USER GUIDE

EVE Screen Designer 3.0

Document Version: 1.0

Date: 09-02-2017

Contents

I Preface

A. Purpose

This document describes the functionality and procedures involved in using the **E**VE **S**creen **D**esigner 3.0.

B. Intended Audience

The intended audience shall be any developer working with EVE products and the ESD3 screen designer tool.

C. Related Documents

Document Name	Document Type	Document Format
FT81x Series Programmers Guide	Programming Guide	PDF
FT81x Datasheet	Datasheet	PDF
FT90x Toolchain Installation Guide	Installation Guide	PDF

D. Feedback

Every effort has been taken to ensure that the document is accurate and complete. However any feedback on the document may be emailed to docufeedback@brtchip.com. For any additional technical support, refer to http://brtchip.com/contact-us/.

II Overview

A. Introduction

EVE **S**creen **D**esigner **3.0** is the next generation of Bridgetek's smart IDE for EVE, making EVE-based GUI development much easier to accomplish. This tool enables users to build one GUI application using a **visual programming**[1] method without needing to know any EVE-specific display list commands.

ESD 3.0 provides a **WYSIWYG** ("What You See Is What You Get") environment for editing graphics, designing visual effects, and defining GUI application user logic, generating **ANSI C code** for the targeted hardware platform. Users can also choose to simulate the whole design within ESD 3.0 to experience the UI before compiling and downloading the generated source code. Furthermore, ESD3 has the capability to work seamlessly with the Bridgetek MCU tool chain.

ESD 3.0 introduces several key new features compared to ESD2, such as high level widget and code generation, providing users the ability to define a UI, including its behaviour, without coding display lists. ESD 3.0 is designed to greatly accelerate EVE based GUI development.

B. Features

The following are some of the key features of ESD 3.0 -

- **WYSIWYG** GUI
- High level widgets
- No EVE display list knowledge required
- Widget based GUI construction
- Drag and drop widget to create screen layout
- Widget communication
- Screen logic creation without coding
- Simulation of screen logic and user touch input using mouse
- Building and downloading the generated "C" code (if FT90X Toolchain is installed)

[1] https://en.wikipedia.org/wiki/Visual_programming_language

C. Known Issues & Limitations

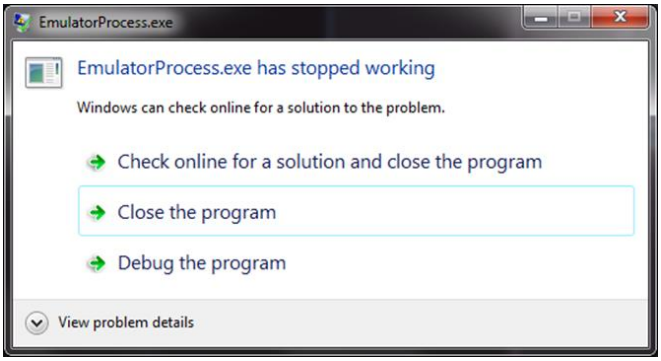
The following are some known issues and limitations of ESD 3.0 -

- Only the FT81X series EVE is supported. FT800 Series EVE is NOT supported
- The C code editor in ESD 3.0 does not support all the features of a code editor.
- Video and Sketch features are not supported.
- If the project file path is too long (i.e. more than 512 bytes), ESD 3.0 may have a problem opening it. Usually the typical error message is like below:

  `"Unable to generate output files, check directory permission at: C:\Users\xxxx.xxxx\Project\ESD3\....."`

  Where

  `"C:\Users\xxxx.xxxx\Project\ESD3\....."` refers to the project folder.
- PALETTED8 format is not yet supported in the Bitmap widget
- Logic node editor background goes white after windows hibernates
- In some rare cases, users may encounter a dialog box (shown below) which will not affect the functionality. In this case, just ignore the dialog box by closing it.



- Users may need to click the "Recompile" button to update the frozen simulation result.

D. Terms & Description

Abbreviations	Description
Actor	One type of logic node which is regularly updated but without visual appearance
DLL	Dynamic Link Library is a collection of small programs, any of which can be called when needed by a larger program that is running in the computer
ESD 3.0	EVE Screen Designer 3.0 or its later versions
EVE Emulator	Bridgetek behaviour-modelling software for EVE. Refer to AN_281_FT800 Emulator Library User Guide for details.
HAL	Hardware Abstraction Layer is a software subsystem for UNIX like operation systems providing hardware abstraction.
IDE	Integrated Development Environment
Logic Node	The node expressing certain logics.(Also referred to simply as "node" in this document)
Logic Node Editor	The place create logics by connecting the logic node
Page	One single screen in design
Simulation	Run the generated C code on PC
Widget	One type of logic node which has visual appearance rendered by EVE

E. Credits

Open Source Software

- TinyCC: http://bellard.org/tcc/ and http://repo.or.cz/tinycc.git available under LGPL.
- Errorlist module: https://github.com/kaetemi/errorlist available under MIT license
- QScintilla: part of PyQt available under PyQt license

  https://www.riverbankcomputing.com/commercial/license-faq

Icons Copyright

Some of the icons used in ESD 3.0 are from:
http://p.yusukekamiyamane.com/icons/search/fugue/ used in compliance with the Creative Commons Attribution 3.0 License.

III Setup & Installation

A. System Requirements

To install ESD 3.0 application, ensure that your system meets the requirements recommended below:

- ✓ Ideally Windows 10; alternatively Windows 8 or 7 with the latest windows updates
- ✓ 1.6GHz or faster processor
- ✓ 1GB of RAM (1.5GB if running on a virtual machine)
- ✓ Multi-Core CPU is highly recommended
- ✓ At least 512MB hard disk space
- ✓ Display resolution 1080 x 800 pixels or higher
- ✓ Write permission to the installation folder of ESD 3.0

B. Hardware Requirements

The exported C source code from ESD 3.0 is targeted at **FT90X MCU Module** and **EVE Module** platforms. Therefore, ESD 3.0 will be able to build and run generated code directly on these platforms:

- **ME810A-WH70R(800x480)**
- **ME811A-WH70C(800x480)**
- **ME812A-WH50R(800x480)**
- **ME813A-WH50C(800x480)**
- **ME810A-HV35R(320x480)**

Plus

- **MM900EV-LITE**
- **MM900EV1A**
- **MM900EV2A**
- **MM900EV3A**

While working on hardware platform (FT90x MM modules), users must ensure that an SD card is inserted into the SD Slot since ESD 3.0 application will try to detect the presence of an SD card. If the SD card could not be detected, then the application will stop working.

C. Dependencies / Pre-Requisites

- **Visual C++ Redistributable for Visual Studio 2015**

  If the PC does not have Microsoft Visual Studio 2015 installed, Visual C++ Redistributable is required. Users can download this from:

  https://www.microsoft.com/en-sg/download/details.aspx?id=48145

- **Windows 10 Universe C Runtime**

  ESD 3.0 has run-time dependency on Windows 10 Universe C Runtime (CRT). You may download it from https://www.microsoft.com/en-us/download/details.aspx?id=48234 and install on your PC should the following problem be encountered:

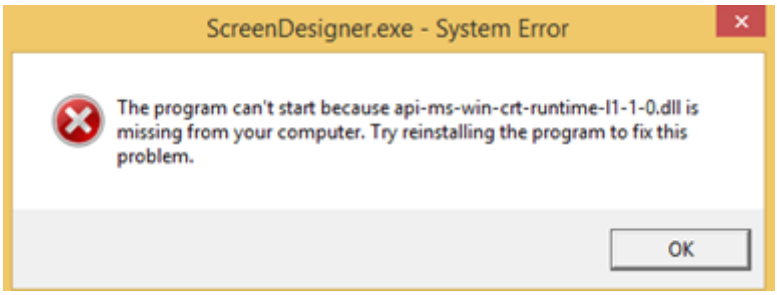


Figure 1 - Screen Designer - System Error

- **FT90X Tool Chain Version 2.1.0 or later**

  To compile and build projects, the FT90X Tool Chain 2.1.0 must be installed on the PC. It is downloadable from http://brtchip.com/ft90x-toolchain/.

  Please ensure that the Tool Chain executable path is defined by the system PATH environment variable.

  Users are advised to check the Known issue and limitation (of FT90x Toolchain) while building the ESD 3.0 project with FT90x Toolchain.

D. Installing ESD 3.0

The following steps will guide you through the ESD 3.0 *Setup/Installation* process.

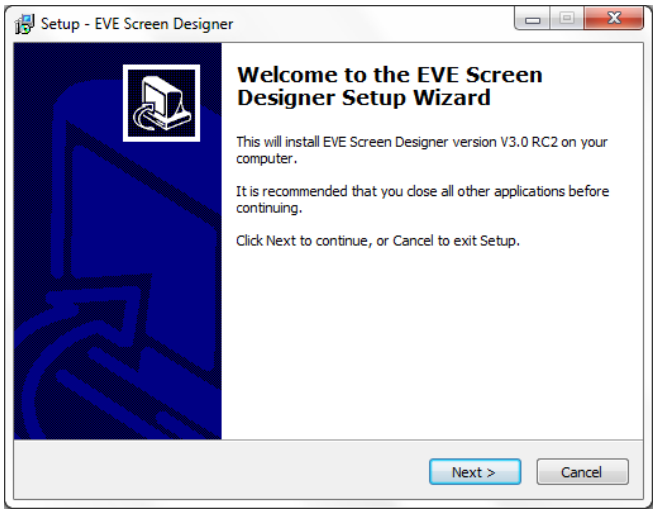
i. Download the package from www.brtchip.com.

ii. When prompted with a download dialog box. Click on **Save**.

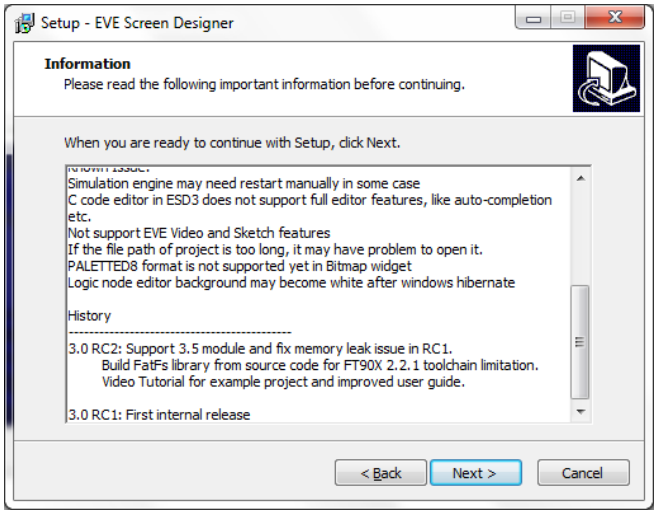
iii. Navigate to the folder under which the package files are downloaded.

iv. Extract the zip file contents. Double click on the executable file – **EVE Screen Designer 3.0.exe**

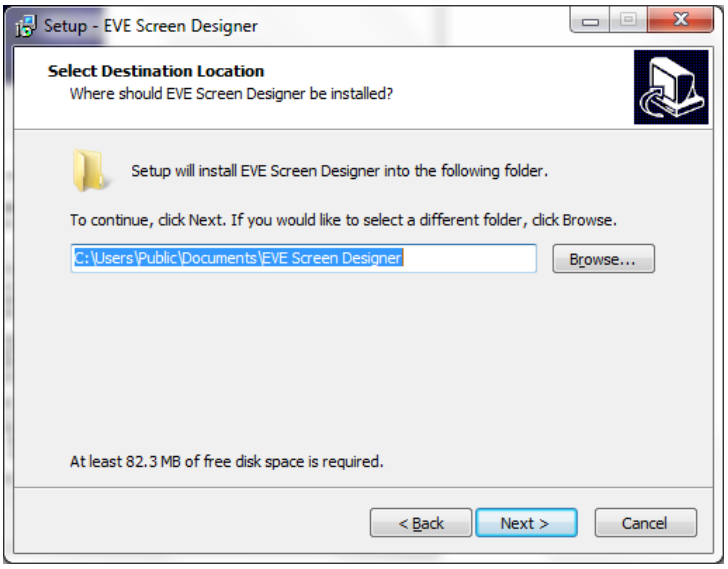
v. The EVE Screen Designer 3.0 Setup Wizard is displayed along with a Welcome message.



