User's Manual

GUIX Studio™

User's Manual: Software

Renesas Synergy™ Platform

synergygallery.renesas.com

Renesas Electronics
www.renesas.com

Rev. 5

User Guide

V5 for Windows

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Part Number: 000-1025
Revision 5.2.6

Contents

About This Guide

This guide provides comprehensive information about GUIX Studio, the Microsoft Windows-based rapid UI development environment specifically designed for the GUIX runtime library from Express Logic.

It is intended for the embedded real-time software developer using the ThreadX Real-Time Operating System (RTOS) and the GUIX UI run-time library. The developer should be familiar with standard ThreadX and GUIX concepts.

Organization

Chapter 1	Provides a basic overview of GUIX Studio and its relationship to real-time development.
Chapter 2	Gives the basic steps to install and use GUIX Studio to analyze your application right out of the box.
Chapter 3	Describes the main features of GUIX Studio.
Chapter 4	Describes how to use GUIX Studio to create and manage your application resources.
Chapter 5	Describes how to use the GUIX WYSIWYG screen designer.
Chapter 6	Describes how your application will utilize the output files and API functions generated by GUIX Studio.
Chapter 7	Describes a simple but complete UI application created by GUIX Studio.

Guide Conventions

Italics	typeface denotes book titles, emphasizes important words, and indicates variables.
Boldface	typeface denotes file names, key words, and further emphasizes important words and variables.
	indicates especially useful information.

Customer Support Center

Support engineers 858.613.6640
Support fax 858.521.4259
Support e-mail support@expresslogic.com
Web page http://www.expresslogic.ccom

Latest Product Information

Visit the Express Logic web site and select the "***Support***" menu option to find the latest online support information, including information about the latest GUIX Studio product releases.

What We Need From You

Please supply us with the following information in an e-mail message so we can more efficiently resolve your support request:

- A detailed description of the problem, including frequency of occurrence and how it can be reliably reproduced.
- Attach the trace file that causes the problem.
- The version of GUIX Studio that you are using (shown in the upper left of the display).
- The version of GUIX that you are using including the ***_gx_version_id***string and ***_gx_build_options*** variable.
- The version of ThreadX that you are using including the ***_tx_version_id***string.

Where to Send Comments About This Guide

The staff at Express Logic is always striving to provide you with better products. To help us achieve this goal, e-mail any comments and suggestions to the CustomerSupportCenter at:

support@expresslogic.com

Enter "***GUIX Studio User Guide***" in the subject line.

Chapter 1

Introduction to GUIX Studio

GUIX Studio is a Microsoft Windows-based rapid UI development environment specifically designed for the GUIX runtime library from Express Logic.

Embedded UI Developers can utilize the GUIX Studio WYSIWYG screen designer to quickly create and update their embedded UI using the GUIX run-time environment. GUIX Studio designs are saved and maintained in a GUIX Studio project file, which has the extension .gxp. When your design is ready for execution on the target, GUIX Studio generates C code that contains all the necessary UI information and code.

GUIX Studio Requirements

In order to function properly, Express Logic's GUIX Studio requires *Windows XP* (or above). The system should have a minimum of 200MB of RAM, 2GB of available hard-disk space, and a minimum display of 1024x768 with 256 colors. In addition, the embedded application must be running on *ThreadX/GUIX V5.0* or later.

To build and run the embedded application as a stand-alone Microsoft Windows executable, you will also need a compiler or build environment capable of compiling C source code to produce a Microsoft Windows executable. The evaluation package included with GUIX Studio also includes MSVC 2005 and MSVC 2010 project files and solutions. If you are using a different compiler, you will need to create your own project files or make files for the purposes of building your example applications.

GUIX Studio Constraints

The GUIX Studio UI design tool has several constraints, as follows:

A maximum of 4 displays per project.
A maximum of 100,000 widgets per GUIX Studio project.
A maximum of 100,000 distinct resources, e.g., colors, fonts, pixelmaps, strings, etc.

Chapter 2

Installation and Use of GUIX Studio

This chapter contains a description of various issues related to installation, setup, and usage of the GUIX Studio UI system design tool.

Product Distribution

GUIX Studio is shipped on a single CD-ROM compatible disk. The package includes an installation program ***Setup.exe*** that automatically runs from the CD. If the GUIX Studio installer does not automatically run, please double-click on the ***Setup.exe*** program in order to install GUIX Studio. The GUIX Studio package also contains an example directory of pre-built traces that should serve as a good starting point for new GUIX Studio users.

The release notes associated with each new GUIX Studio release can be found in the file ***readme_guix_studio.txt***. Please review this file to see what has changed between successive GUIX Studio releases.

GUIX Studio Installation Directory

By default, GUIX Studio is installed in the directory ***c:\Express_Logic\GUIX_Studio_v***, where "***v***" is the version of GUIX Studio being installed. The default location for GUIX Studio installation may be changed via the installation dialog, as shown in the next section.

Note that if you have an error during the GUIX Studio installation process, please try selecting **Setup.exe**, right-click and select "**Run as administrator**".

GUIX Studio Installation

GUIX Studio is easily installed, as shown in **Figure 2.1** through **Figure 2.8**. The installation dialogs are fairly straightforward, but it is worth noting that **Figure 2.4** shows the dialog for changing the default installation directory for GUIX Studio.

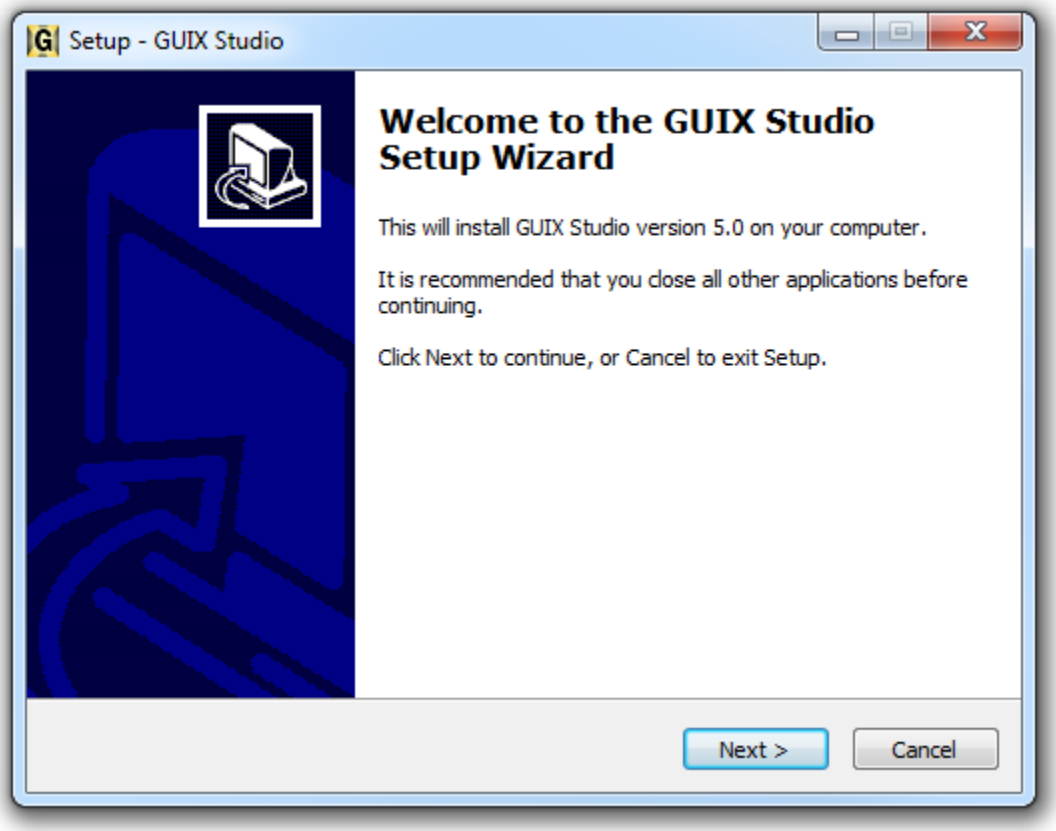


Figure 2.1

Selecting "***Next***" button launches the GUIX Studio installation, as shown in the next figure.

Figure 2.2

Selecting "***Next***" button indicates the terms of the license agreement are agreed and GUIX Studio installation continues, as shown in the next figure.

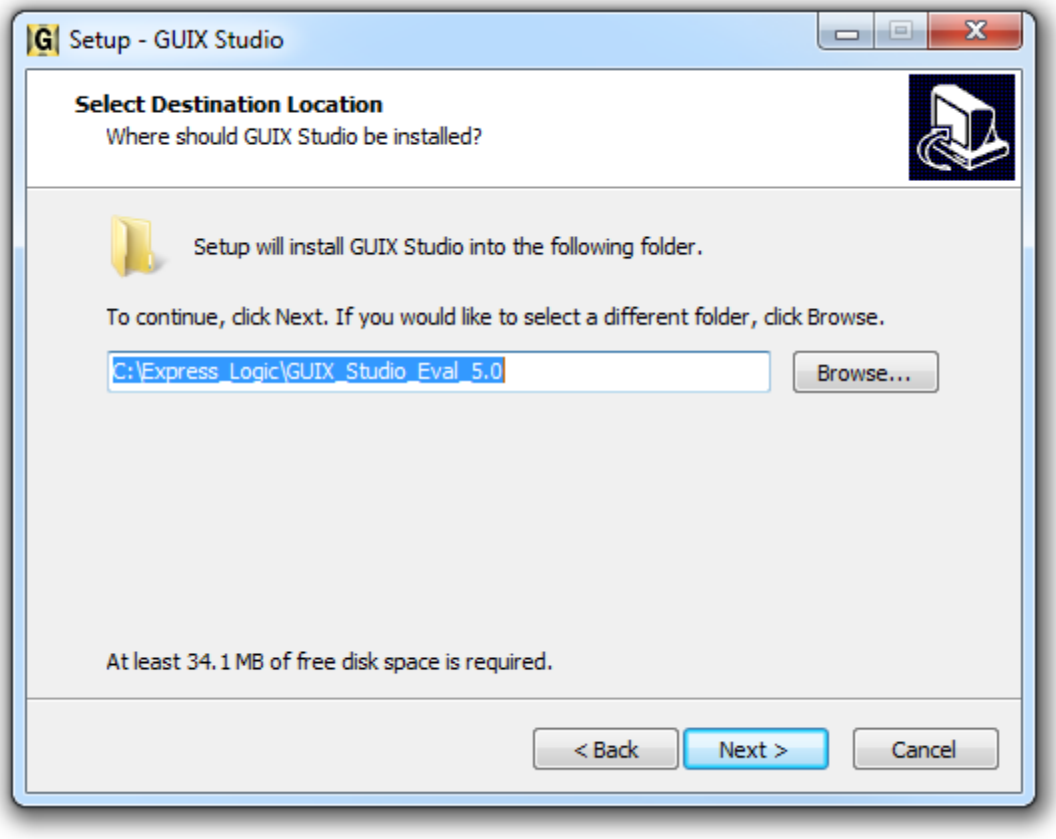


Figure 2.3

If the default installation path is okay, simply select the "***Next***" button to continue the installation, as shown in the next figure.

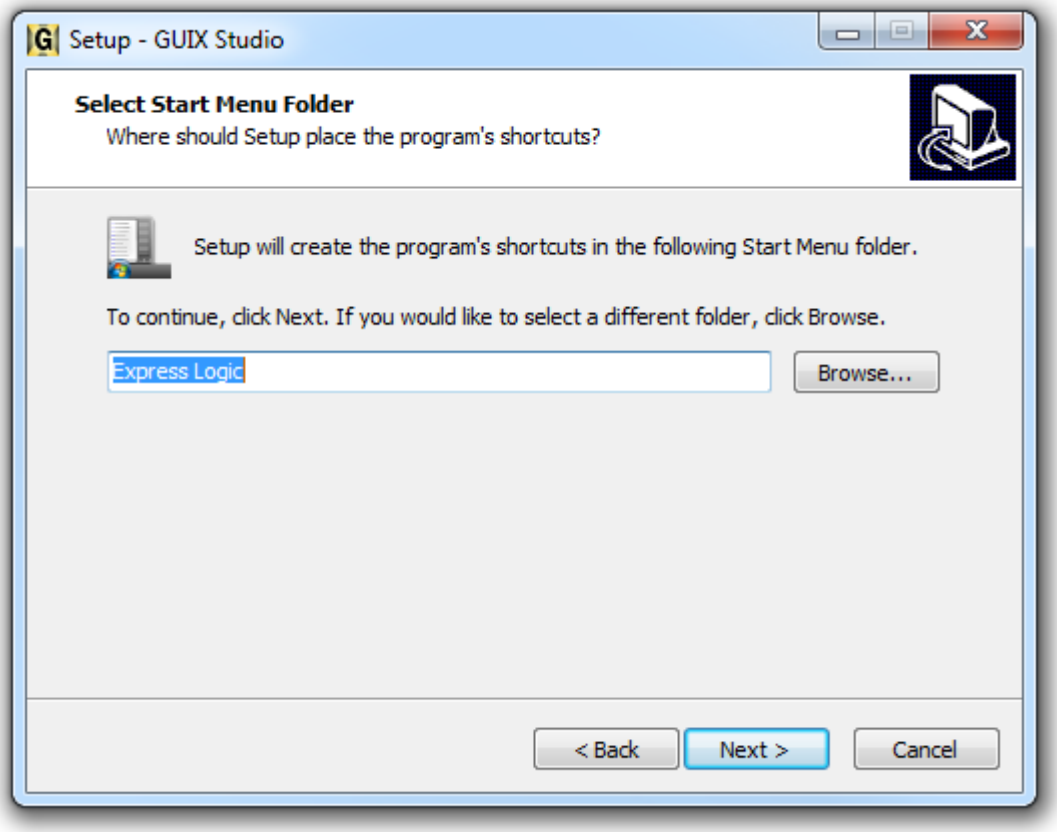


Figure 2.4

If everything is acceptable, simply select the "***Next***" button to continue the installation, as shown in the next figure.

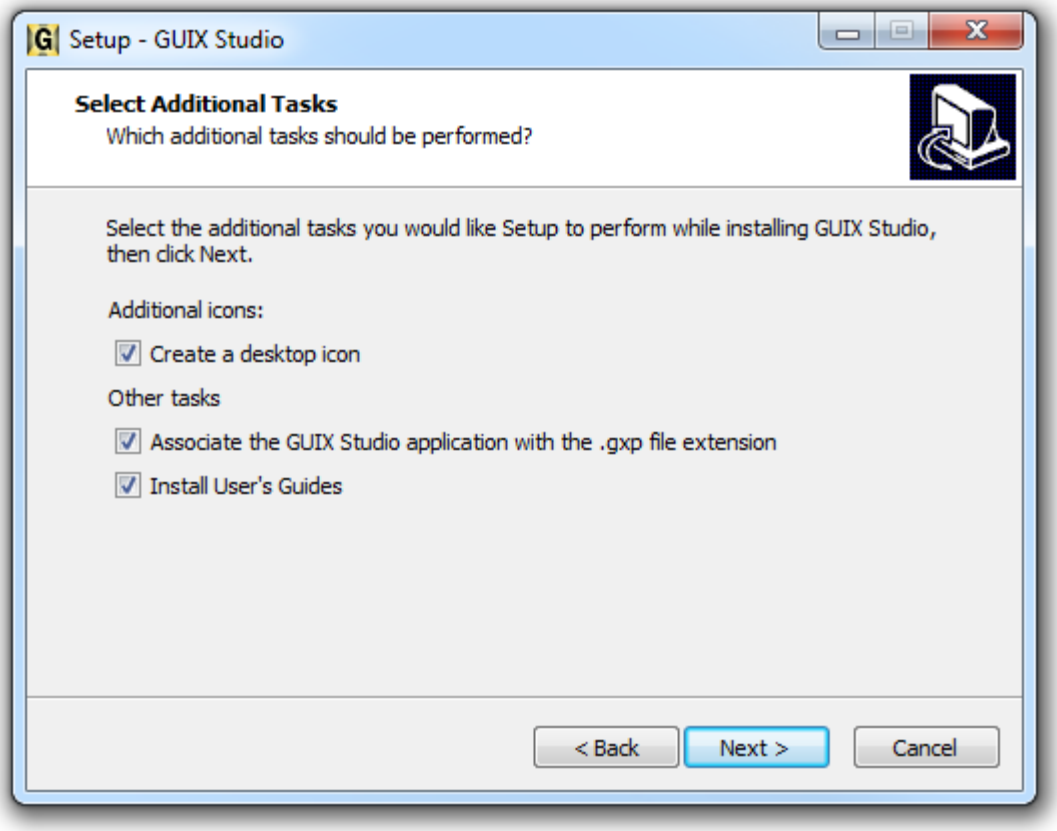


Figure 2.5

If everything is acceptable, simply select the "***Next***" button to continue the installation, as shown in the next figure.

