
Typesetter Minimum Size Increment	0.25q	Normal Small Caps Size %	67
Kerning Units per Em	1000	Normal Ascender Height %	67

Comment	Noto Serif Regular				
Font Family	0			Kerning Data	
Font Variant	0	Primary FAST	501	Pairs File	00501
Range Minimum	1q	Secondary FAST	10001	Track #1	0
Range Maximum	186q	Default FAST	600	Track #2	0
Smallcap Size %	normal	Small Caps FAST	algorithm	Track #3	0
Asc Height %	normal	Lig/Accent Replace	all	Track #4	0

Comment	Noto Serif Bold				
Font Family	0			Kerning Data	
Font Variant	1	Primary FAST	502	Pairs File	00502
Range Minimum	1q	Secondary FAST	10002	Track #1	0
Range Maximum	186q	Default FAST	601	Track #2	0
Smallcap Size %	normal	Small Caps FAST	algorithm	Track #3	0
Asc Height %	normal	Lig/Accent Replace	all	Track #4	0

User-provided information about this Spec.

Header fields containing global information for the Spec.

Rules defining each font family and variant, and information such as the FASTs containing the character information.

Note that the rules define different variants of the same family.

Figure 12-1 *Font Variant Spec*

Header Fields

The entries in the header fields are global entries for the spec; you can override entries in these fields if there is a corresponding rule field.

Typesetter Minimum Size Increment

The minimum increment by which your output device can increase or decrease type size. The minimum increment on some typesetters is 1q; others can adjust point sizes by units such as 0.1q. For example, with an increment of 0.1q, you can adjust the point size from 9q to 8.9q or 9.1q.

Entry	*Description*
number	A positive numeric value followed by a valid unit qualifier. This field entry defaults to 1q.

Kerning Units per Em

The entry in this field determines the value of a kerning unit.

To calculate the value of a kerning unit, the system divides the value in this field by the number of units assigned per em space (PostScript devices are usually 1000 per em.)

You can enter a value in this field that makes the kerning units per em space a relatively small or large number. This gives you extremely fine control.

For example, your output device uses 1000 units per em space and you want 10 kerning units per typesetter unit. Enter 10000 in this field (1000 x 10).

Refer to the description of the *Kerning Data Track #1 - #4* fields.

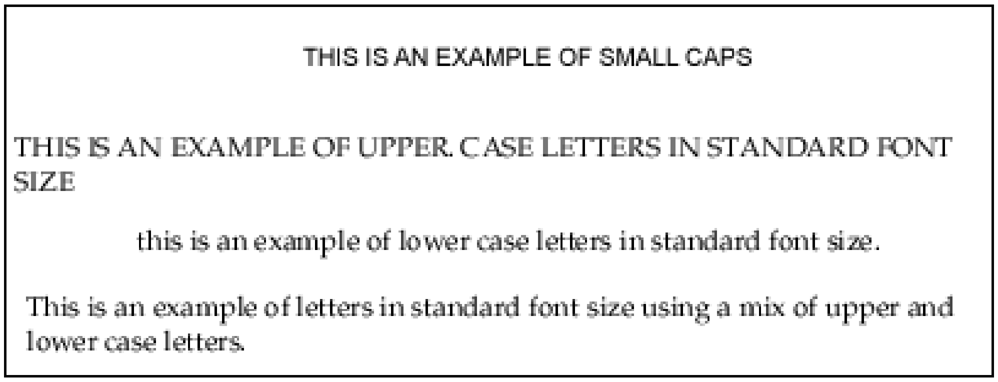
Entry	*Description*
integer	An integer in the range of 1 through 65535. This value divides an em space into kerning units. This field entry defaults to 1000.

Normal Small Caps Size %

In the absence of having an actual small caps font, this field allows you to set up XPP to mimic small caps. Using the value in this field, XPP calculates the height of small capital letters as a percentage of the height of capital letters. For example, using the default entry of **67**, XPP generates small capital letters that are 67% the size of the ascender height of capital letters.

XPP uses this field only if the *Small Caps FAST* field contains the entry **algorithm**. It does not use this field if the *Small Caps FAST* field contains a FAST number or the entry **primary**.

Entry	*Description*
integer	An integer in the range of 1 through 99. This field entry defaults to 67.


THIS IS AN EXAMPLE OF SMALL CAPS

THIS IS AN EXAMPLE OF UPPER CASE LETTERS IN STANDARD FONT SIZE

this is an example of lower case letters in standard font size.

This is an example of letters in standard font size using a mix of upper and lower case letters.

Figure 12-2 *Font size comparison of Small Caps, Upper case, Lower case, and Mixed case letters*

Normal Ascender Height %

Using the entry in this field, the system calculates the percentage of the font height taken up by the ascender. The descender takes up the remaining percentage.

An *ascender* is the part of the glyph that extends above the baseline. The *descender* is the part of the glyph that extends below the baseline. For example, using the default entry of **67**, the ascender height is 67% of the font height and the descender height is 33% of the font height.

In the Item Format Spec, if the Ascender Lead or Descender Lead fields contain the entry **normal**, XPP uses this field to determine the leading. In CSS-XML mode, the same is true for the corresponding `-xpp-ascender-lead` and `-xpp-descender-lead` CSS properties if they are set to **normal**.

Entry	*Description*
integer	An integer in the range of 0 through 100. This field defaults to 67. If the values in the *Ascender Lead* and *Descender Lead* fields in the Item Format Spec, or the `-xpp-ascender-lead` and `-xpp-descender-lead` properties in the CSS Spec, do not correspond to the percentage entered in this field, XPP does not position the glyph correctly relative to the baseline.

Rule Fields

The FV Spec contains a rule for each font family and font variant. While XPP allows you to have multiple rules specifying the different point sizes of the same family and variant in the FV Spec, this option is seldom used due to the standard PostScript environment.

Comment

Information such as the font family and variant names. For example, NotoSerif Regular.

Entry	*Description*
string	Characters consisting of as many as 7 lines of alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, /), and integers 0-9.

Font Family

A number that identifies the font family. The XPP XyMacros Spec (xy_sys in syslib) maps the Main Font Family to font family 0 and the Alternate Font Family to font family 1, using the (*main*) and (*alt*) XyMacros respectively. Also, the Main key is mapped to font family 0 and the Alt key is mapped to font family 1. Therefore, it is recommended that you assign your most frequently used font families to 0 and 1.

When you press [Main] or manually insert the *main* XyMacro, the font changes to font family 0. When you press [Alt] or manually insert the *alt* XyMacro, the font changes to font family 1.

Entry	*Description*
integer	An integer in the range of 0 through 2047.
0	Specifies font family (default = 0). Accessed by the key cap Alt/Main. Mapped to the Main Font Family (*main*) XyMacro primitive */ff;0*[1].
1	Specifies font family 1. Accessed by the key cap Shift + Alt/Main. Mapped to the Alternate Font Family (*alt*) XyMacro primitive */ff;1*.
2-2047	Other valid font family values.

[1]The system first uses the %inmath system variable to check if it is in math. In math, different font family and font variant numbers may be used.

Font Variant

A number that identifies the font variant. The XPP XyMacros Spec maps the variants 0 through 5 to typeface styles. Some keys access these macros. For example, when you press the Ital/Med key or manually insert the Medium Style (*med*) XyMacro, the variant changes to 0.

Entry	*Description*
integer	An integer in the range of 0 through 255.
0	Specifies font variant 0 (default). Accessed by key cap Ital/Med. Mapped to the Medium Style (*med*) XyMacro primitive */fv;0*[1].
1	Specifies font variant 1. Accessed by key cap Ital/Bold. Mapped to the Bold Style (*bold*) XyMacro primitive */fv;1*[1].
2	Specifies font variant 2. Accessed by key cap Shift + Ital/Med. Mapped to the Medium Italic Style (*mdit*) XyMacro primitive */fv;2*[1].
3	Specifies font variant 3. Accessed by key cap Shift + Ital/Bold. Mapped to the Bold Style Italic (*bdit*) XyMacro primitive */fv;3*[1].
4	Specifies font variant 4. Accessed by key cap Ital/Lite. Mapped to the Light Style (*lite*) XyMacro primitive */fv;4*[1].
5	Specifies font variant 5. Accessed by key cap Shift + Ital/Lite. Mapped to the Light Italic Style (*ltit*) XyMacro primitive */fv;5*.
6-255	Other valid font variant values.

[1]The system first uses the %inmath system variable to check if it is in math. In math, different font family and font variant numbers may be used.

Range Minimum/ Range Maximum

The smallest (min. 4.5q) and/or largest (max. 186.1q) point size available for this font on your output device. (Technically, you can specify a smaller point size, but some things having to do with pickup placement and/or leading may not work properly.) The range must also be equal to or within the range specified in the FAST Generation (FGS) Spec. Having multiple rules specifying different point sizes is seldom used.

Entry	*Description*
number	A positive numeric value, followed by a valid unit qualifier, in the range of 0 through 186.1q or the equivalent in p, i, m, n, c, d, k units. The Range Minimum field entry defaults to 1q, however, the minimum point size that XPP can output is 4.5q. (Technically, you can specify a smaller point size, but some things having to do with pickup placement and/or leading may not work properly.)The Range Maximum field entry defaults to 186.1q.

Smallcap Size %

This field overrides the *Normal Small Caps Size %* header field.

The value in this field applies only to the glyphs in the variant defined in this rule.

Entry	*Description*
normal	Size of a small capital letter in relation to a capital letter based on the information in the FAST.
integer	A one- or two-digit number ranging from 1 to 99. This represents a percent of the large cap size.

Asc Height %

This field overrides the *Normal Ascender Height %* header field.

The value in this field applies only to the glyphs in the variant defined in this rule.

Entry	*Description*
normal	The ascender/descender ratio of height for a rule. **normal** uses header field *Normal ASC Height %*.
integer	A one- or two-digit number ranging from 1-99. This represents a percent of the ascender/descender ratio of height for a rule.

Primary FAST

The number you have assigned to this font (usually a text font).

Entry	*Description*
integer	A five-digit number in the range of 0 to 32767. This field entry defaults to 0.

Secondary FAST

The number of the secondary FAST to supplement glyphs in the primary (text) FAST. This FAST usually contains symbols and *Pi* characters.

Refer to ″The FAST Generation Spec (FGS)″, page 9-6, for recommended naming conventions for *Pi* FASTs.

Entry	*Description*
integer	A five-digit number in the range of 0 to 32767. This field entry defaults to 0.

Default FAST

The number of the third FAST (if any) you want the system to check for the glyph information. This may be a catch-all FAST containing glyphs from all of your fonts.

Entry	*Description*
integer	A five-digit number in the range of 0 to 32767. This field entry defaults to 0.

Small Caps FAST

If there is no Small Caps FAST for this font family, **algorithm** is entered in the field. This specifies for the system to use a formula to produce small capital letters based on the values entered in the *Normal Small Caps Size %* or *Smallcap Size %* fields. (The system does not use these fields if the Small Caps FAST field contains a FAST number or the entry **primary**.)

If there is a small caps FAST for this font family, you enter that FAST number in this field.

If the primary FAST in this FV Spec rule is a small caps FAST, you enter **primary** in this field.

Entry	*Description*
algorithm	Specifies using a formula to calculate the size of small capital letters (default).

Entry	*Description*
integer	A five-digit number in the range of 0 to 32767. This field entry defaults to 0.
primary	Specifies using the FAST in the *Primary FAST* field.

Ligature/Accent Replacement

The entry in this field specifies whether to replace accents and use ligatures for characters in this font.

In the Ligature/Accent Replacement Spec, you specify the characters you want to replace and the character that you want to use to replace them. During composition, XPP matches the entry in this field to the entry in the Ligature Mask rule field(s) in the Ligature/Accent Replacement Spec. Refer to “The Ligature/Accent Replacement Spec” on page 16-1 for more information.

If you enter an integer that has a corresponding alphabetic entry (i.e., 1 for ffl, 2 for ffi, and so on), the integer changes to the corresponding alphabetic entry when you exit the field. For example, if you enter **31**, the integer changes to **all** (meaning replace all ligatures) when you exit the field.

If you enter an integer that does not have a corresponding alphabetic entry (i.e., 17, 64, 95, and so on), the integer does not change when you exit the field.

Use the integer entries to create special combinations that you could not otherwise enter in the field. For example, you want to replace only the character combinations f f l and f l with the corresponding ligatures, but that is not a valid entry in this field. Add the integer entry for ffl (1) to the integer entry for fl (16). Enter **17** in the Ligature/Accent Replacement field.

The following alphabetic entries (i.e., **none**, **ffl**, and so on) are available with the *Next Choice, Prev Choice* menu options.

Entry	*Description*
integer	A positive integer in the range 0 through $(2)^{32}$ to represent alphabetic equivalents. The integers that have defined character replacements are shown in the following list. Wherever possible, XPP replaces the integer entry with the alphabetic entry when you exit the field.
string	An alphabetic string for the character replacement. The following is a list of the defined strings.

Defined String	*Integer*	*Specifies*
none	0	No ligature or accent replacement (default)
ffl	1	Replacing ffl with the ffl ligature

Defined String	*Integer*	*Specifies*
ffi	2	Replacing ffi with the ffi ligature
fi	4	Replacing fi with the fi ligature
ff	8	Replacing ff with the ff ligature
fl	16	Replacing fl with the fl ligature
fifl	20	Replacing fi and fl with the fi and fl ligature
all	31	Replacing any ligature character combination (ffl, ffi, fi, ff, fl) with the corresponding ligature.
accents	32	Replacing accents only *(for FASTs with uppercase and lowercase accents).*
both	63	Replacing both ligatures (31) and accents (32) *(for FASTs with uppercase and lowercase accents).*
NA[1]	64	Replacing accents using the Accents (*acm*) XyMacro *(for FASTs with lowercase accents only).*
NA[1]	84	Replacing accents, using acm (64) and fi and fl ligatures (20) *(for FASTs with lowercase accents only).*
NA[1]	95	Replacing accents, using acm (64) and all ligatures (31) *(for FASTs with lowercase accents only).*

[1]NA = Not Available. This integer does not have a corresponding string entry.

Kerning Data: Pairs File

The name or number of the Kerning Pair file (if any) for the system to use in kerning characters in this font family/variant and size range. The Kerning Pair file must be in the pair kerning library (K*library-name) of the same name as the font library (Xlibrary-name*). The system checks the file for matching kerning pairs for every character in a division.

For example, in a Kerning Pair file, you specify the amount of space to add or subtract between a capital "A" and a capital "W." You enter the name of the Kerning Pair file in this field.

When the system finds the capital "A" followed by a capital "W" in the division, it searches the Kerning Pair file for a capital "A" with capital "W". It finds the rule in which you have specified kerning and kerns the spacing between the "A" and the "W."

Entry	*Description*
string	An alphabetic string up to eight characters long.
no entry	Specifies not to use a Kerning Pairs file (default.)

Kerning Data Track #1 - #4

A value specifying positive or negative track kerning. Positive values add white space between each pair of glyphs; negative values remove white space between each pair of glyphs. The entries in the *Track* field are based on relative kerning units (refer to the description of the *Kerning Units per Em* field).

RWS recommends entering **0** in the field for Track #1 so you can easily turn track kerning off in Item Format rules or with the `-xpp-kerning-track` property in the CSS Spec.

Specify the desired kerning track values in the Font Variant Spec; this allows you to kern glyphs in text of different point sizes by varying degrees. For example, you may want to reduce the amount of white space between pairs of glyphs in a large point size used for heads, but not in the point size used for main text. You can also specify the kerning track using the Track Kerning (*tk*) XyMacro or corresponding CSS property.

If you specify track kerning in the Item Format Spec or with the `-xpp-kerning-track` property in the CSS Spec *and* you name a Kerning Pairs file in the Font Variant Spec, the system kerns pairs of glyphs in the Kerning Pairs file by the sum of the two values. If you intend to use both track kerning and the Kerning Pairs file, set up track kerning first.

To calculate the entry for the *Track* field:

1. Determine the value of a kerning unit by dividing the entry in the *Kerning Units per Em* field by the number of units the typesetter assigns per em.
2. Multiply the value of a kerning unit by the number of units of white space adjustment you want between each pair of glyphs.
3. Enter this value in the *Track* field.

For example, the typesetter assigns 1000 units per em. If the *Kerning Units per Em* field contains the entry **1000**, the value of one kerning unit is 1. To specify removing 10 relative units of white space between pairs of glyphs, enter **-10** in this field.

Entry	*Description*
integer	A number in the range of -32767 through 32767. A positive integer specifies adding white space between all pairs of glyphs; a negative integer specifies removing white space between all pairs of glyphs.
0	Specifies no white space adjustment. The following list shows the entries for the *Track* fields for the desired number of relative units for a 1000-unit typesetter based on 10,000 kerning units per em.

Track Field Entries for Relative Units (10,000 kerning units per em)

Relative Units	***1000-Unit Typesetter***
10	100
20	200
30	300
100	1000

Chapter 13

The Typesetter Font Map Spec (TSF)

This chapter contains the following information on the Typesetter Font Map (TSF) Spec:

- Understanding the TSF Spec
- Setting up a TSF Spec

Understanding the TSF Spec

Outputting fonts on PostScript devices requires calling their fonts by their PostScript names. Using a Typesetter Font Map (TSF) Spec, you map the XPP font numbers to the PostScript font name to the so XPP can output and display the font.

Build FAST and GenFAST do not use the TSF Spec information to create a Font Access Table (FAST), but Build FAST updates the TSF Spec.

When do I Need a TSF Spec?

You need to a TSF Spec because the output devices and PostScript/PDF output files access fonts by name rather than by XPP font number.

For example, PostScript devices and files identify fonts by name; however, internally XPP identifies fonts by number. Edit the TSF Spec to map each font number to a PostScript font name so XPP can output the fonts.

If you create a TSF Spec, you must include all the fonts on your system that are going to be used with the font library that contains the TSF Spec. You can only display and output the fonts you have included in the TSF Spec. If you try to output using a font that is not in the TSF Spec, XPP reports the following message during display and output:

```
“ERROR: No mapping for font number #”
```

Naming the TSF Spec

The name of the TSF Spec is **_tsf_system.sde**. No other names are valid. Note that you can use other names while you are developing all your mappings, but then you must copy the necessary rules from that spec into _tsf_system.sde in order for XPP to find and use them.

How XPP Uses the TSF Spec

Using the following lookup procedure:

1. XPP checks the L*font* library specified in the Job Ticket; if the TSF Spec (_tsf_system.sde) exists, XPP uses it.

   XPP checks only the main font library of a numbered series; it does not search any numbered font libraries. For example, it checks noto, but not noto1, noto2, and so on.

2. If the specified L*font* library does not contain the TSF Spec, XPP checks the syslib library. If XPP cannot find the TSF Spec in the syslib library, you cannot edit or print any divisions.

Setting Up the TSF Spec

Setting up the TSF Spec involves accessing the spec, naming the spec, and understanding the structure of the spec.