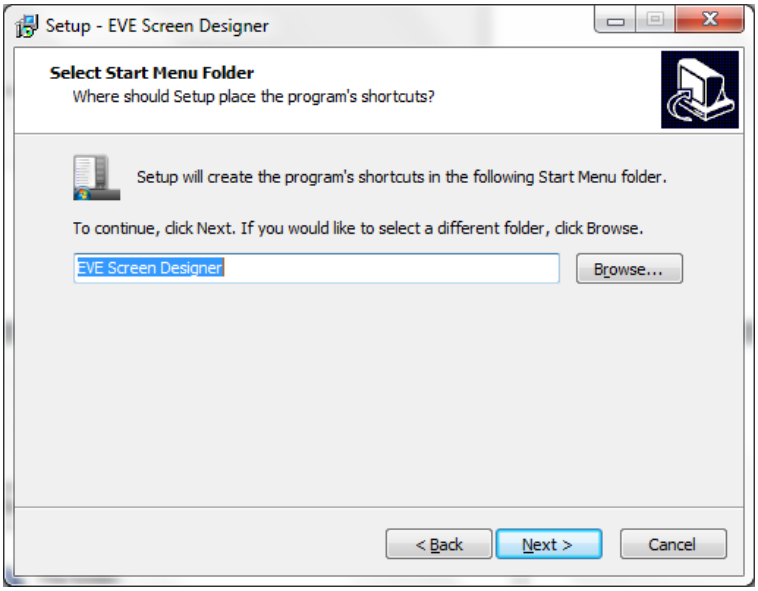
vi. Click **Next** to view the end user information window.



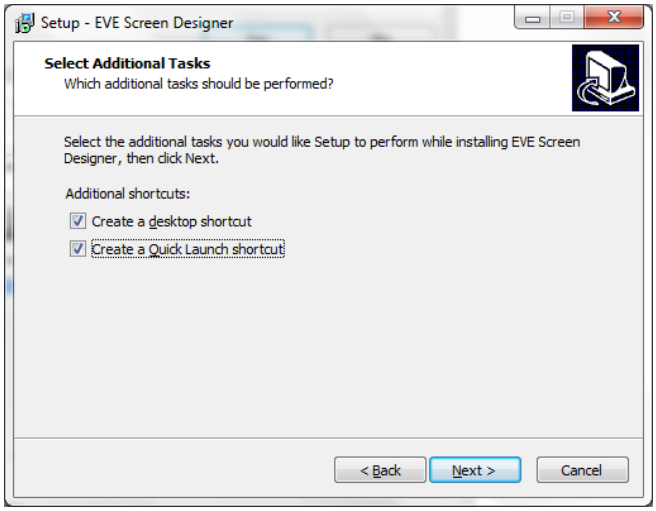
vii. Click **Next** to select a "Destination Folder" for installing the files. Accept the default folder or click **Browse** to specify a different location. Click **Next** to confirm the destination folder and continue.



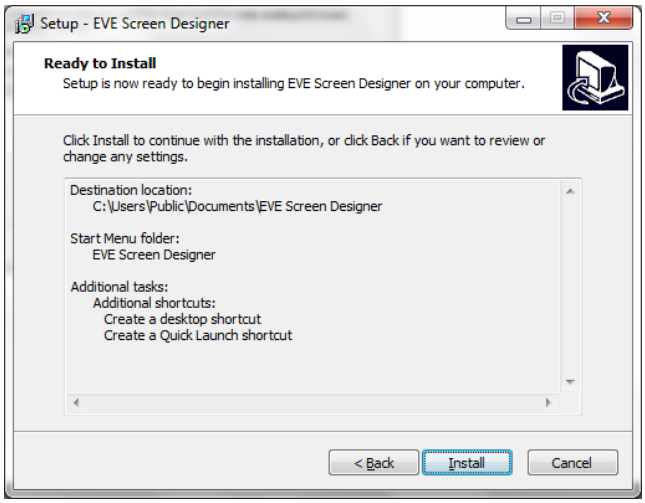
viii. Click **Next** to select a "folder" for creating the program shortcut. Accept the default folder or click **Browse**, to specify a different location. Click **Next** to confirm and continue.



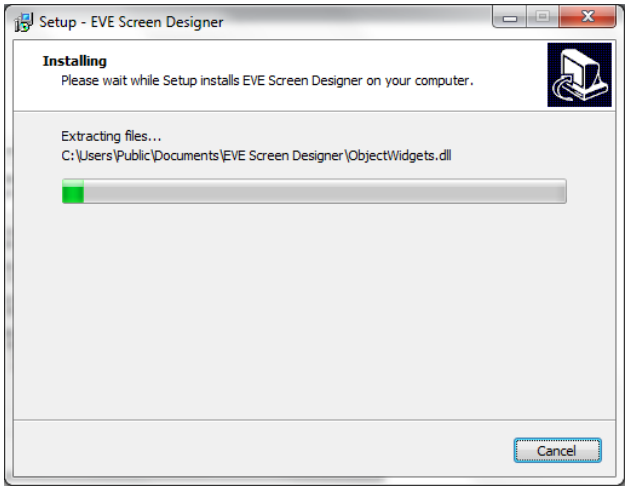
ix. In the **Select Additional Tasks** window, check **"Create a desktop / Create Quick Launch shortcut"** boxes, to have the ESD 3.0 icon and Quick Launch shortcut displayed on the desktop. Click **Next** to prepare for the installation.



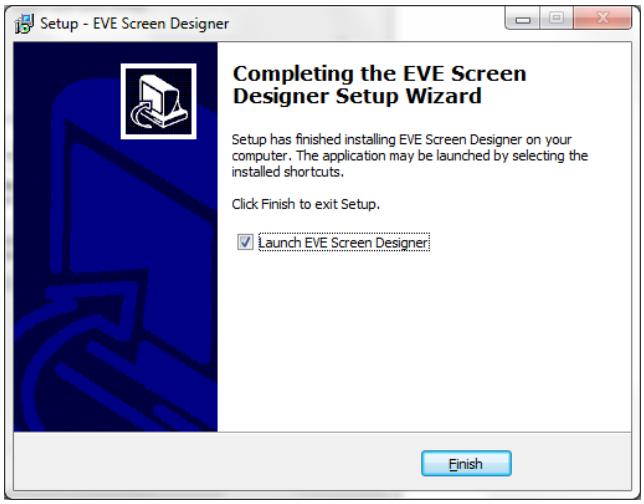
x. The initial setup is completed and the application is ready to be installed.



xi. Click **Install** to start the installation. A progress bar indicates that the installation is in progress.



xii. Upon successful installation, click **Finish**.



xiii. The ESD 3.0 application user interface is displayed.

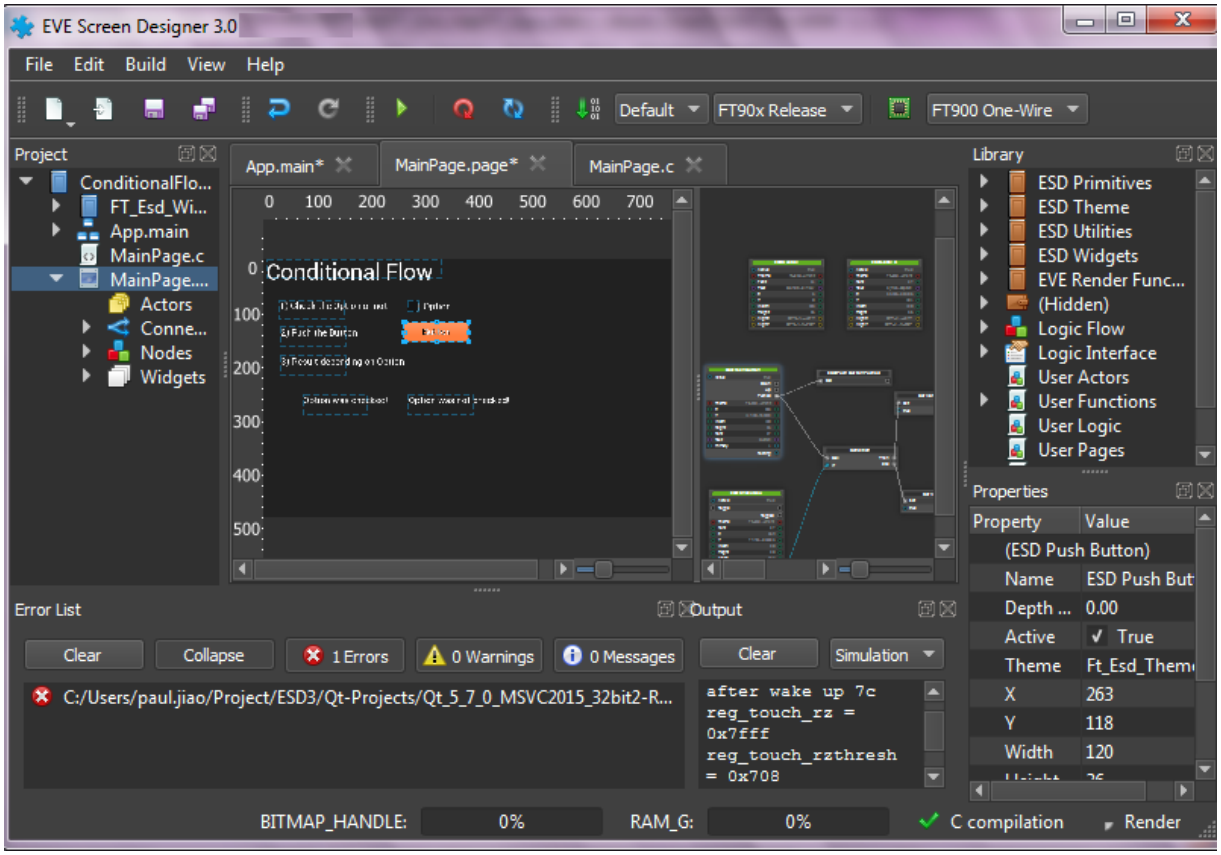


Figure 2 - ESD 3.0 User Interface

F. Installation Folder

Folder Name	Description
Examples	The example project created by ESD 3.0
Imageformats	Qt image format supporting DLL
Libraries	Widget library/application frame work/HAL
Manual	Contains the ESD 3.0 User Guide
Platforms	Qt platform run-time DLL
Settings	Configuration files for image converter and Tool Chain
TinyCC	TinyCC run-time as simulation engine

Table 1 - Installation Folder

IV Getting Started

A. The Graphical User Interface

ESD 3.0 user interface has the following components:

1 *Menu bar*
2 *Toolbar*
3 *Project Explorer*
4 *Screen Layout Editor*
5 *Logic Node Editor*
6 *Library browser*
7 *Error List*
8 *Output*
9 *Property Editor*
10 *Status bar*