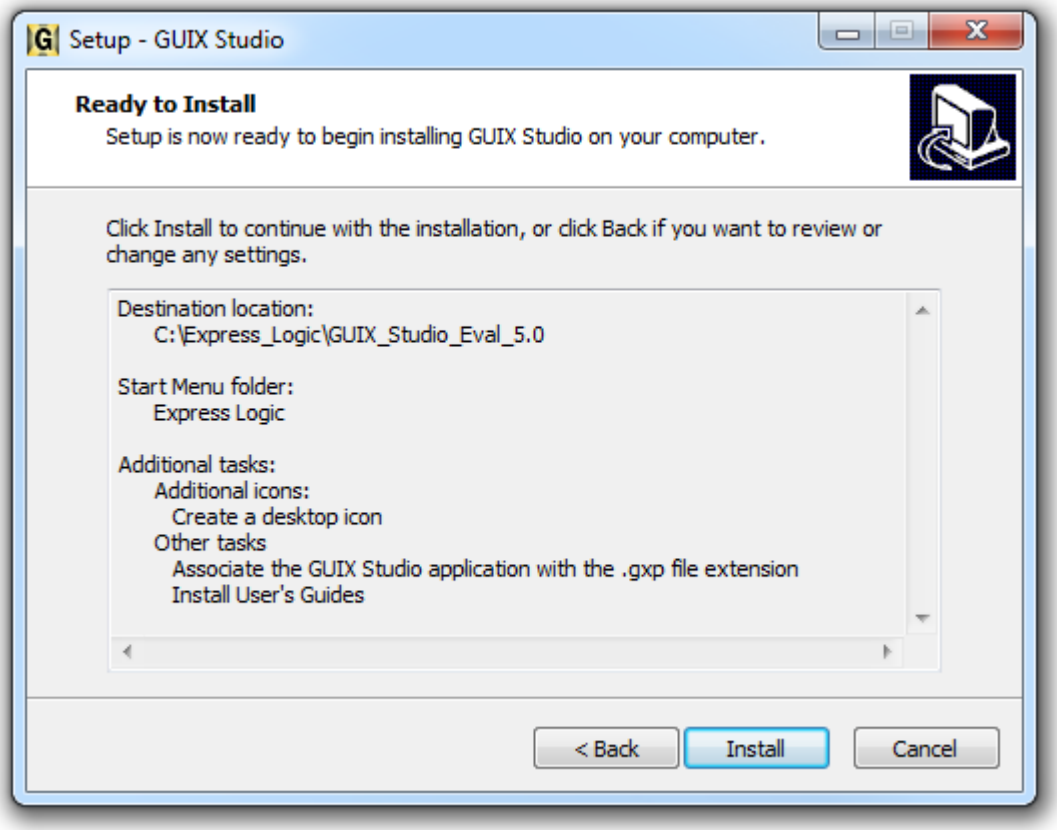


Figure 2.6

If everything is acceptable, simply select the "***Install***" button to continue the installation, as shown in the next figure.

You should now observe the installation of GUIX Studio on your Windows computer.

Figure 2.7

Selecting "***Finish***" button completes the installation and by default brings up the ***GUIX Studio Quickstart Guide***. At this point, GUIX Studio is installed and ready to use!

Using GUIX Studio

Using GUIX Studio is easy - simply run GUIX Studio via the "***Start***" button. At this point you will observe the GUIX Studio UI. You are now ready to use GUIX Studio to graphically create your embedded UI. From here you create a new project or open an existing project, including the GUIX example projects.

Note that you can also double-click on any GUIX Studio project file with an extension of "**gxp,**" which will automatically launch GUIX Studio and open the referenced project.

GUIX Studio Examples

A series of example GUIX Studio project files with the extension "***gxp***" are found in the "***Examples***" sub-directory of your installation. These pre-built example projects will help you get comfortable with using GUIX Studio.

One example project file that is always present is the file ***simple.gxp***. This example project file shows the definition of a simple GUIX UI, as described in ***Chapter 7*** of this document.

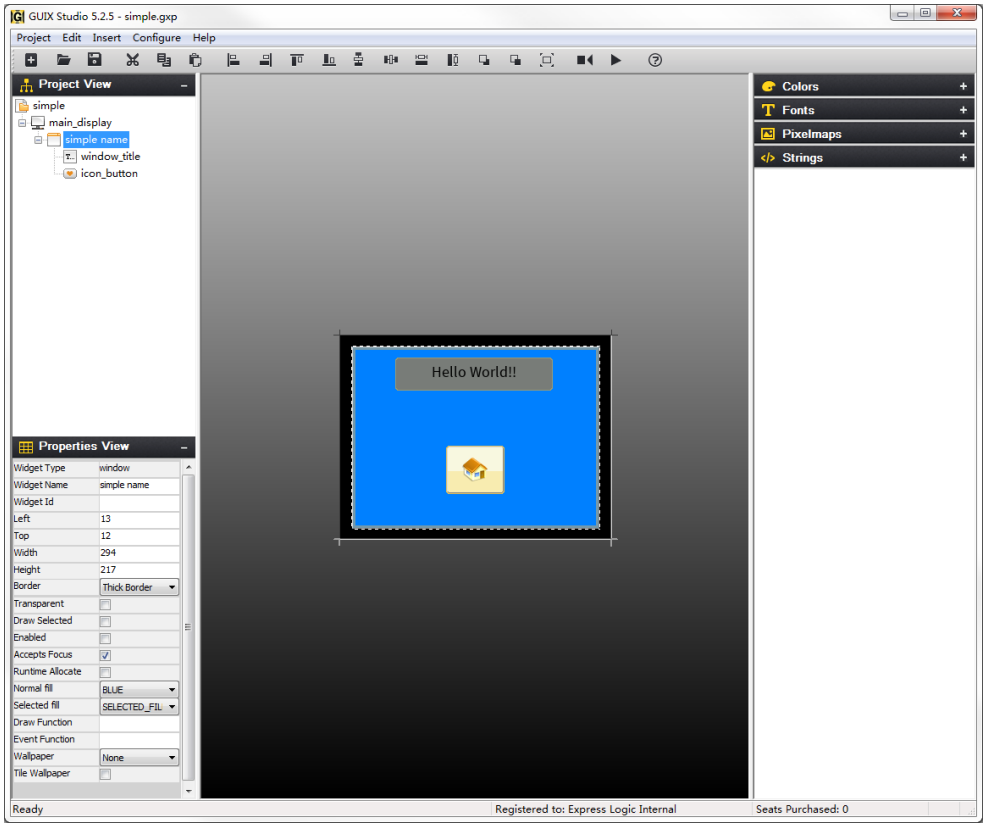


Figure 2.8

Chapter 3

Description of GUIX Studio

This chapter contains a description of the GUIX Studio system analysis tool. A description of the overall functionality of the GUI is found in this chapter.

GUIX Studio Views

There are five principal areas of the GUIX Studio UI, namely the ***Toolbar***, ***Project View***, ***Properties View***, ***Target View***, and ***Resource View***. ***Figure 3.1*** shows the basic GUIX Studio UI. Each of the views is further discussed in the following sub-sections.

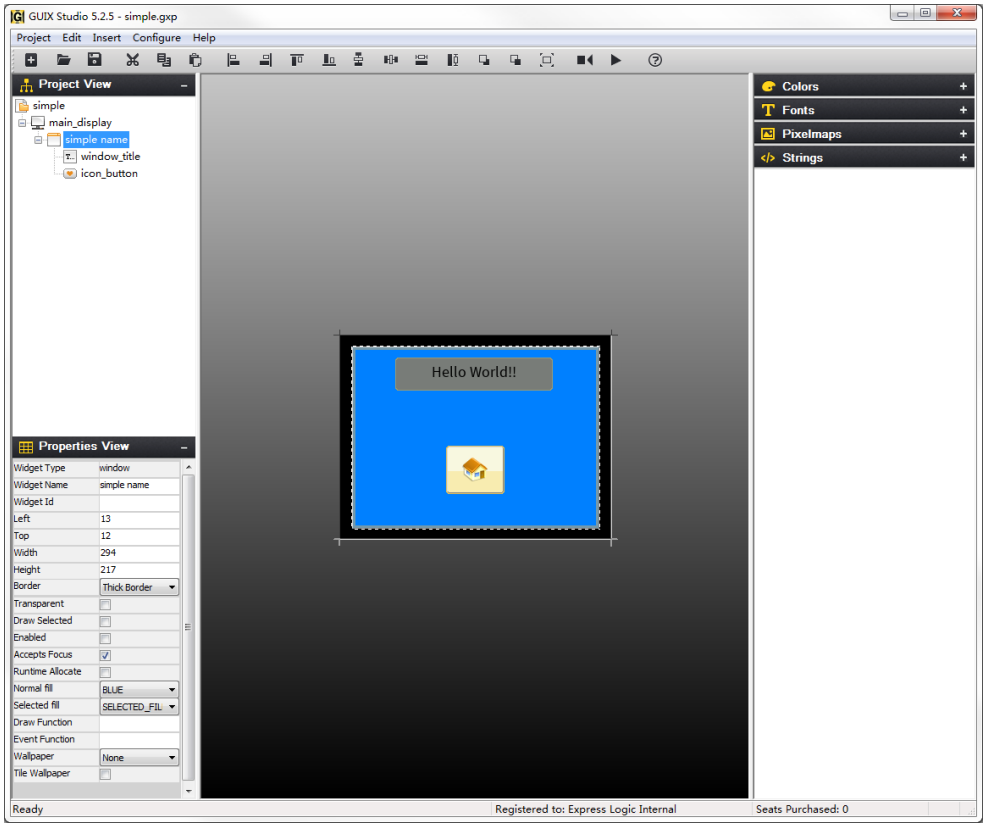


Figure 3.1

Title

The ***Title*** displays the GUIX Studio version as well as the currently open project, as shown at the top of ***Figure 3.1*** previously.

Toolbar

The ***Toolbar*** shows the buttons available to the GUIX Studio developer, as shown in ***Figure 3.2***.

Figure 3.2

The toolbar buttons are defined as follows:


Creates a new GUIX Studio project


Opens an existing GUIX Studio project


Saves the project


Cut widget selected, including children


Copy selected widget, including children


Paste widget and children


Left-align selected widgets


Right-align selected widgets


Top-align selected widgets


Bottom-align selected widgets


Equally space selected widgets vertically


Equally space selected widgets horizontally


Make selected widgets equal width


Make selected widgets equal height


Move selected widgets to front


Move selected widgets to back


Record Macro


Playback Macro


About GUIX Studio

Project View

The ***Project View*** shows the hierarchical list GUIX objects that comprise the embedded UI. New GUIX objects can be added by clicking on the parent object and then selecting an object from the ***Insert*** menu (or by right-clicking on the object and selecting from the right-click menu). ***Figure 3.3*** below shows the GUIX Studio ***Project View***.

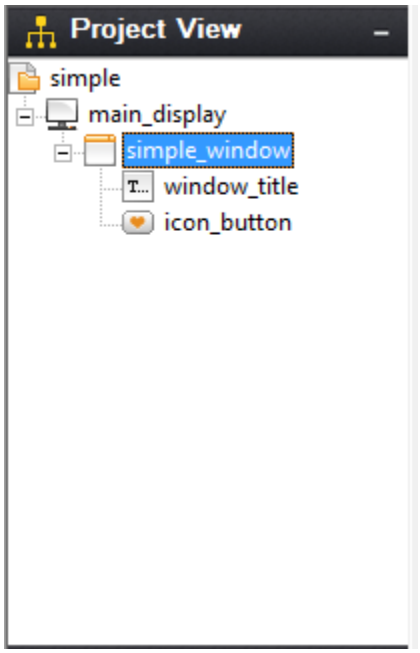


Figure 3.3

Properties View

The ***Properties View*** shows detailed property information of the currently selected GUIX object, which can be selected via the ***Project View*** or by clicking directly on the object in the ***Target View***. ***Figure 3.4*** below shows the GUIX Studio ***Properties View***.

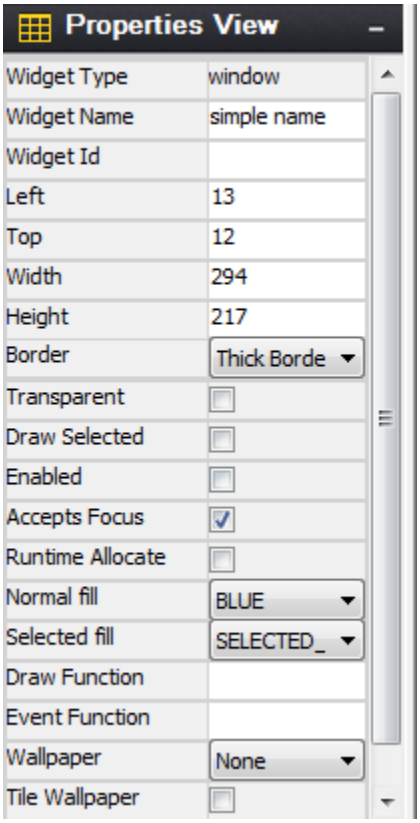


Figure 3.4

Target View

The ***Target View*** is the WYSIWYG screen design and layout area. This view is meant to represent the physical display or displays available on your target hardware. Objects can be selected, moved, resized, etc. via simple mouse operations. In addition, alignment and Z-order button operations are available on selected objects in the Target View. Selecting an object in the ***Target View*** will also result in the properties for that object to be displayed in the ***Properties View***. ***Figure 3.5*** below shows the GUIX Studio ***Target View***.

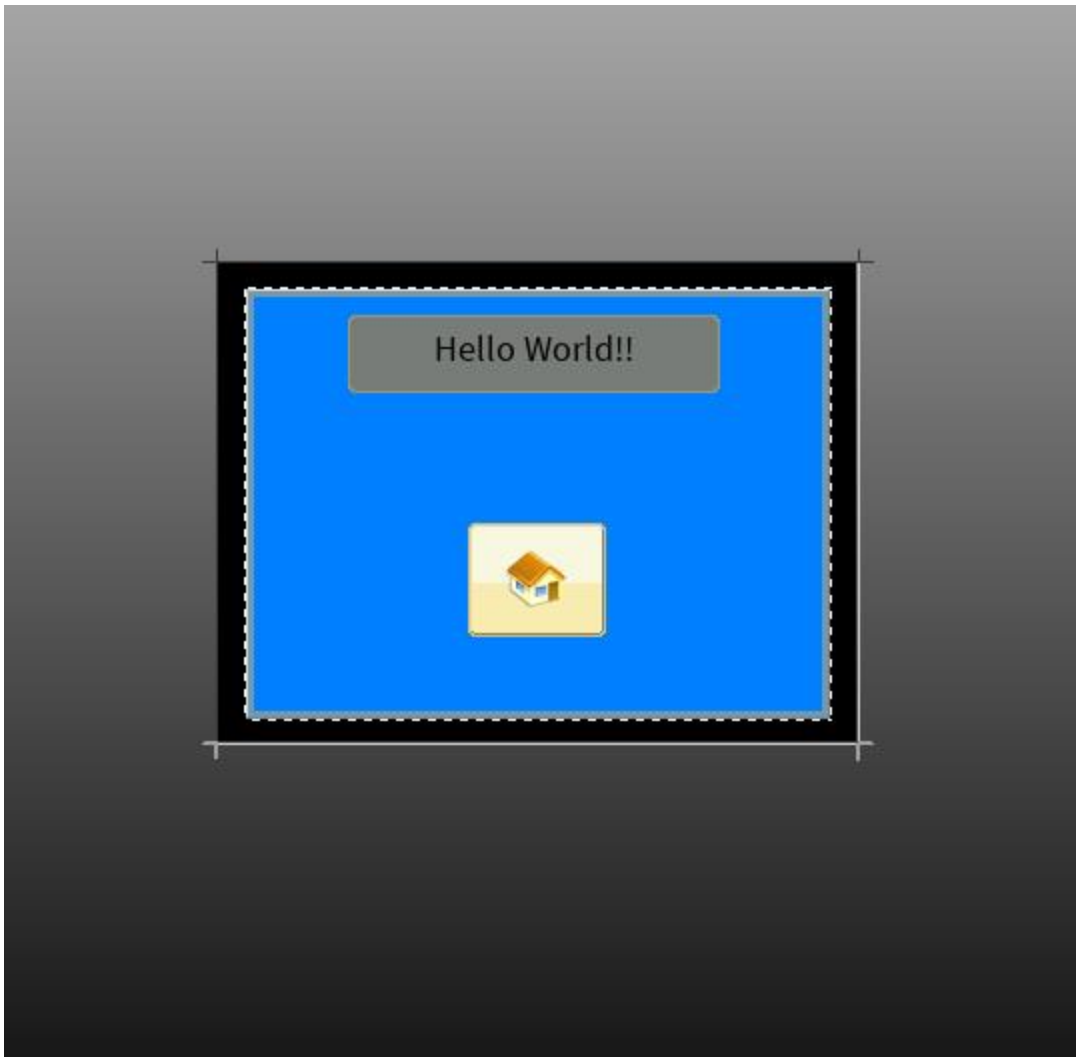


Figure 3.5

Resource View

The ***Resource View*** is used to manage the resources (colors, fonts, pixelmaps, and strings) available to applications screens defined for each display. You can click on the resource view group headers to expand each group and examine the group contents. ***Figure 3.6*** below shows the GUIX Studio ***Resource View***.