Accessing the TSF Spec

Access the TSF Spec from PathFinder using the following sequence:

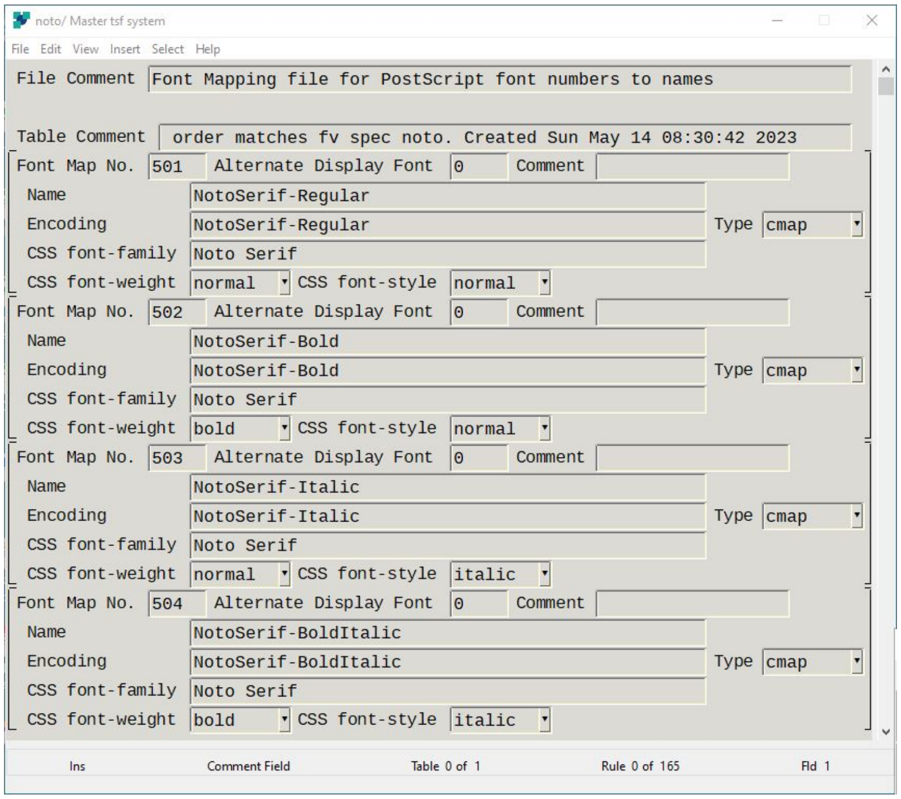
STYLE LIBRARIES > Library > Typesetter Font Maps > *specific TSF Spec*

Structure of the TSF Spec

The TSF Spec consists of a *File Comment* field and one table with the following sections:

- Header — contains the *Table Comment* field
- Rules — contains information such as *Font Map No.*; *Encoding* table name; PostScript font *Name*; encoding *Type*; *Alternate Display Font*; *CSS font-family*, *CSS font-weight*, *CSS font-style*; and *Comment*

The following figure shows the structure of the TSF Spec.



```
File Comment   Font Mapping file for PostScript font numbers to names

Table Comment   order matches fv spec noto. Created Sun May 14 08:30:42 2023
Font Map No.  501   Alternate Display Font  0    Comment
 Name             NotoSerif-Regular
 Encoding         NotoSerif-Regular                      Type  cmap
 CSS font-family  Noto Serif
 CSS font-weight  normal   CSS font-style  normal
Font Map No.  502   Alternate Display Font  0    Comment
 Name             NotoSerif-Bold
 Encoding         NotoSerif-Bold                         Type  cmap
 CSS font-family  Noto Serif
 CSS font-weight  bold     CSS font-style  normal
Font Map No.  503   Alternate Display Font  0    Comment
 Name             NotoSerif-Italic
 Encoding         NotoSerif-Italic                       Type  cmap
 CSS font-family  Noto Serif
 CSS font-weight  normal   CSS font-style  italic
Font Map No.  504   Alternate Display Font  0    Comment
 Name             NotoSerif-BoldItalic
 Encoding         NotoSerif-BoldItalic                   Type  cmap
 CSS font-family  Noto Serif
 CSS font-weight  bold     CSS font-style  italic

      Ins          Comment Field        Table 0 of 1        Rule 0 of 165        Fld 1
```

Header Fields

The header fields of the TSF Spec are comment fields, allowing you to enter up to 7 lines of information about the file or the table.

Rule Fields

The rules map the typesetter font number specified in each PTS Spec to a PostScript font name.

Font Map No.

The same number you enter in the *Font Number* field in the Typesetter Information section of the PTS Spec.

Entry	*Description*
integer	A positive integer in the range of 0 through 65535. This field entry defaults to 0.

Alternate Display Font

This field is rarely used. It allows you to use one font for output but specify an alternate font for screen display in the XyView (WYSIWYG Editor).

Entry	*Description*
integer	A positive integer in the range of 0 through 65535. This field entry defaults to 0.

Comment

A comment pertaining to the font accessed by this rule.

Entry	*Description*
string	A comment containing up to 7 lines of alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, /), and the integers 0-9.

Name

Enter the name of the font. It is essential that this name exactly matches the actual PostScript font name; it is case sensitive.

Entry	*Description*
string	As many as 62 alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, /), and the integers 0-9.

Encoding

The name of the encoding table or CMap used. The encoding table name for encoding *Type:* **standard** for Type 1 fonts is often *extended* for text fonts or *none* for *Pi* fonts. If you are using a *Pi* font, you must enter **none** in this field. If you leave this field blank, the default for *Type:* **standard** is *extended* encoding and the default for *Type:* **cmap** is the same name as is entered in the *Name* field.

Type

Asks whether you want to access glyphs using a CMap file for non-Type 1 fonts or a standard encoding file for Type 1 fonts.

Entry	*Description*
cmap	The font in use is a CID or OpenType font. XPP looks for a CMap file.
standard	The font in use is a PostScript Type 1 font. XPP processes the font accordingly.

CSS

These fields are used if your DIV is in css-xml mode; they correspond to the `font-family`, `font-style`, and `font-weight` properties in your `.css` style sheet definitions. Refer to the *Styling Content with CSS* publication for more information on using fonts with CSS. That manual also discusses using the *afmtotsf.pl* utility, which can be run from *PathFinder*, to read the font AFM files and populate any empty CSS fields in a TSF spec.

Entry	*Description*
font-family	The base family name for the font. The *CSS font-family* name field value is an alias to the actual font used in XPP, as the CSS font name can be different than the PostScript font names used in XPP. *CSS family-name* value where *CSS family-name* is a valid font family name as defined in the CSS specification.
font-style	The *CSS font-style* field value specifies whether the font is italic or not. ***Note:*** *The font could be oblique if the actual font referenced is implemented as an oblique font (rather than a true italic font). However, in the TSF spec, the font is specified as* **italic** *in the CSS font-style field.* `normal | italic`

Entry	*Description*
font-weight	The *CSS font-weight* field value specifies whether the font is bold or not. `normal | bold`

Chapter 14

Encoding Tables

This chapter describes encoding tables for PostScript fonts and provides some common encoding tables.

Encoding Tables for PostScript Fonts

All PostScript Type 1 fonts have a default encoding table that specifies the codes used to access the glyphs. This default table resides within the font itself.

All OpenType fonts also contain an internal table that maps Unicode code points to the glyphs in the font. Font Copy uses this to create the font-specific CMap.

CID fonts that are not OpenType contain neither a default encoding nor a Unicode mapping table. They must be addressed using separate CMaps that are built for the particular arrangement of glyphs in the font.

The default PostScript character encoding supplied by the Adobe Type 1 Roman text fonts includes 149 glyphs, 32 reserved codes, and 75 unused codes. They may also contain "unencoded" characters, which are represented in the *.afm* file with -1 as the character code (C value).

XPP used to provide access to 75 common unencoded characters in a Type 1 font encoding table called **extended**. Most of these glyphs are accented, such as Aacute (Á). The complete pathname is XYV_EXECS/sys/od/ps_dlf/encodings/extended.

Caution

RWS recommends that you do not edit the *extended* file, although it is no longer delivered. This file was used for the standard Adobe PostScript Type 1 fonts that XPP used to deliver.

The first few lines of the **extended** file that used to be deivered are shown here:

```
8#201 /Aacute        8#232 /Udieresis

8#202 /Acircumflex   8#227 /Ugrave        8#376 /minus

8#204 /Adieresis     8#233 /Yacute        8#177 /mu

8#200 /Agrave        8#366 /Ydieresis     8#375 /multiply

8#205 /Aring         8#300 /Zcaron        8#336 /ntilde
```

Each entry in a Type 1 font encoding file is in the format:

8#*access-code   /character-name*

where:

Entry	*Description*
8#	Specifies that the next value is a 3 digit octal.
3 digit access-code	Specifies the glyph access code (position in the font).
/*character-name*	Specifies the PostScript glyph name, preceded by a slash.

Non-Standard Type 1 Font Encodings

The Xyvision Character Set can be applied to most Type 1 text fonts. However, all Type 1 *Pi* fonts and some Type 1 text fonts have different layouts, such as non-English language fonts and phonetic fonts. The XPP extended encoding must be disabled for these Type 1 fonts.

To disable the extended encoding, enter **none** in the Build FAST *Encoding/ CMap Table* name field and in the TSF Spec *Encoding* table name field.

The default encoding when the TSF Spec *Encoding* field is blank for a *Type* **standard** Type 1 font is **extended**, so in that case or when **none** was not entered when running Buld FAST, you must explicitly enter or change it to **none** in the TSF Spec *Encoding* field.

Reconciling Unencoded Characters for Type 1 Fonts

This section provides information on how to use more glyphs in PostScript Type 1 fonts that contain more than 256 glyphs or unencoded characters beyond the 75 that XPP used to provide access to with the **extended** encoding file. Note that this information is not necessary with CID or OpenType fonts.

The procedures described in this section **require** a solid understanding of the XPP font environment for Type 1 fonts that XPP used to deliver. If you do not have this foundation, RWS strongly advises that you read the other chapters in this manual before attempting any of these procedures.

Handling More than 256 Glyphs

If a font has more than 256 glyphs, or more than 75 unencoded characters, it exceeds the range of character positions in an encoding table. If you need access to more than 256 glyphs in the font, it is necessary to create one or more additional encoding tables. You accomplish this by building two or more PTS Specs against the single *.afm* file. A single FAST is then built against the multiple PTS Specs.

You do not modify the Type 1 font *.afm* file in any way.

Unencoded Characters Beyond the 75 XPP Used to Provide

If Build FAST encounters ʺunencodedʺ characters, it means that the font has glyphs that are not in the encoding table that you specified. You need to compare the encoding table with the unencoded characters in the *.afm* file.

If the encoding table includes access to characters that are not in the *.afm* file, you can replace these characters with the ʺunencodedʺ characters from the *.afm* file to provide access to those characters. However, the *.afm* file may contain all 75 characters in the **extended** encoding file and perhaps many more.

Reconciling a Few Unencoded Characters

If you run Build FAST with a standard Type 1 Roman text font and use the **extended** encoding file, you may get a message that the following characters are unencoded: onesuperior, twosuperior, threesuperior and logicalnot. If you receive this message or one similar, and you need to use these glyphs, follow the method outlined below. If you have more than 75 unencoded characters, refer to page 14-6.

To reconcile a **few** unencoded characters:

1. Run Build FAST for the primary glyphs, using the **extended** encoding table.
2. Copy the **extended** character encoding file (XYV_EXECS/sys/od/ps_dlf/encodings/extended) to another name (e.g., extended2).
3. In extended2:
    - Overwrite the character names with the unencoded character names, using any four numbers (positions). (Be sure to use the postscript character name exactly as it appears in the font's *.afm* file.)
    - Record the position numbers you are using, which are in octal notation.
    - You may then eliminate the remaining characters, if you wish.
4. Create a new PTS Spec with a unique name that contains one rule for each of the unencoded characters. (e.g., If the original PTS Spec is _pts_05001, name the new one _pts_05002)
    - Fill in the *UNICODE NUMBER* and *CHAR WIDTH* fields as appropriate.
    - Fill in the *CHAR CODE* field with the same position number value used in the extended2 table.
      You may want to convert the octal to decimal notation.
    - The value in the *Font Map Number* field in the header must match the unique number you used in naming the spec.
5. In the original PTS Spec (created from running *Build FAST*), delete the rules for the unencoded characters (they are assigned *CHAR CODE* of d0).
6. Add a rule to the FGS Spec that was created by your initial run of Build FAST.
   The *PTS/PSF Spec* field must contain the name of the second PTS Spec that you created. (e.g., FGS Spec already contains pts_05001; add _pts_05002).
7. Run GenFAST against this FGS Spec.
   Right-click the FGS Spec and select **Tools > Generate FAST**.

   GenFAST creates a single FAST. (e.g., 05001)
8. Run View FAST to display the FAST you just created.

   In the **X***library*, right-click the original FAST number and select

   Tools > View FAST.

   In the *Font #* column, notice that not all the glyphs come from the original PTS Spec (e.g., 05001). Depending on the number of PTS Specs listed in the FGS Spec, the FAST displays a different font number for these extra glyphs, as well as the position number you assigned.

9. Add a rule to the TSF Spec (L*library*).
   - The *Font Map No.* field should contain the same number you used in naming the new PTS Spec you created. (e.g. 05002).
   - The *Encoding* table name field should contain the new encoding table name (e.g., extended2).
   - The PostScript font *Name* field must contain the same name as the *Name* field in the original rule of the TSF Spec.

   The TSF Spec should now have two rules in it with the same PostScript font name, but different encodings and font numbers.

10. (Optional) Run a Font Width Test of this FAST to verify access to and correct widths for all glyphs.
    - Right-click the FAST in the X*Library* that you want to verify.
    - Select Width Test from the pop-up menu.

Reconciling Many Unencoded Characters

If you have a font with many unencoded characters, you follow the same basic procedure listed for a few unencoded characters. To facilitate the setup, however, you may run Build FAST twice (or more) using a different encoding table name each time. Running Build FAST automatically creates a new PTS Spec (Step 4) and adds a rule to the TSF Spec (Step 7). You then need to reconcile the resulting multiple PTSs and FASTs.

To reconcile **many** unencoded characters:

Follow the steps on page 14-5. The exceptions are noted below:

- In Step 4, unlike on page 14-5, you do not need to create the PTS Spec since it is automatically created when you run Build FAST the second time.
- In Step 5, remove **all** unencoded characters from **each** PTS Spec.
- In Step 6, list **all** the PTS numbers in the FGS Spec.
- Step 7 is unnecessary since Build FAST automatically adds a rule to the TSF Spec when you run the utility a second time.
- You may delete the FGS and PSF Specs that result from the subsequent runs of Build FAST, as they have no use.

Note: *The positions used in the unique encoding table may vary depending on the default encoding table for that font. Remember that Pi fonts generally use* **none** *as the encoding table.*

Chapter 15

The Kerning Pairs Spec (KP)

This chapter contains the following information on the Kerning Pairs (KP) Spec:

- Understanding the KP Spec
- Kerning pairs libraries
- Accessing KP Specs
- Setting up a KP Spec

Understanding the KP Spec

Using the KP Spec, you specify a pair of characters and the amount of space to add or subtract between them. The pair of characters you specify is the *kerning pair*; the spacing between the characters is the amount of *kerning*. You can expand or reduce the space between the two characters.

The following figure shows the characters A and V placed without kerning and with kerning.

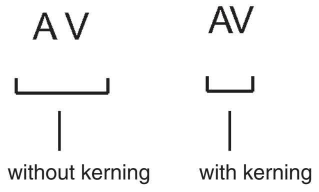


Figure 15-1 *Characters Placed Without and With Kerning*

The Build FAST utility automatically generates a KP Spec if kerning pairs information is present in the AFM file.

Differentiating Between the KP Spec and the KP File

The KP Spec contains the kerning information that determines the spacing between two characters. When you store/exit a KP Spec in an L*font* library, XPP creates a machine-readable _kp_#####.x file and places it in the K*font* library. XPP programs use the machine-readable file for faster performance.

You cannot view or edit the_kp_#####.x file. To change any of the kerning information, you need to edit the KP Spec in the L*font* library and store/exit the spec to update the _kp_#####.x file in the K*font* library.

Note: *RWS strongly recommends that you do not delete* **_***kp***_***#####.sde (the KP Spec) unless you are absolutely sure that you will never need to add, delete, or modify any kerning pairs for that font. Should you ever decide that you need to update its corresponding* **_***kp***_***#####.x file, there is no way to retrieve the active kern pairs and values from the .x file without the* **_***kp***_***#####.sde Spec file.*

Note: *Kerning pairs values provided by the font vendor may not meet your typographic aesthetic standards. If so, you can repeatedly run the Kerning Pairs utility, adjust the Kerning Pairs Spec (storing out will cause the file in the Kfont library to be updated), then run the Kerning Pairs utility again. When satisfied, compose your documents to decide if further modifications are desired. But then do not re-run Build FAST on the font again because that will overwrite all your custom kerning pairs modifications.*

How Composition Uses the KP File

You specify the KP file in the Font Variant (FV) Spec. As composition processes text, it reads the Font Variant Spec and uses the glyph widths from the specified FAST to set the glyphs.

If the currently active rule in the FV Spec specifies a KP file, composition searches the text for the characters specified in the KP Spec. When composition finds matching character pairs, it does the following:

1. Places the first glyph using the glyph width specified in the FAST.
2. Moves left or right by the amount specified in the KP file.
3. Places the second glyph of the pair using the glyph width specified in the FAST.

For example, the KP file contains a rule defining a pair of characters where the letter "A" is the first character and the letter "W" is the second character. The rule specifies removing space between the glyphs so they are set closer together. If composition finds this pair of characters, it places the "A," moves left by the amount specified in the KP file, then places the "W."

XPP also applies track kerning if the *Kern Track* field in the Item Format Spec or `-xpp-kerning-track` property in the CSS Spec specifies a track and the current FV rule defines a value for kerning in that track.

If the currently active rule in the FV Spec does not specify a KP file or the specified spec does not exist in the library, composition does not change the amount of space between the glyphs unless you have specified track kerning by entering a value in the *Kern Track* field in the Item Format Spec or `-xpp-kerning-track` property in the CSS Spec.

Note: *If you have specified any replacements in the Ligature/Accent Replacement field of the Font Variant Spec, the system replaces any specified characters before applying kerning.*

The following figure shows how the system processes the KP file fields and determines whether to use kerning pairs and/or track kerning.

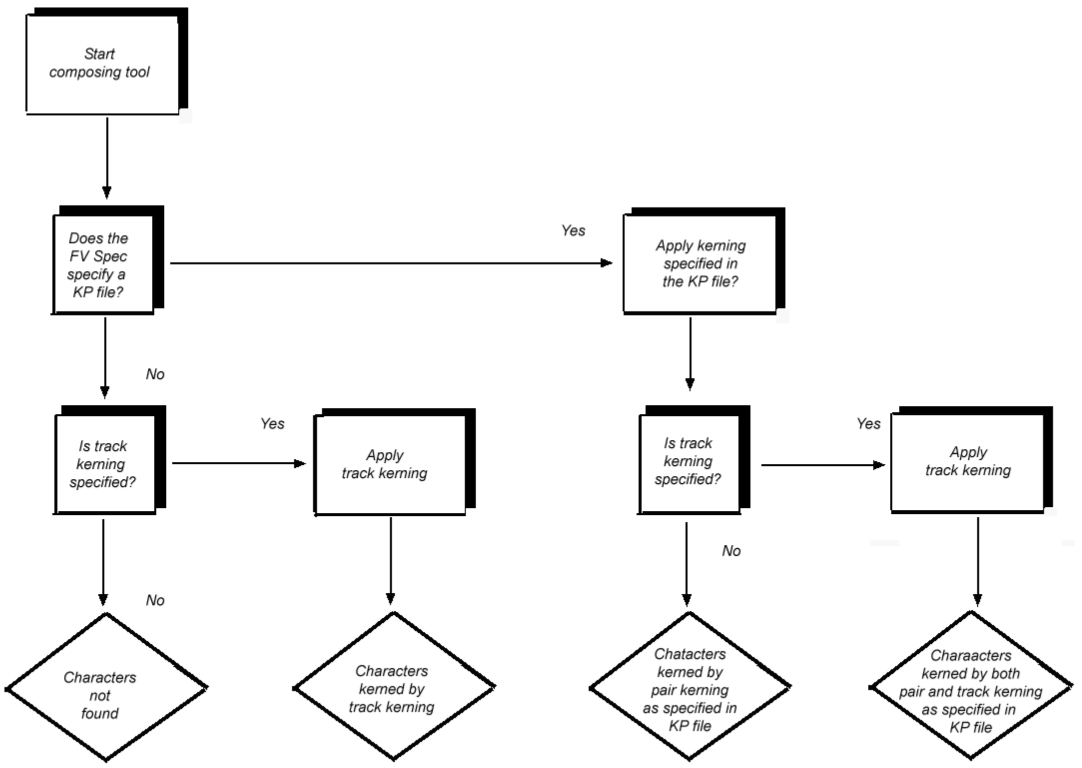

Start composing tool
Does the FV Spec specify a KP file?
Yes
Apply kerning specified in the KP file?
No
Is track kerning specified?
Yes
Apply track kerning
Is track kerning specified?
Yes
Apply track kerning
No
No
Characters not found
Characters kerned by track kerning
Chatacters kerned by pair kerning as specified in KP file
Charaacters kerned by both pair and track kerning as specified in KP file

Figure 15-2 *How the System Applies Pairs and Track Kerning*

When to Edit KP Specs

Edit KP Specs when one of the following conditions is true:

- You add a font and want to specify kerning.
- You want to add a kerning pair to a spec for an existing font.
- You want to adjust the amount of kerning specified for an existing pair.

The widths of glyphs differ from typeface to typeface; the amount of kerning for a pair in one typeface may not be right for the same glyphs in another typeface. You may need to create a separate KP Spec for each typeface or size range.

Delivered KP Specs

XPP delivers KP Specs (based on kerning pairs data in the AFM files) for the delivered Noto PostScript text fonts to the K*noto* library. XPP also delivers their corresponding machine-readable _kp_#####.x files to that library. Build FAST generates KP Specs for PostScript fonts when those fonts include kerning pairs data from the PostScript font vendor.