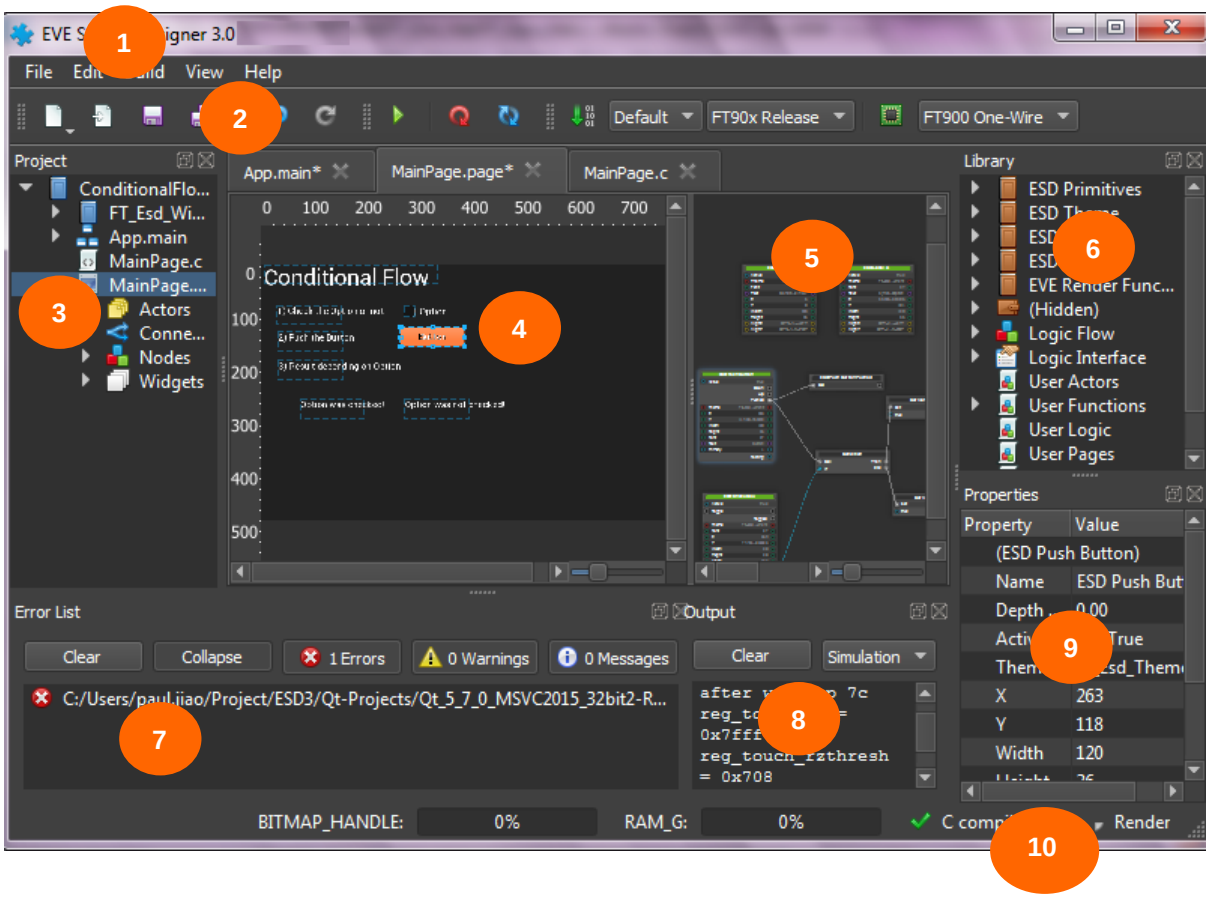


Figure 3 - EVE Screen Designer 3.0 User Interface

1 Menu bar

The ESD 3.0 *Menu bar* displays the headings for each drop-down menu. According to the function, the commands are grouped under each of these menu headings.

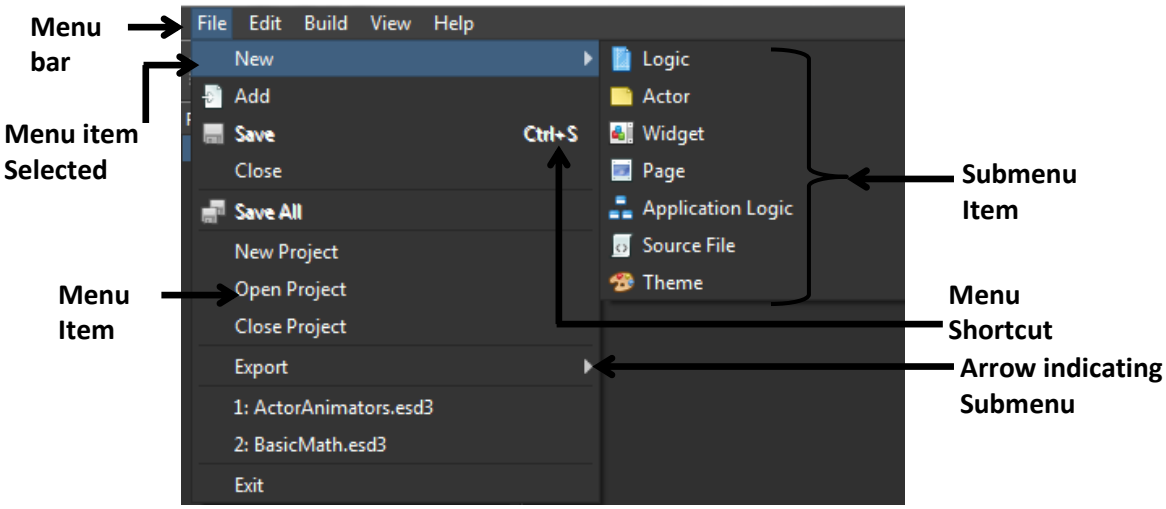


Figure 4 - Menu bar

The following table provides the list of Menu/Submenu and its description -

Menu	Submenu		Description
File	New	Logic	Creates new file for the existing project
		Actor	
		Widget	
		Page	
		Application Logic	
		Source File	
		Theme	
	Add		To add existing resource (such as bitmap, C source file etc.) to the project
	Save		To save the currently open file
	Close		To close the file that you are currently working on
	Save All		To save all the files in the current project
	New		To open a new project

	Project		
	Open Project		To open an existing project
	Close Project		To close a project
	Export	Eclipse Project	To export eclipse project file
		Export As…	To export all the files into a new folder
		Launch Eclipse	To launch the FT90x Eclipse IDE
	Exit		To close the ESD 3.0 application
Edit	Undo		To reverse the action of an recently performed action
	Redo		To revert the effects of the undo action
Build	Build Executable		To generate a executable file
	Build and Upload to Hardware		To generate a executable file and upload to hardware
	Browse to Executable		To navigate to the folder under which the executable file is located
View	Project		To display or hide a component view. By default all the components are displayed.
	Library		
	Properties		
	Error List		
	Output		
Help	User Guide		Opens the ESD 3.0 user guide
	About		Displays the ESD 3.0 version details
	Third Party		Displays the copyright, disclaimer and license information of the third party software

Table 2 - Menu & Description

2 Toolbar

The *Toolbar* provides an easy access to those functions (in the form of icons) such as new file, save file, undo, redo etc., that are commonly used.

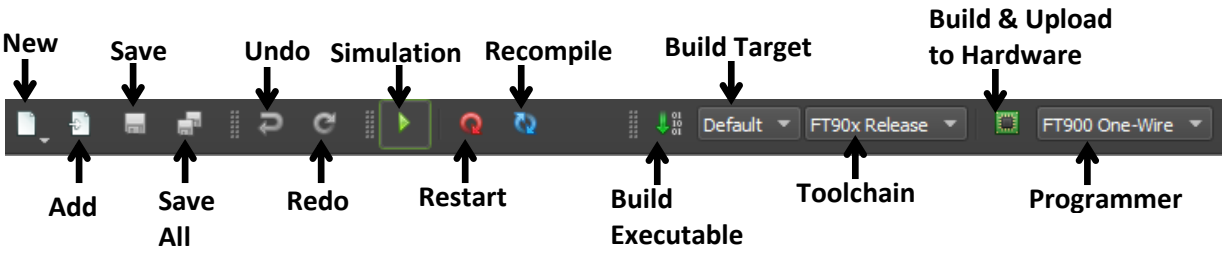


Figure 5 - Toolbar

The following table provides the list of toolbar functions and its description –

Toolbar Function	Description
New	To open new *Logic/Actor/Widget/Page/Application Logic/Source File/Theme*
Add	To add existing resource (such as bitmap, C source file etc.) to the project
Save	To save the currently open file
Save All	To save all the files in the current project
Undo	To reverse the action of a recently performed action
Redo	To revert the effects of the undo action
Simulation	A toggle button which starts or stops the simulation mode.  - This state indicates that the ESD 3.0 is in simulation mode. Clicking this button stops the simulation.  - This state indicates that the ESD 3.0 is out of simulation mode. Users can drag/drop widgets or edit them safely.
Restart	To automatically restart the EVE emulator. Clicking this button will force the simulated screens to be re-drawn.
Recompile	To recompile the whole project's source code.
Build Executable	To build the project and generate the executable file
Build Target	Currently not in use
Toolchain	Building configuration of the Toolchain

Build and Upload to Hardware	To build the project; generate executable file and upload it into the selected hardware (e.g. MM900EV2A)
Programmer	It is defaulted to FT900 One-Wire programmer.

Table 3 - Toolbar & Description

3 Project Explorer

The *Project Explorer* window organizes all the files used in the project in a tree view. It also lists out all the resources used by each file. Project explorer allows users to navigate each page of the project.

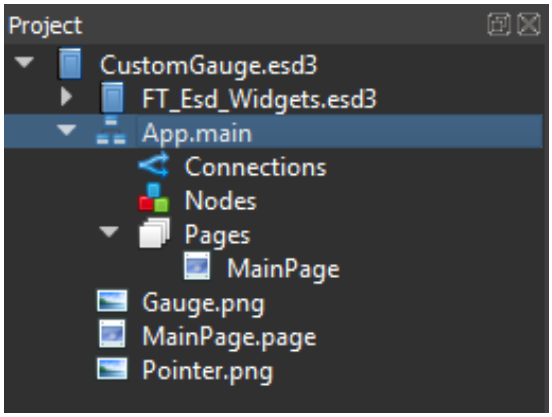


Figure 6 - Project Explorer window

4 Screen Layout Editor

The *Screen Layout Editor* component displays the rendering output of EVE and allows users to edit C source code. The page/widget/main files can be opened and edited in this editor.

Figure 7 - Screen Layout Editor – App.main

```
MainPage.c    App.main
1  #include "Ft_Esd.h"
2  #include "MainPage.h"
3
4  #include "Ft_Esd_Layout.h"
5  #include "Ft_Esd_PushButton.h"
6  #include "Ft_Esd_ScrollPanel.h"
7
8  #include <stdlib.h>
9
10 // These will probably be moved to something generic, but use this for now
11 ft_int16_t Ft_Esd_ScrollPanel_ChildX(Ft_Esd_ScrollPanel *context);
12 ft_int16_t Ft_Esd_ScrollPanel_ChildY(Ft_Esd_ScrollPanel *context);
13 ft_int16_t Ft_Esd_ScrollPanel_ChildWidth(Ft_Esd_ScrollPanel *context);
14 ft_int16_t Ft_Esd_ScrollPanel_ChildHeight(Ft_Esd_ScrollPanel *context);
15 void Ft_Esd_ScrollPanel_Remove(Ft_Esd_ScrollPanel *context, Ft_Esd_LayoutChild *child);
```

Figure 8 - Screen Layout Editor - MainPage.c (C Source Code)

It shares one view port with the logic node editor. Users can adjust the size of the screen layout editor by dragging the splitter handle.

Users can drag and drop the widgets from the library browser to form the layout when simulation mode is off.

5 Logic Node Editor

The *Logic Note Editor* allows users to layout logic nodes and to establish connections to create logic maps. Users can drag and drop a logic node from the Library Browser component to the Logic Note Editor in order to create connections.