Figure 3.6

Each resource group in the view above can be expanded or collapsed by clicking on the group header. A more detailed description of each resource groups follows in the next chapter.

The GUIX Studio Project

A GUIX Studio project maintains information about your UI screen design and UI resources. The project data is saved to an XML format file with the extension ".***gxp***". Since the project file is an XML schema file, it can be versioned controlled and shared similar to any other source file.

When you first start using GUIX Studio, you will need to either open one of the example projects provided with the distribution or create a new project. All of your work is saved to the project data file.

GUIX Studio also produces ANSI C source files. These source files contain either your application resources or data structures describing your designed screens. GUIX Studio also writes to these generated source files API functions that know to utilize the generated data structures to dynamically create your application screens. Your application software will simply invoke the provided API functions to create the screens you have designed within GUIX Studio.

As you progress in designing your user interface, you will periodically want to use GUIX Studio to generate the GUIX compatible output files that will allow you to build and run the interface you have designed. You can

compile and run the generated source files for either your target hardware or on your Windows desktop that simulates ThreadX and GUIX.

GUIX Studio Project Organization

It is helpful to have some knowledge of the basic organization of a GUIX Studio project to understand how to use GUIX Studio effectively and to understand the information presented in the Project View of the GUIX Studio IDE. The Project View is a summary visual representation of all of the information contained in your project.

Before describing the project, it is necessary to define few terms. First, we use the term **Display** to mean a physical display device. This is most often an LCD display device but it could be using other technology. The next term is **Screen**, which mean a top-level GUIX object, usually a GUIX Window, and all of its associated child elements. A Screen is a software construct that can be defined and modified at runtime. Finally, a **Theme** is a collection of resources. A theme includes a table of color definitions, font definitions, and pixelmap definitions that are designed to work well together and present your end user with a consistent look and feel.

The project first includes a set of global information such as the project name, number of displays supported, the resolution and color format of each display, the number of languages supported, the name of each supported language. The project name is the first node displayed in the Project View.

The project next organizes all of the information required for up to 4 physical displays and the screens and resources available to each display. The display names are the next level nodes in the Project View tree.

A unique feature of the GUIX Studio application is built-in support for multiple physical displays, each with its own x,y resolution, color format, screens, and resources. While the vast majority of GUIX applications utilize only one physical display, this capability is important for those making a product that must support multiple simultaneous physical displays.

Beneath each display definition are the top-level windows or screens defined for that display. The screen definitions can be nested to any level depending on the number and nesting of child widgets on each screen. This screen and child widget organization is displayed in a graphical manner in the Project View.

Also associated with each display are the Themes supported by the display and the resource content composing each Theme. If your project includes multiple displays, you will notice that the Resource View changes its content when you select one display and then another. This is because the resource content is linked to each display. Not only the color format may be different, but the pixelmaps, colors, and fonts you choose to use may vary from one physical display to another.

The final component maintained by the project is the string table data associated with each display. Since displays can be of very different x,y resolutions, the string data is maintained independently for each display defined in the project.

Chapter 4

GUIX Studio Resources

GUIX Studio provides management of all UI resources the application will use for colors, fonts, pixelmaps and strings. The sections that follow describe how to add, modify, and delete resources within your UI screen design.

All resource management is done within the ***Resource View*** of the GUIX Studio UI, as shown below in ***Figure 4.1***.

Figure 4.1

Color Resources

In order to manage color resources the ***Colors*** section of the ***Resource View*** must first be expanded by clicking on the **+** field, resulting in the dialog shown below in ***Figure 4.2***:

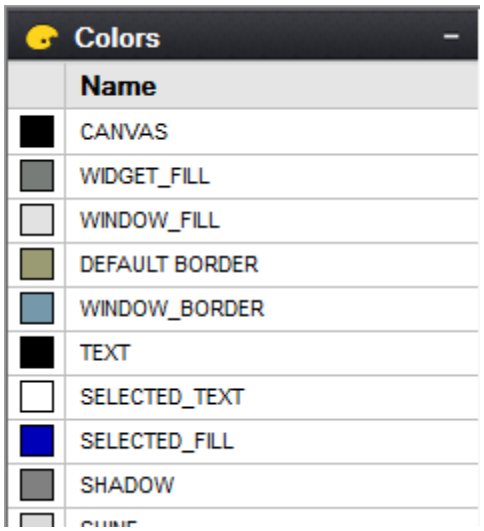


Figure 4.2

Color resources consist of one or more colors, each with a unique logical name. For example, in **Figure 4.2** the logical name **CANVAS,** which is the system color ID for the screen background fill color, is associated with the physical color black. This color resource is used whenever the application specifies **GX_COLOR_ID_CANVAS** as the color in the object properties.

The color “swatch” indicating the color RGB value is shown on the left, followed by the color ID name. You can change the RGB value associated with any ID name at any time. You cannot change the pre-defined system color ID names because these are used internally by the GUIX library. You can however change any of the color values. Changing a system color value is a “global change”, meaning that any widget that does not have a specific color assignment will take on the new system color value.

You can change both the color name and color value for custom colors that you have added to the Theme.

Modifying a color resource is easy, simply double-click (or right-click and menu select) on the color resource. This brings up the color-definition dialog. From this dialog the color resource can be modified to match the application’s UI needs. ***Figure 4.3*** shows the modification dialog when **CANVAS** is double-clicked. Note that the appearance of this dialog will change based on the color-format settings of the target display.

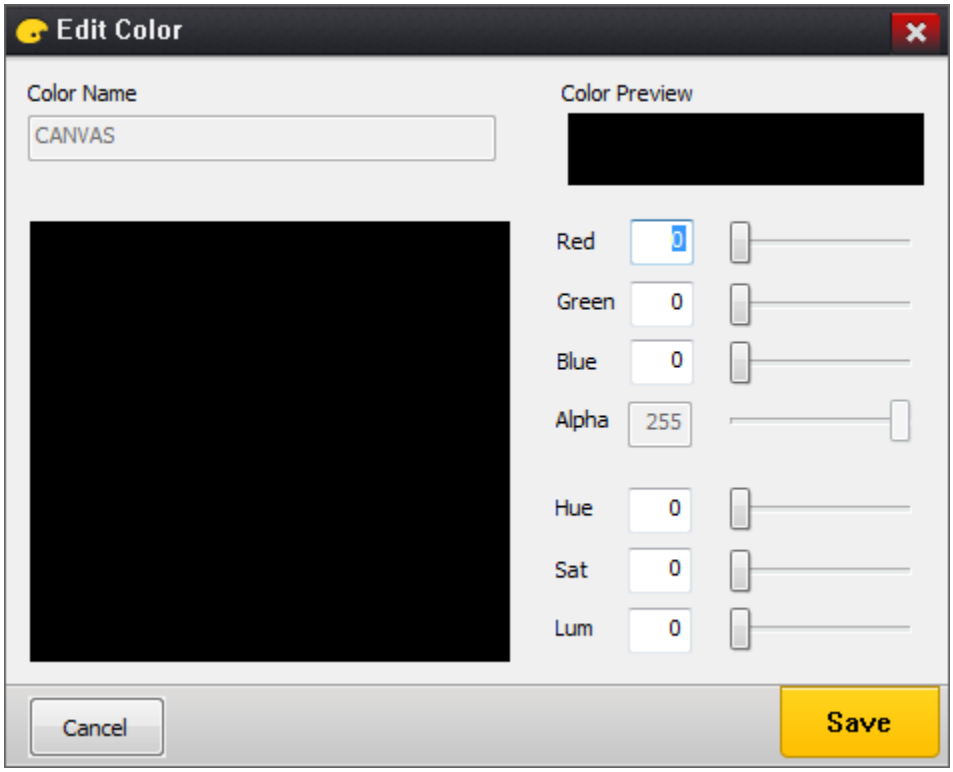


Figure 4.3

Adding a new color resource is easy, from the ***Colors*** section of the ***Resource View*** select the following:

Simply use the resulting color dialog to add a new color resource, as shown below in ***Figure 4.4****:*

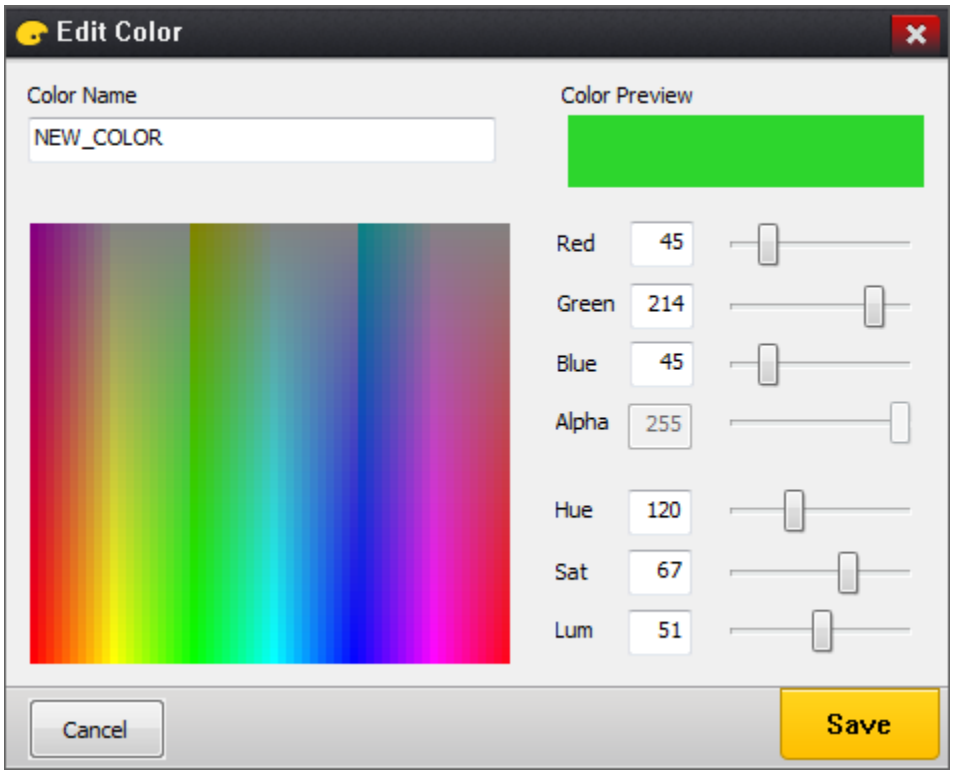


Figure 4.4

By selecting **Save** a new color resource with the name **NEW_COLOR** with the physical color green will be available for the application to use.

Font Resources

In order to manage font resources the ***Fonts*** section of the ***Resource View*** must first be expanded by clicking on the **+** field, resulting in the dialog shown below in ***Figure 4.5***:

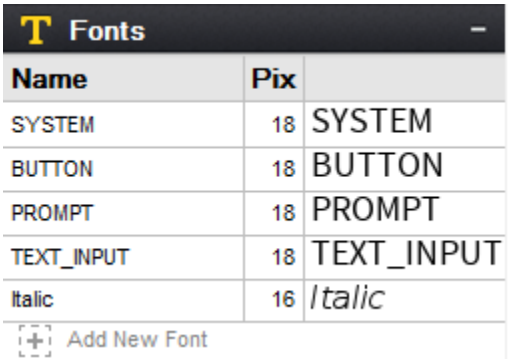


Figure 4.5

Font resources consist of one or more fonts, each with a unique logical name. For example, in **Figure 4.5** the logical name **SYSTEM** is associated with a specific font. This font resource is used whenever the application specifies **SYSTEM** as the font in the object properties.

The font group shows you the Font ID name on the left, the font height in pixels, and a WYSIWYG preview of the font glyphs on the right.

In the view above, the first four fonts are the pre-defined default fonts that are required by the GUIX library. You can change the font data associated with these fonts, however you cannot change these font ID names.

The last font shown above, named “Italic”, is a custom font that has been added to the project by the user.

Modifying a font resource is easy, simply double-click (or right-click and menu select) on the font resource. From this dialog the font resource can be modified to match the application’s UI needs. ***Figure 4.6*** shows the modification dialog when **SYSTEM** is double-clicked.

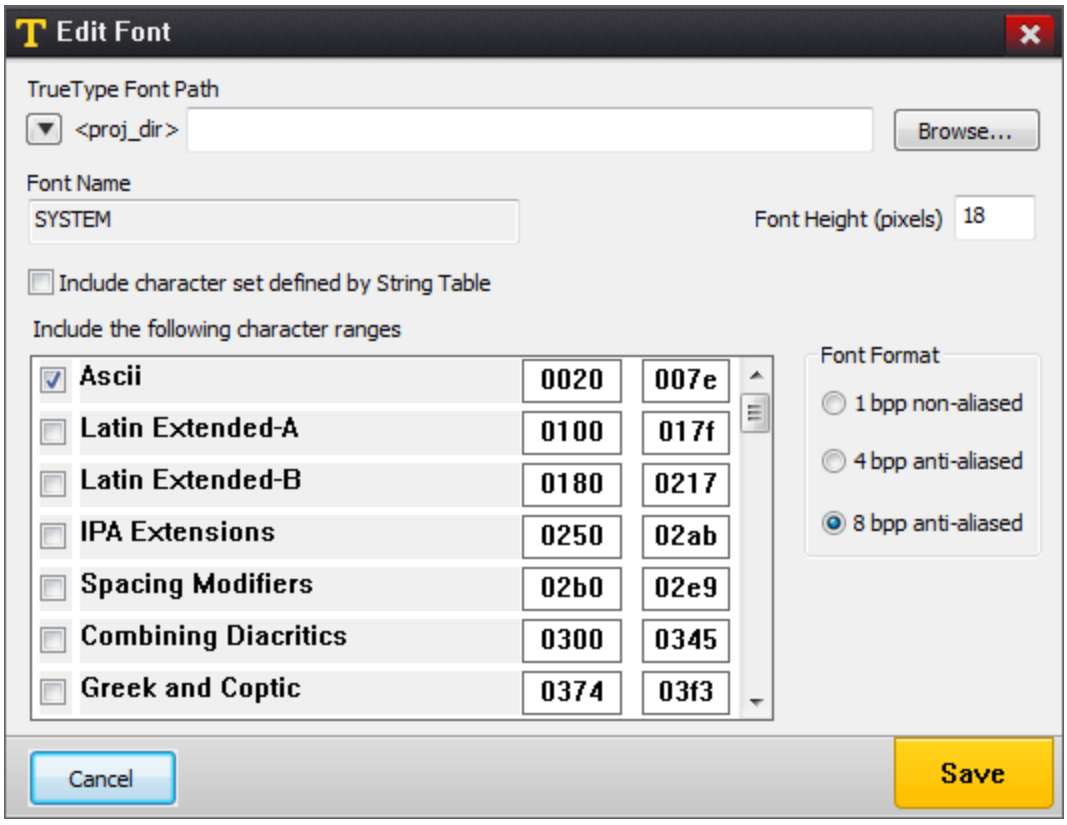


Figure 4.6

Adding a new font resource is easy, from the ***Fonts*** section of the ***Resource View*** select the following:

Simply use the resulting font dialog to add a new font resource, as shown below in ***Figure 4.7****:*

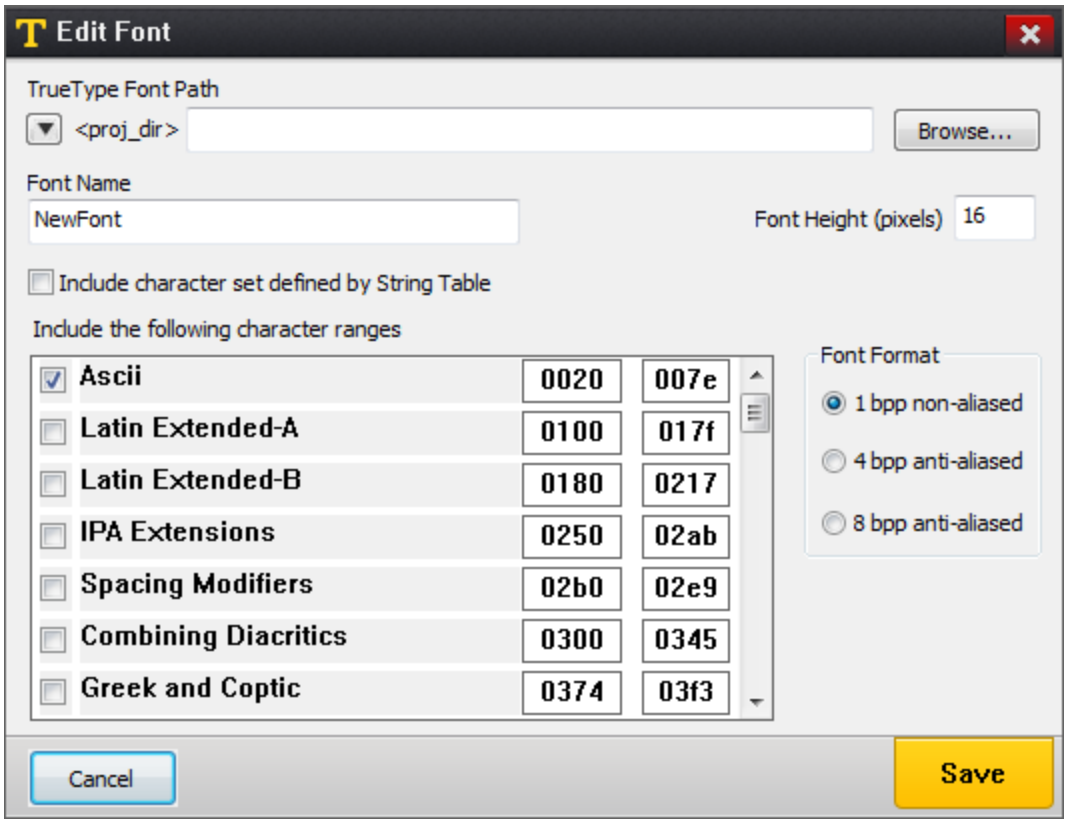


Figure 4.7

New GUIX fonts are created by GUIX Studio rendering a chosen TrueType font at a particular size. Therefore the dialog above first requires a TrueType font path. You can use the browse button to browse to a directory containing font files on your development system. Several TrueType fonts are also included in the GUIX/fonts sub-folder wherever you have installed GUIX Studio.