Kerning Pairs Data in Type 1 Font AFM Files

Some Type 1 PostScript font vendors supply kerning pairs data in the Adobe Font Metrics (AFM) files. Some Type 1 font vendors do not supply kerning pairs data in the AFM files. The number of kerning pairs varies from vendor to vendor.

Note: *In AFM files, negative values remove white space from between two glyphs. However, in the XPP KP Specs, positive values remove white space.*

In the following example, a line of kerning pairs data, in an AFM file, reads:

```
KPX A y -92
```

- The -92 moves the A and y closer together by 92 units (based on 1000 units per em).
- The corresponding entry in an XPP KP Spec contains 92 in the *Kern Amount* field (not -92). Conversely, positive values in AFM files are entered as negative values in KP Specs.

Kerning Pairs Libraries

The first time you store a KP Spec in a new L*libraryname*, XPP automatically creates a destination kerning pairs library called K*libraryname* if it does not already exist. It then creates the machine-readable version of the spec and places it in the destination library. If you edit other KP Specs in that source library, the system places a machine-readable version of the corresponding specs in the destination library.

You can create and edit Kerning Pairs Specs in the same font libraries that contain source font specs, such as the PTS and FGS Specs and so on.

KP Specs in the Font Spec Library

The following figure shows an example of the source and destination libraries when KP Specs are created and edited in the same library as the font specs.

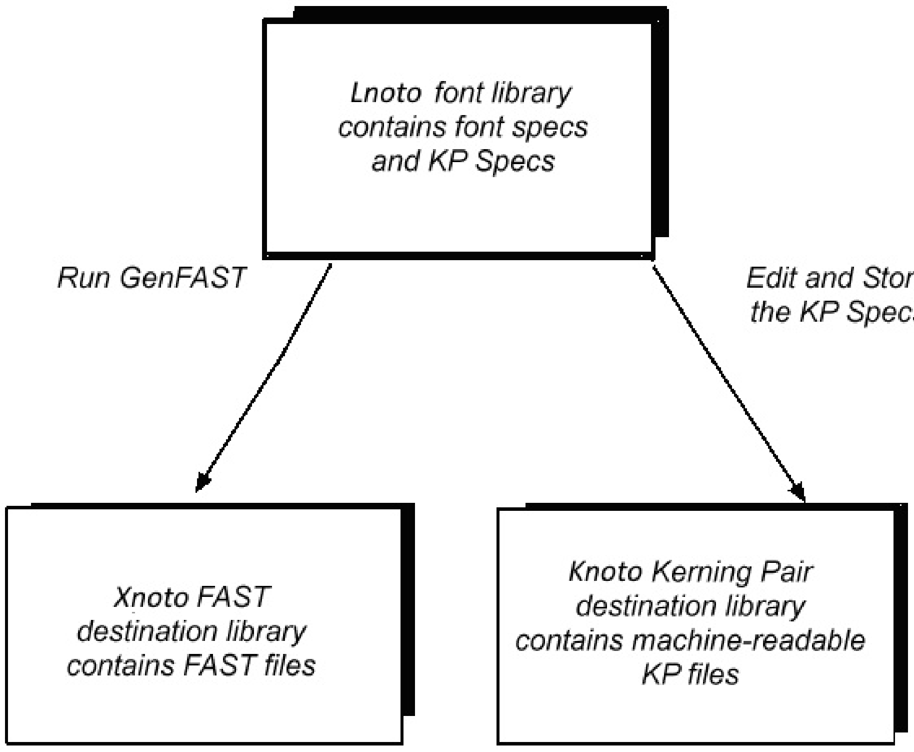

Lnoto font library contains font specs and KP Specs
Run GenFAST
Edit and Store the KP Specs
Xnoto FAST destination library contains FAST files
Knoto Kerning Pair destination library contains machine-readable KP files

Figure 15-3 *Example of KP Specs in the Font Library*

Setting Up a KP Spec

Naming a KP Spec

When creating a KP Spec, you assign it a name consisting of the _kp_ prefix followed by a name consisting of up to eight alphanumeric characters, For example, you can give the spec a numeric name such as _kp_00001.sde. Or you can give the spec a name that reflects the font with which it will be used, such as _kp_timesrom.sde.

Accessing KP Specs

Access the Kerning Pairs Spec from PathFinder using the following sequence:

STYLE LIBRARIES > L*library* **> Kerning Pairs > Specific KP Spec**

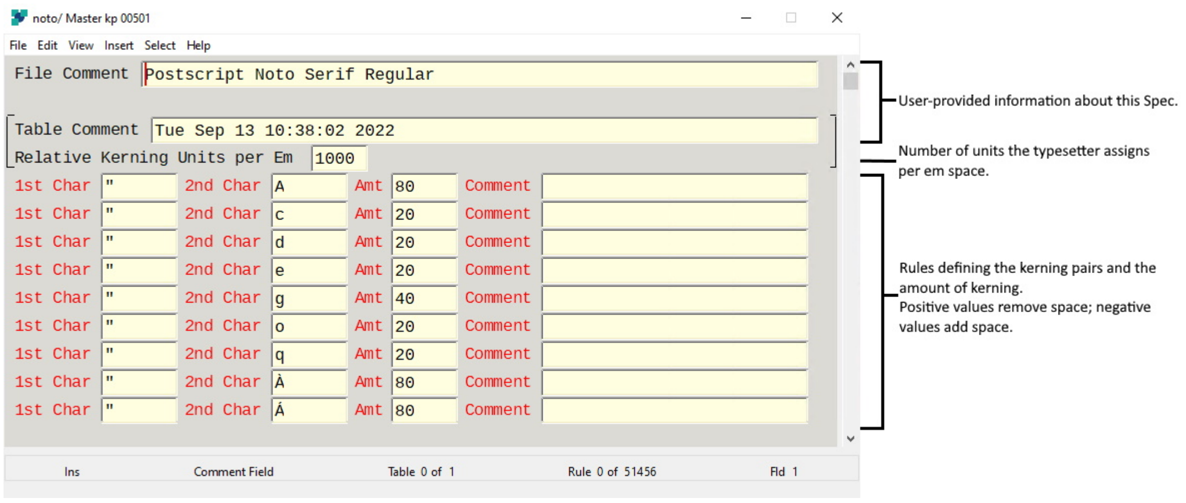
You may find Kerning Pairs Specs in more than one library.

Structure of a KP Spec

The KP Spec consists of one table with the following sections:

- Header — contains a File Comment field, a Table Comment field, and a Relative Kerning Units per Em Field.
- Rule — contains fields for specifying the characters in a kerning pair, the amount of space you want to add or subtract between the glyphs, and a comment.

The following figure shows the structure of the KP Spec.


noto/ Master kp 00501
File Edit View Insert Select Help
User-provided information about this Spec.
Number of units the typesetter assigns per em space.
Rules defining the kerning pairs and the amount of kerning.
Positive values remove space; negative values add space.
Ins
Comment Field
Table 0 of 1
Rule 0 of 51456
Fld 1

File Comment: Postscript Noto Serif Regular

Table Comment: Tue Sep 13 10:38:02 2022

Relative Kerning Units per Em: 1000

1st Char	"	2nd Char	A	Amt	80	Comment	
1st Char	"	2nd Char	c	Amt	20	Comment	
1st Char	"	2nd Char	d	Amt	20	Comment	
1st Char	"	2nd Char	e	Amt	20	Comment	
1st Char	"	2nd Char	g	Amt	40	Comment	
1st Char	"	2nd Char	o	Amt	20	Comment	
1st Char	"	2nd Char	q	Amt	20	Comment	
1st Char	"	2nd Char	À	Amt	80	Comment	
1st Char	"	2nd Char	Á	Amt	80	Comment	

Header Fields

The header fields in the KP Spec consist of: *File Comment*, *Table Comment*, and *Relative Kerning Units per Em.* The *File Comment* and *Table Comment* fields contain information provided by the user or Build FAST. The *Relative Kerning Units per Em* field is described below.

Relative Kerning Units per Em

Enter the number of units your typesetter assigns per em space. For example, PostScript devices use 1000 units per em space.

Entry	*Description*
integer	An integer in the range 0 through 65535. This field entry defaults to 1000.

Rule Fields

Create a rule in the KP Spec for each kerning pair you want to define. XPP can handle a maximum 6,553,500 kerning pairs.

Rule Field: 1st Char and 2nd Char

Enter the first character in the kerning pair you are specifying in this rule. This field accepts only one character; you cannot enter multiple characters. If the font has ligatures, such as *fl*, you can enter a ligature in this field. The system treats a ligature as a single character.

To enter characters from alternate keyboards, such as ligatures and *Pi* characters:

1. Access the keyboard. For example, the *fl* ligature is on the Default keyboard. Press Shift + F1 + d to access the Default keyboard.
2. Press the key for the desired character. In this example, press [L] (Shift + l) for the *fl* ligature;

   The system puts the *fl* ligature in the field.

To enter a Unicode value, type Shift + F2 and you will be prompted to enter up to 6 hex digits, then press Enter.

You will see either the character itself, a XYASCII representation of the character, or a character representation that starts with a ’tree’ symbol, 2 stacked triangles, followed by hex digits. For example:

1. Press Shift + F2.
2. Enter 391.
3. Press Enter.

   XPP displays the Greek capital letter Alpha which has the Unicode value 0x0391.

If the Xyvision Character Set (XCS) Spec defines an ASCII escape sequence for the character, the system may enter a representation of that sequence in this field. In this case, the vertical bar (|) that appears in the XCS Spec appears as four dots (::) in this field. The remaining two characters of the escape sequence follow the four dots.

Entry	*Description*
character	A valid XSF character or escape sequence.

Kern Amount

Enter the amount of space to add or subtract between the glyphs in the kerning pair.

- Positive values *remove* space from between the glyphs; that is, move the second glyph to the left, closer to the first glyph.
- Negative values *add* space between the glyphs; that is, move the second glyph to the right, away from the first glyph.

Entry	*Description*
integer	An integer in the range -32767 to 32767. Default is 0.

Comment

A comment describing the kerning pair.

Entry	*Description*
string	A comment as long as 26 alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, /), and integers 0-9.

Generating Kerning Pairs Test Divisions

Use the Kerning Pairs Utility (*kp_pairs.pl*) to generate test divisions in the CLS_xpputils/GRP_kern/JOB_*library* from existing Kerning Pairs (KP) Specs. The utility prompts you for the name of the Font (or FAST) Library, a Font Variant Spec, and typeface parameters.

Based on the information you provide, the *kp_pairs* utility generates a composed division with four sets of pages — one set of pages for each of the four kerning tracks defined in the Font Variant Spec. Each set of pages contains the pairs from the Kerning Pairs Spec in the font family, font variant and point size specified, followed by a short piece of sample text that shows the results of using kerning pairs and track kerning.

Typeset and review the test division. If you want to change the aesthetics of a typeset page, you can change the kerning pair values in the KP Spec and/or the kerning track values in the Font Variant Spec, then rerun the utility.

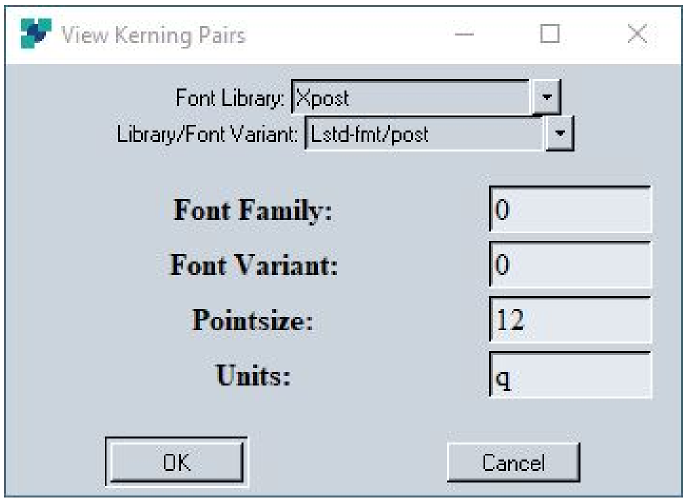
Running the kp_pairs Utility

To run the *kp_pairs* utility from PathFinder:

1. Access the KP Spec you want to process, using the following sequence:
   STYLE LIBRARIES > L*fontlib* **> Kerning Pairs**
   PathFinder displays the KP Specs in the PathFinder List View.

2. Right-click the **KP Spec** that you want to process.
   PathFinder displays a pop-up menu.

3. Select **Tools > Kerning Pairs Utility**.
   XPP displays the View Kerning Pairs dialog box.


View Kerning Pairs

Font Library: Xpost
Library/Font Variant: Lstd-fmt/post

Font Family: 0
Font Variant: 0
Pointsize: 12
Units: q

4. In the dialog box, fill in the following fields:
   a. *Font Library:* field— select the FAST library.
      This utility adds the library name to the *Font Width Library* field in the Job Ticket, which is used for the resulting division.
   b. *Library/Font Variant:* field—specify a library-level Font Variant Spec (FV).
   c. Enter values in the *Font Family*, *Font Variant*, *Pointsize*, and *Units* fields.
5. Click the **OK** button to start processing.
   The *kp_pairs* utility displays a message box stating that the process is complete and displays the location of the kerning pairs division: CLS_xpputils/GRP_kern/JOB_*fontlib*/DIV_*kpname* path. The job name is the name of the Kerning Pairs library and the division name is the name of the KP Spec. If the job path does not already exist, the utility creates it for you.

   For example, if you generate a test division for Kerning Pairs Spec **00502** in the **noto** library, the utility generates the following test division:
   CLS_xpputils/GRP_kern/JOB_noto/DIV_00502.

   The message box asks if you want to examine the division?
6. Click the **yes** button.
   XPP opens the division in the XyView.

   —or—

   Click the **no** button.
   You can view it later.

Notes About the Utility

When using the utility, note the following:

- Each time you run the *kp_pairs* utility and specify a new FAST library and/or Font Variant Spec, the *kp_pairs* utility updates the *Font Width Library* and *Font Variant Spec* fields in the Job Ticket with the new names. If you run the utility with a different FAST library or FV Spec, then recompose previously-generated divisions, the changes in the Job Ticket may affect the appearance of kerning pairs and other text in the divisions.
- The test division displays the current kern value (if any) after each kerning pair. This value helps eliminate confusion about what kerning pairs values were in effect before track kerning was applied and helps you while editing the KP Spec to make adjustments.

Related Information

- If you have specified any replacements in the *Ligature/Accent Replacement* field of the Font Variant Spec, XPP replaces any specified characters before applying kerning. For more information, see “The Font Variant Spec (FV)” on page 12-1 and “The Ligature/Accent Replacement Spec (RP)”on page 16-1.
- For information about setting track kerning, see “The Font Variant Spec (FV)” on page 12-1.

Chapter 16

The Ligature/Accent Replacement Spec (RP)

This chapter contains the following information on the Ligature/Accent Replacement (RP) Spec. When using OpenType fonts, you may not need to use the RP Spec or will use it only for common ligature replacements:

- Understanding the RP Spec
- The standard RP Spec
- Modifying the RP Spec
- Viewing ligature/accent replacement

Understanding the RP Spec

Using the RP Spec, you specify whether you want the system to replace either one of the following:

- Accents with a different accent or with the same accent positioned differently.
- Characters with the corresponding *ligature*—a combination of characters, usually with less space between them than if you entered them individually—that the system treats as one character.

You specify the characters you want to replace in the *Lig/Accent Replace* field of the FV Spec. Using the Ligature/Accent Replacement Spec affects composition, screen display, and output.

Unicode Non-spacing Marks

XPP 8.x and later uses the Unicode non-spacing marks model, which means that a Unicode base character, followed by a number of non-spacing marks (typically accents), dynamically stack the marks over (or under) the base character. A base character can be an actual character, a fixed space, variable space, non-breaking space, or a ligature. Using XPP 8.x and later, you can “float” a maximum of 10 marks.

When you use this model, composition behaves in the following ways:

- Uses the Unicode character table combining class to determine whether it is a non-spacing mark.
- Ignores the widths of the non-spacing marks and justifies the line accordingly.
- Does not hyphenate between the base character and any non-spacing marks.
- Accepts a limit of ten (10) non-spacing marks.
- If there is a custom RP Spec rule for the same base/accent combination, that will override the dynamic non-spacing mark behavior to ensure backward compatibility with the RP Spec behavior.

Using this method means that you no longer need to set up all the character combinations in the Replace Table Spec and instead, can just use the Unicode non-spacing marks you need.

You can turn on non-spacing marks from the *Enable non-spacing marks* field in the Division Ticket and in the Job Ticket. In the Division Ticket, you can select "**yes, no, default**" and in the Job Ticket, you can select "**yes, no**". Default in the Division Ticket means to use the *yes/no* setting in the Job Ticket. All existing jobs have the Job Ticket set to *no* by default and the Division Ticket set to *default*.

Replacing Accents

You can replace certain accents with other accents. For example, your font has both uppercase and lowercase accents. When a lowercase accent appears with an uppercase character, you can replace the lowercase accent with an uppercase accent.

Or, if your font has only lowercase accents, using rules in the RP Spec, you can properly place the accents using the Accents (*acm)* XyMacro.

Replacing Characters with a Ligature

You may want to replace certain characters with the corresponding ligature. For example, you could specify replacing the character combination f i with the corresponding ligature fi.

How Composition Uses the RP Spec

You specify the characters you want to replace in the *Lig/Accent Replace* field of the Font Variant (FV) Spec. You can specify certain ligatures, all ligatures, accents (in various settings), or both ligatures and accents. If the *Lig/Accent Replace* field in the currently active rule of the FV Spec contains the entry **none**, composition does not use the RP Spec to replace any characters in text.

Composition uses the following look-up procedure to determine which characters to replace:

1. Processes the *Lig/Accent Replace* field entry in the FV Spec to determine which rules to look for in the RP Spec (refer to the section "How Composition Uses the *Lig/Accent Replace* Field" on page 16-4).
2. Checks the RP Spec for rules with *Ligature Mask* fields that match according to the results of step 1.
3. Checks the *Input String* field to determine which characters to look for in the division when it finds a matching *Ligature Mask* field.
4. Checks the text in the division for the characters.

5. Replaces the characters with the contents of the *Output String* field of the corresponding RP Spec rule.

The following figure shows how composition processes these specs and the text in the division.

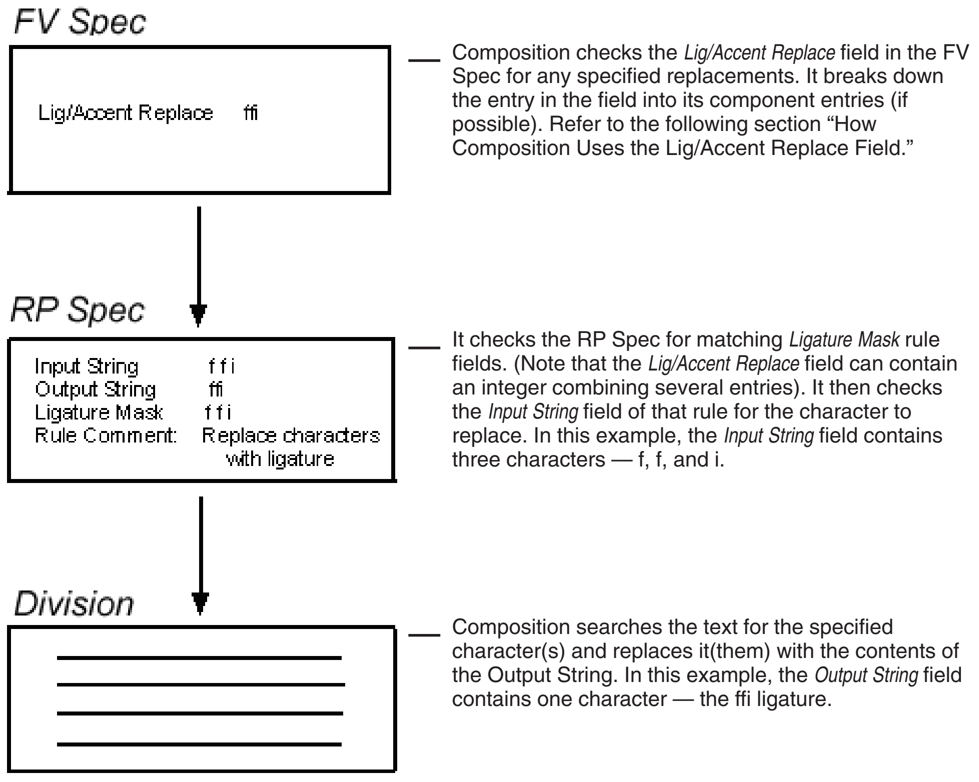

FV Spec
Lig/Accent Replace ffi
Composition checks the Lig/Accent Replace field in the FV Spec for any specified replacements. It breaks down the entry in the field into its component entries (if possible). Refer to the following section "How Composition Uses the Lig/Accent Replace Field."
RP Spec
Input String f f i
Output String ffi
Ligature Mask f f i
Rule Comment: Replace characters with ligature
It checks the RP Spec for matching Ligature Mask rule fields. (Note that the Lig/Accent Replace field can contain an integer combining several entries). It then checks the Input String field of that rule for the character to replace. In this example, the Input String field contains three characters — f, f, and i.
Division
Composition searches the text for the specified character(s) and replaces it(them) with the contents of the Output String. In this example, the Output String field contains one character — the ffi ligature.