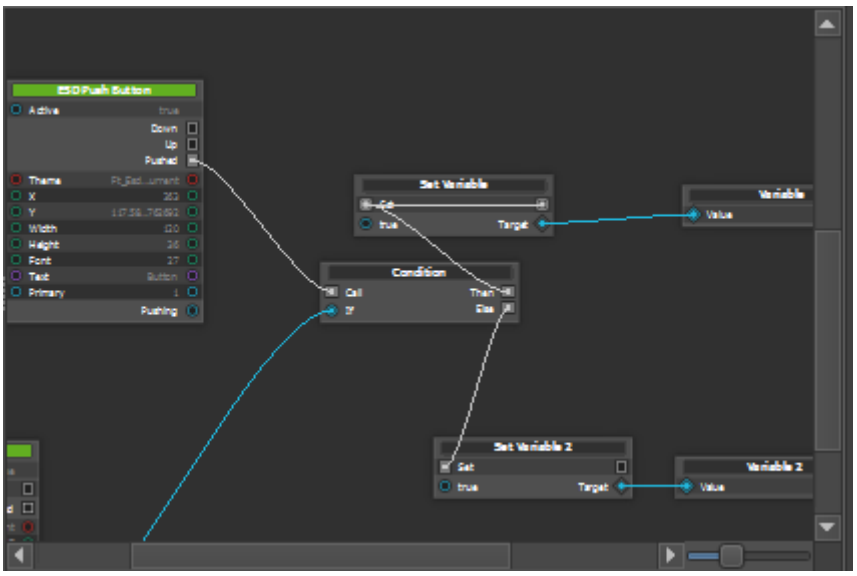


Figure 9 - Logic Node Editor

6 Library Browser

The *Library Browser* allows storing all the available logic nodes and resources in ESD 3.0 - that includes both built-in and user-defined ones. Users can view these logic nodes by category and select one for their project.

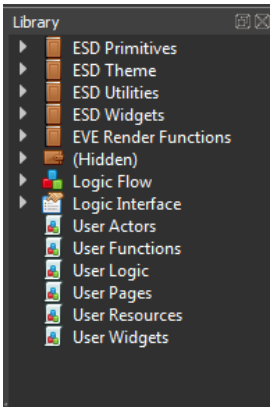


Figure 10 - Library Browser

The following table provides the ESD 3.0 libraries and its description –

Library Name	Description
ESD Primitives	ESD built-in basic widgets
ESD Theme	ESD built-in basic theme manipulation functions
ESD Utilities	ESD built-in utilities
ESD Widgets	ESD built-in widgets
ESD Render Functions	ESD built-in Render functions
Logic Flow	ESD built-in logic node for control flow
Logic Interface	ESD built-in logic node for interface
User Actors	User defined actor logic node
User Functions	User defined functions
User Logics	User defined logics
User Pages	Pages added by user
User Resources	Resources added by users (For example: bitmap)
User Widget	Widgets created and added by users

Table 4 - ESD 3.0 Libraries

7 Error List

The *Error List* is a docked window which shows the message output from ESD 3.0 while saving and recompiling the project files. Any error message displayed in this window indicates that the generated source code for the current project is unable to be executed successfully. Users need to double check the logic defined in *logic node editor* or the user-defined source code in the project.

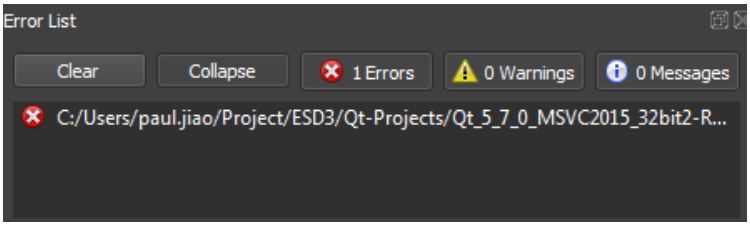


Figure 11 - Error List window

8 Output

The *Output* component is a docked window which shows the message output from the EVE emulator and Toolchain.

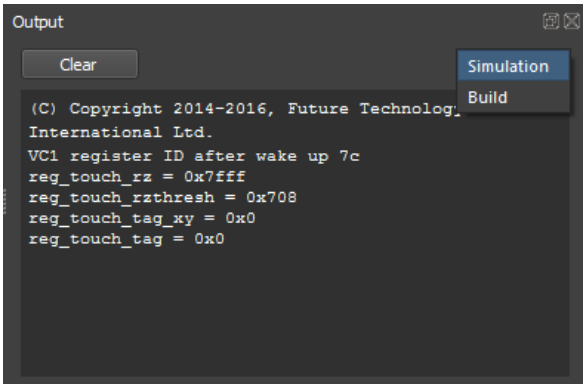


Figure 12 - Output window

To check a message from the EVE emulator, click **"Simulation"** from the drop down list.

To check a message from the EVE Toolchain while building generated source code, click **"Build"** from the drop down list.

This window is automatically updated during simulation or building.

9 Property Editor

The *Property Editor* allows user to edit the properties of selected logic nodes. The sample screenshot given below shows the property editor of an ESD Push button.

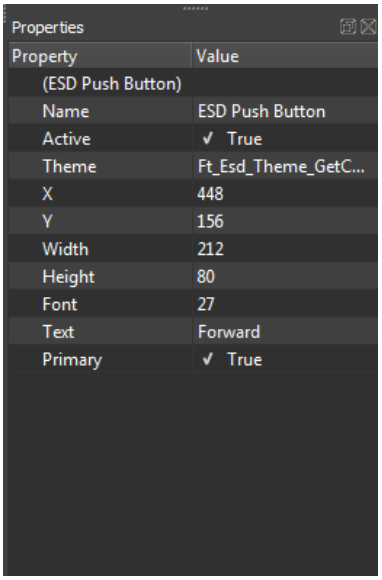


Figure 13 - Property Editor (ESD 3.0 Push Button)

10 Status bar

The *Status bar* is found at the bottom of the ESD 3.0 user interface. It primarily displays the current status of any of process or job that is being handled by the ESD 3.0 application. For example, if the user is performing a C compilation, then the compilation status of the generated C code is displayed.

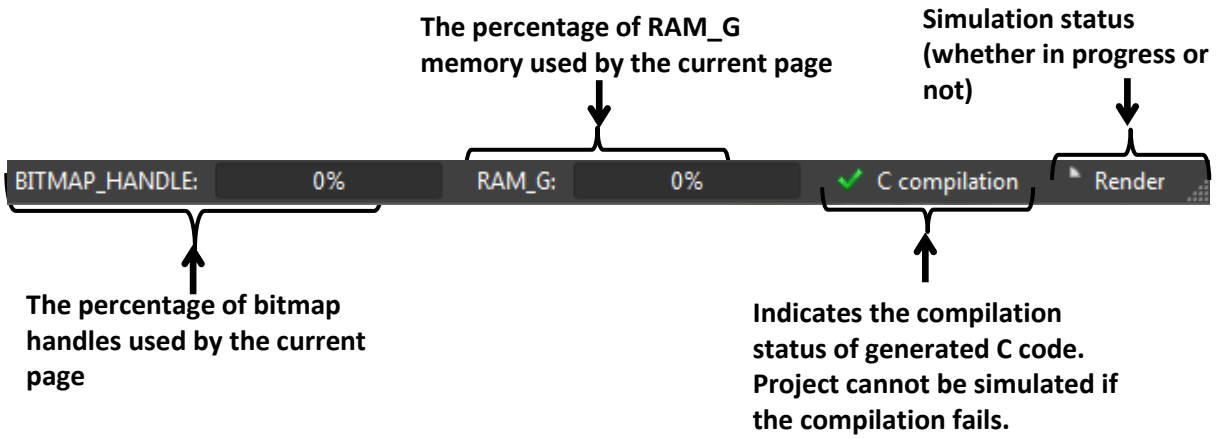


Figure 14 - Status bar

B. Application Project Structure

An Application consists of one or more pages. Each page in turn may contain one or more widgets. Widgets and pages are connected via connection lines so that screen logic is defined.

The application model follows a strict hierarchical structure, where all the pages are owned and managed by the application, and all the widgets are owned by their respective pages. The visibility and memory lifetime of the widgets are thus managed on a page-by-page basis.

By design, all widgets and pages are generated to be entirely modular and self-sufficient. Any interaction between widgets and pages is therefore required to be routed through the hierarchical chain. (Only node connections exist between directly connected nodes in the hierarchy.)

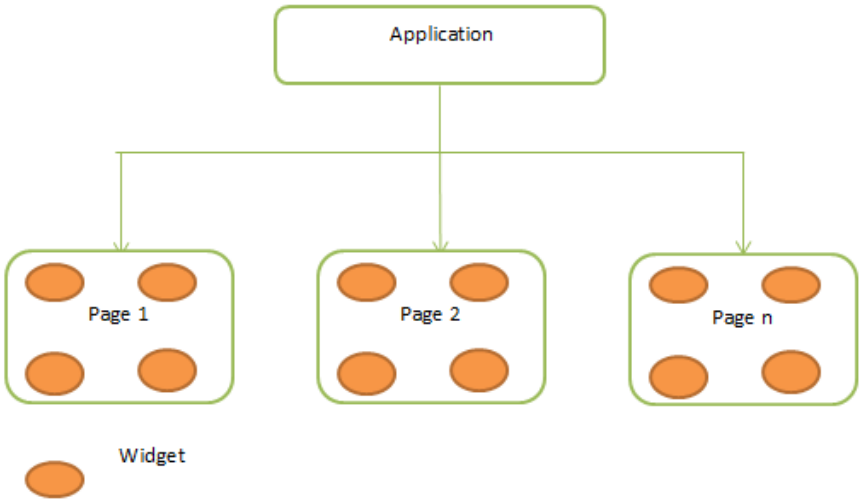


Figure 15 - Application Project Structure

File Name	Description
*.esd3	*Project* file - This XML file describes what files are included in this project. This file is unique for the whole project.
*.main	*Logic* file - This XML file describes the application level logic. By default, it is named as "app.main" and is unique project wide.
*.page	*Page Node Definition* file - This XML file describes what widgets are contained and connected in each page.
*.widget	*Widget Node Definition* file – This XML file describes how a widget is defined.
*.actor	*Actor Node Definition* file – This XML file describes how the actor nodes are defined.
*.logic	*Logic Object Definition* file – This XML file describes how the logic is defined.

C. Application Workflow

The workflow given below illustrates how to create an EVE based GUI application using ESD 3.0.

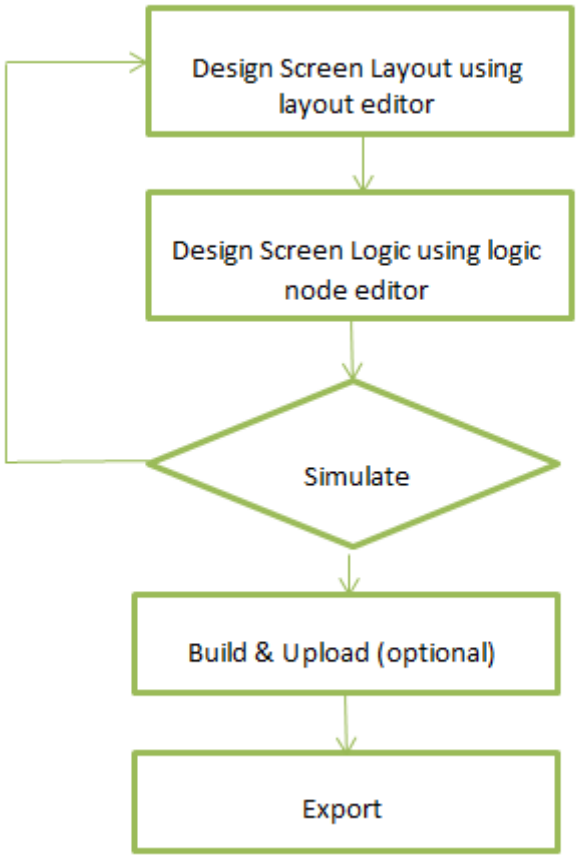


Figure 16 - Application Workflow

Design Screen Layout using Layout Editor

Users need to determine how many pages are included in the project and what widgets are to be contained in each page. After a page is created, users can use drag and drop the widgets to design the appearance of the page in the screen layout editor. This process determines how many screens are shown in the project and what they look like together.