If possible the location of the TrueType font file is stored internally using a project-relative path. For this reason it is important to keep all of your font files in a common location and use a common directory tree structure for your projects and font files in order to enable you to move GUIX Studio projects from one development station to another.

The Font Name field allows you to specify the font resource ID name. This is the resource ID that will be used in the code generated by GUIX Studio and also used by your application when referencing the font. This name must follow C variable naming syntax requirements.

Once you have chosen a TrueType font file to use as input, enter a font logical name.

If you plan to use a particular font only to display some specified data, you should check “**Include character set defined by String Table**” to keep the font size to a minimum.

For extended or non-ASCII character sets, simply check the Unicode range list with characters desired. Of course if your source font must also contain the needed characters in order for GUIX Studio to produce meaningful glyphs for each character value.

Once the character range is determined, specify the font height in pixels and the font format. Both anti-aliased and binary fonts are supported. Binary font require less static data storage area but anti-aliased fonts produce the best appearance on targets running at 4-bpp grayscale or higher color depths.

When all fields are completed, click on the OK button to create a new font resource. GUIX Studio will generate a GUIX compatible font with the chosen properties, add that font to the project resources, and make the font available for the application to use.

Pixelmap Resources

In order to manage pixelmap resources the ***Pixelmaps*** section of the ***Resource View*** must first be expanded by clicking on the **+** field, resulting in the dialog shown below in ***Figure 4.8***:

When the Pixelmap group is expanded, you should see a preview similar to this:

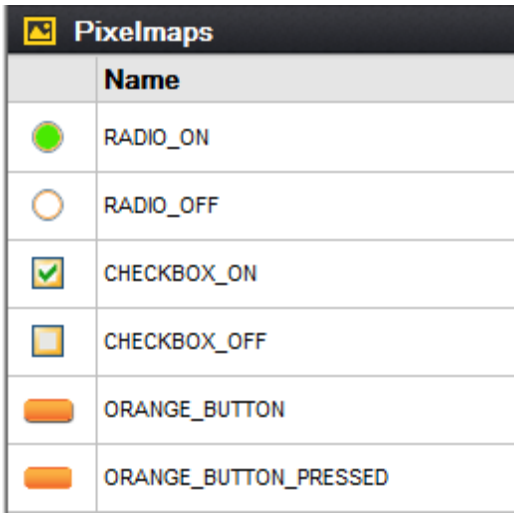


Figure 4.8

Pixelmap resources consist of one or more pixelmaps, each with a unique logical name. In this case, a scaled thumbnail image of the each pixelmap is shown on the left, and the pixelmap ID name is shown on the right.

The first group of pixelmaps comprises the pre-defined system pixelmaps required by GUIX widgets such as radio buttons and checkboxes. You can change the pixelmap data associated with the system pixelmaps, however you cannot change these pixelmap ID names. Also shown above are two custom pixelmaps named “ORANGE_BUTTON” and “ORANGE_BUTTON_PRESSED”. These are examples of pixelmaps a user has added to the project that might be used to render a GX_PIXELMAP_BUTTON widget.

Modifying a pixelmap resource is easy, simply double-click (or right-click and menu select) on the pixelmap resource. From this dialog the pixelmap resource can be modified to match the application’s UI needs. ***Figure 4.9*** shows the modification dialog when **RADIO_ON** is double-clicked.

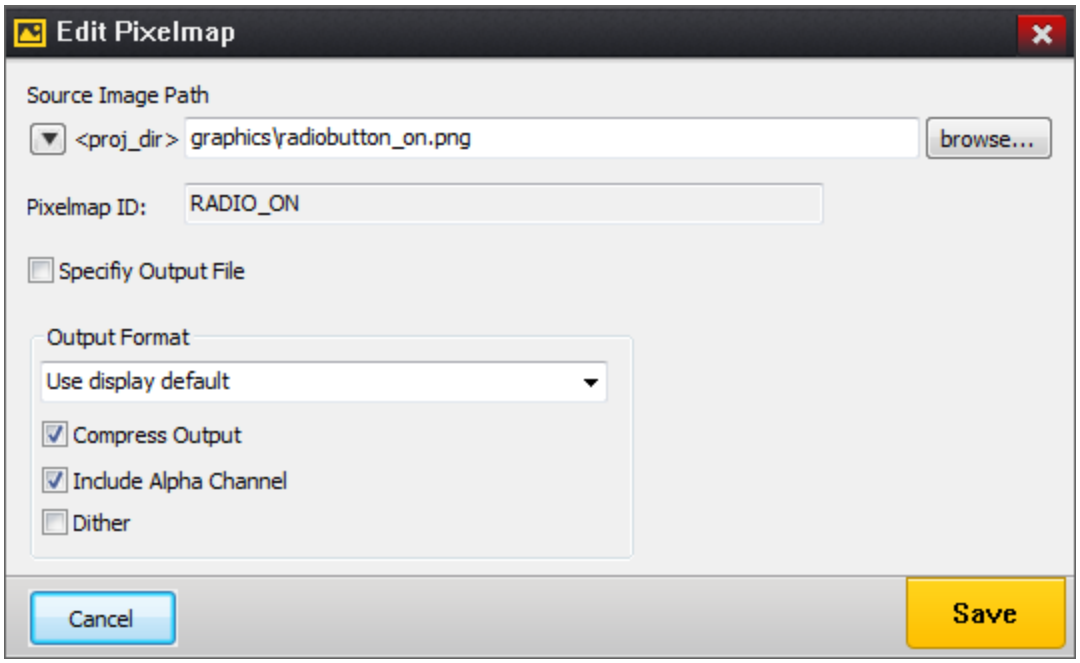


Figure 4.9

The Edit Pixelmap dialog allows you to define a new pixelmap or modify the content of an existing pixelmap. Behind the scenes, GUIX Studio reads the input image and converts the image to the GUIX GX_PIXELMAP format that can be used by the GUIX library. GUIX Studio also converts the color space of the incoming image to the color space of the display on which this pixelmap will be used.

The first field of this dialog is the path to the source image. GUIX Studio supports the input of PNG (.png) or JPEG (.jpg) format image files. You can use the “browse” button to find the desired input file on your local file system.

If possible the location of the input image file is stored internally using a project-relative path. For this reason it is important to keep all of your image files in a common location and use a common directory tree structure for your projects and image files in order to enable you to move GUIX Studio projects from one development station to another and not lose track of input image data.

The Pixelmap ID fields allow you to specify the logical name of the Pixelmap resource. This name typed here must be unique and must follow C variable naming syntax rules.

The Specify Output File checkbox allows you to specify a unique output file for each pixelmap. If this checkbox is not selected, the pixelmap data is written to the default resource file for this display. If the checkbox is

selected, you can type a specific filename into which the data for this pixelmap will be written. The purpose for this option is to allow you to divide your pixelmap data, which can be very large C arrays, into multiple output files. Certain compilers struggle to handle C files that are hundreds of thousands of source lines.

The “Compress Output” checkbox allows you to specify if the pixelmap output is uses a proprietary GUIX compression algorithm. Compressed output files are generally smaller, but they also require processor time to render on the target. Most often you will choose compression for your large pixelmaps, and use non-compressed format for your smaller pixelmaps.

The “Include Alpha Channel” checkbox determines how GUIX Studio utilizes alpha channel information that might be present in .png format input files. If this checkbox is checked and the display is running at 16-bpp color depth or higher, GUIX Studio will preserve the full incoming alpha channel data in the output file. If this checkbox is not checked, GUIX will produce a smaller output file that may include transparency, but will not include full alpha-blending channel information.

Finally, the “Dither” checkbox instructs GUIX Studio to optionally apply an advance dithering algorithm when down-sampling the input image to a lower color depth display data format. Dithering is usually enabled, but can cause larger output files if compression is used because there will be fewer “repeating” pixel color values.

Once all options are set as desired, click the OK button to produce a new pixelmap resource. GUIX Studio will read the input image file, decompress it, perform color space conversion and dithering, optionally re-compress the data, and save the data in GUIX compatible GX_PIXELMAP format. The new pixelmap is added to the project resources and made available for the application to use.

Adding a new pixelmap resource is easy, from the ***Pixelmaps*** section of the ***Resource View*** select the following:

String Resources

When the Strings group is expanded you should see a preview of the project string table, as shown below:

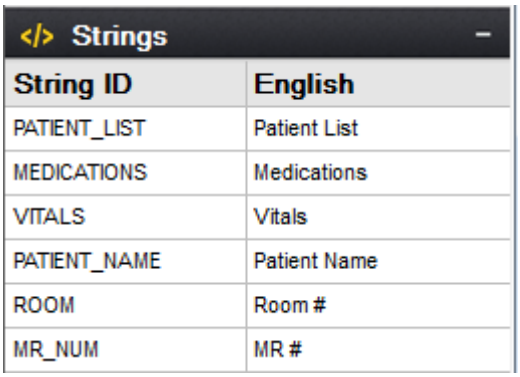


Figure 4.11

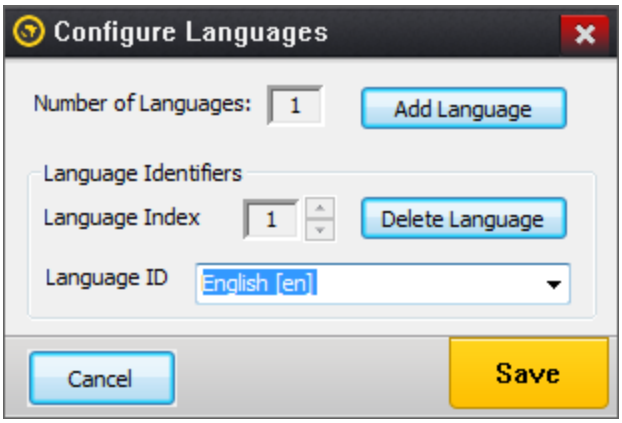
String resources consist of one or more strings, each with a unique logical name. For example, in **Figure 4.11** the logical name “PATIENT_LIST” is associated with the string “Patient List” shown on its right. This string resource is used whenever the application specifies PATIENT_LIST as the string in the object properties.

Always remember that your ID names for all resource types must be C syntax compatible variable names. These names will be used extensively when your project resource files and specifications files are produced by Studio.

Modifying a string resource is easy, simply double-click (or right-click and menu select) on the string resource to invoke the ***String Table Editor*** dialog. From the ***String Table Editor*** dialog the string resource can be modified to match the application’s UI needs. ***Figure 4.12*** shows the modification dialog when **STRING_13** is double-clicked.
In this case, the string ID name is shown on the left, which the string content for the first or reference language is shown on the right. Of course the exact string content is very specific to your application, however the layout of the String group preview is consistent.

GUIX Studio supports static text and multi-lingual application by defining and maintaining a String Table. The String Table defines one string ID for each record, and one string constant for each record for each supported language.

The languages to be supported by your application are defined by using the Language Configuration Dialog, show here:



The Language Configuration Dialog is invoked by using the Configure | Languages command on the application menu. This dialog allows you to define the number of languages to be supported by your application and the name or language ID to be associated with each language. The languages supported can be modified after your project has been created, however if a language is removed you should be aware that the string data associated with that language is also removed and cannot be retrieved.

The first language or “Index 1” language is referred to as your “reference language”. This is the language that GUIX Studio will use when you are defining and editing your UI design. All other languages in your string table are referred to as Translation Languages. GUIX Studio supports exporting and importing the string table data in industry standard XLIFF format data files, convenient for exchanging string information with translators who might assist the application developer with adding translations for the languages to be supported other than the reference language. When you export the GUIX string table to an XLIFF file, the reference language along with one translation language are included in the XLIFF string data exchange file. Similarly, when you import an XLIFF file, the imported data is used to populate one translation language in your GUIX String table.

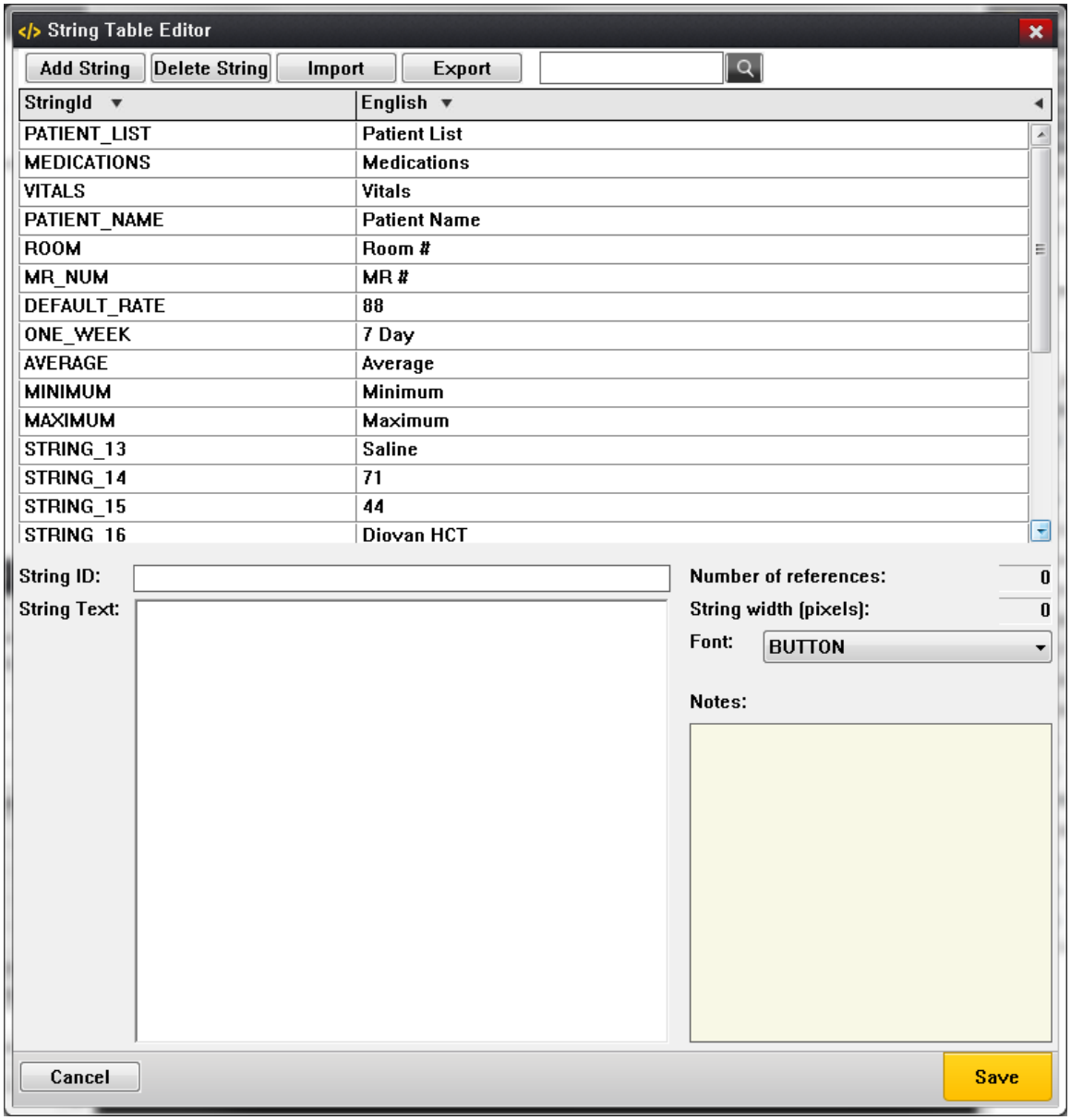


Figure 4.12

The String Table Editor dialog first displays a list of string IDs on the left, followed by the reference language string data. If more than one language is defined, a third column shows any one of the supported translation languages. You can open and close the third column by clicking on the small arrow at the top-right of the reference language column.

When the translation language column is visible, you can cycle through the translation languages contained in the project by clicking on the small arrows at the top-right of the translation language column of the string list.

You can edit a string record by clicking on the record in the table to select it. When a record is selected, the record String ID and string content are shown in the fields below the table view. You can type new values into these fields to modify the string ID and string content.