Figure 16-1 *How Composition Uses the RP Spec*

How Composition Uses the Lig/Accent Replace Field

To determine which RP Spec rules to use, composition first breaks the *Ligature/Accent Replace* field entry in the FV Spec into powers of 2 (e.g., 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024).

For example, the *Ligature/Accent Replace* field in the FV Spec contains the entry **all**. This is equivalent to an entry of 31 which breaks down into the sum of the following powers of 2: $2^4 = 16$ (fl), $2^3 = 8$ (ff), $2^2 = 4$ (fi), $2^1 = 2$ (ffi), and $2^0 = 1$ (ffl). Composition uses the RP Spec rules with **16**, **8**, **4**, **2**, and **1** in the *Ligature Mask* field.

Accessing the RP Spec

Access the Ligature/Replacement Spec in the Lsyslib library using the following sequence:

STYLES LIBRARIES > Lsyslib > Ligature–Accent Replacement > sys

Editing the RP Spec

The system uses only the Ligature/Accent Replacement Spec, named *_rp_sys.sde*, in the Lsyslib library. Do not create other RP Specs or rename *_rp_sys.sde*. Although you may modify this spec, generally, you do not need to.

However, you should check the *_rp_sys* Spec to determine whether the existing rules meet your needs. If you determine that you need to create a new rule, gather the following information before beginning:

- Accents—additional accents you want to replace and the characters with which you want to replace them.
- Other characters—any characters you want to replace and the characters with which you want to replace them.

To specify which characters you want composition to replace, perform one of the following:

- Use the entry in the *Ligature Mask* field as the entry in the *Lig/Accent Replace* field of the FV Spec
- Combine the entries of multiple *Ligature Mask* fields to use as the entry in the *Lig/Accent Replace* field of the FV Spec

You see the effects of edits to the RP Spec on text in divisions the next time you compose.

Note: *If you add or delete rules in the standard rp_sys Spec, the rule numbers referred to in this document may no longer apply.*

The Standard RP Spec

XPP delivers the Ligature/Accent Replacement Spec named *_rp_sys.sde* to the library named Lsyslib. This spec contains characters that composition can replace, including ligatures and accents.

The system will only use the *_rp_sys.sde* Spec. Do not rename this spec or create additional RP Specs. However, you may modify this spec.

The rules in *rp_sys* are organized according to function:

- Rules 1 - 5 are for ligature replacement.
- Rules 6 - 11 are for fonts with both lowercase and uppercase accents.
- Rules 11-24 are for fonts with only lowercase accents.

Ligatures

Rules 1 through 5 in the standard XPP-delivered spec specify replacement of characters with the corresponding ligature. The following table shows the character combinations and the ligature that replaces them.

Table 16-1 *Ligature Replacement Rules*

Rule	*If composition finds the characters...*	*It replaces them with the ligature...*
1	f f l	ﬄ
2	f f i	ﬃ
3	f i	ﬁ
4	f f	ﬀ
5	f l	ﬂ

Although the contents of the *Output String* field appear to be the same as the contents of the *Input String* field, they are not. The *Input String* field contains individually entered characters; the *Output String* field contains one character — a ligature. Position the cursor on the contents of these fields to see that the contents differ.

If the *Lig/Accent Replace* field of the FV Spec specifies **all** (alone or in combination with other entries), composition uses all the ligature rules.

Refer to the section “Structure of the RP Spec” on page 16-11 for descriptions of the fields and their valid entries.

Accents

Rules 6 through 24 in the standard XPP-delivered *_rp_sys.sde* Spec are for various types of replacing accents. The following sections describe the contents of the rules and how composition uses the information.

Note: *Frequently, multiple rules have the same ligature mask. This is because each rule may denote a different contiguous range or set of Unicode codes.*

Lower and Uppercase Accents in Font

Rules 6 through 11 are useful for configurations in which the font has both lowercase and uppercase accents but does not have combined accented characters. These rules replace lowercase accents appearing over uppercase letters with uppercase accents in floating accents only, that is, accents entered using the Accent (F2) key. These rules do not affect accents entered using the Alternate Keyboard (F1, *Pi* keys, or non-spacing accents automatically being floated over base characters.

The following table explains the character combinations that composition replaces and the corresponding replacement characters. The RP Spec rules contain the escape sequences that correspond to these Unicode values. Refer to the XPP document, *Xyvision Character Set*, to determine the characters corresponding to the referenced Unicode values.

Table 16-2 *Rules for fonts with Both Lower- and Uppercase Accents*

Rule	*If composition finds...*	*It replaces...*
6	a lowercase accent with Unicode 0x327, 0x30A, 0xF02EC, 0x311, 0x328, 0x313, 0x301, 0x300, 0x315, 0x303, 0x306, 0x304, 0x30C, 0x302, 0x308, x307, or 0x30B followed by an uppercase letter	the accent with the corresponding uppercase accent (Unicode 0xF0416–xF0425 or 0x02DD).
7	a lowercase math accent with Unicode 0x20D6, O x20D7, 0x20D0, ox20D1, or 0xF048F–0xF0493 followed by an uppercase letter	the accent with the corresponding uppercase accent (Unicode 0xF0427–0xF042F)
8	a lowercase accent with Unicode 0x30A (lowercase bolle) followed by a lowercase i	the i with a dotless i (ı)
9	a lowercase accent with Unicode 0x311 (lowercase inverted breve) followed by a lowercase i	the i with a dotless i (ı)
10	a lowercase accent with Unicode 0x313, 0x301, or 0x300 followed by a lowercase i	the i with a dotless i (ı)

Table 16-2 *Rules for fonts with Both Lower- and Uppercase Accents (Continued)*

Rule	*If composition finds...*	*It replaces...*
11	a lowercase accent with Unicode 0x303, 0x306, 0x304, 0x30C, 0x302, x0308, 0x307, or 0x30B followed by a lowercase i	the i with a dotless i (ı)

The *Ligature Mask* field for these rules contains the entry **accents**. If the *Lig/Accent Replace* field of the FV Spec specifies **accents** (alone or combined with other entries), composition uses all these rules.

Rules 8 through 11 replace a dotted i with a dotless i (ı) in cases where the accent would interfere with the dot over the i. There are four rules (8 through 11) for replacing the i with a dotless i because the Unicode values are not contiguous. Unicode values 0xF02EC, 0x328, and 0x315 are for accents where it is not necessary to replace the i.

It is not necessary to replace the i with a dotless i in cases where the accent floats under or next to the base character, such as the lowercase virgule (Unicode 0xF02EC), the lowercase hook (Unicode 0xF02EE), and the lowercase apostrophe beside accent (Unicode 0x315). The spec also does not contain lowercase math accents because they are not typically used with a dotless i.

Lowercase Accents Only in Fonts

Rules 12 through 24 are useful for configurations in which the font has lowercase accents only and does not have combined accented characters. These rules replace uppercase or lowercase accents over uppercase characters with lowercase accents properly positioned over the uppercase character. These rules affect floating accents only, that is, accents entered using the Accent(F2) key; they do not affect accents entered using the Alternate Keyboard (F1), [Pi] keys, or non-spacing accents automatically being floated over base characters.

Note that the *Ligature Mask* field for rules 12 through 24 contains the entry **64**. If the *Lig/Accent Replace* field of the FV Spec specifies **64** (alone or in combination with other entries), composition uses all these rules.

Rules 12 through 20 output an Accents (*acm*) XyMacro with the appropriate base and accent characters. Using rules 12 through 20, composition corrects instances where the text contains a lower- or uppercase accent over an uppercase character, but the font has only lowercase accents. Composition does not replace accents that appear below the base character. The Accents (*acm*) XyMacro floats the accent over an uppercase character.

To position the base character and the accent, the standard Accents (*acm*) XyMacro does the following:

1. Outputs the uppercase base character.
2. Shifts the baseline up by 2⁄10 of the current font height.
3. Moves left by half the width of the character and half the width of the accent.
4. Outputs the lowercase accent (now properly positioned over the base character).
5. Restores to the horizontal and vertical position past the character.

The following table describes the character combinations that composition replaces when using rules 12 through 24. The RP Spec rules contain the escape sequences that correspond to these Unicode values. Refer to the XPP document *The Xyvision Character Set* to determine the characters corresponding to the referenced Unicode values.

Table 16-3 *Rules for fonts with Lowercase Accents Only*

Rule	*If composition finds...*	*It replaces...*
12	uppercase accents with Unicode numbers 0xF0417–0xF0419 preceding an uppercase letter	the accent with the corresponding lowercase accent using the Accents (*acm*) XyMacro
13	uppercase accents with Unicode numbers 0xF041B–0xF0425, or 0x2DD preceding an uppercase letter	
14	uppercase math accents with Unicode numbers 0xF0427–0xF042F preceding an uppercase letter	

Table 16-3 *Rules for fonts with Lowercase Accents Only (Continued)*

Rule	*If composition finds...*	*It replaces...*
15	a lowercase math accent with Unicode number 0xF02E7 preceding an uppercase letter	the accent and character with the Accents (*acm*) XyMacro using the same accent and character
16	lowercase accents with Unicode numbers 0x30A, 0xF02EC, or 0x311 preceding an uppercase letter	
17	lowercase accents with Unicode numbers 0x313, 0x301, 0x300, 0x315, 0x303, 0x306, 0x304, 0x30C, 0x302, 0x308, 0x307, or 0x30B preceding an uppercase letter	
18	lowercase math accents with Unicode numbers 0xF046C–0xF0475 preceding an uppercase letter	
19	lowercase math accents with Unicode numbers 0xF0478–0xF047E preceding an uppercase letter	
20	lowercase math accents with Unicode numbers 0x20D6, 0x20D7, 0x20D0, 0x20D1, or 0xF048F–0xF0493 preceding an uppercase letter	
21	lowercase accent with Unicode number 0x30A followed by a lowercase dotted i	the lowercase dotted i with a lowercase dotless i (ı)
22	lowercase accent with Unicode number 0x311 followed by a dotted i	
23	lowercase accents with Unicode numbers 0x313, 0x301, or 0x300 followed by a lowercase dotted i	
24	lowercase accents with Unicode numbers 0x303, 0x306, 0x304, 0x30C,0x302, 0x308, 0x307, or 0x30B followed by a lowercase dotted i	

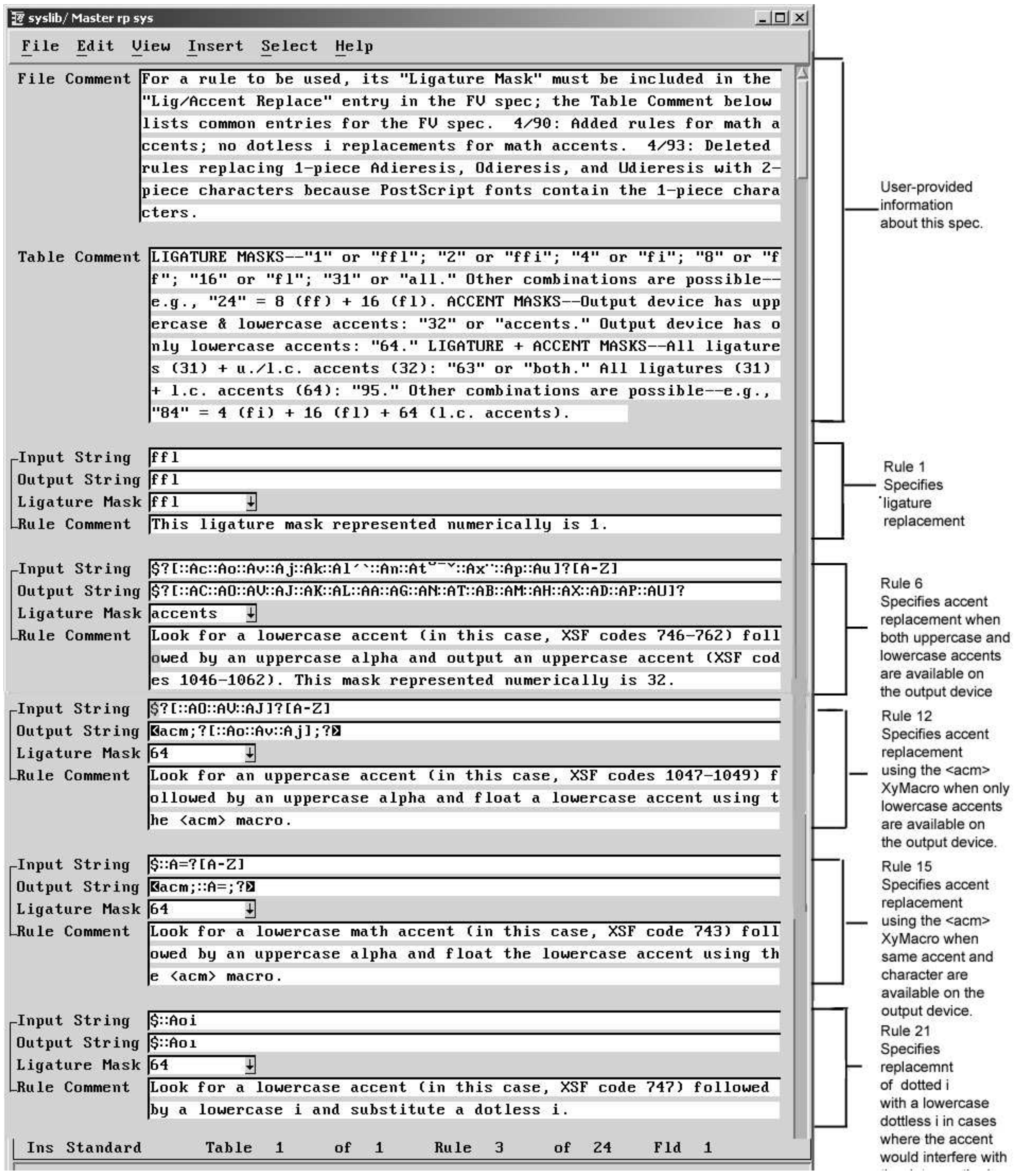
Rules 21 through 24 replace a dotted i with a dotless i (ı) in cases where the accent would interfere with the dot over the i. These rules function the same as 8 through 11 with one exception: the *Ligature Mask* field contains the entry **64** rather than **accents**. The rules are repeated this way so that they are available using either entry (alone or in combination with other entries) in the *Lig/Accent Replace* field of the FV Spec.

Structure of the RP Spec

The RP Spec consists of the following sections:

- Header — contains comment fields
- Rules — rules specifying the character to replace, the character to replace it with and a comment

The following figure shows the structure of the first, sixth, twelfth, fifteenth, and twenty-first rule of the *rp_sys* Spec.


syslib/ Master rp sys
File Edit View Insert Select Help
User-provided information about this spec.
Rule 1 Specifies ligature replacement
Rule 6 Specifies accent replacement when both uppercase and lowercase accents are available on the output device
Rule 12 Specifies accent replacement using the <acm> XyMacro when only lowercase accents are available on the output device.
Rule 15 Specifies accent replacement using the <acm> XyMacro when same accent and character are available on the output device.
Rule 21 Specifies replacemnt of dotted i with a lowercase dottless i in cases where the accent would interfere with

```
File Comment  For a rule to be used, its "Ligature Mask" must be included in the
              "Lig/Accent Replace" entry in the FV spec; the Table Comment below
              lists common entries for the FV spec.  4/90: Added rules for math a
              ccents; no dotless i replacements for math accents.  4/93: Deleted
              rules replacing 1-piece Adieresis, Odieresis, and Udieresis with 2-
              piece characters because PostScript fonts contain the 1-piece chara
              cters.

Table Comment LIGATURE MASKS--"1" or "ffl"; "2" or "ffi"; "4" or "fi"; "8" or "f
              f"; "16" or "fl"; "31" or "all." Other combinations are possible--
              e.g., "24" = 8 (ff) + 16 (fl). ACCENT MASKS--Output device has upp
              ercase & lowercase accents: "32" or "accents." Output device has o
              nly lowercase accents: "64." LIGATURE + ACCENT MASKS--All ligature
              s (31) + u./l.c. accents (32): "63" or "both." All ligatures (31)
              + l.c. accents (64): "95." Other combinations are possible--e.g.,
              "84" = 4 (fi) + 16 (fl) + 64 (l.c. accents).

Input String  ffl
Output String ffl
Ligature Mask ffl
Rule Comment  This ligature mask represented numerically is 1.

Input String  $?[::Ac::Ao::Av::Aj::Ak::Al´`::An::At˘¯ˇ::Ax˙˙::Ap::Au]?[A-Z]
Output String $?[::AC::AO::AV::AJ::AK::AL::AA::AG::AN::AT::AB::AM::AH::AX::AD::AP::AU]?
Ligature Mask accents
Rule Comment  Look for a lowercase accent (in this case, XSF codes 746-762) foll
              owed by an uppercase alpha and output an uppercase accent (XSF cod
              es 1046-1062). This mask represented numerically is 32.

Input String  $?[::AO::AV::AJ]?[A-Z]
Output String ◀acm;?[::Ao::Av::Aj];?▶
Ligature Mask 64
Rule Comment  Look for an uppercase accent (in this case, XSF codes 1047-1049) f
              ollowed by an uppercase alpha and float a lowercase accent using t
              he <acm> macro.

Input String  $::A=?[A-Z]
Output String ◀acm;::A=;?▶
Ligature Mask 64
Rule Comment  Look for a lowercase math accent (in this case, XSF code 743) foll
              owed by an uppercase alpha and float the lowercase accent using th
              e <acm> macro.

Input String  $::Aoi
Output String $::Aoı
Ligature Mask 64
Rule Comment  Look for a lowercase accent (in this case, XSF code 747) followed
              by a lowercase i and substitute a dotless i.

 Ins Standard       Table  1     of   1      Rule   3     of  24      Fld  1
```

Figure 16-2 *Ligature/Accent Replacement Spec*

Header Fields

File Comment and Table Comment

Enter information about the table.

Entry	*Description*
string	A comment as long as 7½ lines (512 alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, /) and the integers 0-9).

Rule Fields

Input String and Output String

Enter the character or string defining the characters you want composition to look for in the text by their Unicode numbers. When composition finds the specified characters, it replaces them according to the entry in the *Output String* field.

Entry	*Description*
string	Up to 65 alphanumeric characters including: ● Direct input of characters from the Standard keyboard (0). ● ASCII escape sequences for characters (from alternate keyboards). The system displays some ASCII escape sequences as character strings and other ASCII escape sequences as single characters. For example, the *Input String* field in rule 10 of the standard RP Spec contains an ASCII escape sequence (::Al) and the grave accent (`). Both characters were entered by pressing Shift + Alternate Keyboard (F1) + Shif + a to activate the Accents keyboard (keyboard A), then selecting the desired character from that keyboard. ● The Accent key (F2) generates the |$$ XYASCII sequence followed by the floating accent character. In a Spec field, the \|$$ XYASCII sequence looks like he accent indicator symbol, ⩚. However, in the RP Spec, a "$" input string represents a floating accent. To enter accent characters, use the Accents Keyboard (Shift + F1 + Shift + a); do not use the Accent key (F2). ● ? for a single wild card.

Entry	*Description*
	• The characters "[" and "]" after "?" to define ranges or sets of Unicode numbers. The system interprets a field entry of [A-Z] as the range of Unicode number 0x41 to Unicode number 0x5A. The multiple character wild card symbol (*) is not a valid entry in this field.

Ligature Mask

Composition uses the entry in the *Lig/Accent Replace* field of the FV Spec along with the entry in the *Ligature Mask* field to determine whether to use this rule.