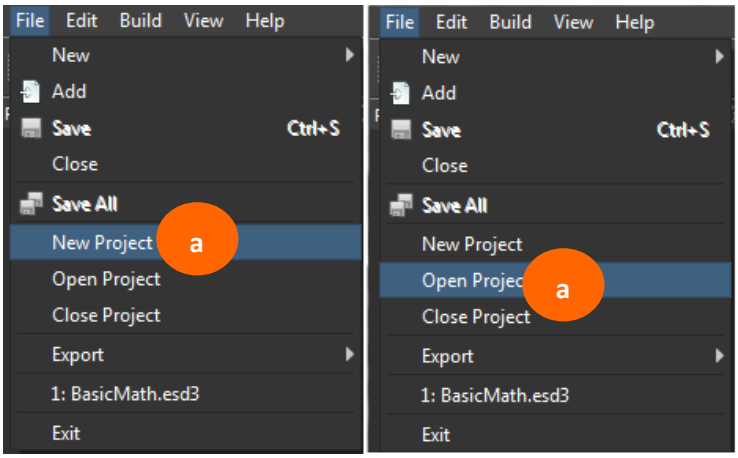
Custom Widget

ESD 3.0 provides a set of built-in widgets for users to construct screen layouts. Additionally, if the built-in widgets do not meet the design requirement, users may define and design custom widgets.

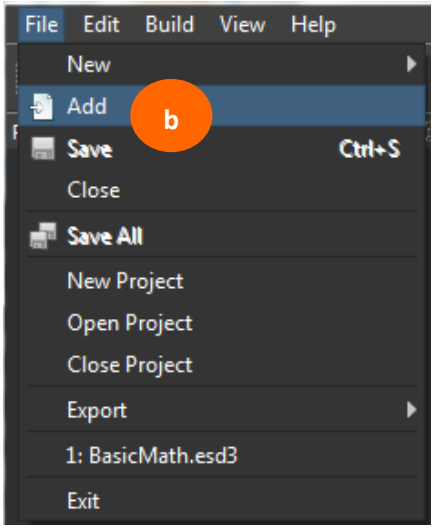
Add Bitmap Resource

To display a bitmap on the page, users need to add a bitmap resource into the project. The following steps will provide guidance on how to add a bitmap resource:

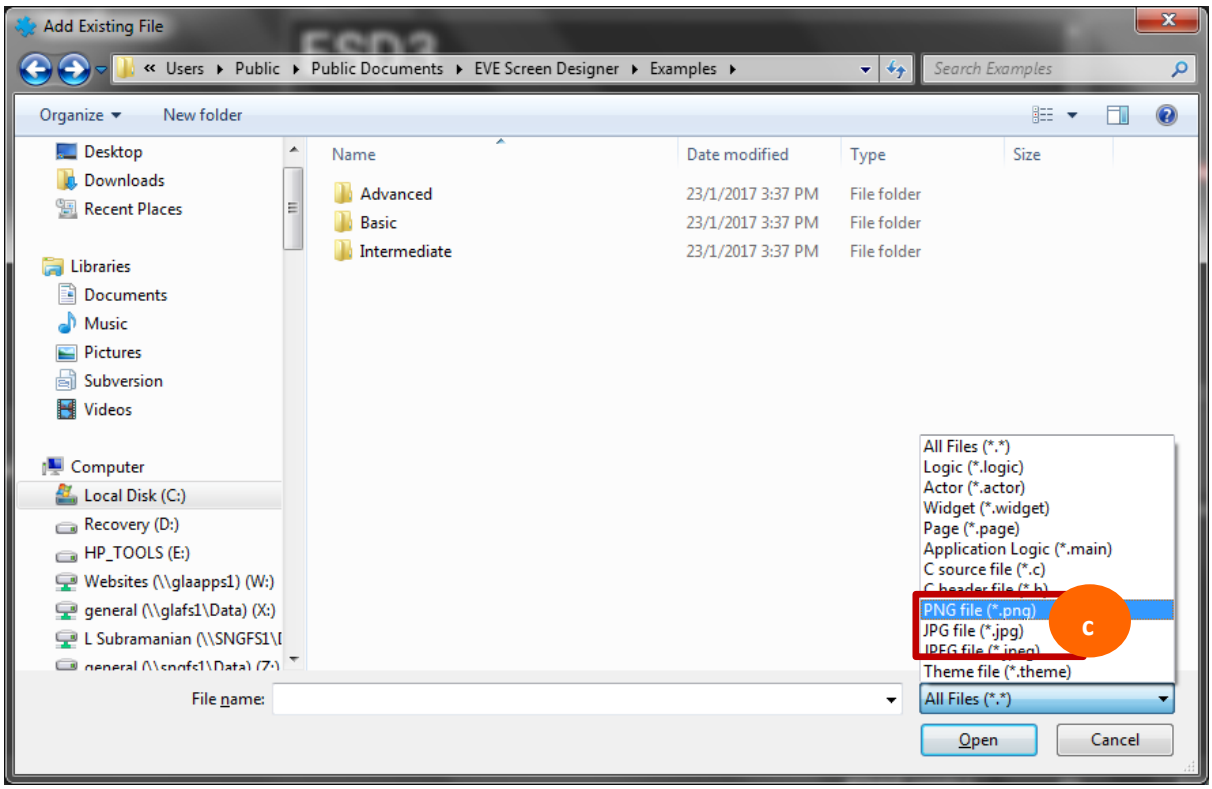
a. Create a New Project (*File → New Project*) or Open an existing Project *(File → Open Project)*.



b. Click *File → Add → Bitmap* from the menu.



c. Browse and select the image file(s) from the file explorer window.



Upon adding the file, users can drag and drop the ESD Bitmap widget to the desired page and select the bitmap cell accordingly.

Configure Bitmap Resource

Users can configure a bitmap resource via the Property Editor. Select the image file from the project browser and check the Property Editor.

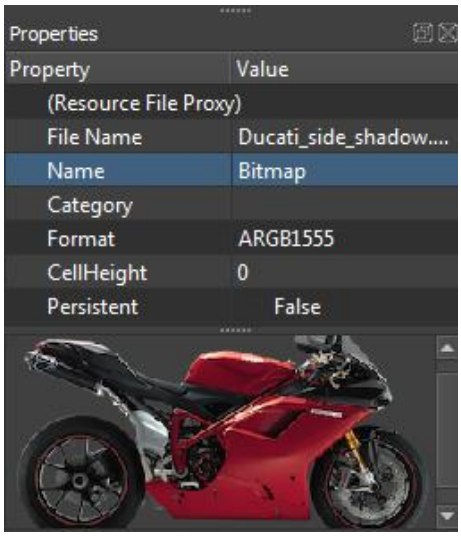


Figure 17 - Property Editor

Property	Value/Description
File Name	Name of the image file
Name	Resource Type Name. In this case, the selected resource type is Bitmap.
Category	Category of the selected object
Format	Image Format – The target format of bitmap can be – • L1/L2/L4/L8 • RGB332/RGB565 • ARGB2/ARGB4 • PALETTED4444/PALETTED565
CellHeight	The pixel number of one cell. For example, if the bitmap width is 240 and height is 240, then it can be defined as two cells with "CellHeight" - 120 or four cells with "CellHeight" – 60.
Persistent	Keeps the memory persistent even if the bitmap is not displayed when it is true.

Table 5 - Bitmap Resource Properties

Design Screen Logic using Logic Node Editor

Users can define the dynamic behaviour of an application in this phase. The Logic Node Editor employs the innovative visual programming idea to help users define logic flows without coding.

During this phase, users can decide the behaviour mode of both inner- and inter-pages. For inner-page behaviour, users can connect the widgets on the page via the logic node editor, adding more logic nodes from the library browser, if necessary.

Inter-page logic behaviour is captured in the application logic, which is normally named as "app.main". Users need to drag and drop the predefined pages into the logic node editor and connect these pages via the logic interface. This will define the inter-pages behaviour like screen logic.

Simulate

To validate the behaviour of the screen logic, users may need to simulate the project on the PC before compiling and downloading the generated C code into the target device. When application logic is selected, users can simulate the whole project by clicking the **"Simulation"** button on the toolbar.

When the application is in simulation mode, users cannot drag and drop the widgets into the layout editor. The PC mouse will act as a touch stylus on the touch panel to interact with application directly.

Before performing a simulation, it is strongly recommended to save the current project by clicking the **"Save all"** button. Once the simulation is completed, ensure to switch off the simulation mode by clicking the **"Simulation"** button again.

Simulation State	Description
	ON State - Indicates that the simulation is in progress
	OFF State - Indicates that currently simulation is not in progress

Build & Upload

Upon installing the FT90x Toolchain, ESD 3.0 will invoke its compiler and uploader so that the current project can be built and uploaded successfully into the target hardware.

Click on the **"Build Target"** button to build a project

Click on the **"Build and Upload to Hardware"** button to build the program and upload it to the target device

Export

Upon completing the screen design, it's time to move the project to the next phase of the workflow – **"Export"**. The export function is used to prepare all the necessary files for the MCU tool chain so that users can make further enhancements such as connecting sensors or external hardware to a full HMI solution.

Once the export is completed, ESD 3.0 copies all the generated "C" source code and necessary bitmap resources into a user defined folder. In addition, the eclipse project file *".cproject"* and *".project"* files are also generated so that the users can open the project in the FT90X eclipse IDE[2]. All the generated files are named as "*_generated.c".

[2] http://brtchip.com/ft90x-toolchain/

Exported Folder Structure

The following table provides the list folders and files that are created upon exporting the project –

Folder / Files	Description
.cproject	Eclipse project file
.project	Eclipse project file
FT_Esd_Framework	Contains the application framework files. **It is recommended not to make any changes to this folder/ files**
FT_Esd_Widgets	Contains the widget files **It is recommended not to make any changes to this folder/ files**
FT_Esd_Hal	Contains the hardware abstraction layer files. These files may be changed to support a different MCU.
Data	Contains the compressed bitmap data in *.bin format.
$(ProjectName)	Contains the application specific source code

Bitmap Resource

If a *bitmap resource* is used in the project, users need to download the converted bitmap resource into an SD card root directory and insert the SD card into the hardware module.

The converted bitmap resource is in compressed .bin file format and is located at:

$(ProjectFolder)\build\default\data

Or

$(ExportFolder)\data

The decompression is done by using the EVE engine command *CMD_INFLATE*. As per the Hardware Platform requirements, the SD card must be formatted as a FAT32 file system.

D. Screen Layout Editor

The *Screen Layout Editor* provides the facility to preview pages as well as the whole project. ESD 3.0 employs the EVE emulator to render an EVE display onto the screen layout editor. It provides users the interface to view the result of the screen design and its logic.

The following file types are displayed in screen layout editor:

- *Page file* - generally with ".page" extension
- *Application logic file* - generally with ".main" extension
- *Logic node file* - generally with ".logic" extension
- *C source file* - generally with ".c" extension

Page File

Page stands for a single screen which is rendered by ESD 3.0. One application consists of at least one page. Page has its own life cycle and manages all the widgets on it. Only widgets can be dragged and dropped into pages, however, all the nodes from a library can be dragged and dropped into the logic node editor.

"Active" Property

The *"Active" property* is a Boolean value to control the page, and is created or released while the application is updating the pages. By default, the "Active" property is true. The page is created and shown, unless "Switch" end is connected.

Page Switch

Users can define multiple pages and show at runtime by connecting the "Switch" end to another signal end. For example, in the application logic, i.e. "App.main", the "Start" slot is required to connect with the "Switch" property. Refer to the below sample picture.

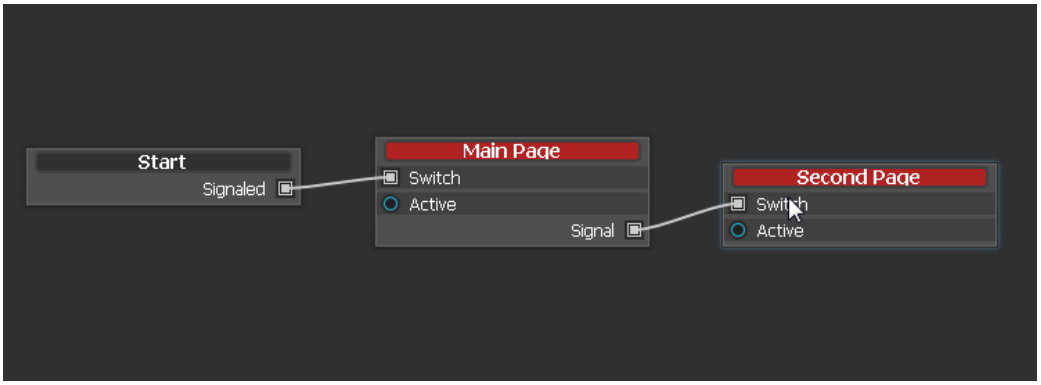


Figure 18 - Sample Page Switch

Page Persistence

To allow a page to remain in memory with its current state when it is made inactive, enable the *Persistence checkbox* in the page's Properties from the application logic. This feature can help users to reserve the information of a page even when it is not visible.