The fields to the right of the string content include:

“Number of references”: This field indicates how often a particular string ID is used within the GUIX Studio project. If the reference count is 0, this string may be obsolete and may optionally be removed by the user.

String Width (pixels) indicates the display width of the string using the indicated font.

The “Notes” field is an optional comment field that allows you to add information about the purpose or use of each string. These notes are included in any exported XLIFF string data files to aid translators in making accurate and meaningful string translations.

Any time you have the ***String Table Editor*** dialog open you can add additional strings to your project by clicking on the Add String button at the top of the dialog. Obsolete or unused strings can be removed from the project by first selecting the string, then clicking on the Delete String button at the top of the dialog.

In addition to manually adding new strings to your project using the String Table Editor dialog, you can also add new strings indirectly by simply typing string content in the “Text” field of the Properties View of any widget that supports text. In other words, when you are adding new widgets in the target view or typing text information in the properties view, these actions are automatically creating new entries in the project string table.

Adding Language Translations

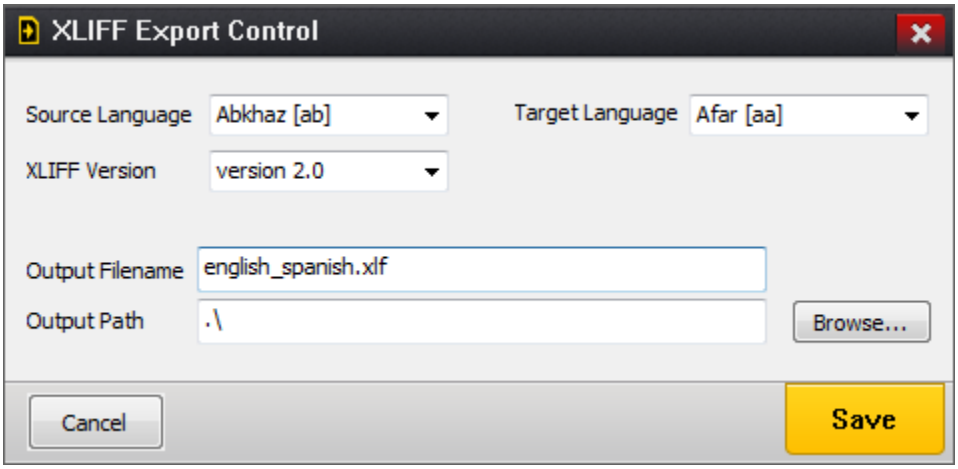
The GUIX Studio string table editor supports a language definition workflow that allows the developer to create an application using his primary language, then export the string data to a standard schema XML file to be sent to a language translation expert. The translation file is then returned to the developer, who can import the language translations back into his Studio project, thereby adding support for a new language to his application.

This facility is invoked by using the Export (to write the string data to a file) and Import (to read the translated strings) buttons at the top of the String Table Editor. The Export button is used to create an XLIFF schema XML

file which contains your reference language strings. This file can be utilized by a translator using tools and editors that support the standard XLIFF file format.

When a translation expert returns the XLIFF file to you with the new string translations, you can use the Import button to read the data from this XLIFF file. If the XLIFF file contains a new language, the new language is added to your project. If the XLIFF file contains new string data for an existing language, this new data is imported into your project. The reference language strings are not modified by the Import operation.

When you click the Export button, the XLIFF Export Control dialog, show below, is displayed:



The Source Language and Target Language fields specify which string table columns will be written to the XLIFF file as the reference language and the translation language. The Source language is the reference strings, and the Target Language is the language for which your translator will provide translated string data.

The XLIFF version field specifies one of two main XLIFF file format versions, either version 1.2 or version 2.0 (and later). These XLIFF file format standards are incompatible, and you need to know which version your tools utilize before using the XLIFF Export/Import commands. More information about the XLIFF schema and XLIFF standards can be found here:

version 1.2: http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html
version 2.0: http://docs.oasis-open.org/xliff/xliff-core/v2.0/os/xliff-core-v2.0-os.pdf

The output filename and output path fields allow you to specify the filename and location to which the output file will be written. The filename is entirely up to the user, however we suggest that you use names that indicate the source and target languages contained within the exported file.

Chapter 5

GUIX Studio Screen Designer

Designing application screens is the primary purpose of GUIX Studio. Screen design is accomplished through all the various views described previously in Chapter 3. However, the main element of screen design in GUIX Studio is the ***Target View***, which is where all the screen elements are shown visually and in exactly the same manner they will appear on the embedded target display. These screen elements can be selected, moved, resized, etc. via simple mouse and button operations. In addition, alignment and Z-order button operations are available on selected object(s). The following sub-sections describe various features of GUIX Studio screen design.

Creating/Configuring Projects

Creating projects in GUIX Studio is straightforward – simply select the ***New Project*** button or the menu selection ***Project -> New Project***. Next, GUIX Studio presents the ***Configure Project*** dialog. From this dialog, basic display settings, as well as path information for where to locate code generated by GUIX Studio is specified.

When a new project is created, the configure project dialog is presented. This is where the developer specifies the number of hardware displays available on the target and the properties each display. Properties include the display's logical name, x/y resolution, color depth and format, and other display properties. GUIX Studio supports multiple displays in the same project. If additional displays are required, the ***Number of Displays*** field should be changed to match the number of displays on the embedded device. The maximum number of displays in a project is 4. ***Figure 5.1*** shows the Configure Project dialog.

Modifying the project and/or display settings is accomplished by either the menu option ***Configure -> Project/Displays*** or by selecting the project or display, right-clicking, and selecting ***Configure Project/Display***. In either case, the ***Configure Project*** dialog is presented to facilitate changes to the project settings and/or display(s).

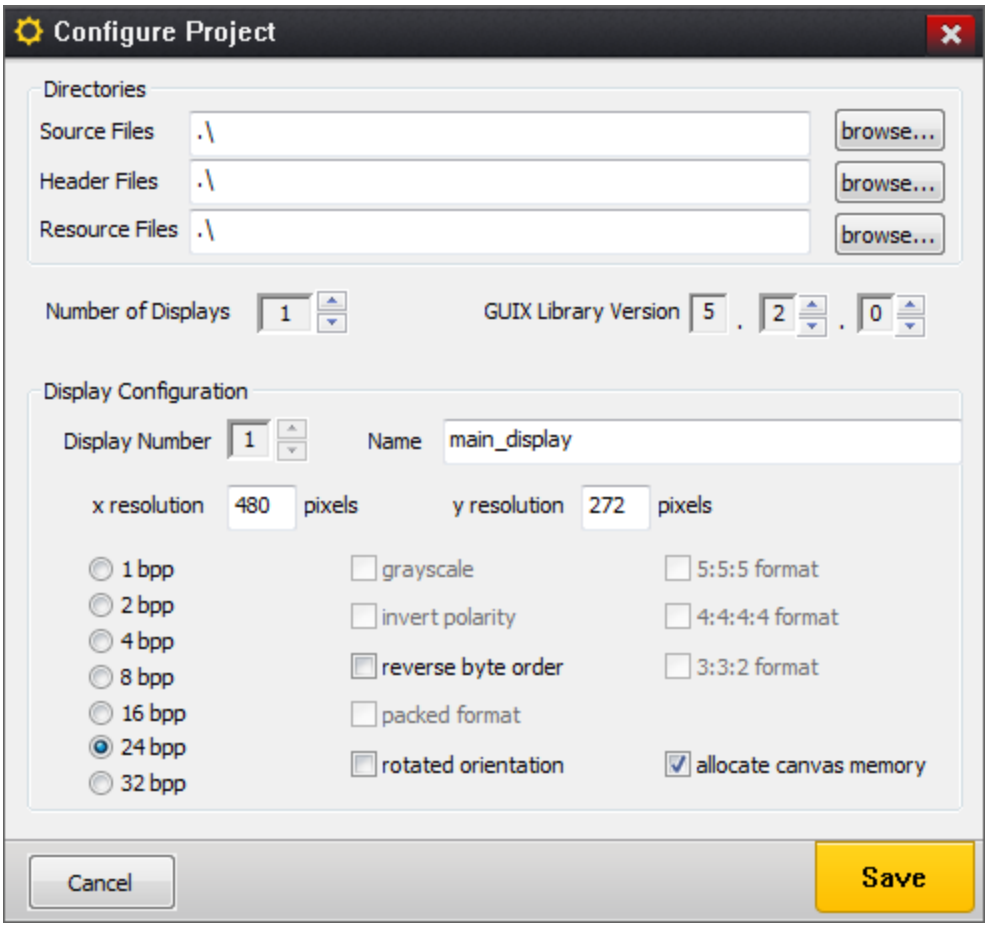


Figure 5.1

The Directories group is where you can specify the default output directories for the C source and header files produced by Studio. These directories are normally saved relative the project location to make it easy to move projects from one computer to another or from one filesystem to another.

When you invoke the Studio “Generate Application” or “Generate Resources” commands, these are the default directories into which those source files will be written. Of course you can override these directory locations at any time by entering new locations in the Output Directory dialog.

Selecting Widgets

Selecting widgets is done by either clicking on the widget in the ***Project View*** widget tree or by clicking on the widget visible in the ***Target View*** area. When a single widget is selected, its properties are displayed in the ***Property View*** area. ***Figure 5.2*** shows the widget “***button***” selected.

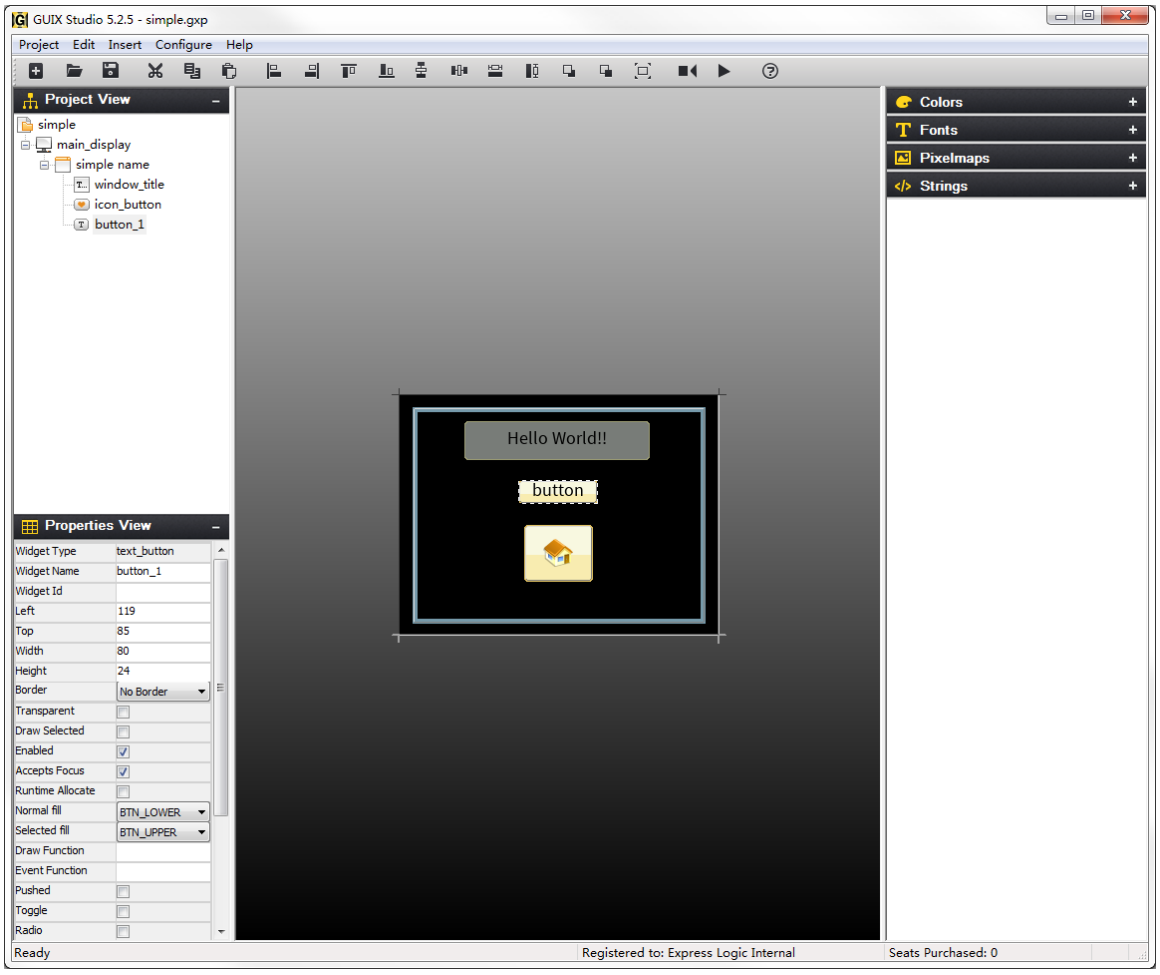


Figure 5.2

Using Properties

As mentioned previously, the properties for a selected widget are presented in the ***Properties View***. All widgets have a common set of properties as well as some properties that are specific to the particular widget type. For example, a button widget has a ***Pushed*** property while a window widget does not. The following are the common set of widget properties:

Property	Meaning
Widget Type	Type of widget, for reference
Widget Name	Name of widget, passed to the widget create function and used for variable naming in the generated source files.

Widget ID	ID of widget. This ID value is used to generate signals from child widgets to their parent screens.
Left	Left-most coordinate of widget
Top	Top-most coordinate of widget
Width	Width of widget in pixels
Height	Height of widget in pixels
Border	Type of widget border
Transparent	Should be checked if the widget is partially transparent
Draw Selected	Should be checked if the widget should initially draw itself in the selected state.
Enable	Should be checked if the widget can be selected or clicked by the end user.
Accepts Focus	Should be checked if the widget accepts focus.
Runtime Allocate	Should be checked if the widget control block should be allocated dynamically.
Normal Fill	Normal fill color resource id
Selected Fill	Selected fill color resource id
Draw Function	User-defined custom drawing function Name. If this field is blank, the standard drawing function for that widget type is used.
Event Function	User-defined custom event handling function name. If blank, the standard event handling for this widget type is used.

Figure 5.3 shows the properties of a simple window widget.

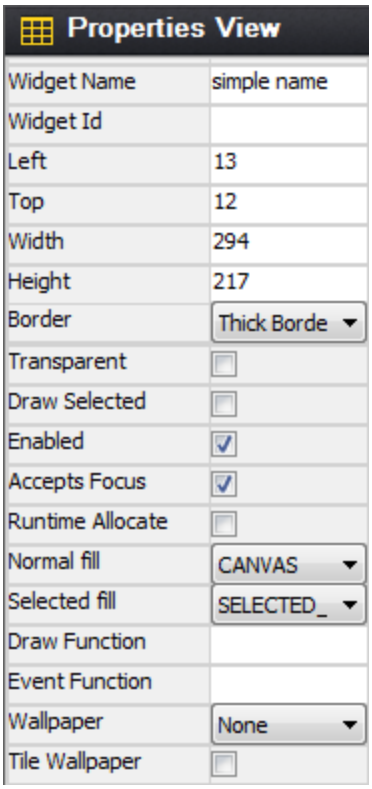


Figure 5.3

Many widget types have additional properties specific to each widget type. For example, in Figure 5.3 above, the Window widget type supports a Wallpaper pixelmap Id, and a style setting indicating if the wallpaper should be centered or tiled.

Text widgets support a string ID field, along with text alignment styles and a font specification. The additional widget properties are normally very intuitive once you have read the description of each widget type and the available styles and Create function parameters for that widget type.

Manipulating Widgets

To manipulate a widget, is first must be selected. This is done by either clicking directly on the widget in the ***Target View*** or by selecting it in the ***Project View*** widget tree. Once selected, the widget will have a dashed-outline. In this state, it may be moved by simply clicking on the widget and dragging it to the desired location on its parent. If the widget is a top-level widget, dragging the widget is effectively setting the widget's initial position on the target display. Of course it is always possible to move or resize any widget at any time using the GUIX API.

To resize the widget's height, position the mouse on the top edge of the widget and wait for the mouse pointer to change to an up-down arrow. At this point the widget height may be changed by simply moving the mouse while the right mouse button is depressed. The width of the mouse may be resized in a similar fashion by positioning the mouse pointer on the left edge of the widget. ***Figure 5.4*** shows the "***button***" widget resized and moved to the left/top area of the parent window.