There are two types of valid entries: Integers and Alphabetic strings

There is a corresponding integer entry for each alphabetic string. You can combine rules by adding the integer entries and entering that sum in this field. The integer entries and their the corresponding alphabetic strings are shown in a following list.

If you enter an integer that has a corresponding alphabetic entry (i.e., 1 for ffl, 2 for ffi, and so on), the integer changes to the corresponding alphabetic entry when you exit the field. For example, if you enter **31**, the integer changes to **all** when you exit the field.

If you enter an integer that does not have a corresponding alphabetic entry (i.e., 17, 64, and so on), the integer does not change when you exit the field.

When creating a user-defined rule, use the next available power of 2. The entry should not already exist and the system should not be able to break it down into predefined entries as described in the section "How Composition Uses the Ligature/Accent Replacement Field" on page 16-4, earlier in this chapter. For example, if the RP Spec contains a rule specifying 128 (the value of 2^7), use the next power of 2 which is 256 (the value of 2^8).

Entry	*Description*
integer	A power of 2 in the range 0 through 2^{31} to represent character replacements (e.g., 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024). When assigning a Ligature Mask to a user-defined rule, use an integer that is a power of 2. Otherwise, the system breaks the entry into powers of 2 and you do not get predictable results. For example, if the *Ligature Mask* field contains the entry **100**, composition breaks the entry into 64, 32 (accents), and 4 (fi). The integers that have defined strings are shown below.

Entry	*Description*
string	An alphabetic string for the character replacement. These are also available with the *Next Choice, Prev Choice* menu options. Below is a list of the defined strings.

Table 16-4 *Defined Entries for the Ligature Mask Field*

Integer	*String*	*Specifies*
0	none	no ligature or accent replacement (default). This value is meant mainly for use in the *Lig/Accent Replace* field of the FV Spec.
1	ffl	replacing the string with the ffl ligature.
2	ffi	replacing the string with the ffi ligature.
4	fi	replacing the string with the fi ligature.
8	ff	replacing the string with the ff ligature.
16	fl	replacing the string with the fl ligature.
20	fifl	replacing the string with the fi and fl ligatures. This value is meant mainly for use in the *Lig/Accent Replace* field of the FV Spec.
31	all	replacing any ligature character combination (ffl, ffi, fi, fl, ff) with the corresponding ligature. This value is meant mainly for use in the *Lig/Accent Replace* field of the FV Spec.
32[1]	accents	replacing accents only.
63[1]	both	replacing both ligatures (31) and accents (32). This value is meant mainly for use in the *Lig/Accent Replace* field of the FV Spec.
64[2]	NA[3]	replacing accents only (using the Accents (*acm*) XyMacro).
84[2]	NA[3]	replacing accents (64) (using the Accents (*acm*) XyMacro) and fi and fl ligatures (20). This value is meant mainly for use in the *Lig/Accent Replace* field of the FV Spec.
95[2]	NA[3]	replacing both accents (64) (using the Accents (*acm*) XyMacro) and ligatures (31). This value is meant mainly for use in the *Lig/Accent Replace* field of the FV Spec.

[1] Use this entry if your font has both uppercase and lowercase accents.
[2] Use this entry if your font has only lowercase accents.
[3] NA = Not Available. This integer does not have a corresponding string entry.

Rule Comment

Enter information about the rule, such as its purpose.

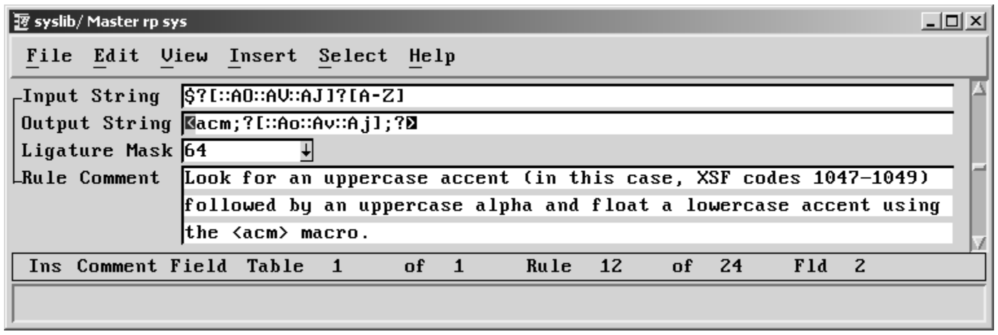
Entry	*Description*
string	A comment as long as 7½ lines (512 alphanumeric characters, including uppercase and lowercase characters, spaces, symbols (such as $, &, /), and the integers 0-9).

Examples of RP Spec Rules

This section contains an example of how composition processes a rule in the *_rp_sys.sde* Spec and an example of the type of rule you may want to add to the *_rp_sys.sde* Spec.

Example 1

This example explains the purpose of rule 12 of the *_rp_sys.sde* Spec, shows how composition uses the rule, and explains the contents of the Ligature Mask, Input String, and *Output String* fields. The following figure shows rule 12 of the *_rp_sys* Spec. Note that if you have added or deleted any rules from the spec, this rule may no longer be rule 12.



```
Input String     $?[::AO::AV::AJ]?[A-Z]
Output String    <acm;?[::Ao::Av::Aj];?>
Ligature Mask    64
Rule Comment     Look for an uppercase accent (in this case, XSF codes 1047-1049)
                 followed by an uppercase alpha and float a lowercase accent using
                 the <acm> macro.
 Ins Comment Field  Table  1    of  1     Rule  12    of  24    Fld  2
```

Rule 12 replaces uppercase accents using Unicode numbers 0xF0417–0xF0419 with the corresponding lowercase accent (Unicode number 0x30A, 0xF02EC, or 0x311). This character becomes the first argument of the Accents (*acm*) XyMacro. The second character of the floating accent string becomes the second argument in the Accents (*acm*) XyMacro (i.e., the uppercase base character). If your font has only lowercase accents but not combined accented characters, this replacement is desirable.

When composition uses rule 12 in the RP Spec *_rp_sys.sde*, it looks for an uppercase accent with an Unicode number 0xF0417 (ASCII escape sequence ::AO) through 0xF0419 (ASCII escape sequence ::AJ) followed by an uppercase character.

If composition finds this character combination, it replaces the characters with the same base character and the appropriate lowercase accent, shifted up by the Accents (*acm*) XyMacro so it will not interfere with the base character.

Following is a description of the *Input String, Output String,* and *Ligature Mask* fields for rule 12 in *rp_sys*.

The *Ligature Mask* field contains the following entry: **64**

The entry **64** denotes replacing floating accents with accents created with the Accents (*acm*) XyMacro. Rules with a **64** in the *Ligature Mask* field are useful for configurations in which the font has only lowercase accents but not combined accented characters.

The *Lig/Accent Replace* field in the FV Spec must contain this entry (alone or in combination with other entries) for composition to use this rule.

The *Input String* field contains the following entry: **$?**[::**AO**::**AV**::**AJ**]**?**[**A-Z**]

where:

$	Tells composition to look for a floating accent, i.e., an accent created using the Accent(F2) key.
?[::**AO**::**AV**::**AJ**]	The first character of the floating accent string is one of the ASCII escape sequences ::AO, ::AV, or ::AJ. This means that the character is an uppercase bolle (Unicode number 0xF0417), virgule (Unicode number 0xF0418), or inverted breve (Unicode number 0xF0419).
?[**A-Z**]	The second character of the floating accent string is an uppercase letter in the range of Unicode number 0x40 through Unicode number 0x5A.

Using the *Input String* field of rule 12, composition looks for a floating accent where the first character has an Unicode number between 0xF0417 and 0xFF0419 and the second character is an uppercase letter.

The *Output String* field contains the following entry: **◀acm;?**[::**Ao**::**Av**::**Aj**]**;?▶**

where:

◀	The opening character of a XyMacro.
acm	The Accents (*acm*) XyMacro. This XyMacro is in the standard XPP Macros Spec, _xy_sys.sde.
;	The XyMacro argument separator.
?[::**Ao**::**Av**::**Aj**]	Composition converts the first character of the input string to the corresponding lowercase character. The Unicode number for the replacement character is 0x30A (ASCII escape sequence ::Ao), 0xF02EC (ASCII escapte sequence ::Av), or 0x311 (ASCII escape sequence ::Aj). For example, the uppercase virgule (Unicode number 0xF0418) is converted to a lowercase virgule (0xF02EC). It then puts the converted lowercase character into the first argument of the Accents *acm* XyMacro.
?	Composition puts the second character of the input string (the uppercase letter) into the second argument of the Accents (*acm*) XyMacro.
▶	The closing character of a XyMacro.

Using the *Output String* field of rule 12, composition replaces the uppercase accents with Unicode numbers 0xF417–0xF419 to the corresponding lowercase accent (Unicode number 0x30A, 0xF02EC, or 0x311). It uses this character as the first argument of the Accents (*acm*) XyMacro. That is, this is the character that composition moves up and properly places over the base character. Composition then uses the second character of the floating accent string as the second argument in the Accents (*acm*) XyMacro (i.e., the uppercase base character).

Example 2

This example shows the type of rule you may want to add to the *rp_sys* Spec.

You want to use the Case Mode (*cm*) XyMacro to change text to small caps. Composition changes the lowercase characters to small caps, but still uses lowercase accents (since uppercase accents are not available in all fonts). If your font has uppercase accents, you may want to add a rule for using the uppercase accents over small caps.

Note: *If your font has only lowercase accents available (e.g., Type 1 fonts), you cannot use the RP Spec to place these accents over small caps characters. Refer to "Accents Over Algorithmic Small Caps" on page 17-1 for information on how to properly place lowercase accents over small caps.*

The following rule replaces lowercase accents over lowercase letters with uppercase accents over lowercase letters. Case mode then changes the lowercase letters to small caps letters. The result is uppercase accents over small caps letters.

The *Ligature Mask* field contains the following entry: **128**

This is a unique entry since it is a power of 2. To use this rule, the *Lig/Accent Replace* field of the active rule in the FV Spec must contain 128 (alone or combined with other entries).

The *Input String* field contains the following entry:
$?[**::Ac::Ao::Av::Aj::Ak::Al::Aa::Ag::An::At::Ab::Am::Ah::Ax::Ap::Au**]**?**[**a-z**]

where:

$	Tells composition to look for a floating accent (i.e., an accent created using the Accent key [F2]).
?[**::Ac::Ao::Av ::Aj::Ak::Al::Aa ::Ag::An::At::Ab ::Am::Ah::Ax ::Ap::Au**]	The first character of the floating accent string is Unicode 0x327, 0x30A, 0xF02EC, 0x311, 0x328, 0x313, 0x301, 0x300, 0x315, 0x303, 0x306, 0x304, 0x30C, 0x302, 0x308, 0x307, or 0x30B These are Unicode numbers for lowercase accents.
?[**a-z**]	The second character of the floating accent string is a lowercase letter.

Using the *Input String* field of this rule, composition looks for a floating accent string consisting of a lowercase accent over a lowercase letter.

Output String Field

The *Output String* field contains the following entry:

$?[::**AC**::**AO**::**AV**::**AJ**::**AK**::**AL**::**AA**::**AG**::**AN**::**AT**::**AB**::**AM**::**AH**::**AX**::**AD**::**AP**::**AU**]**?**

where:

$	Tells composition to output a floating accent**.**
$?[::**AC**::**AO** ::**AV**::**AJ**::**AK** ::**AL**::**AA**::**AG** ::**AN**::**AT**::**AB** ::**AM**::**AH**::**AX** ::**AD**::**AP**::**AU**]**?**	For the first character of the floating accent string, output the uppercase accent that corresponds to the lowercase accent in the input string.
?	Output the second character of the input string as the second character of the floating accent string (i.e., the base character).

Using the *Output String* field of this rule, composition outputs an uppercase accent as the first character of the floating accent string, followed by the second character of the floating accent string (the lowercase letter). The Case Mode (*cm*) XyMacro changes the lowercase letter to an uppercase letter.

Viewing Ligature/Accent Replacements

You can view information about ligature/accent replacements using:

- The Line Editor
- The Softkey menu
- Status window

Line Editor

When you are working in the Line Edit window, you can see when ligature or accent replacements are in effect by the up and down arrows that surround the characters being replaced. The Expand RP button on the Line Editor is a toggle switch that controls whether the Line Editor displays only the input portion of a ligature or expands the display to show both the input and the output portions of any ligature or other replacements from the RP Spec. The Expand RP button works the same whether you are using XML/SGML mode or standard XPP mode.

For example, when in the Line Edit window, you see the input characters:
`↑⇑::AAE↓`

↑↓	Indicates that the characters in between the arrows are being replaced by something else.
⇑	Indicates that the next character is a floating accent, and the character following the accent is the base character. This is a 2 piece accented character.
`::AA`	Is the escape sequence for an uppercase acute accent. In the Line Edit window, the :: represents `|` (pipe).
`E`	Is the base character, the character that the accent is placed over.

To see the output characters (what replaces the input characters):

- Toggle the Expand RP button.
  The Line Editor now displays more information:

↑⇑::AAE ⇄ <acm;';E> 'EE'↓

↑↓	These arrows now surround both the input and output characters.
⇑`::AAE`	Input string.
⇄	Separates input characters from output characters.
<acm;';E> 'EE'	Output string.

In this example, the uppercase acute accent (|AA) is Unicode number 0xF041C, which is in the range of characters listed in rule 13 of the RP Spec. Rule 13 can be used when the font does not have uppercase accents available and also does not have combined accented characters and indicates to replace the uppercase accent with the related lowercase accent and furthermore to use the Accents (*acm*) XyMacro to properly position the lowercase accent. The output string above shows this happening: ::`AA` is replaced with `'` as the first argument of the Accents (*acm*) XyMacro. The base character, `E`, becomes the second argument of the Accents (*acm*) XyMacro. `'EE'` is generated by the Accents (*acm*) XyMacro.

Note: *The lowercase acute accent has an escape sequence of |Aa and is assigned to Unicode number 0x301. The reason that* `'` *rather than* ::`Aa` *displays in the Line Edit window is that the XCS Spec specifies to display the vuem* `'` *character instead. Refer to "The Xyvision Character Set Spec (XCS)", on page 3-1 for information on vuem characters.*

Softkey Menu

You can also use the Softkey menu to change the display of Ligature/Accent replacements.

1. Select **Display > Other Display >Expand RP On/Off** on the Softkey menu.
2. Bring up the Line Edit window again to see any replacements.

The Status Window

The Status window provides helpful font information for character replacement. When you place the cursor on a complicated replacement from the RP Spec (for example, when replacing a floating accent with an Accents (*acm*) XyMacro or with a different floating accent), the Status window displays the names and numbers for the first two characters in the replacement and font numbers only for the third through fifth characters (if there are more than two characters in the replacement). For example, if there are five or more characters in the replacement, the Status window would show something like **Symbol (29)/Times Roman (30)/(30)/(29)/(30)**.

If you place your cursor on a floating accented character and the font for the floating accent is different from the font for the base character, the Status window displays both font name and font number with the same pattern as previously stated, that is, **Symbol (29)/Times Roman (30)**.

If the font is the same for all the characters of a floating accented character or replacement, then the Status window displays only one font name and font number. If the font number of a character can be determined, but there is no entry for the font in the TSF Spec (that is, composition reports “No mapping for font number #”), the Status window displays the font information as **? (#)** instead of **-none-**.

If the Status window displays **(VPX #####)** for the Font information, this indicates that the character is a pseudo character. To see the definition of the pseudo character, open the X*fontlib* entry under STYLE LIBRARIES, click FASTs, right click ##### and choose View Pseudo Characters. If you right-click the character in the Line Editor, XPP displays the Unicode value. You can look up this value in the VPX spec that View Pseudo Characters creates.

Chapter 17

Accents Over Algorithmic Small Caps

This chapter describes how to adjust the placement of accents over algorithmic small caps fonts, which often do not contain all the uppercase accent characters needed.

Moving Accents Over Characters

In fonts where the small caps are created algorithmically (that is, they are not true small caps fonts) and the fonts do not contain uppercase accents, lowercase accents that should display over a glyph crash into the glyph (overprint). You cannot use the Accents (*acm*) XyMacro in the RP Spec to move the lowercase accents up because the Accents (*acm*) XyMacro is active only in normal case mode (upper- and lowercase).

Many Type 1 fonts do not include uppercase accents. But, if your font has uppercase accents available, you can set up the RP Spec to use them instead of having to do one of the following actions.

If your font does not have uppercase accents , you have a few choices:

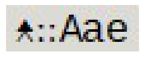
- **Purchase true small caps fonts**

  You may be able to purchase or obtain true small caps fonts.

- **Use one-piece accented characters**

  You may be able to use one-piece accented characters. However, these characters have different Unicode values and therefore must be accessed by different ASCII sequences than with floating accents.

  For example, using a floating accent sequence "|$$|Aae" within algorithmic small caps mode centers a lowercase acute accent (Unicode x301) over a lowercase e (Unicode x65) that is being displayed as an algorithmic small caps E. If you enter this syntax in an ASCII file and use ToXSF to bring the file into XPP, you would see the following in the Line Editor:

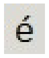
  

  ▲::Aae

  but, you would see the following in the XyView if algorithmic small caps mode is enabled and where the lowercase accent crashes into the top left of the small caps E (Unicode x45):

  Note: XPP supports the |$$ syntax in XML/SGML, but RWS does not recommend that you use it because it produces non-portable XML/SGML.

  By contrast, "|fe" accesses the one-piece accented lowercase character e acute (Unicode xE9). If you enter this syntax in an ASCII file and use ToXSF to bring the file into XPP, you would see the following in the Line Editor:

  é

  but, you would see the following in the XyView if you were using a true small caps font:

  É

- **Set up Pseudofont (PSF) Specs to move the accents up**

  You can set up Pseudofont (PSF) Specs to move the accents up. The following section describes how to do this.

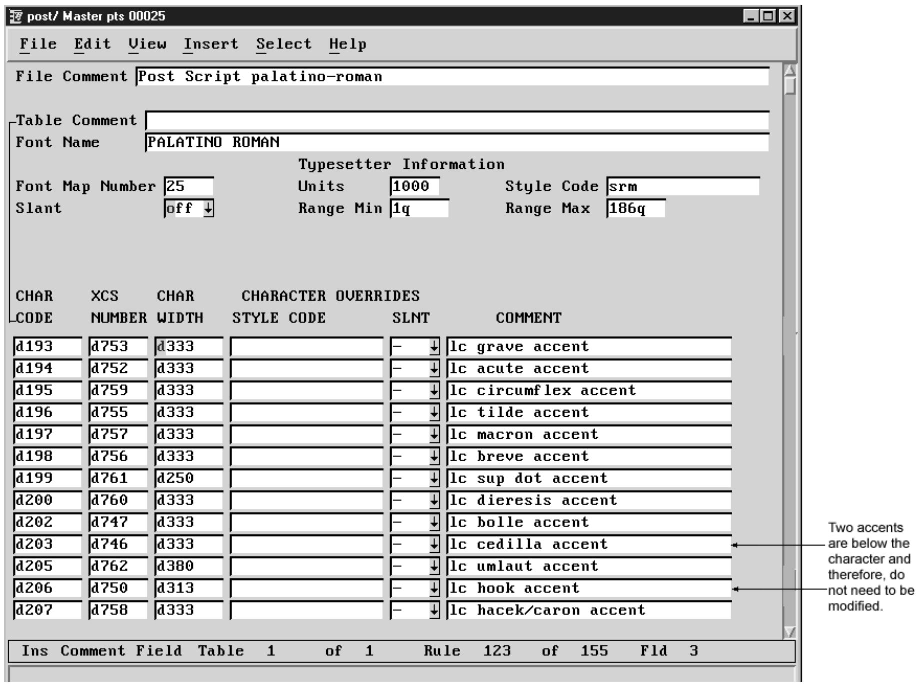
Setting Up PSFs for Correct Accent Placement

To set up PSFs:

1. Determine which accents you want to use over small caps. Fonts may include accents that are positioned above the glyph.

2. Determine which fonts you want to use in small caps, then find the numbers that have been assigned for each font (look at the *_tsf_system.sde* Spec in your font library).

3. In the PTS Spec for each font, print out the rules for the accents. The widths are typically the same for most accents in a font.


post/ Master pts 00025
File Edit View Insert Select Help
Two accents are below the character and therefore, do not need to be modified.