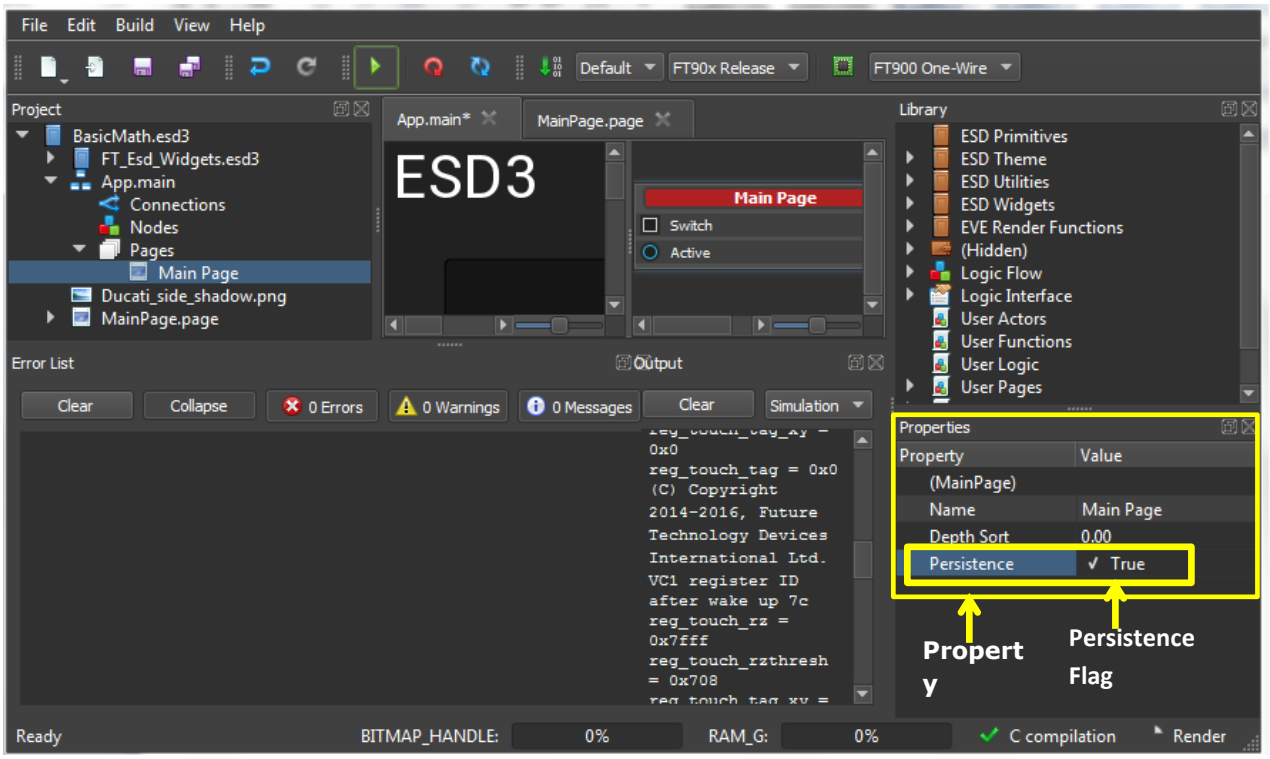


Figure 19 - Properties Editor - Persistence Flag

User Defined Function for Page File

Users can write their own code to handle the "Pushed" signal of the "ESD Push Button" widget. Users need to select the "ESD Push Button" widget in the logic node editor and double click it. A function will be generated to replace the user defined function.

```
ESD_METHOD(MainPage_ESD_Push_Button_Pushed, Context = MainPage)
void MainPage_ESD_Push_Button_Pushed(MainPage *context)
{
        // Users can write their code here
}
```

The code will be located at ***$(PageFileName).c*** file.

```
MainPage.page*   App.main   MainPage.c

#include "Ft_Esd.h"
#include "MainPage.h"

ESD_METHOD(MainPage_ESD_Push_Button_Pushed, Context = MainPage)
void MainPage_ESD_Push_Button_Pushed(MainPage *context)
{
    // …
    printf("This push button is pushed\n");
}
```

Figure 20 - Sample Source File

Zoom In & Out

The visual area of the screen layout editor can be zoomed when the current file is a page file, i.e., *".page"* file. The mouse wheel button provides *zoom-in* and *zoom-out* functionality, allowing users to view more details about the screen design. Refer to the below sample picture.

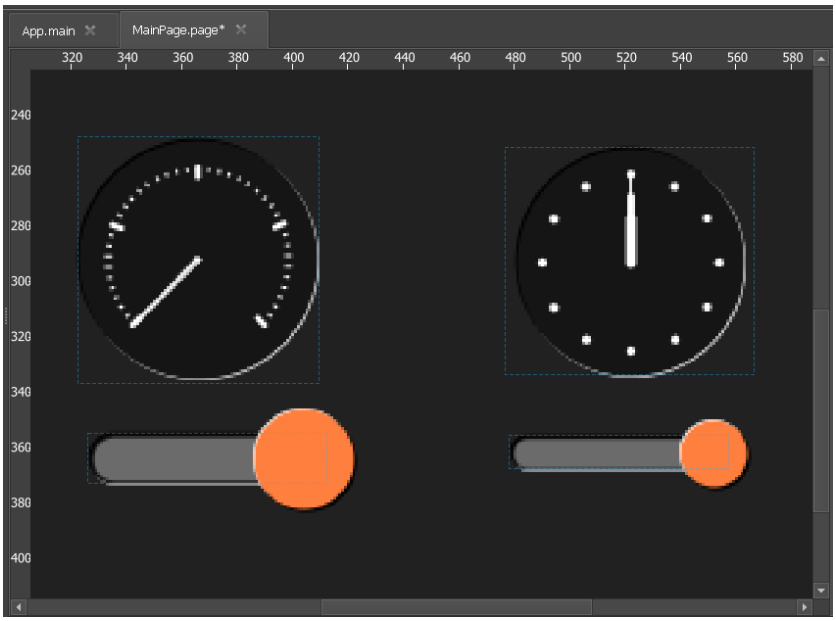


Figure 21 - Zoom In & Out

Main File

To view the application logic file (*.main) file, double click the file within the project explorer. The screen layout editor will show the first page defined in the logic node editor. To view how the application logic is defined, click the **"Simulation"** button and check it in the screen layout editor.

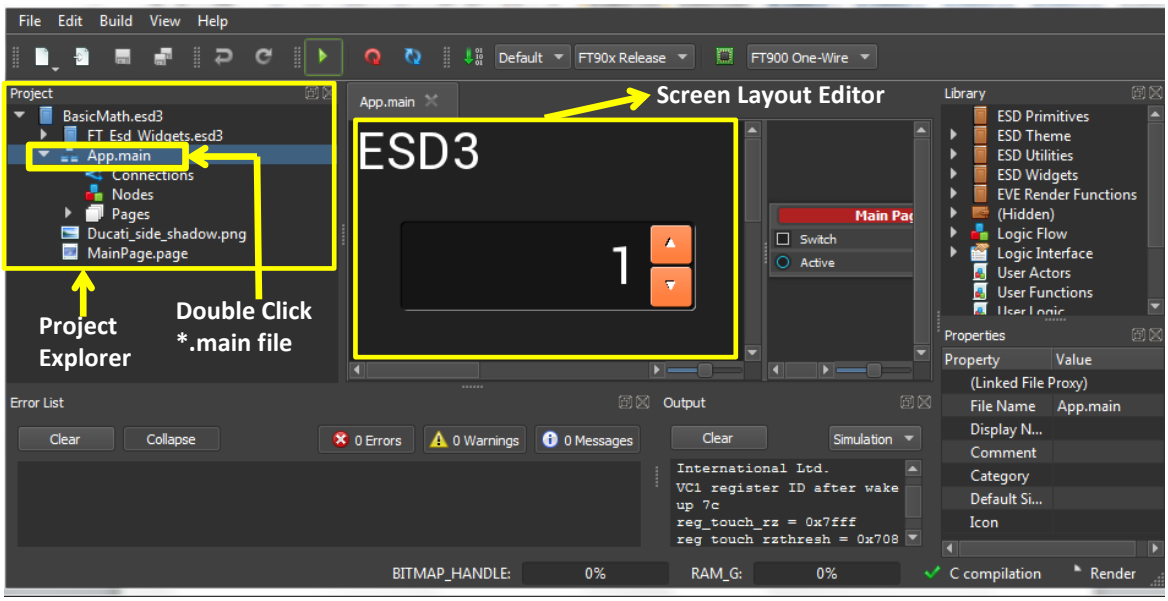


Figure 22 - Main File

Logic File

The logic file contains certain logic that is used to encapsulate the user defined logic. It works similar to a C function. A logic file does not render any user interface.

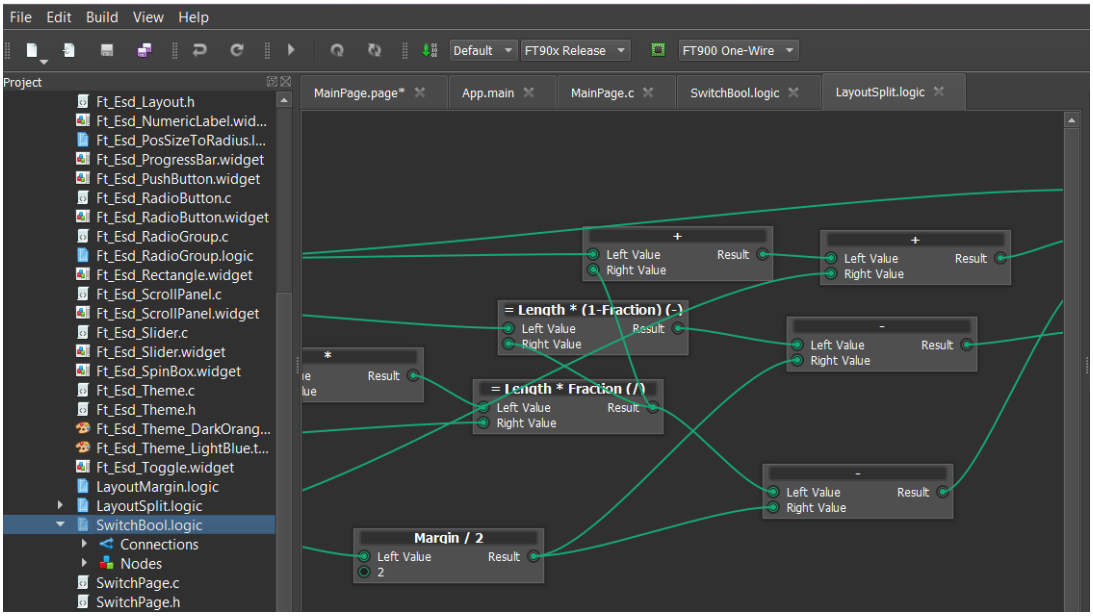


Figure 23 - Sample Logic File

C File

The screen layout editor provides a simple C editor when users double click a ".c" file in the project browser. Note that it is not a full-fledged C editor and many features may be implemented in the future.