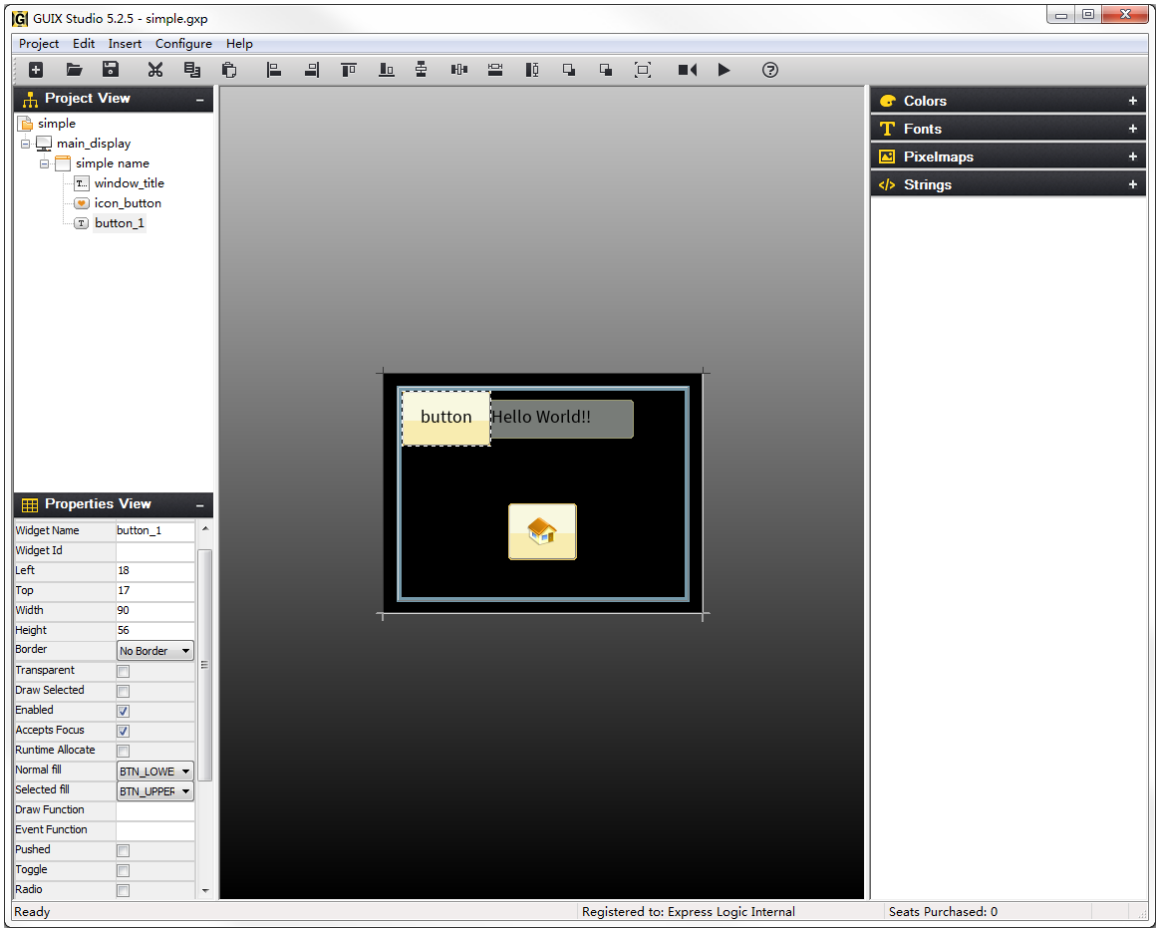


Figure 5.4

Manipulating Multiple Widgets

Selecting multiple widgets is accomplished by clicking on multiple widgets in the target view while holding the ***Ctrl*** key down. Doing this will show each of the widgets selected with a dashed-outline around it. Note that when selecting multiple widgets each widget in the selection group must a child of the same parent.

Once multiple widgets are selected, they may be simultaneously moved by clicking inside one on the selected widgets and moving the mouse with the right mouse button pushed down. In addition, the alignment buttons on the ***Tool Bar*** may be used to align the group of selected widgets. ***Figure 5.5*** shows both the "***button***" and "***new button***" widgets selected and ***Figure 5.6*** shows the result of the ***Align-Left*** button selection while these widgets are selected.

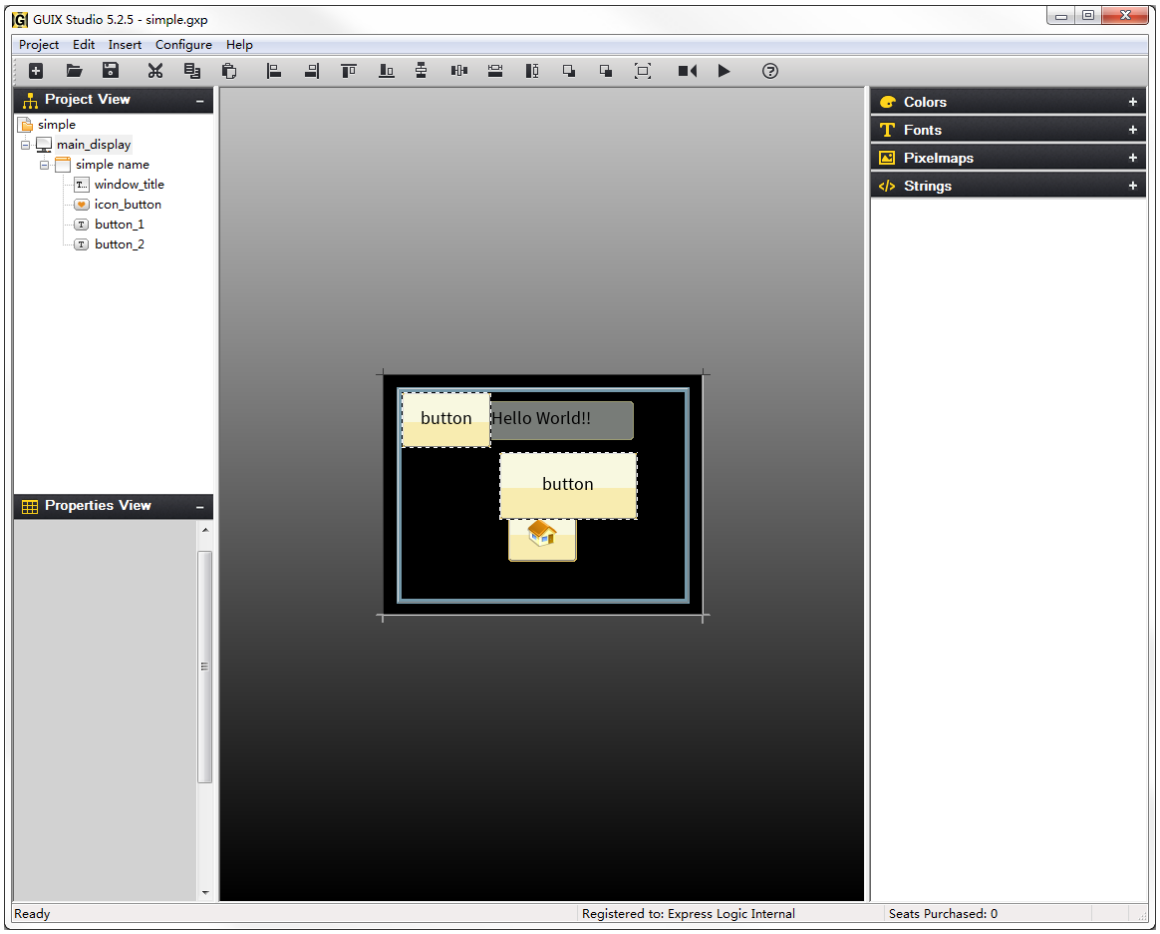


Figure 5.5

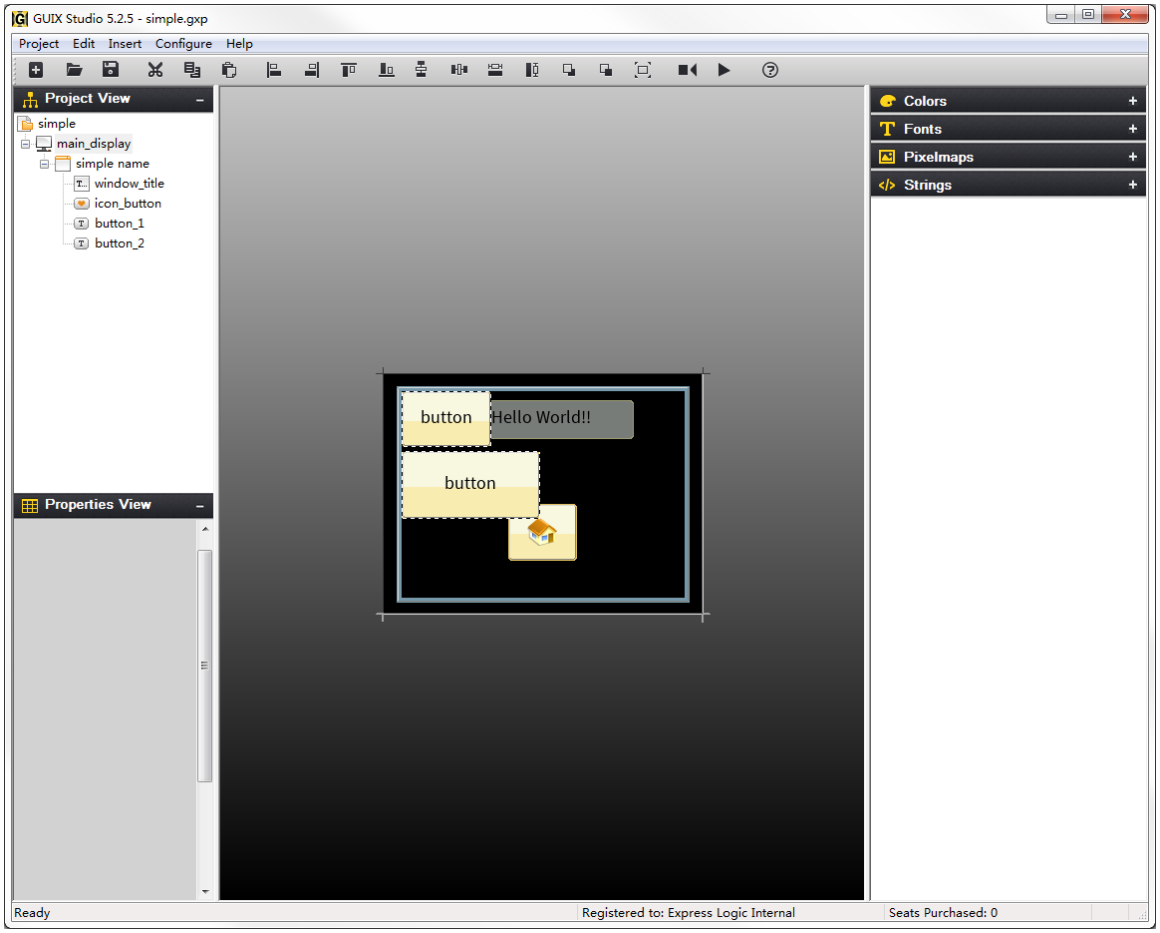


Figure 5.6

Cut/Copy/Paste Operations

A selected widget in the ***Target View*** may be cut, copied, and pasted in standard fashion. The ***Tool Bar*** has buttons for cut, copy, and paste. There are also the same options in the Edit menu option. Note that when pasting a widget, the parent widget should be selected before pasting the new widget. ***Figure 5.7*** shows the result of selecting the "***button***" widget, copying it, and pasting the copy in the same window.

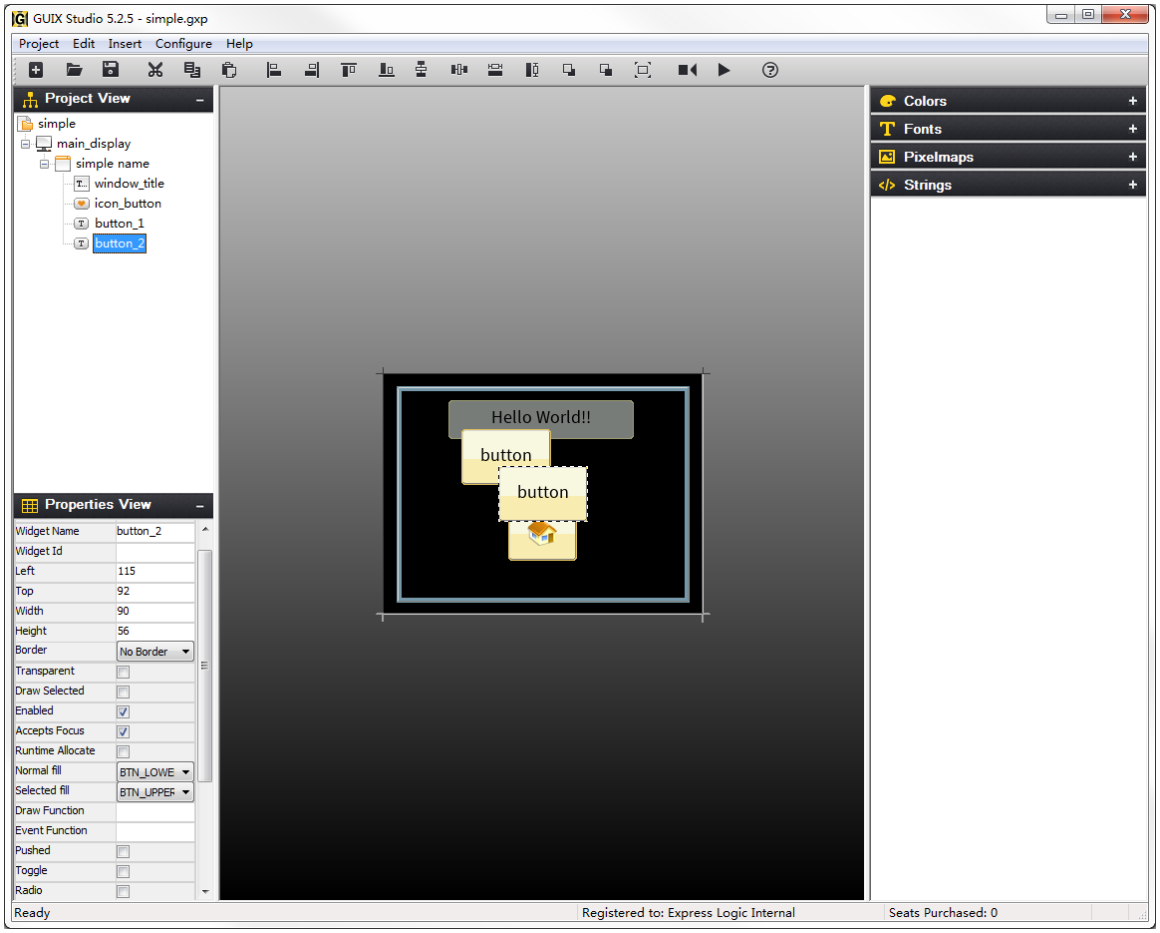


Figure 5.7

Changing Z-Order

Widgets can easily be moved in front of or behind other widgets. This is accomplished by selecting the widget and selecting either the ***Move to Front*** or ***Move to Back*** buttons on the ***Tool Bar***. ***Figure 5.8*** shows the moving the second button to the back.

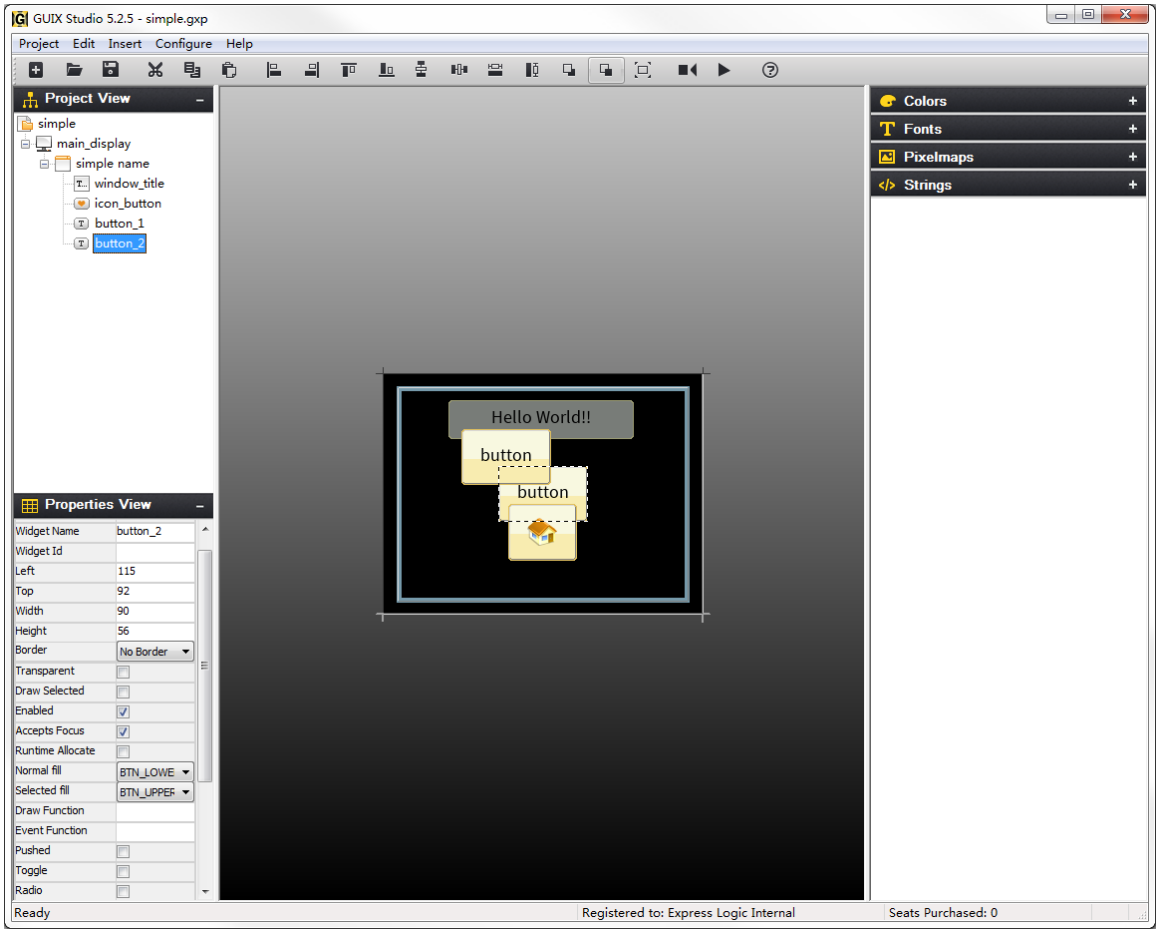


Figure 5.8

Assigning Colors, Fonts, and Pixelmaps

In addition to selecting colors, fonts, and pixelmaps in the Properties View for a selected widget, a shorthand drag-and-drop method of assigning resources to widgets is also supported. To use this feature, simply left-click on a resource such as a color of font in the resource view, and drag the resource over the desired widget in the target view. Drop the resource by releasing the left mouse button over the widget.