File Comment: Post Script palatino-roman

Table Comment:

Font Name: PALATINO ROMAN

Typesetter Information

Font Map Number: 25 Units: 1000 Style Code: srm

Slant: off Range Min: 1q Range Max: 186q

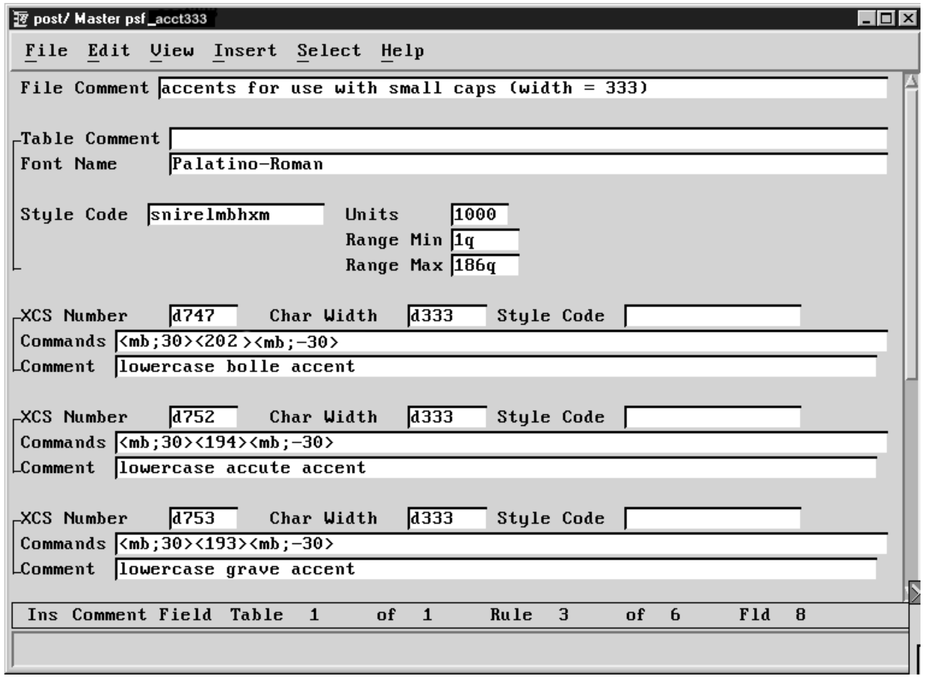
CHAR CODE	XCS NUMBER	CHAR WIDTH	CHARACTER OVERRIDES STYLE CODE	SLNT	COMMENT
d193	d753	d333		–	lc grave accent
d194	d752	d333		–	lc acute accent
d195	d759	d333		–	lc circumflex accent
d196	d755	d333		–	lc tilde accent
d197	d757	d333		–	lc macron accent
d198	d756	d333		–	lc breve accent
d199	d761	d250		–	lc sup dot accent
d200	d760	d333		–	lc dieresis accent
d202	d747	d333		–	lc bolle accent
d203	d746	d333		–	lc cedilla accent
d205	d762	d380		–	lc umlaut accent
d206	d750	d313		–	lc hook accent
d207	d758	d333		–	lc hacek/caron accent

```
Ins  Comment Field  Table  1    of  1    Rule  123   of  155   Fld  3
```

Figure 17-1 *PTS Spec Rules for Accents pts_00025*

4. Using the information from the PTS Specs, set up a PSF Spec for each width. Name the spec to indicate the width (for example, *psf_acct333*). Use <mb;30> to move the accent up by 30% of the font height, or experiment with other values. After placing the accent, move down the same amount to return to the original baseline (for example, <mb;-30>). For more information, refer to ″The Pseudofont Spec (PSF)″ on page 7-1.

The following figure shows sample PSF Spec rules.

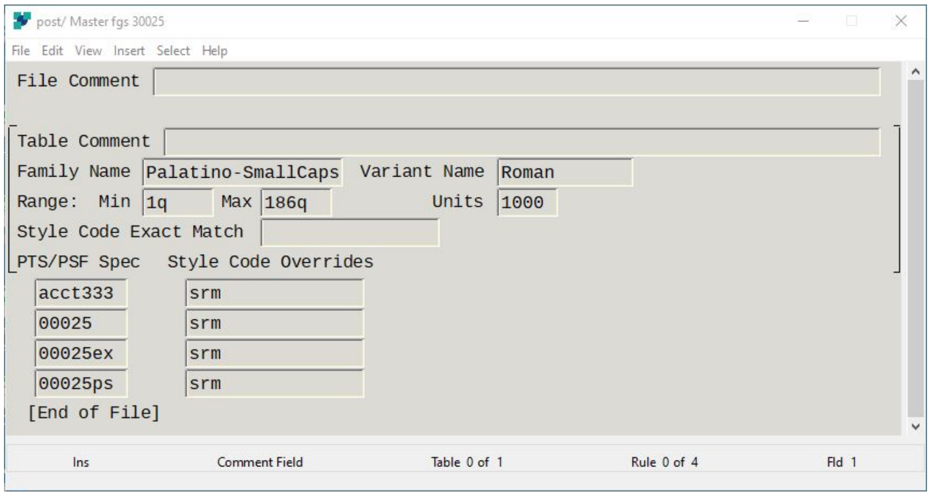


```
File Comment accents for use with small caps (width = 333)

Table Comment
Font Name     Palatino-Roman

Style Code  snirelmbhxm        Units      1000
                               Range Min  1q
                               Range Max  186q

XCS Number    d747     Char Width  d333     Style Code
Commands  <mb;30><202 ><mb;-30>
Comment   lowercase bolle accent

XCS Number    d752     Char Width  d333     Style Code
Commands  <mb;30><194><mb;-30>
Comment   lowercase accute accent

XCS Number    d753     Char Width  d333     Style Code
Commands  <mb;30><193><mb;-30>
Comment   lowercase grave accent

 Ins  Comment Field  Table  1    of   1     Rule  3    of   6     Fld  8
```

Figure 17-2 *PSF Spec psf_acct333 Rules*

5. For each small caps font, copy the existing FGS Spec to a new number. To avoid confusion, use a consistent numbering scheme, such as adding 30,000 to the original number. Edit the new FGS Spec. Modify the *Family Name* field to indicate small caps, and add a new rule for the PSF Spec *before* the rule for the PTS Spec containing the accents.

   The following figure shows a sample FGS Spec.



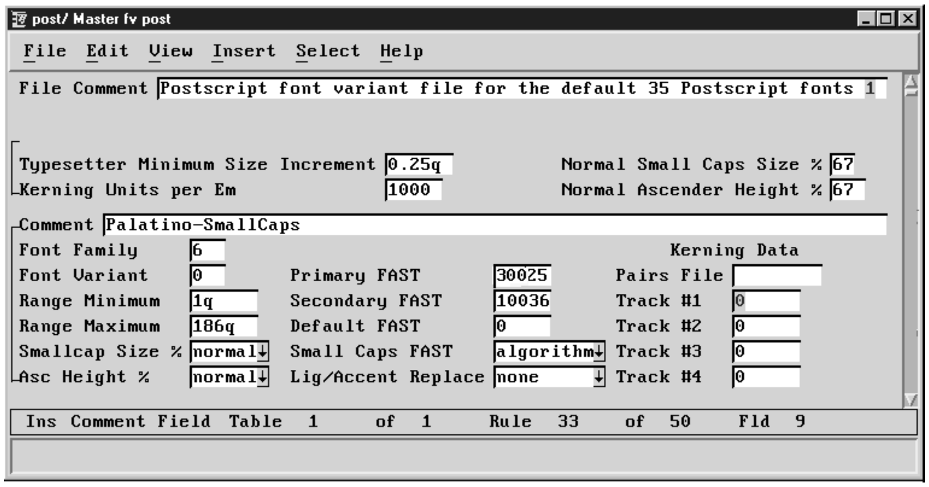
```
File Comment

Table Comment
Family Name  Palatino-SmallCaps  Variant Name  Roman
Range:  Min  1q       Max  186q          Units  1000
Style Code Exact Match
PTS/PSF Spec   Style Code Overrides
  acct333        srm
  00025          srm
  00025ex        srm
  00025ps        srm
 [End of File]

Ins        Comment Field        Table 0 of 1        Rule 0 of 4        Fld 1
```

Figure 17-3 *New FGS Spec*

6. Run GenFAST on the new FGS Spec to create a new FAST; optionally update the *font_desc* file.

7. Edit the master Font Variant Spec. Copy the existing rule and change the *Font Family* and *Font Variant* to a unique set of values and change the value in the *Primary FAST* field of the rule using **algorithm** for the Small Caps FAST to the new FAST number you selected in step 5.

   The following figure shows a sample FV Spec entry, using 30025 as the new FAST number.



```
File Comment Postscript font variant file for the default 35 Postscript fonts 1

Typesetter Minimum Size Increment 0.25q          Normal Small Caps Size % 67
Kerning Units per Em              1000           Normal Ascender Height % 67

Comment Palatino-SmallCaps
Font Family      6                                        Kerning Data
Font Variant     0       Primary FAST        30025      Pairs File
Range Minimum    1q      Secondary FAST      10036      Track #1   0
Range Maximum    186q    Default FAST        0          Track #2   0
Smallcap Size %  normal  Small Caps FAST     algorithm  Track #3   0
Asc Height %     normal  Lig/Accent Replace  none       Track #4   0

Ins Comment Field  Table  1    of  1     Rule  33    of  50     Fld  9
```

Figure 17-4 *New Rule in the FV Spec*

Appendix A

Spec Quick Reference

Table A-1 is a quick-reference showing all the font-related specs, spec mnemonics, spec naming conventions, and the purpose of each spec.

Table A-1 *Spec Quick Reference*

Mnemonic	***Spec***	***Naming Conventions***	***Description***
FGS	Font Generation	A five-digit number between 00001 and 65535. *Example:* _fgs_00015.sde	Lists the PTS and PSF Specs for GenFAST to use in making a FAST.
FGX	Font Generation Exception (optional)	A five-digit number between 00001 and 65535; it must be the same number as the corresponding FGS Spec. *Example:*_fgx_00015.sde	Includes glyphs, not in a PTS or PSF Spec, for inclusion in a FAST.
FV	Font Variant	Up to 8 alphanumeric characters. *Example:* _fv_doc.sde	Maps the font specified in tags or CSS font properties (using the TSF Spec) and the Font Family (*ff*) and Font Variant (*fv*) XyMacros to the FASTs.

Table A-1 *Spec Quick Reference (Continued)*

Mnemonic	*Spec*	*Naming Conventions*	*Description*
FX[1]	FAST (Font Access Table)	A five-digit number between 00001 and 65535; it must be the same number as the corresponding FGS Spec. *Example*: _fx_00015.sde	Provides all font information needed for composition, screen display, and output. The FX file is not accessible for editing. You can view the information through the system-generated VFX Spec.
KB	Keyboard Map	A single alphanumeric character (A-Z, a-z, 0-9).	Maps key caps to Unicode values.
KP	Kerning Pairs *(optional)*	Up to 8 alphanumeric characters, often named the same as the FGS Spec.	Adds/removes space from between specified characters. Source spec file located in L*library*.
KP	Kerning Pairs *machine-readable (optional)*	Up to 8 alphanumeric characters, often named the same as the FGS Spec.	Adds/removes space from between specified characters. Machine-readable .x file located in L*library*.
PSN	PostScript Name	_psn_unicode.sde _psn_custom.sde _psn_ps2xcs.sde	Maps PostScript character (or glyph) names to the Unicode character numbers.
PSF[1]	Pseudofont *(optional)*	Up to 8 alphanumeric characters; do not use the same name as a PTS. *Example:* _psf_math.sde	Modifies typesetter output of glyph(s). Also combines existing glyphs to create a new glyph.
PTS	Phototypesetter	Up to 8 alphanumeric characters, often the same name as the corresponding FGS Spec. *Example:* _pts_00015.sde	Maps Unicode numbers to a manufacturer-assigned phototypesetter code, also provides character information such as width, style, etc.
RP	Ligature/ Accent Replacement *(optional)*	_rp_sys.sde	Specifies replacing an accent with another accent and characters with corresponding ligatures. Used with the Ligature/ accent Replacement field in the Font Variant Spec.

Table A-1 *Spec Quick Reference (Continued)*

Mnemonic	*Spec*	*Naming Conventions*	*Description*
TSF	Typesetter Font Map	_tsf_system.sde	Maps font numbers to font names or to an alternate font number for screen display. For CSS, maps CSS font properties to a FAST and its font name.
XCS	Xyvision Character Set	_xcs_default.sde	Assigns each XSF code a name, unique ASCII sequence, optionally Unicode number(s), optionally named character entities, and a string for use in the Line Editor, spec fields, etc.

[1]FASTs are machine-readable only files. The system can generate the VFX and VPX files for you to view.

Appendix B

Status, Tips, & Troubleshooting

This appendix describes how to check font status, explains font-related error messages, and gives tips and hints for font troubleshooting

Checking Font Status

When editing a division, you can obtain information on fonts by viewing the Status window.

To display the Status window:

- Select **Menu > Status**

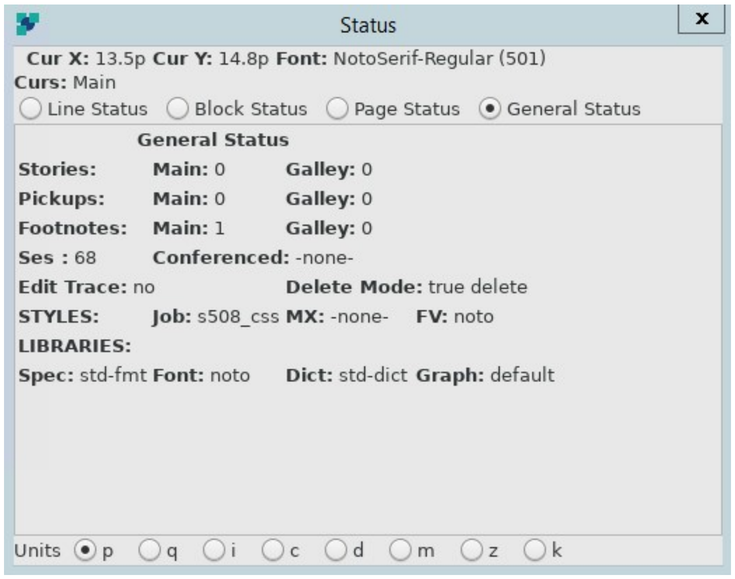
XPP displays the Status window. The XyView also displays the Status menu on the Softkey menu. Its options correspond to the radio buttons in the Status window.

- *Line Status*
- *Block Status*
- *Page Status*
- *General Status*

From Line, Block, Page, or General Status, you can get the name of the font on which your cursor is positioned. (Remember, the Status Window does not dynamically refresh by moving the cursor with the arrow keys.)

- For example, Font family and variant by name—NotoSerif-Regular (501). (501) is the *Font Map No.* field in the TSF Spec, which corresponds to the PostScript font *Name* field.

  Note: *If a "substitute" font is being used for display, the font named in the Status window has a* **$** *in front of it. If a pseudo character is being used, the font will be displayed as* **(VPX #####)**. *Refer to page 16-23 for additional information.*



Status

Cur X: 13.5p **Cur Y:** 14.8p **Font:** NotoSerif-Regular (501)
Curs: Main

Line Status Block Status Page Status [✓ General Status]

```
                 General Status
Stories:       Main: 0       Galley: 0
Pickups:       Main: 0       Galley: 0
Footnotes:     Main: 1       Galley: 0
Ses : 68       Conferenced: -none-
Edit Trace: no               Delete Mode: true delete
STYLES:        Job: s508_css MX: -none-    FV: noto
LIBRARIES:
Spec: std-fmt  Font: noto    Dict: std-dict Graph: default
```

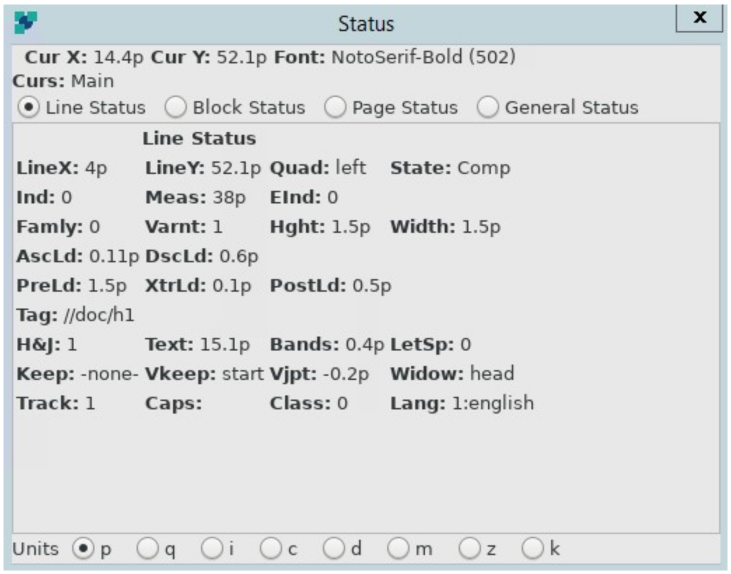
Units [✓ p] q i c d m z k

Figure B-1 *Font Information in the General Status Window*

General Status shows the names of the libraries and specs that composition is currently using for the open division, as defined in the Job Ticket and/or the Division Ticket

In this example, the Font library is noto.

From Line Status, you can obtain the following information on the font that is at the present cursor position:



```
Cur X: 14.4p Cur Y: 52.1p Font: NotoSerif-Bold (502)
Curs: Main
```

[✓ Line Status] Block Status Page Status General Status

```
             Line Status
LineX: 4p     LineY: 52.1p Quad: left   State: Comp
Ind: 0        Meas: 38p    EInd: 0
Famly: 0      Varnt: 1     Hght: 1.5p   Width: 1.5p
AscLd: 0.11p DscLd: 0.6p
PreLd: 1.5p   XtrLd: 0.1p  PostLd: 0.5p
Tag: //doc/h1
H&J: 1        Text: 15.1p  Bands: 0.4p LetSp: 0
Keep: -none- Vkeep: start Vjpt: -0.2p  Widow: head
Track: 1      Caps:        Class: 0     Lang: 1:english
```

Units [✓ p] q i c d m z k

Figure B-2 *Font Information in the Line Status Window*

In this example,
—The Font Family number is 0.
—The Font Variant number is 1.
—The Font Height is 1.5p.
—The Font Width is 1.5p.

Font Messages

The system generates a variety of error messages for fonts. These messages appear when you run Build FAST, access a division for editing, compose text, and so on. The messages may appear in the dialog area of the XyView, the composition log, the print queue log, and so on.

Error messages in the dialog area of the XyView may not show the full message depending on the size of the dialog area. Use Search > View Log on the Softkey menu to display complete error messages after composition. Refer to the XPP document *XML Professional Publisher: Managing XPP* for information on viewing messages in the View Log window.

Table B-1 lists the common font error messages and describes the meaning of each message.

Table B-1 *Font Messages*

Message	***Description***
`Record out of sequence: xsf #### xcs inconsistency(s), cannot continue.`	This message may appear when you run GenXCS. The XCS Spec contains rules with the same XCS field entry. Locate the rule with the specified XCS code; the XCS field must have a unique entry.
`Dup. esc. seq.: xsf #### xcs inconsistency(s), cannot continue.`	This message may appear when you run GenXCS. The XCS Spec contains rules with the same ASCII field entry. XPP has assigned an ASCII character or escape sequence to each XCS code. Do not modify this field. If you have modified this field, locate the rule with the specified XCS code and search for a rule with the same ASCII field entry. Each ASCII field must contain a unique entry.
`The default FAST number 30 in “fv” record 1 is missing.`	This message may appear when you attempt to open a division. It indicates that FAST 30, which is specified in the first record (rule) of the active FV Spec, is not present in the font width library (the “X” library) that is specified in the Job Ticket.
`Unspecified typesetter character! (9) U+E13`	This message may appear when you compose a division. It indicates that Unicode character U+E13 (d3603) was input but is not in the current primary, secondary, or default FAST.

Table B-1 *Font Messages (Continued)*

Message	*Description*
`font family/variant error (11): ff,fv,sh,sw=0,7,11,11`	One or more of these values (specified either in the Item Format Spec or by CSS font properties or in an override macro) is invalid (does not exist in the active Font Variant Spec). The four values in this example are ff=0, fv=7, sh=11, sw=11. In this example, fv 7 does not exist in ff 0 in the active FV Spec.
`Variant 0, family 0, size 10; Missing FAST or FFVAR entry. Font family 0, variant 0 must be defined in your Font Variant Spec.`	This message may appear when you access a division for editing. This message means one of two things: Either the active FV Spec does not include a rule for Font Family 0, Variant 0 that includes 10 point, or the FV Spec rule for Font Family 0, Variant 0 points to a FAST that is not present in the font width library specified in the Job Ticket.
`ERROR: No mapping for font number 30`	Indicates that the TSF Spec in the active font width library does not contain an entry for font 30 (in the Font Map Number field of the TSF Spec), so XPP cannot determine the PostScript font name.