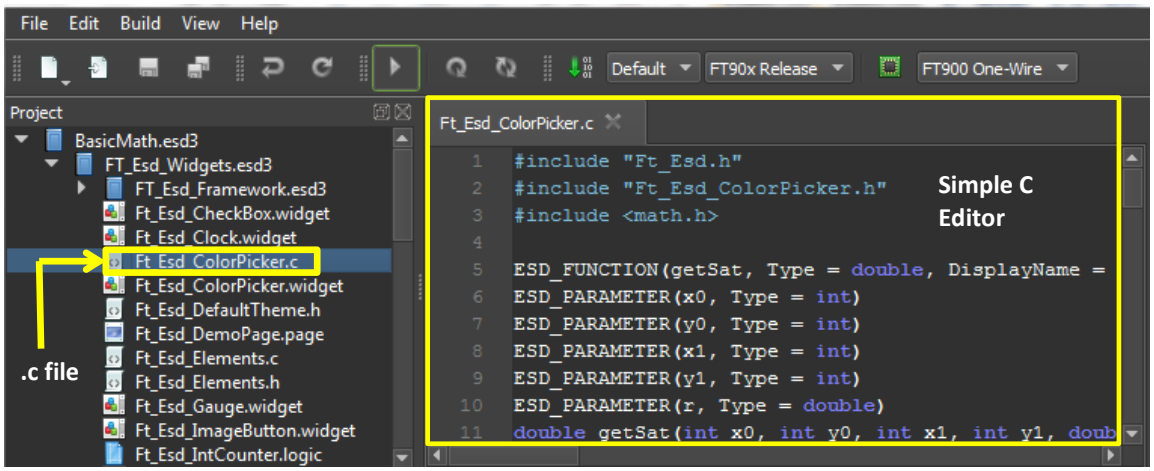


Figure 24 - C File / C Editor

To effect the changes made, ensure that all the changes are saved to the project by clicking **"File → Save All"** from the menu or from the toolbar.

E. Logic Note Editor

Using the *Logic Node Editor* (LNE), users compose screen logic by connecting logic nodes. The Logic Note Editor is designed to open the following file formats:

- Application logic file , *.main file
- Screen logic file, *.page file
- Widget file, *.widget
- Actor file, *.actor
- User defined logic file, *.logic file

Basic Logic Node

ESD 3.0 has some built-in predefined *basic logic nodes*. These basic logic nodes provide basic control flow and logic interfaces as well as basic functions. A typical basic logic node is represented in the logic node editor as below, the name of which is shown in white.

Figure 25 – Logic Node Editor - Basic Logic Node

Within the library browser, the basic logic nodes are located at *"Logic Flow"*, *"Logic Interface"*, *"ESD Utilities"* and *"EVE Render Functions"*.

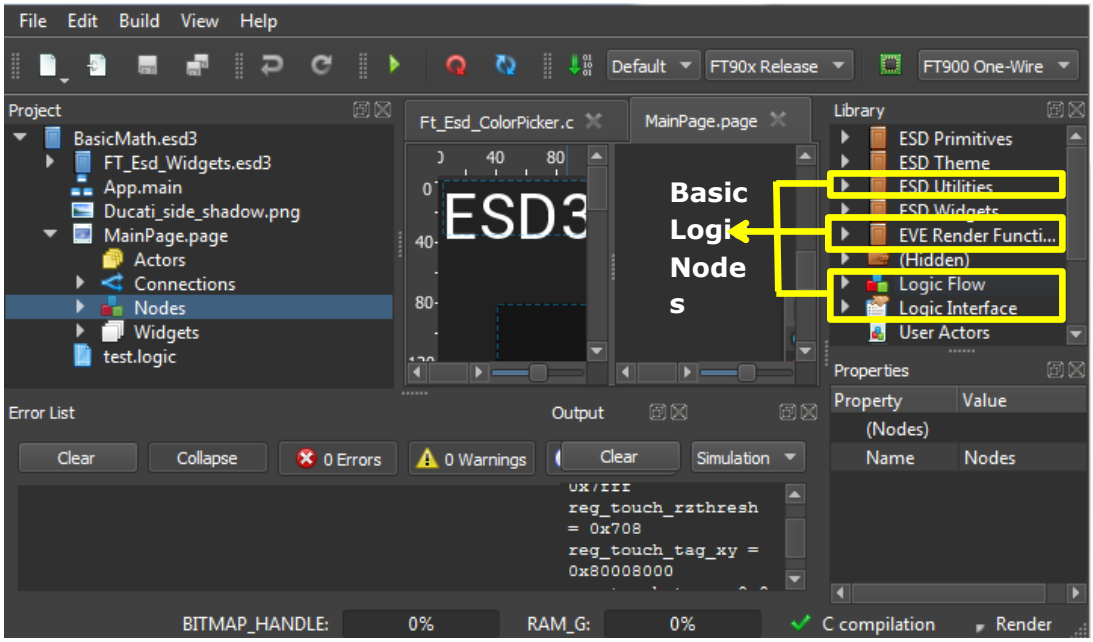


Figure 26 - Library Browser - Basic Logic Nodes

Composite Logic Node

A *Composite Logic Node* is made up by connecting multiple basic logic nodes. A typical composite logic node is defined in a standalone document in XML format and is shown in the logic node. A sample picture is given below –

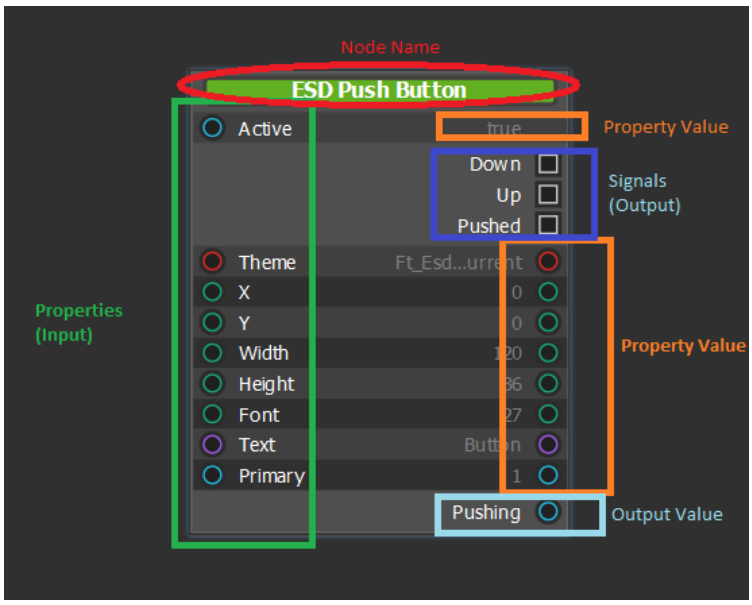


Figure 27 - Composite Logic Node

The following are the different types of composite logic nodes defined in ESD 3.0 –

- *Page Node*
- *Widget Node*
- *Actor Node*
- *Logic Object*

Page Node

When users create a single page, it is represented by a logic node under the "User Page" category in the library browser. Users can connect one page node to another within the logic node editor. It has its own property and defined output.

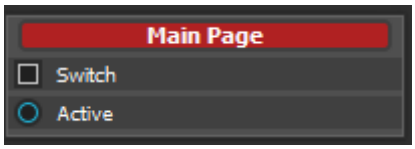


Figure 28 - Composite Logic Node - Page Node

Widget Node

The *Widget Node* is one type of logic node which has visual appearance and is able to handle user input. It has the built-in *Start*, *Update*, *Idle*, *Render* and *End slots*.

For more information on built-in slots, please refer to Built-in Slot.

Within the logic node editor, the widget name is highlighted with the green color as shown in the sample picture below -

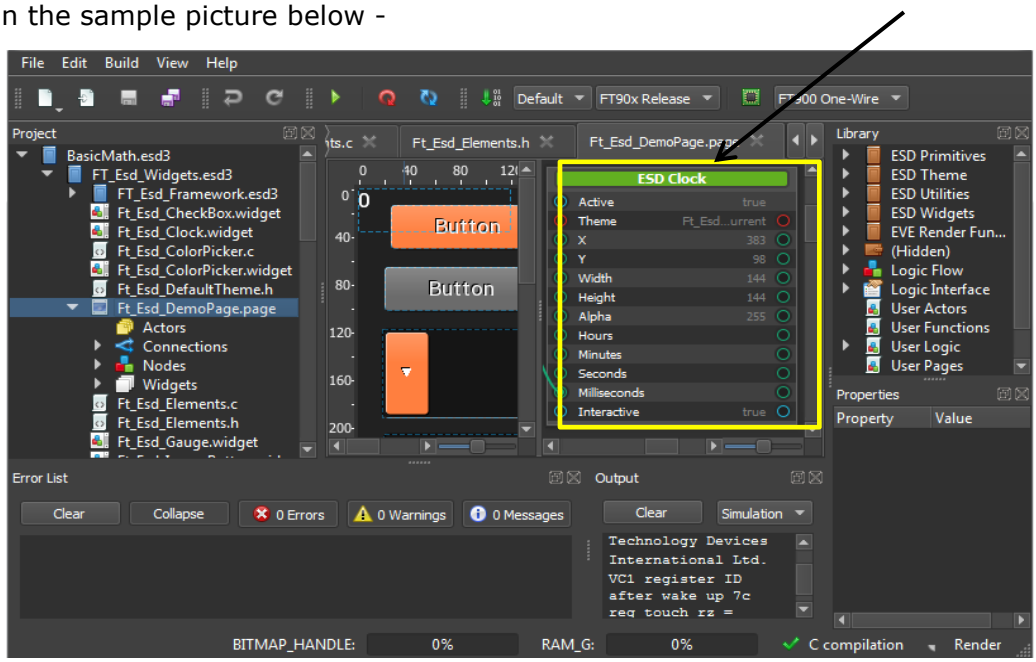
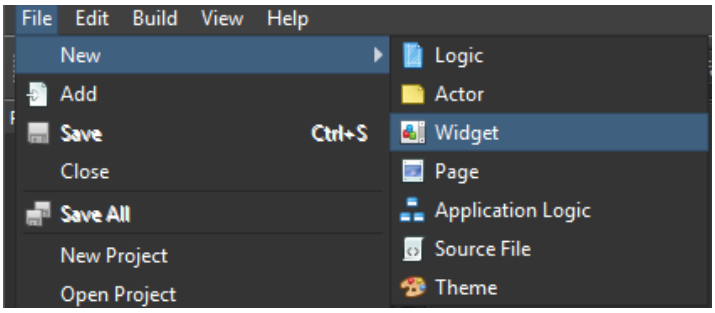


Figure 29 - Logic Node Editor – Widget Node

The following sections provide information on adding user widgets; the position and size properties; rendering the widget and widget theme.

Adding User Widgets

A *User Widget*, also called as *custom widget* is a reusable user interface element created by a user. To create a widget, from the menu, click and select **"File → New → Widget"**.



In order to make use of these slots, users need to drag the slots from the library browser and drop them into the logic node editor. By renaming it to the slot defined above, users can connect widget control flow to application control flow.

Position & Size Properties

As a user interface element, widgets have at least position and size properties so that users can control them using a visible handle. Position property is defined by an "Input" node with "ft_int16_t" type named as "X" or "Y". Size property is defined by an "Input" node with "ft_int16_t" type named as "Width" or "Height". A sample picture of the "X" property is given below –

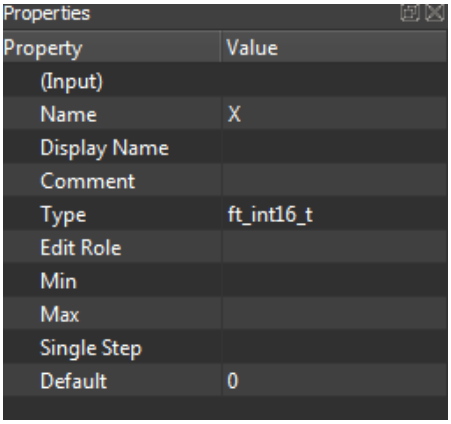


Figure 30 - User Widget - Position & Size Properties

Rendering the widget

Rendering of a widget may be implemented fully in the logic editor, or directly in the code using the logic editor to glue together the code with the generated structures. To create the render function, simply add a *Slot* from the Logic Interface category of the library browser, name it Render, and double click to create the user method for handling the rendering. The widget is expected to restore any graphics state it changes, with the exception of the state values specified below.