Color resources are always assigned to the widget normal background color when using the drag and drop method. Other colors such as selected color or selected text color must be assigned using the Properties View.

Similarly, pixelmap resources are assigned to the “normal” or “fill” pixelmap field of a widget that supports pixelmap display. To assign other

fields to a widget that supports multiple pixelmaps, you must use the Properties View.

Record and Playback Macro

Macro record and playback functions help you record and playback keystrokes and mouse events.

Recording to a macro file is accomplished by selecting the ***Record Macro*** toolbar button or the menu selecting ***Edit -> Record Macro***. GUIX Studio will presents the ***Record Macro*** dialog which allows you to specify the pathname for your macro file. After making this selection, click the ***Record*** button to start recording. After you have finished recording, again select the ***Record Macro*** toolbar button or use the pull-down menu selecting ***Edit -> End Macro*** to end macro recording.

Playback of a macro file is accomplished by selecting the ***Playback Macro*** toolbar button using the main pull-down menu to select the ***Edit -> Playback Macro*** command. GUIX Studio presents the ***Playback Macro*** dialog which allows you to specify the previously recorded macro file to be run.

When recording macros that choose input or output files, such as adding a font or image, it is important to use the keyboard to type the file name, rather than using the mouse to select from the file browser. Since the macro recorder records mouse and keyboard events, and since your file browser may change over time, it is more reliable to type the filename than to select the file graphically.

Chapter 6

GUIX Studio Generated Code

When you are done editing your screens and resources, GUIX Studio produces a set of source files that can be incorporated into your embedded application. The source files are generated by selecting ***Generate Resource Files*** and ***Generate Specifications*** from the Project menu item. The files generated by GUIX Studio are intended to be compiled and linked with the embedded application source code.

The user's embedded application code makes references to the code generated by GUIX Studio. Furthermore, the GUIX Studio generated code expects all custom widget drawing, event handling, and memory allocation functions specified in the project to be defined in the user's embedded application code. If they are not, link errors will be present when building the application.

The user should never have to modify the code generated by GUIX Studio and should resist doing so. All UI modifications should be made in the associated GUIX Studio project. This will keep the project synchronized with the embedded application.

Generating Resource Files

Resource files generated by GUIX Studio contain C preset data structures that define all of the GUIX Studio resources (colors, fonts, pixelmaps, and strings), which is effectively all the resources defined in the ***Resource View*** of the project. By default, there are two files generated, one file is a standard C source code file and the other is a C header file that provides external references and constants that are necessary for the application code to access the GUIX resources defined in the project. The file names are of the form:

{project-name}_**resources.h**
{project-name}_**resources.c**

For example, the Resource files created for the "***simple***" GUIX Studio project are:

simple_resources.h
simple_resources.c

Generating the Resource files is accomplished by selecting ***Generate Resource Files*** option in the ***Project*** menu option. The destination of the resource files is specified in the ***Configure Project*** dialog, which is accessible via the ***Configure Project/Displays*** option in the ***Configure*** menu item.

For Pixelmap and Font resources, you can specify a custom output filename for each pixelmap and font in the associated resource editing dialogs. This feature allows you to put very large resources in distinct files, rather than putting all resources in one common output file. If you do not specify an overridden filename for a font or pixelmap resource, those resources are written into the common resource file.

Generating Specification Code

The Specification files generated by GUIX Studio contain all the C code to create the UI designed in GUIX Studio. This code also references the Resource files generated for this project. The user's application code will make calls to this code to actually create the UI objects defined in the project. Furthermore, the user's application code contains all custom widget drawing, event handling, and memory allocation functions specified in the project. By default, there are two files generated, one file is a standard C source code file and the other is a C header file that provides external references and constants that are necessary for the application code to access the GUIX Studio Specifications. The file names are of the form:

{project-name}_**specifications.h**
{project-name}_**specifications.c**

For example, the Specification files created for the "***simple***" GUIX Studio project are:

simple_specifications.h
simple_specifications.c

Generating the Specification files is accomplished by selecting ***Generate Specification Files*** option in the ***Project*** menu option. The destination of the Specification files is specified in the ***Configure Project*** dialog, which is accessible via the ***Configure Project/Displays*** option in the ***Configure*** menu item.

Integrating with User Code

Integrating the Resource and Specification files generated by GUIX Studio is straightforward, simply follow these steps:

1. Either copy or make the Resource and Specification files accessible via path settings to the embedded build environment

2. Add all Resource and Specification files to embedded IDE project or makefile

3. Ensure the application embedded code calls the necessary functions to initialize and create the UI contained in the Resource and Specification files

4. Ensure the application embedded code contains all necessary custom widget drawing, event handling, and memory allocation functions

5. Build the application (compile and link)

6. Execute the application!

Chapter 7

Simple Example Project

This chapter describes how to create a simple example project in GUIX Studio and execute the example on GUIX.

Create New Project

The first step is creating a new project and configuring the displays and languages that the project will support. When you first start GUIX Studio, you will see the ***Recent Projects*** screen:

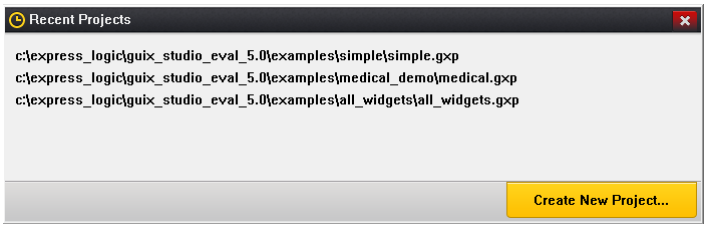


Figure 7.1

Click on the ***Create New Project***… button to begin a new project. You will be presented with the ***New GUIX Project*** dialog, shown here:

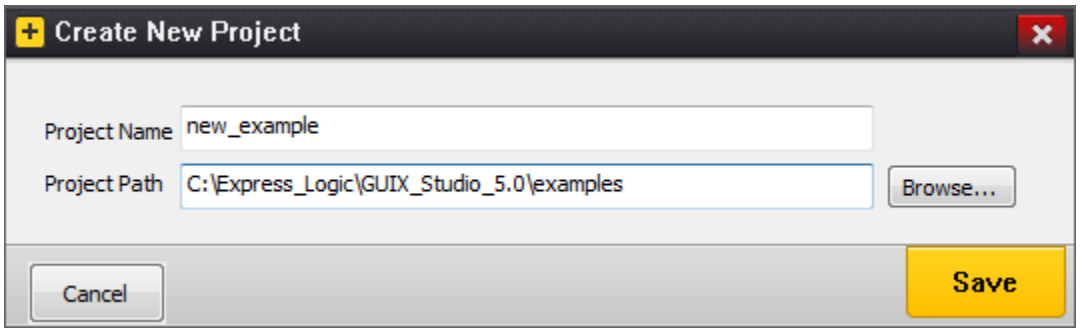


Figure 7.2

Type the name "***new_example***" as the project name. Note that project names should use standard C variable naming rules, i.e. no special or reserved characters. Type the path to the location where the project should be saved. The path must be a valid file system directory, i.e. GUIX Studio will not create the directory if it does not exist. Click "OK" to create the project.

The next screen shown is the Project Configuration screen, shown here:

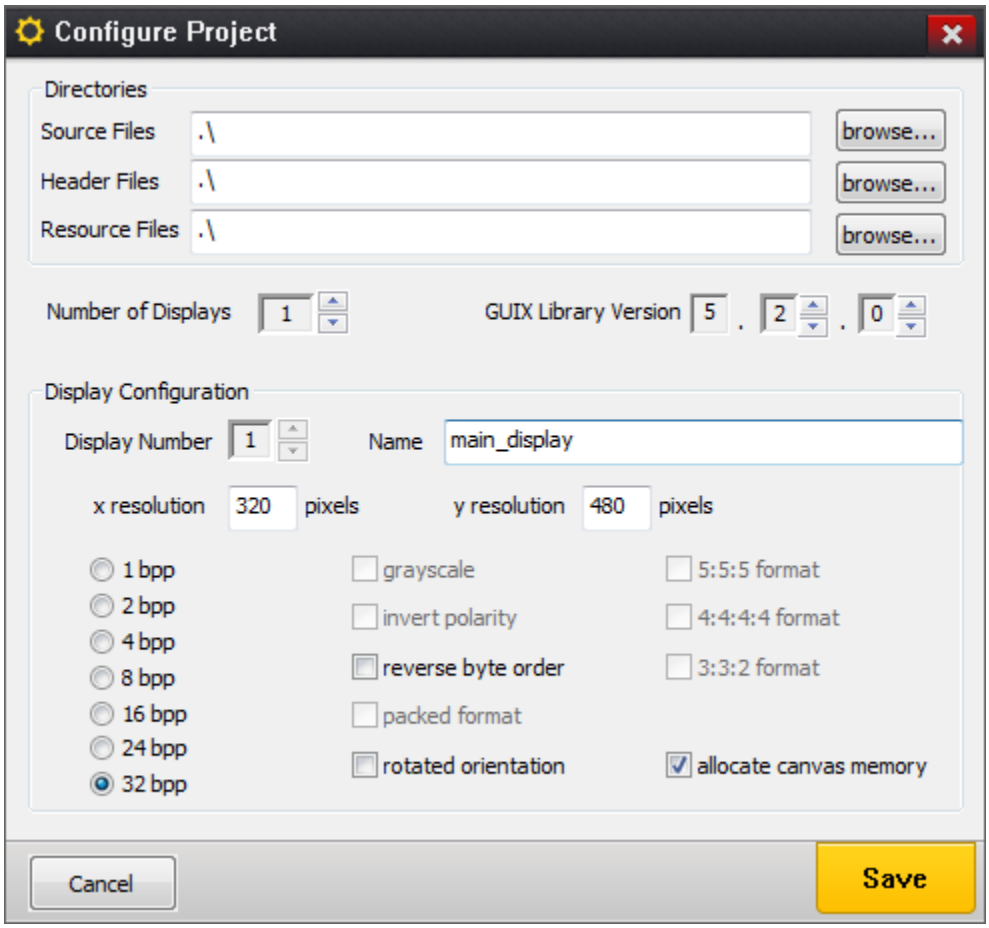


Figure 7.3

This dialog allows you to specify how many displays your project will support, and give a name to each display. You must also specify the color format supported by each display, and optionally type a pathname for the output files generated by Studio for each display. The default directory for the output files is "**.**", meaning the C output files are written to the same directory as the project itself.

For this example, leave the ***Number of Displays*** set to "1", type the name "***main_display***" in the display name field, and check "***allocate canvas memory***". Leave the resolution, color format, and directory fields at their default values, and click ***OK***.

You should now see your project open with the Studio IDE, similar to this:

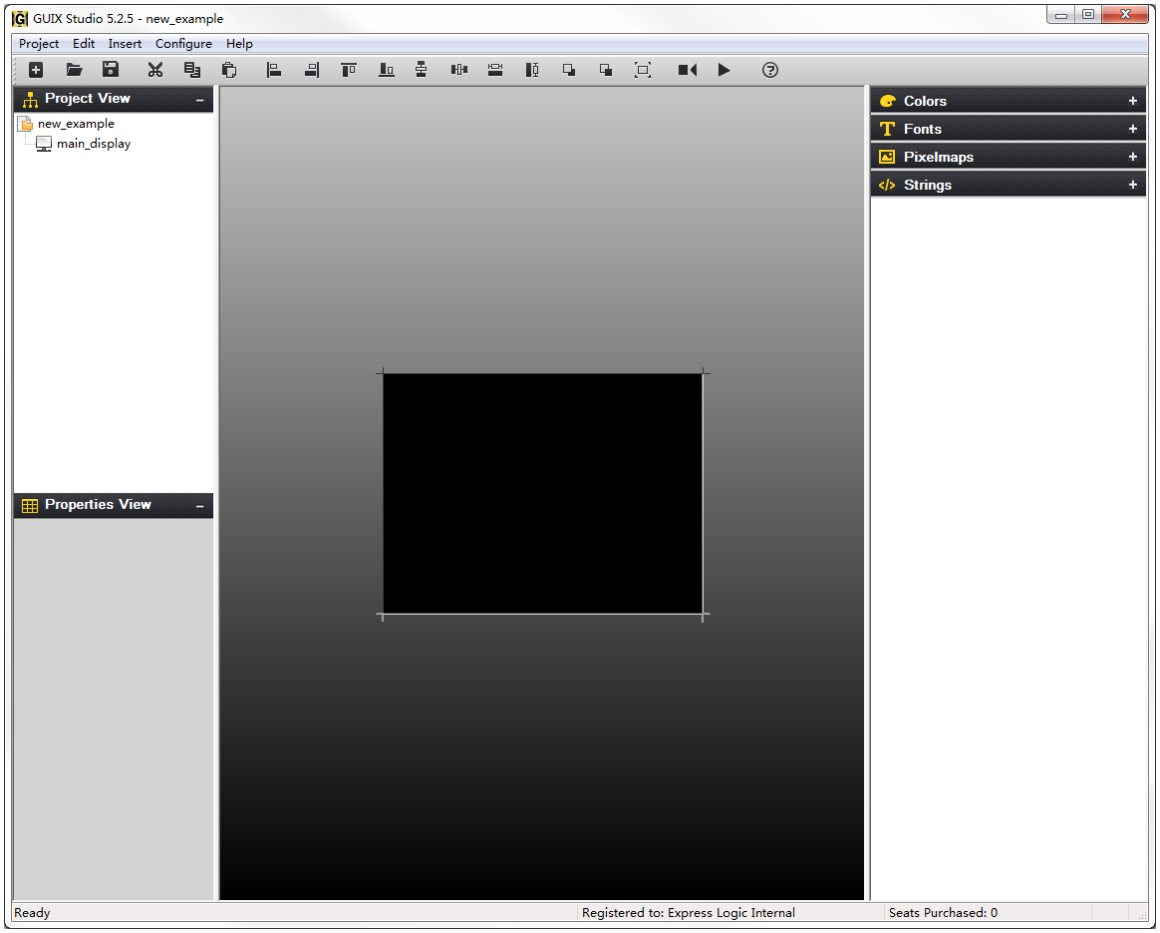


Figure 7.4

The next step is to create a screen to be shown on the display. To do this, click on the display name "***main_display***" in the project view, and use the menu command ***Insert -> Window -> Window*** to add a window to the display.

You should now see a rather plain gray window centered within the "***main_display***" within the Target View:

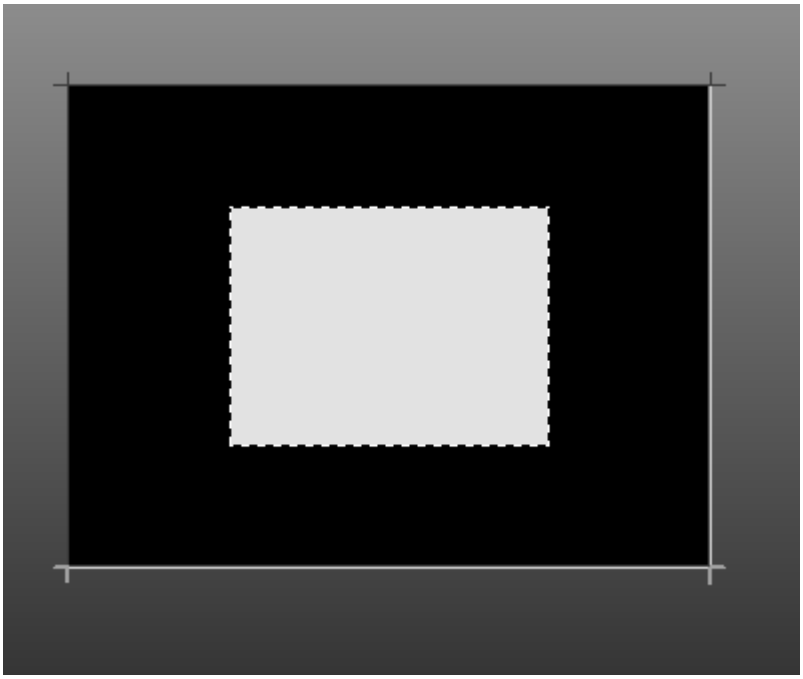


Figure 7.5

If the window is not selected, click on the window so that the dashed selection box is drawn around the window. Now in the ***Properties View***, change the ***Widget Name***, ***Widget Id***, ***Left***, ***Top***, ***Width***, ***Height***, and ***Border*** to match those settings shown below. As you make these changes, you should see your changes immediately taking effect within the Target View.