Font Messages from Build FAST

Table B-2 *Messages when you click the* Apply *button in Build FAST*

Error Message	*Description*
`WARNING: Unencoded character` *`charactername`*`, using 0`	The Type 1 font AFM source file contains *charactername* with a character code (`C` value) of -1. This flags the utility to check for any encoding table which should be applied to the font. The utility then scans the encoding table named in the Font Specification window looking for *charactername*. It does not find it and therefore gives this glyph a character code value of 0 in the PTS Spec. If you need access to this glyph, you can copy the encoding table to a unique name and substitute this *charactername* for a glyph you do NOT need access to and RERUN the utility with the new encoding table name specified in the Build FAST window.
`WARNING: line 10 field 'fontvar' (`*`fontname`*`) too long - limit is 12`	This means that the Font Name exceeds the 12-character limit for the Font Family and Font Variant The offending name will be truncated. This will be noticeable in the Status Window display. To see what the utility has done and edit it if you want, edit the FGS Spec fields.

Table B-2 *Messages when you click the* Apply *button in Build FAST (Continued)*

Error Message	***Description***
`No (`*`character name`*`) PSN`	Enter the character name(s) in *psn_custom* or *psn_unicode* in *Lsyslib*, with an appropriate Unicode number and rerun Build FAST.
`WARNING: Font contains (`*`number`*`) kerning pairs; KP table can only hold first 6,553,500; rest will be ignored`	The KP table cannot hold more than 6,553,500 kerning pairs.
`WARNING: CMap code (`*`number`*`) collides with xyps, 0xf800-f8ff; characters in this range will be ignored.`	XPP reserves the code range f800-f8ff for its own use, to define xyps, as allowed by the Unicode specification. Fonts must not define characters in this range.
`WARNING: fontname already exists as font `*`FAST number`*`.`	The FAST number you have selected in the Font Specification window has already been used in this font width library. If you continue, you will overwrite the existing one. The utility detects this by scanning the TSF Spec where it finds the font number already in use.

Tips and Hints

Here is a list of some common problems with fonts and possible causes for each problem.

Table B-3 *Tips and Hints for Fonts*

Symptom	***Possible Cause***
Text appears justified onscreen; however, it is ragged on the output.	Font is not loaded on the output device. Either load the font on the PostScript printer, or download the font on output. —or— Your PTS glyph widths do not match the actual font glyph widths.
Text appears ragged onscreen; however, it is justified on the output.	A substitute font is being used for screen display in the TSF Spec. The substitute font has different widths than the font used for output.
When you enter a character from an alternate keyboard, a reverse-video question mark appears onscreen.	The Unicode character is not defined in any of the FASTs you are using in the current Font Family and Variant rule of the FV Spec.
When you try to enter a character from an alternate keyboard, the system beeps.	The key you pressed is not mapped to a character or string in the KB Spec.
You do not know the ASCII sequence required for creating accented characters.	To obtain the ASCII sequences, you can either: 1. Look in the xcs_default Spec. —or— 2. Enter the characters in a division, run FromXSF, then view the ASCII file.
Lowercase letters in small caps text appear to be kerned incorrectly.	Set up a separate KP Spec for use with small caps fonts.
Added a KP entry to the FV Spec but not seeing any change.	Kerning Pairs Spec does not exist in Font Width library. The FV Spec you edited is not the one that is active (you may have a job-level FV Spec that is active). To access the active FV Spec, go into your division and select *Activity*, *Edit Job Styles*, *Font Variant*.
Added new FAST and assigned it to FV 7 in the FV Spec. Getting FF/FV error.	Same as previous entry.

Troubleshooting Type of Font

If you have trouble with a particular font, validate that the font is likely to be an acceptable PostScript font. To do this, run **xyprfont.pl** from the operating system command line:

```
$XYV_EXECS/bin/xyprfont.pl filename [options]
```

where *filename* is the font file that you want to check. The *xyprfont.pl* program produces a PostScript output file in the current directory. For more information, refer to “Troubleshooting Fonts” on page 1-23.

You can view the output file using a PostScript viewer or convert it to PDF and then view it with a PDF viewer.

XPP provides a PostScript viewer. Execute the following at the command line:

```
XYV_EXECS/gs/gsx filename.ps
```

where *filename.ps* is the output file produced by *xyprfont.pl*.

Note: *If you are running xyprfont.pl in the XYV_EXECS/psres/fonts directory tree, be sure to remove the .ps file when you finish testing. Otherwise, the next time you run makepsres, you will receive a warning about an invalid font file.*

Troubleshooting Output of OpenType/TrueType Fonts

If you are downloading OpenType/TrueType fonts to a printer, the printer must have a PostScript engine with a version number of 2015 or greater. To check the version number of your printer, you can print the file XYV_EXECS/sys/od/psversion.ps to it. This will tell you the PostScript version number of the printer and also if the printer can support Type 42 fonts. The information generated is presented with the following format:

- Product name:
- Product revision:
- PostScript version:
- Support Type 42 fonts?
- Support CMaps?
- Maximum memory:

Mapping Unmapped Open Type Font Glyphs with Psfmtdrv

To map the unmapped glyphs with *psfmtdrv* (this will not work with *divpdf*):

1. Create a workspace area for this activity that is outside of the XYV_EXECS/psres/fonts folder for storing backups and extra files created during this process.

2. Copy the *fontname*.otf or *fontname*.ttf file to your workspace directory.

3. Run `xyprfont.pl` on the font file in your workspace directory. This will create a PostScript file in this directory.

4. Distill the PostScript file created by `xyprfont.pl` either with Adobe Distiller or by executing %XYV_EXECS%\gs\gs.exe (Windows) or $XYV_EXECS/gs/gs (Unix) on the PostScript file.

5. Page through or go to the last pages of the distilled file to see if there are unmapped glyphs in the font. There may be more than one page of unmapped glyphs. Mapped glyphs will display in a grid, while unmapped glyphs will display in a two-column format showing the glyph itself and its name.

6. Determine which and how many unmapped glyphs you need to map manually.

7. Keep the *fontname*.ps and *fontname*.pdf files in your workspace area to avoid errors the next time makpsres or Font Copy is run. Extra files in the XYV_EXECS/psres/fonts directory or its subdirectories will cause error messages when updating the PSres.upr file.

8. Determine which Unicode range (with values xFFFF and lower) is not used in the current .cmap file that can accommodate the number of characters you want to map manually.

Note: *If you want to use Unicode values in the "private use" range (such as xF0001) for use in XPP, there is a slight difference in what you put into the PTS spec in step 15. If you use Unicode values in the "private use" range for these characters in XPP, then you need to edit the PTS spec manually; you will not be able to just modify the .afm file and rerun BuildFAST.*

9. Copy the current cmap file to your workspace area to modify it and save a backup copy of the original cmap file.

10. Edit the cmap file adding a range of Unicode values not already in use in this font.

    In the following example, values <f01d>, <f01e>, and <f01f> are used to map three unmapped glyphs. Edit the last codespacerange section of the cmap file as seen in the following example and add the range in the order shown. Change `3 begincodespacerange` to `4 begincodespacerange` as a fourth range is being added to this section. Then, in order, identify the range you are adding, in this case <f01d> through <f01f>. There can be no more than 100 ranges in each codespacerange section.

    ```
    4 begincodespacerange
    ```

```
<0020> <o3c0>
<2010> <25ca>
<f01d> <f01f>
<fb01> <fb02>
endcodespacerange
```

11. You need to add these characters to the end of the file, in order, in a bfchar section as well. Keep in mind that each bfchar section can contain no more than 100 characters. Go to a section that holds fewer than 100 characters, if there is one, and add the new ones with the codes you have chosen in the correct alpha/numeric order. Otherwise, add a new section.

```
56 beginbfchar
...
<f01d> /f_i
<f01e> /f_i
<f01f> /a.superior
<fb01> /fi
<fb02> /fl
endbfchar
```

12. Save the cmap file and then copy the modified file to the appropriate XYV_EXECS/psres/fonts directory.

Note: *An alternative to steps 12 through 15, if using the same Unicode value in XPP as is used in the modified .cmap file, is to copy a backup of the .afm file into your workspace directory and then edit the .afm file, locate the glyphs being mapped (at the end of the CharMetrics section) and enter the hex Unicode values being used where you see* `UNX 0000`*. Then run BuildFAST on the modified .afm file for each FAST that uses the font.*

13. Open the PTS spec for the font on which you are working.

14. Go to the bottom of the spec, where you will find entries for the unmapped glyphs.

15. There will be `x0` in the *Char Code* and *Unicode Number* fields. For already mapped glyphs, these values equate to the Unicode value in the grid you see after running `xyprfont.pl` and distilling the PostScript file. Notice the glyph widths for the unmapped glyphs are present in the PTS spec so you do not need to add them. These values are in the afm file in the font directory.

16. Using the values you have chosen for the unmapped glyphs, update the PTS spec's *Char Code* and *Unicode Number* fields with the Unicode value you used in the cmap file. Alternatively, if you want to use the "private use" numbers for XPP, put the "private use" number in the

Unicode Number field and the Unicode value from the cmap file in the *Char Code* field. For example, for the `f_i` glyph in the previous example, you could enter `xF01D` in the *Char Code* field and `xF1001` in the *Unicode Number* field of the PTS spec to use that Unicode value in the XPP data.

17. Run GenFAST on the FAST(s) using the modified PTS spec.

Note: *Running Font Copy on the font again will overwrite the modified .cmap file (and possibly the .afm file if it was modified), so it is a good idea to save the original and modified versions of these files in another location. Also, running BuildFAST on the font again will overwrite the manually modified PTS spec (unless the .afm file was modified along with the .cmap file and BuildFAST was used to modify the FASTs).*

Appendix C

Sample Type 1 Font Encoding Tables

This appendix provides samples of some Type 1 font encoding tables:

- Adobe StandardEncoding Table
- XPP Extended Character Set encoding table
- Symbol Character Set encoding table
- Zapf Dingbat Character Set encoding table

Adobe StandardEncoding Table

All Type 1 PostScript fonts have a default encoding table. In Adobe Type 1 Roman text fonts, the Adobe “StandardEncoding” table is the default encoding table.

The following figure displays the Adobe “StandardEncoding” table.

StandardEncoding Encoding Table

octal	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x		!	"	#	$	%	&	’
\05x	(	)	*	+	,	-	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	@	A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	_
\14x	‘	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	
\20x								
\21x								
\22x								
\23x								
\24x		¡	¢	£	⁄	¥	ƒ	§
\25x	¤	'	“	«	‹	›	ﬁ	ﬂ
\26x		–	†	‡	·		¶	•
\27x	‚	„	”	»	…	‰		¿
\30x		`	´	ˆ	˜	¯	˘	˙
\31x	¨		˚	¸		˝	˛	ˇ
\32x	—							
\33x								
\34x		Æ		ª				
\35x	Ł	Ø	Œ	º				
\36x		æ				ı		
\37x	ł	ø	œ	ß				

Figure C-1 *Adobe Standard Encoding Table*

This table includes glyphs in the 128-character ASCII set (a-z, A_Z, 0-9, and common punctuation symbols) and some additional glyphs. Such fonts often contain many more glyphs, such as accented characters and other publishing symbols. You may access these glyphs through the remaining character codes 129-256. The StandardEncoding table does not address all of these additional glyphs. You can see them in a font's .afm file with "−1" as the character code (`C` value), indicating that they are "unencoded."

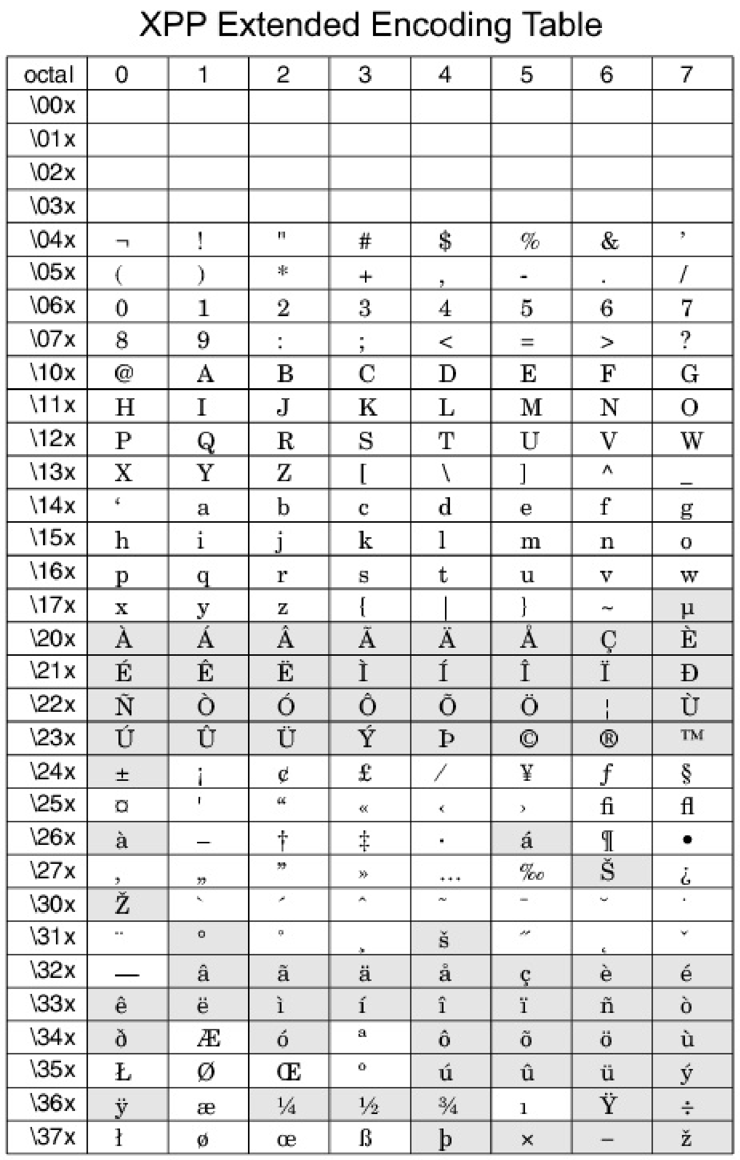
The StandardEncoding table assigns 149 positions. There are 75 positions available for additional glyphs. There are 32 positions reserved for control characters—it is, therefore, recommended that you do not assign them otherwise.

The shaded boxes in this table show the 75 positions that may be filled by extended encodings.

XPP Extended Character Set Encoding Table

XPP used to provide an encoding table called “extended” that extended the Adobe StandardEncoding table. This “extended” encoding table allowed you to encode additional glyphs generally found in Type 1 Roman text fonts up to the 75 available positions not used by StandardEncoding.

The following figure displays the XPP “extended” encoding table:



XPP Extended Encoding Table

octal	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x	¬	!	"	#	$	%	&	’
\05x	(	)	*	+	,	-	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	@	A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	_
\14x	‘	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	µ
\20x	À	Á	Â	Ã	Ä	Å	Ç	È
\21x	É	Ê	Ë	Ì	Í	Î	Ï	Đ
\22x	Ñ	Ò	Ó	Ô	Õ	Ö	¦	Ù
\23x	Ú	Û	Ü	Ý	Þ	©	®	™
\24x	±	¡	¢	£	⁄	¥	ƒ	§
\25x	¤	'	“	«	‹	›	ﬁ	ﬂ
\26x	à	–	†	‡	·	á	¶	•
\27x	‚	„	”	»	…	‰	Š	¿
\30x	Ž	`	´	ˆ	˜	¯	˘	˙
\31x	¨	˚	°	¸	š	˝	˛	ˇ
\32x	—	â	ã	ä	å	ç	è	é
\33x	ê	ë	ì	í	î	ï	ñ	ò
\34x	ð	Æ	ó	ª	ô	õ	ö	ù
\35x	Ł	Ø	Œ	º	ú	û	ü	ý
\36x	ÿ	æ	¼	½	¾	ı	Ÿ	÷
\37x	ł	ø	œ	ß	þ	×	−	ž

Figure C-2 *XPP “extended” Encoding Table*

The shaded boxes show the 75 glyphs that “extended” added to the StandardEncoding table.

In the case of a Standard Type 1 roman text font, if the value in the *tsf_system encoding* field was "none" and you were trying to access a character that was unencoded, the result would be a blank glyph. If you were to use the XPP "extended" character encoding instead, you would have gotten the appropriate character. (In XPP, this assumes that you had set up your PTS Spec appropriately.)

For example, in Helvetica, a lower-case a grave (character code d176) is not mapped in StandardEncoding and is, therefore, "unencoded". If, in XPP, you had set the values as follows:

- encoding = none— you get a blank space.
- encoding = extended—you get the correctly accented character.

Extended PostScript Character Set

The standard Adobe PostScript character set includes approximately 150 characters. The XPP Extended PostScript Character Set included an additional 75 ISO latin1 characters.

The following table displays the XPP Extended Character Set, that used to be delivered, by font position (octal) and PostScript Name:

Table C-1 *XPP Extended Characters by Font Position and Name*

Position (octal)	***Glyph***	**PostScript Name**
\177	µ	mu
\200	À	Agrave
\201	Á	Aacute
\202	Â	Acircumflex
\203	Ã	Atilde
\204	Ä	Adieresis
\205	Å	Aring
\206	Ç	Ccedilla
\207	È	Egrave
\210	É	Eacute
\211	Ê	Ecircumflex
\212	Ë	Edieresis

Table C-1 *XPP Extended Characters by Font Position and Name (Continued)*

Position (octal)	*Glyph*	**PostScript Name**
\213	Ì	Igrave
\214	Í	Iacute
\215	Î	Icircumflex
\216	Ï	Idieresis
\217	Đ	Eth
\220	Ñ	Ntilde
\221	Ò	Ograve
\222	Ó	Oacute
\223	Ô	Ocircumflex
\224	Õ	Otilde
\225	Ö	Odieresis
\226	¦	brokenbar
\227	Ù	Ugrave
\230	Ú	Uacute
\231	Û	Ucircumflex
\232	Ü	Udieresis
\233	Ý	Yacute
\234	Þ	Thorn
\235	©	copyright
\236	®	registered
\237	™	trademark
\240	±	plusminus
\260	à	agrave
\265	á	aacute
\276	Š	Scaron
\300	Ž	Zcaron

Table C-1 *XPP Extended Characters by Font Position and Name (Continued)*

Position (octal)	***Glyph***	**PostScript Name**
\311	°	degree
\314	š	scaron
\321	â	acircumflex
\322	ã	atilde
\323	ä	adieresis
\324	å	aring
\325	ç	ccedilla
\326	è	egrave
\327	é	eacute
\330	ê	ecircumflex
\331	ë	edieresis
\332	ì	igrave
\333	í	iacute
\334	î	icircumflex
\335	ï	idieresis
\336	ñ	ntilde
\337	ò	ograve
\340	ð	eth
\342	ó	oacute
\344	ô	ocircumflex
\345	õ	otilde
\346	ö	odieresis
\347	ù	ugrave
\354	ú	uacute
\355	û	ucircumflex
\356	ü	udieresis

Table C-1 *XPP Extended Characters by Font Position and Name (Continued)*

Position (octal)	***Glyph***	**PostScript Name**
\357	ý	yacute
\360	ÿ	ydieresis
\362	¼	onequarter
\363	½	onehalf
\364	¾	threequarters
\366	Ÿ	Ydieresis
\367	÷	divide
\374	þ	thorn
\375	×	multiply
\376	−	minus
\377	ž	zcaron

Symbol Character Set Encoding Table

A PostScript Type 1 font may contain a set of characters that are not text characters, that is, not alphanumeric characters. This is especially true of *Pi* fonts, which usually contain a set of characters that are only symbols.

When using *Pi* fonts, the encoding table entry in the TSF Spec must be set to “none”, meaning the default internal encoding table of the font is used.

The following figure displays the Type 1 Symbol font Character Set encoding table. Notice how different the font is from the default encoding of a text font.

Symbol Encoding Table

octal	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x		!	∀	#	∃	%	&	∋
\05x	(	)	∗	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	≅	Α	Β	Χ	Δ	Ε	Φ	Γ
\11x	Η	Ι	ϑ	Κ	Λ	Μ	Ν	Ο
\12x	Π	Θ	Ρ	Σ	Τ	Υ	ς	Ω
\13x	Ξ	Ψ	Ζ	[	∴	]	⊥	_
\14x	‾	α	β	χ	δ	ε	ϕ	γ
\15x	η	ι	φ	κ	λ	μ	ν	ο
\16x	π	θ	ρ	σ	τ	υ	ϖ	ω
\17x	ξ	ψ	ζ	{	|	}	∼	
\20x								
\21x								
\22x								
\23x								
\24x		ϒ	′	≤	⁄	∞	ƒ	♣
\25x	♦	♥	♠	↔	←	↑	→	↓
\26x	°	±	″	≥	×	∝	∂	•
\27x	÷	≠	≡	≈	…	⏐	⎯	↵
\30x	ℵ	ℑ	ℜ	℘	⊗	⊕	∅	∩
\31x	∪	⊃	⊇	⊄	⊂	⊆	∈	∉
\32x	∠	∇	®	©	™	∏	√	⋅
\33x	¬	∧	∨	⇔	⇐	⇑	⇒	⇓
\34x	◊	〈	®	©	™	∑	⎛	⎜
\35x	⎝	⎡	⎢	⎣	⎧	⎨	⎩	⎪
\36x		〉	∫	⌠	⎮	⌡	⎞	⎟
\37x	⎠	⎤	⎥	⎦	⎫	⎬	⎭	

Figure C-3 *Symbol (non-text) Encoding Table*

Zapf Dingbat Charater Set Encoding Table

The Zapf Dingbat font Character Set encoding table is another example of a PostScript Type 1 non-text character set. Notice how different these characters are from the text character sets and from the Symbol character set.

Zapf Dingbats Encoding Table

octal	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x		✁	✂	✃	✄	☎	✆	✇
\05x	✈	✉	☛	☞	✌	✍	✎	✏
\06x	✐	✑	✒	✓	✔	✕	✖	✗
\07x	✘	✙	✚	✛	✜	✝	✞	✟
\10x	✠	✡	✢	✣	✤	✥	✦	✧
\11x	★	✩	✪	✫	✬	✭	✮	✯
\12x	✰	✱	✲	✳	✴	✵	✶	✷
\13x	✸	✹	✺	✻	✼	✽	✾	✿
\14x	❀	❁	❂	❃	❄	❅	❆	❇
\15x	❈	❉	❊	❋	●	❍	■	❏
\16x	❐	❑	❒	▲	▼	◆	❖	◗
\17x	❘	❙	❚	❛	❜	❝	❞	
\20x								
\21x								
\22x								
\23x								
\24x		❡	❢	❣	❤	❥	❦	❧
\25x	♣	♦	♥	♠	①	②	③	④
\26x	⑤	⑥	⑦	⑧	⑨	⑩	❶	❷
\27x	❸	❹	❺	❻	❼	❽	❾	❿
\30x	➀	➁	➂	➃	➄	➅	➆	➇
\31x	➈	➉	➊	➋	➌	➍	➎	➏
\32x	➐	➑	➒	➓	➔	→	↔	↕
\33x	➘	➙	➚	➛	➜	➝	➞	➟
\34x	➠	➡	➢	➣	➤	➥	➦	➧
\35x	➨	➩	➪	➫	➬	➭	➮	➯
\36x		➱	➲	➳	➴	➵	➶	➷
\37x	➸	➹	➺	➻	➼	➽	➾	

Figure C-4 *Zapf Dingbat encoding table*

Glossary

Accent/Ligature Replacement spec
A font spec that defines replacing an accent with a different accent, characters with the corresponding ligature, or any character(s) with another character(s). Also referred to as the RP spec.

alternate keyboard spec
A keyboard spec other than the Standard XPP Keyboard spec (keyboard 0). The system supports up to 61 alternate keyboard specs. Groups of related characters appear in the XPP-delivered alternate keyboard specs. You can create an alternate keyboard spec and map the characters you frequently use to the key caps.

ASCII
The American Standard Code for Information Interchange method of representing characters as symbols.

ASCII escape sequence
The unique ASCII character sequence RWS has assigned to each character in the XCS spec.

ASCII to XSF file
A file used during ToXSF for converting ASCII format files to XPP format files. The ASCII to XSF file maps ASCII escape sequences to Unicode numbers.

ascender
The portion of the glyph that extends above the baseline.

ascender height
The distance from the baseline to the top of the ascenders on lowercase letters (or the top of uppercase letters).

baseline
An imaginary line at the bottom of the ascender of each glyph (excluding the descender and any extra lead) in a line of type. The baseline serves as a basis for the horizontal alignment of glyphs and also as a basis for calculating the positions of other text elements.

base font
A font whose glyphs are addressed by PostScript character names, as opposed tocharacter identifiers (CIDs). This term is commonly used to describe OpenType fonts that are not also CID fonts.

bitmap character
A glyph whose image is designed for a particular point size and is therefore not scalable.

Build FAST
An XPP utility that creates basic font specs based on fonts in the XYV_EXECS/psres/fonts directory. This is very helpful when you are loading new fonts.

CID
A character identifier—352—used to access glyphs in a CID font.

CID font
A font whose glyphs are addressed by a CID rather than a PostScript character name. CID fonts generally have hundreds or thousands of glyphs. A CID font may or may not also be an OpenType font. Non-OpenType CID font filenames may not have a filename extension.

CFF
Compact Font Format—the type of data representation used in a "PostScript-flavored" OpenType font. It is not the same as Type 1.

CMap
A type of encoding file that accesses hundreds or thousands of glyphs. A CMap that uses CIDs must be used with a CID font, whether or not it is an OpenType font. A CMap that uses PostScript character names is usually the preferred way to access glyphs in an OpenType font that is not also a CID font.

CSS spec
A Cascading Style Sheet (CSS) spec that defines the typographic style of named text elements (tags), such as doc, h1, section, or figure. Tag names embedded in the text refer the system to style definitions stored in the CSS spec.

decimal notation
A method of representing a numeric value using base 10.

descender
The portion of the glyph that extends below the baseline. For example, the vertical stroke in the lowercase letter q.

destination library
The font libraries containing the machine-readable font specs such as the FASTs and, optionally, the machine-readable Kerning Pair files. The files in the destination libraries can not be edited.

division
The level of the XPP database containing one or more pages of text. Typically, one division corresponds to one document or one logical portion of a document (for example, a chapter).

Division Ticket
A supporting data spec containing information that the system uses to compose and paginate a division. Some of the information in the Division Ticket is in addition to the information in the Job Ticket; some of the information in the Division Ticket is alternative information to that in the Job Ticket.

em
A unit of type measurement, usually equal to the width of an uppercase M, that is exactly as wide as the point size being set.

embold
A characteristic of type where the characters are set in heavier lines of type. For example, **bold**. The system allows three levels of electronic emboldening for screen display and output of characters (if the output device has emboldening capability).

em dash
A dash that is equal in width to an em space.

em space
A fixed amount of blank space, equal in width to an em in the current point size. The space is not altered when the line is justified.

FAST Generation Exception spec
A font spec that defines a glyph(s) to include in a FAST that is either not in the specified PTS/PSF specs or is an exception to the glyphs in the specified PTS/PSF spec. Also referred to as the FGX spec, it must have the same name as the FAST Generation spec.

FAST Generation spec
A font spec that defines the PTS and PSF specs from which to include glyphs and the style codes to include/exclude when building a FAST using the GenFAST program. Also referred to as the FGS spec.

FAST
A Font Access Table— a machine-readable file produced by running the GenFAST (Generate a FAST) program. The FAST contains the glyph widths and glyph access information needed for composition, display, and output.

You cannot edit, view, or print a FAST. The viewable versions of the FAST are the system-generated VFX and VPX specs.

field notation
Representing numeric values within spec fields in decimal, hexadecimal, or octal.

FGS spec
See FAST Generation spec.

FGX spec
See FAST Generation Exception spec.

floating accent
An accent that is centered over the base glyph.

font
A complete assortment of alphanumeric glyphs and special glyphs grouped together according to their unique appearance or style.

Font Access Table
See FAST.

Font Copy
An XPP utility that copies font files into the XYV_EXECS/psres/fonts directory tree and makes them available to XPP by adding their names to the *PSres.upr* file. For OpenType fonts, Font Copy also creates an AFM and a CMap file.

font descriptor file
A file containing a list of the FASTs, by family and variant name and number, that are available in a destination font width library to represent your fonts available with that font library.

Font download table

A font download table is a simple text file, and may be created and/or modified using any ASCII or text editor.

You need font download tables to enable/disable font downloading by font when you generate a PostScript file in XPP (`PS to file` and `PS to PDF file`) or output to a PS device using *psfmtdrv*.

font family

A set of fonts that share certain visual characteristics that include shapes of serifs, relative position of thick and thin strokes, and unique decorative characteristics. For example, NotoSerif, NotoSans, NotoSansMono.

font height

The total distance from the top of a font's highest ascender to the bottom of the font's lowest descender (i.e., the vertical space). Also referred to as point size.

font width library

A library containing the FAST.

font name

The name of the font, for example, NotoSerif-Regular, NotoSans-Medium, NotoSansMono-Regular, and so on.

font specs

The set of specifications defining the fonts, glyph widths, and other information needed to access fonts on an XPP system.

font variant

The name (or associated number) of the variant, for example, bold, medium, italic, and so on.

Font Variant spec

A spec that maps the fonts specified in the tags or by CSS font properties or Font Family (*ff*) XyMacro and Font Variant (*fv*) XyMacro to the FASTs. Using the rules in the Font Variant spec, you can also specify a Kerning Pairs (KP) spec, set up kerning tracks, specify percentages for the height of small caps and ascenders, and specify which rules to use in the Accent/Ligature Replacement (RP) spec. The Font Variant spec is also referred to as the FV spec.

font width

The horizontal measure of an em in the current point size, usually the same as the font height.

FV spec

See Font Variant spec.

GenFAST

The Generate a FAST program. This program creates a FAST by combining information from the PTS and PSF (if any) specs referenced in the specified FGS spec and corresponding FGX spec (if any). GenFAST reads the specs, selects the glyphs, and puts the information in the FAST in order by Unicode number.

GenXCS

The Generate XCS program. This program creates four files — the ASCII to XSF file (two versions — one machine-readable only, the other, viewable), the Xyvision to ASCII file (machine-readable version only), and the XSF to Terminal file (machine-readable version only). The system uses these files when running ToXSF and FromXSF and for displaying characters in the Line Edit window and spec fields.

hexadecimal notation

A method of representing a numeric value using base 16.

italic
A form of printing where the glyphs are slanted. For example, *italic*. *See also* roman.

Item Format spec
A style spec that defines the typographic style of named text items (tags), such as subhead, list, footnote, or caption. Tag names embedded in the text refer the system to style definitions stored in the Item Format spec. The Item Format spec is also referred to as the IF spec.

Job Ticket
A spec that acts as the link between a job and the specs that define its style. The Job Ticket identifies the style specs, fonts, and dictionaries for use in processing a job. It also identifies the graphics libraries for the job.

KB spec
See Keyboard spec.

kern
To reduce the amount of space (kerning) between specified characters. (Reverse kern is to add space.)

kerning
The adjustment of white space between characters. The system can apply pair kerning and/or track kerning. *See also* pair kerning, track kerning.

kerning pair
A pair of characters between which you modify the amount of white space.

Kerning Pairs spec
A spec defining pairs of characters (kerning pairs) and the amount of space to add or remove between them. Also referred to as the KP spec. When edited and stored, a post processor creates a machine-readable kerning pairs file and places it in the K*fontlib* directory.

Keyboard spec
A spec mapping keystrokes to Unicode numbers (so you can enter characters in a division) and character strings. A character string can consist of tags and XyMacros as well as characters. Also referred to as the KB spec.

K*library*
The destination library for the machine-readable versions of the Kerning Pairs specs. If the KP specs are in the font library, the *library* name is the same as the source font library name. If the KP specs are in a separate library, the *library* name is the same as the name of the source library. This name must be specified in the Pair Kerning Library field of the Job Ticket.

KP spec
See Kerning Pairs spec.

ligature
A combination of characters, usually with less space between them, that the system treats as one character. For example, *ff*.

L*library*
The source library for the font specs and, optionally, the Kerning Pairs specs.

machine-readable
The versions of the FAST and KP files that the system actually reads and processes. You can not directly view or edit machine-readable files.

macro
A command embedded within text that makes local exceptions or changes to the typographic style of a text item or component in a division. A macro consists of a begin character, a mnemonic, and an end character. It may also contain one or more arguments. Macros include XyMacros (a set of macros provided by XPP) and user-defined macros. *See also* XyMacro. In XML/SGML, macros embedded in content take the form of processing instructions.

main font library
Used with numbered font libraries, the main library wass usually named for the output device that used the fonts. The main library contained specs such as the PTS and FGS specs for *Pi* fonts. Typically only main libraries are used.

mean line
A horizontal line placed at the height of a lowercase x.

numbered font library
One of a group of font libraries with related names used when one output device had more than 250 fonts. Numbered libraries consisted of a main library named for the output device that used the fonts and additional font libraries named by appending a number to the end of the name of the main library. The main library usually contained specs such as the PTS and FGS specs for *Pi* fonts. The additional libraries usually contained specs such as the PTS and FGS specs for text fonts. Typically numbered libraries are no longer used anymore.

octal notation
A method of representing a numeric value using base 8.

OpenType font
A type of font designed to be platform-independent and contain hundreds or thousands of glyphs. OpenType fonts may contain PostScript (CFF) or TrueType font data, both of which are supported by XPP. An OpenType PostScript font may be a base font or a CID font. OpenType font filenames have a *.otf* or a *.ttf* filename extension.

pair kerning
Adjusting the amount of space between specified pairs of characters. The character pairs are specified in the KP spec.

path prefix
The path to a directory under which a collection of data resides.

Phototypesetter spec
A spec that maps each vendor-assigned character code to a Unicode number and provides information on the glyph width, style, and so on. Also referred to as the PTS spec.

***Pi* character**
Glyphs that are not among the alphanumeric set, such as boxes, arrows, triangles, math characters, and so on.

pica
A linear measurement of type. There are 12 points to a pica, 6 picas to an inch.

point
A unit of measurement used to specify type size. One point equals approximately .0138 inch, or 1⁄12 of a pica.

point size
The total vertical space measuring from the top of a font’s highest ascender to the bottom of the font’s lowest descender. Also referred to as font height.

PostScript Name spec
A spec for mapping PostScript character names to Unicode numbers. Most character names are mapped in the _psn_ps2xcs.sde spec, any custom character names can be mapped in _psn_custom.sde or _psn_unicode.sde.

Pseudofont spec
A spec for defining glyphs whose output characteristics you have modified. For example, in large point sizes, the open parenthesis looks too wide. In the Pseudofont spec, you can specify the open parenthesis and reduce its width. Also referred to as the PSF spec.

PSF spec
See Pseudofont spec.

PSN spec
See PostScript Name spec.

PTS code
The character access code preceded by a letter denoting the notation (d for decimal, x for hexadecimal, o for octal). Use the vendor-provided character access code as the entry for the *PTS Char Code* field.

PTS spec
See Phototypesetter spec.

resolution
The number of dots per inch used to depict characters onscreen or on a particular output device.

roman
A form of printing where the glyphs are upright. *See also* italic.

RP spec
See Accent/Ligature Replacement spec.

sans serif
Not having the short strokes added to the basic strokes needed to represent characters. *See also* sans serif typeface, serif, serif typeface.

sans serif typeface
A form of printing in which the glyphs do not have serifs added to the basic strokes needed to represent characters. For example, sans serif. *See also* serif, serif typeface, stroke.

serif
A short stroke added to the basic strokes needed to represent a character. Glyphs with serifs are referred to as being in a serif typeface. For example, A represents the capital letter a using the necessary basic strokes. With the serifs, the letter appears as A. *See also* sans serif, sans serif typeface, serif typeface, stroke.

serif typeface
A form of printing in which the glyphs have serifs added to the basic strokes needed to represent the character. For example, **serif**. *See also* stroke, sans serif, sans serif typeface, serif.

slant
Electronically slanting glyphs to give them an italic appearance. You can slant roman glyphs; you can also give italic glyphs more slant. *See also* italic and roman.

small caps
A font in which all the lowercase alpha glyphs are capital letters but are the same size as the x-height.

source library
The font library (or numbered libraries) containing the source font specs used to generate the FASTs and machine-readable Kerning Pair files. The specs in the source library are editable.

stroke
The basic lines needed to represent a character. *See also* serif.

style code
An alphabetic symbol XPP has assigned to denote the weight and style of glyphs. For example, the style code *r* means roman, the style code *h* means heavy.

style spec
The set of specifications describing the overall appearance of the divisions within a job. Style specs determine font types, font sizes, page layouts, spacing, and all other physical attributes of the finished document.

Style specs include the Item Format (IF) spec, the Page Column Override (PC) spec, the Page Layout (PL) spec, and so on. The font specs (the Phototypesetter [PTS] spec, the Pseudofont spec, and so on) are a complement of the style specs.

track kerning
Adjusting the amount of space between all glyphs. You specify track kerning in the Font Variant spec.

TrueType font
See OpenType font.

TSF spec
See Typesetter Font Map spec.

Type 1 font
The oldest type of PostScript font, now deprecated in XPP. Glyphs are addressed in a Type 1 font via an old-style encoding. This cannot address more than 256 different characters.

Type42 font
XPP converts a TrueType font into a Type42 font for the PostScript workflow, since PostScript does not natively support TrueType fonts. When a TrueType font contains no glyph names, XPP creates a permanent Type42 font (*.t42*) file in the XYV_EXECS/psres area, along with a special AFM file that will work with such a font, that are needed for the PostScript workflow. XPP uses the *ttftotype42* utility to convert TrueType fonts into Type42 fonts.

Typesetter Font Map spec
This spec, also referred to as the TSF spec, maps a FAST number to a PostScript font name and encoding. XPP uses the information in the TSF spec when viewing or outputting text.

variant
The weight (light, medium, bold, and so on) and attributes (roman, italic, small caps, and so on) of a particular font family. For example, regular and italic are variants of NotoSerif.

VFX spec
The viewable version of a FAST. The VFX spec contains a rule for each non-pseudofont glyph; the rules are in order by Unicode number. *See also* VPX spec.

VPX spec
A viewable file containing information on pseudofont (customized) glyphs from a FAST. *See also* VFX spec.

vuem codes
Numerical codes that XPP has assigned to special characters that are required on the XPP system, such as for tags, pgrafs, and macros.

XCS number
A unique number that XPP has assigned to some characters. The Xyvision Character Set spec contains entries for more than 4,000 characters to define ASCII sequences, and optionally Unicode values and named character entities for those characters.

XCS spec
See Xyvision Character Set spec.

x-height
The distance from the baseline to the mean line. This is the height of a lowercase x.

XSF
The electronic format in which XPP document pages are stored.

XSF to ASCII file
A file used during FromXSF for converting XSF files to ASCII format files. The Xyvision to ASCII file maps Unicode numbers to ASCII escape sequences. When processing the XSF file, the system reads the file and replaces the Unicode number with the corresponding ASCII sequence.

XSF to Terminal file
A file containing information such as the Unicode number and the contents of the *Ascii* field from the XCS spec for each character. The system cannot represent all characters in the Line Edit Window or in spec fields. In these cases, the system uses the information in the XSF to Terminal file. The system searches the file and displays the ASCII string that corresponds to the Unicode number.

XyMacro
Any macro designed and documented for XPP and furnished with the XPP system. XyMacros perform a variety of tasks, such as quadding a line or drawing a box. You can use XyMacros to change or override typographic parameters defined in the style specs. A XyMacro *must* contain a begin character, a mnemonic, and an end character. It *may* also contain one or more arguments. XyMacros provide a flexible way to meet specific typographical changes, and to produce highly individualized single pages or exception pages. *See also* macro.

Xyvision Character Set
A standard set of more than 4,000 characters. The XCS spec maps each character to an XCS number and to an ASCII string, and optionally to Unicode values and named character entities.

Xyvision Character Set spec
A spec containing definitions for more than 4,000 characters. Each character has a unique Xyvision Character Set (XCS) number. For each XCS number, the spec assigns a name, a unique ASCII character sequence, a character for the system to use when it cannot display the actual character (e.g., in the Line Edit window and in spec fields), and optionally Unicode values and named character entities. Also referred to as the XCS spec.

Xyvision Standard Format
A system of using numbers to represent symbols and characters. These numbers are Xyvision Character Set (XCS) numbers. *See also* XCS number.

Index

RWS Group
201 Edgewater Drive
Wakefield, MA 01880-6216

www.rws.com