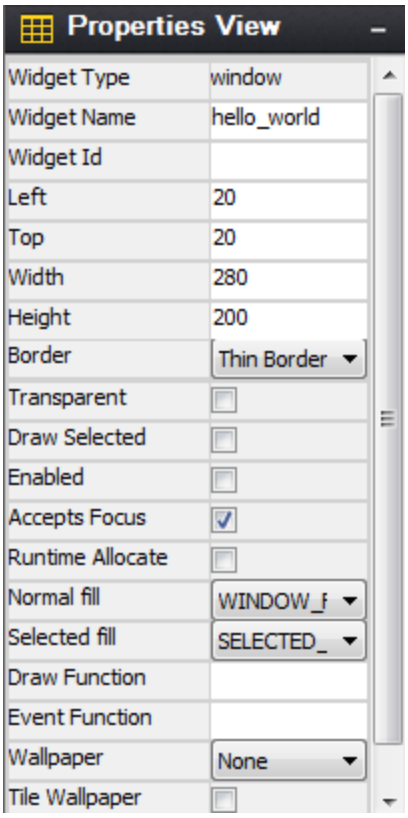


Figure 7.6

Next we will add a pixelmap resource to be used within a ***GX_ICON*** widget. Several icons are provided with your GUIX Studio distribution that will work fine for this example. Expand your ***Pixelmaps*** Resource View and click the ***Add New Pixelmap*** button:

Browse to your GUIX Studio installation folder, and within the ***./graphics/icons*** folder select the file named ***i_history_lg.png***. Click ***Open*** to add this resource to your project. Your ***Pixelmaps*** Resource View should now show a preview of the just added icon image:

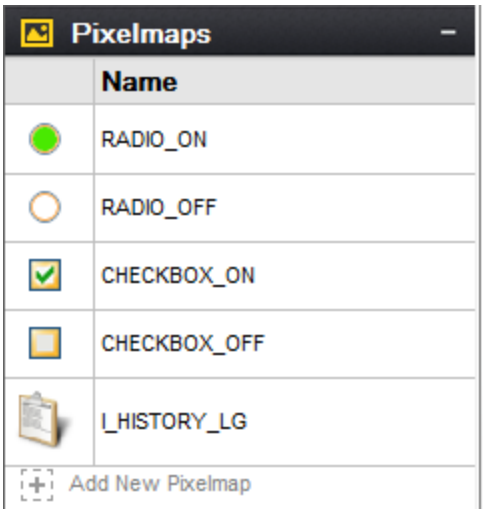


Figure 7.7

We will use this new image resource later as part of our UI design.

Similar to adding a pixelmap resource, we will add a new font resource to our toolbox so that we can use this font later in our design. Expand the ***Fonts*** Resource View and click on the ***Add New Font*** button. This will bring up the ***Add/Edit*** font dialog. Next, browse to the supplied GUIX fonts in the GUIX Studio installation folder, ***.\fonts\verasans*** and select the TrueType font file named ***VeraIt.ttf***. Type font the font name "***MEDIUM_ITALIC***" in the font name field, and type "***30***" in the height field. Your dialog should now look like this:

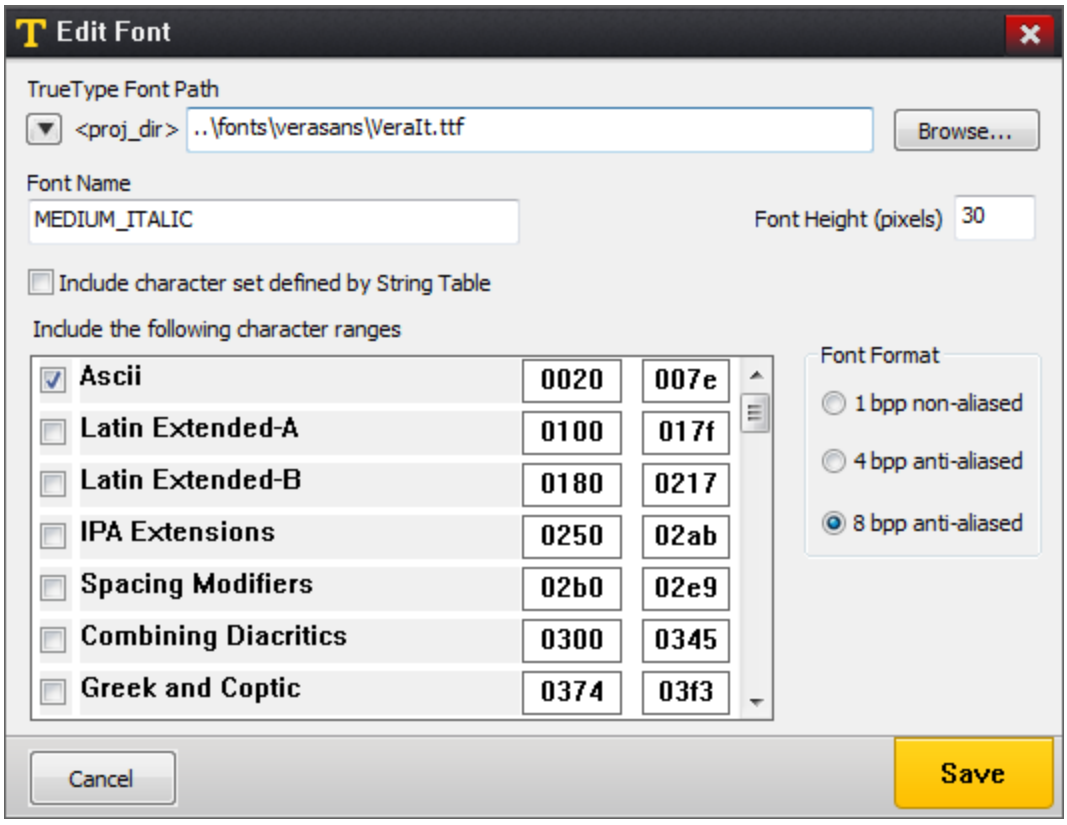


Figure 7.8

Click ***OK*** to add this font to your project. You should now see the font in your Resource View:

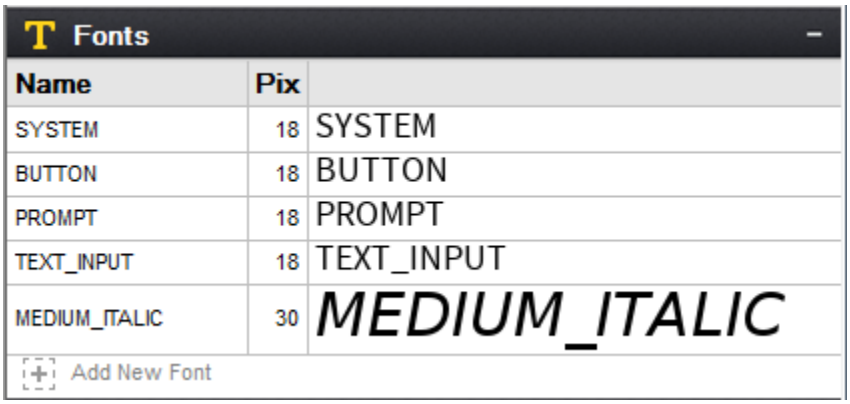


Figure 7.9

We will use this new font later in our UI design.

Now that we have some new resources available, we need to add some child widgets to our screen that can utilize these resources. Select the previously created window named "***hello_world***" by right-clicking on the

window in the Target View. In the pop-up menu that is now displayed, select the menu command ***Insert ->Text -> Prompt*** to insert a new ***GX_PROMPT*** widget and attach the widget to the background window. Your window should now look like this:

Figure 7.10

Click on the font named "***MEDIUM_ITALIC***" in the ***Fonts*** Resource View, and drag and drop the font on the prompt widget. Next, edit the prompt properties as show below to resize the prompt, set the prompt transparency, and change the prompt text and style:

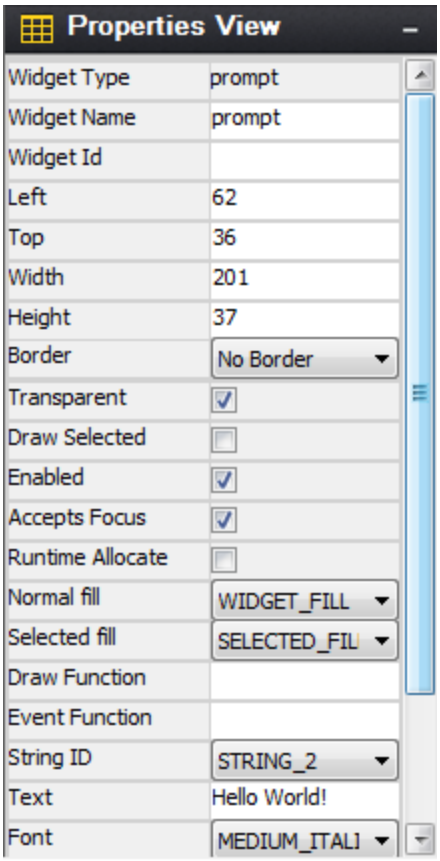


Figure 7.11

You may need to scroll up and down in the Properties View to see each of these settings depending on your screen resolution. After making these changes, your Target View should now look like this:

Figure 7.12

Next we will place an Icon Button style widget on the screen. Select the background window by clicking on it, and use either the top-level menu or the right-click pop-up menu to select ***Insert -> Button -> Icon Button*** to add a new ***GX_ICON_BUTTON*** to the window. Click on the Icon named ***I_HISTORY_LG*** that we added earlier and drag it to the icon button. Using the properties view, adjust the icon position and size as show below:

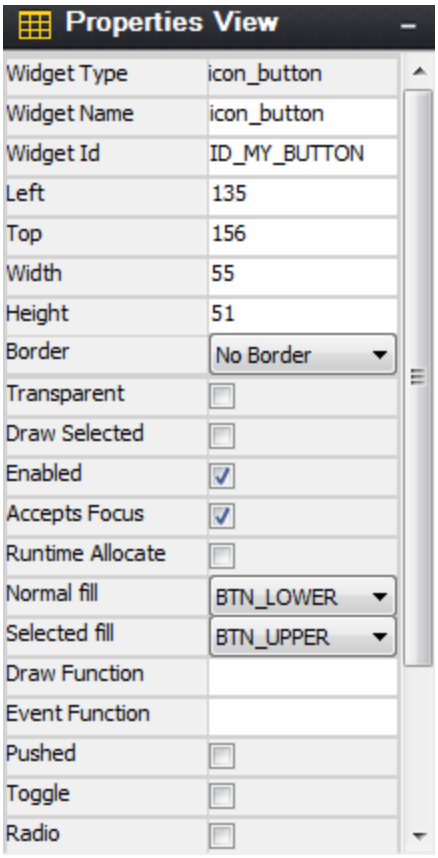


Figure 7.13

Your screen should now look like this:

Figure 7.14

We will call this complete for the simple example screen design. Of course your actual application screens will likely be much more sophisticated, but this is enough to show you how to use GUIX Studio to create your own application screens.

Generate Resource and Application Code

The next step is to generate the resource file and specification file that define the embedded GUIX run-time UI. To do this you will need right-click on the ***main_display*** node in the Project View, and select the ***Generate Resource Files*** command. You should observe an information window that indicates your resource files have been generated, as show below:

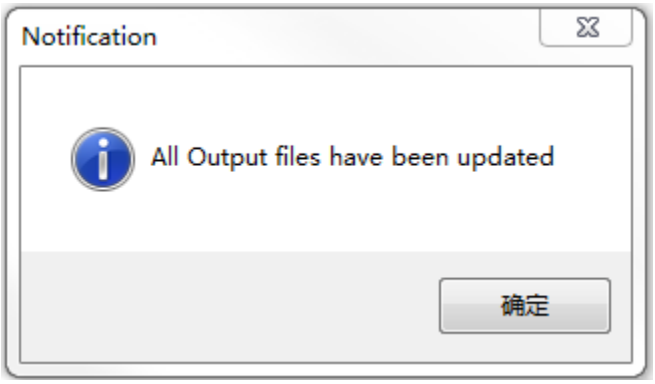


Figure 7.15

Click ***OK*** to dismiss this notification, and use the same procedure to right-click on the ***main_display*** node and select the ***Generate Specification Files*** command. You should observe a similar notification window. You have now generated your simple UI application files.

Create User Supplied Code

The next step is to create your own application file that will invoke the GUIX Studio generated screen design. Using your preferred editor, create a source file named ***new_example.c***, and enter the following source code into this file:

```
/* This is an example of the GUIX graphics framework in Win32. */

/* Include system files.  */

#include <stdio.h>
#include "tx_api.h"
#include "gx_api.h"


/* Include GUIX resource and specification files for example.  */

#include "new_example_resources.h"
#include "new_example_specifications.h"
```

```
/* Define the new example thread control block and stack.  */

TX_THREAD   new_example_thread;
UCHAR       new_example_thread_stack[4096];


/* Define the root window pointer.  */

GX_WINDOW_ROOT    *root_window;

/* Define function prototypes.   */

VOID  new_example_thread_entry(ULONG thread_input);
UINT  win32_graphics_driver_setup_24bpp(GX_DISPLAY *display);

int main()
{
    /* Enter the ThreadX kernel.  */
    tx_kernel_enter();

    return(0);
}


VOID tx_application_define(void *first_unused_memory)
{

    /* Create the new example thread.  */
    tx_thread_create(&new_example_thread,
                     "GUIX New Example Thread",
                     new_example_thread_entry, 0,
                     new_example_thread_stack, sizeof(new_example_thread_stack),
                           1, 1, TX_NO_TIME_SLICE, TX_AUTO_START);
}


VOID  new_example_thread_entry(ULONG thread_input)
{

    /* Initialize the GUIX library */
    gx_system_initialize();

    /* Configure the main display.  */
    gx_studio_display_configure(MAIN_DISPLAY,         /* Display to configure*/
                win32_graphics_driver_setup_24bpp,  /* Driver to use       */
                LANGUAGE_ENGLISH,                   /* Language to install */
                MAIN_DISPLAY_DEFAULT_THEME,         /* Theme to install    */
                &root_window);                      /* Root window pointer */

    /* Create the screen - attached to root window.  */
    gx_studio_named_widget_create("hello_world", (GX_WIDGET *) root_window, GX_NULL);

    /* Show the root window to make it visible.  */
    gx_widget_show(root_window);

    /* Let GUIX run. */
    gx_system_start();
}
```

The source code above creates a typical ThreadX thread named “***GUIX New Example Thread***” with a stack size of 4K bytes. The interesting work begins in the function named ***new_example_thread_entry***. This is where the GUIX specific thread begins to run.

The first call is to the function named ***gx_system_initialize***. This call is always required before any other GUIX APIs are invoked to prepare the GUIX library for first use.

Next, the example calls the function ***gx_studio_display_configure***. This function creates the GUIX display instance, installs the requested language of the application string table, installs the requested theme from the display resources, and returns a pointer to the root window that has been created for this display. The root window is used as the parent of all top-level screens that our application will display.

Next the example calls ***gx_studio_named_widget_create*** to create an instance of our ***hello_world*** screen. This function uses the data structures and resource produces by GUIX Studio to create an instance of the screen as we have defined it. We pass the root window pointer as the second parameter to this function call, meaning we want the screen to be immediately attached to the root window. The last parameter is an optional return pointer that can be used if we want to keep a pointer to the created screen.

Next ***gx_widget_show*** is called, which makes the root window and all of its children, including the ***hello_world*** screen, visible.

Finally, the example calls ***gx_system_start***. This function begins executing the GUIX system event processing loop.

Build and Run the Example

Building and running the simple example is specific to your build tools and environment. However we can define the general process:

1) Create a new directory and application project
2) Add these files to the project:

    new_example_resources.c
    new_example_specification.c
    new_example.c

3) Add the Win32 run-time support files from the GUIX Studio installation path ***./win32_runtime***. This includes the ThreadX and GUIX Win32 header and run-time library files.
4) Add the GUIX Win32 library (***gx.lib***) to the project
5) Add the ThreadX Win32 library (***tx.lib***) to the project
6) Compile, Link, and Run the application!

Index

Renesas Synergy™ Platform

User's Manual: Software

Publication Date: Rev.5 October, 2015

Renesas Synergy™ Platform

User's Manual: Software

Renesas Electronics Corporation

R11UM0002EU0500