Preferably use the Ft_Esd_Dl_ utility functions for displaying list drawing when available. Several state parameters which may be expected to be changed often to the same value implement de-duplication here to shorten the display list. The de-duplicated values are: *Palette source, Color RGB, Color A, Handle, Cell,* and *Vertex Format*. These graphic state values are not required to be restored to their original values when the Ft_Esd_Dl_ functions are used. These values should be expected to be any undefined value at the start of the Rendering function.

Access the value of the X input property as given below:

```
int x = context->X(context->Parent);
```

This calls the `get()` function assigned by the parent to the X property, passing the parent's context to the function to handle it there.

Theme

Widgets should provide property input to override the default application theme. An input property with type *Ft_Esd_Theme* * should be added by dragging one input node from the "Logic Interface" category of the library browser.

Colours from the theme may be accessed from the logic editor using the utility functions under the ESD Theme category, or from the user code as given below –

```
Ft_Esd_Theme *theme = context->Theme(context->Parent);
if (theme == NULL) theme = Ft_Esd_Theme_GetCurrent();
ft_rgb32_t primaryColor = theme->PrimaryColor;
```

Touch Input

In order to handle the *touch input*, add a Touch Tag node from *"ESD Utilities"* category of the library browser to the logic editor, and either handle the signals or access its state through the Touch Tag interface. Users need to make use of the "Tag" value provided by the "Touch Tag" node during the Render function, and restore the "Tag" to value 0 after the relevant portion of the rendering.

To handle any of the signals from Touch Tag in user code, simply create an ESD_METHOD style function in the same format as the Render user code function, and it will be available from the User Functions category in the library to drag into the logic editor and connect to any of the signals from the Touch Tag node.

Actor Node

Actor type is an extended type of logic node which functions similarly to the widget node, but without the *Render* slot. Therefore, it has no visual appearance. When users create a new actor node, it does not show in the layout editor. It has the built-in *Start, Update, Idle* and *End* slots which are not available in a regular logic node.

For more information about built-in slots, please refer to Built-in Slot.

An Actor node is useful for implementing both continuous and background running behaviours, such as animations, timers, and input data polling. The actor node name is highlighted with the yellow color within the logic node editor.

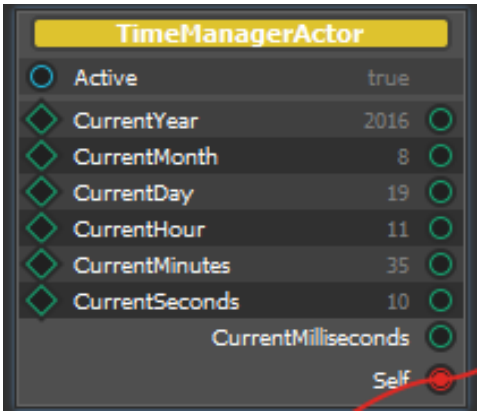
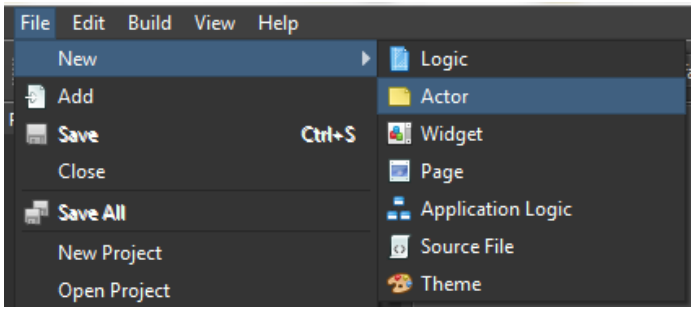


Figure 31 - Actor Node

To create an actor, from the menu, click and select **"File → New → Actor"**.



Refer to the example project **"AlarmClock.esd3"** under the **"EVE Screen Designer → Examples → Advanced → Alarm Clock"** folder for adding and using actor node.

Logic Object

A *Logic Object* is a basic logic block designed to express certain logic, which has NO built-in slot. Hence, it is not possible to invoke it through the built-in render slot or update slot. However, it may have its own private or public property and output, signal and slot, depending on the users' implementation.

A logic object is generally used to simplify complex logic by splitting it into multiple smaller and simpler logic objects. It is defined and saved in a "*.logic" file.

Within the logic node editor, the logic object name is highlighted with the blue colour.

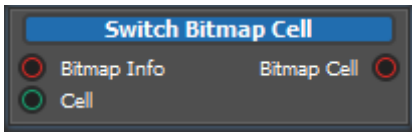


Figure 32 - Logic Object

To add a logic object, from the menu, click and select **"File → New → Logic"**.

Connections

Connections are lines between two nodes. The connection line's colour is determined by the ends. The following table provides the different types of ends and their description.

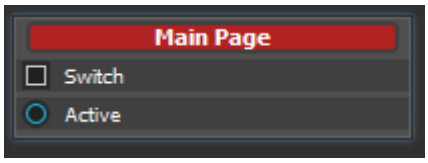
Connection Line End Type	Description
Event slots and Signals	Event slots and signals are implemented as functions which only take a pointer to the node context as a parameter. The node context contains the state of the node as well as a pointer to the parent node to signal back.
Data property bindings / /	Data property input and output bindings are implemented as get-functions which only take a pointer to the node context as a parameter and return a value. This means that these connections are completely evaluated every time the property is read, and consequently the property always reads as the latest value. A buffer variable can be used to cache values if necessary. The different colours denote the different data types.
Variable write	The write interface is translated into a function pointer during the code generation. Variables are plain variables, which can be written using the Set Variable node. They may also be set through the Writer interface which is implemented using a pointer to a set-function.

Table 6 - Connection End Types

In general, all of the input properties that can be connected can also be specified from the property editor when they are constant values; this is sometimes referred to as the "Default" value. In addition, it is possible to specify the Minimum and Maximum values for the input bindings and variables.

Rendering Order

In the application logic file "app.main", the pages added on the logic node editor are to be represented with the symbols shown in the sample picture.



When there are more than two pages, the order in which the pages were created, determines the rendering order.

The following implicit rule of the logic node editor is worthy to be noted, since this rule applies to all the logic nodes.

"Depth Sort" is used to determine the rendering order of the logic nodes. The greater the value of "Depth Sort", the later the logic node is rendered and higher the layer in which it is shown. When logic nodes have the same "Depth Sort" value, the logic node's "Y" coordinate (within logic node editor) determines the rendering order. The page node with the lowest value of the "Y" coordinate is rendered first.

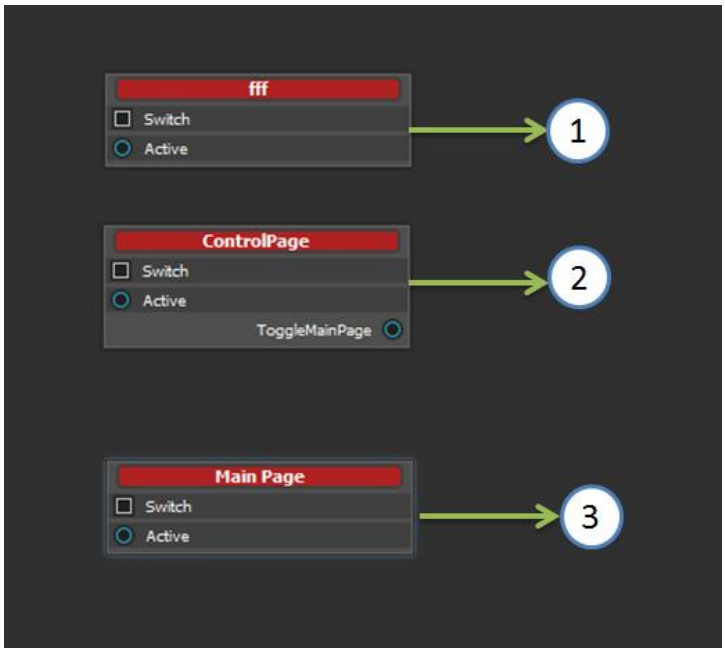


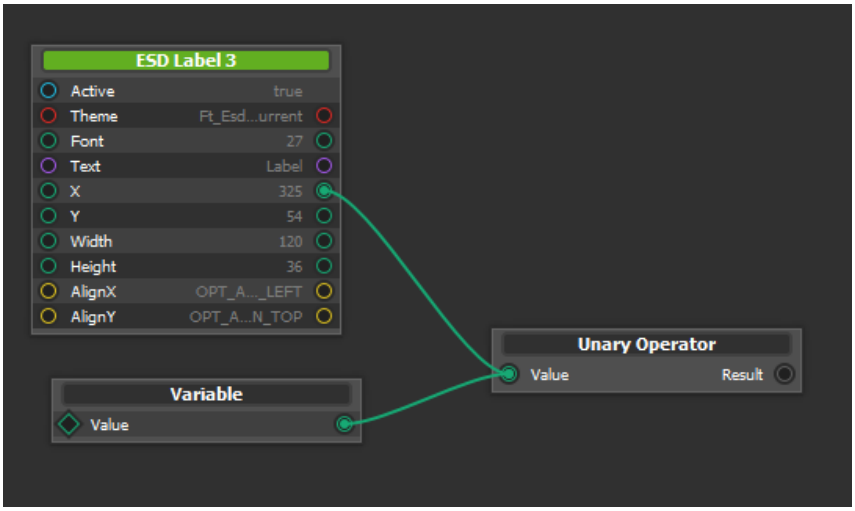
Figure 33 - Rendering Order Example

Zoom In & Zoom Out

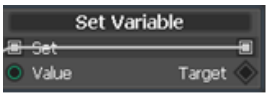
The Logic Node Editor enables *Zoom-in* and *Zoom-out* functionality. Users can scroll the mouse wheel button to zoom in and zoom out.

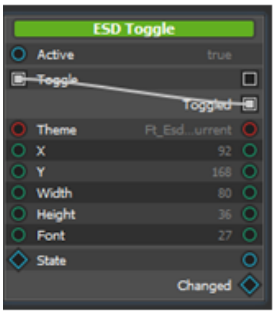
DON'TS

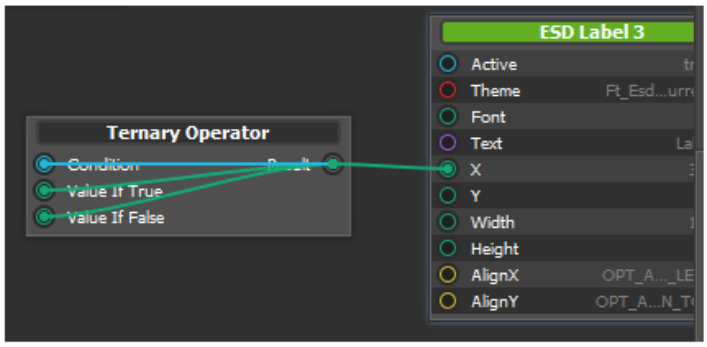
Avoid two connections connecting to the same port. Refer to the sample picture given below.



Avoid endless dead loops caused by connecting the input and output of the same object. It will cause an unexpected error. Refer to the sample pictures given below.







To effect the changes made to the logic note editor, ensure that all the changes are saved to the project, by clicking **"File → Save All"** from the menu or toolbar. In some cases, click the **"Recompile"** button to recompile the source code generated by ESD 3.0.

F. Library Browser

The *Library Browser* consists of the basic components which are predefined and built in as part of ESD 3.0, as well as user defined components and/or resources. The components in the library include *Theme*, *Primitive, Widgets*, *Logic Flow* and *Logic Interface*. Users can select the appropriate components and drag them into the logic node editor or layout editor.

The library view depends on the currently selected node. The library browser will only show the contents which applies to the currently opened node.

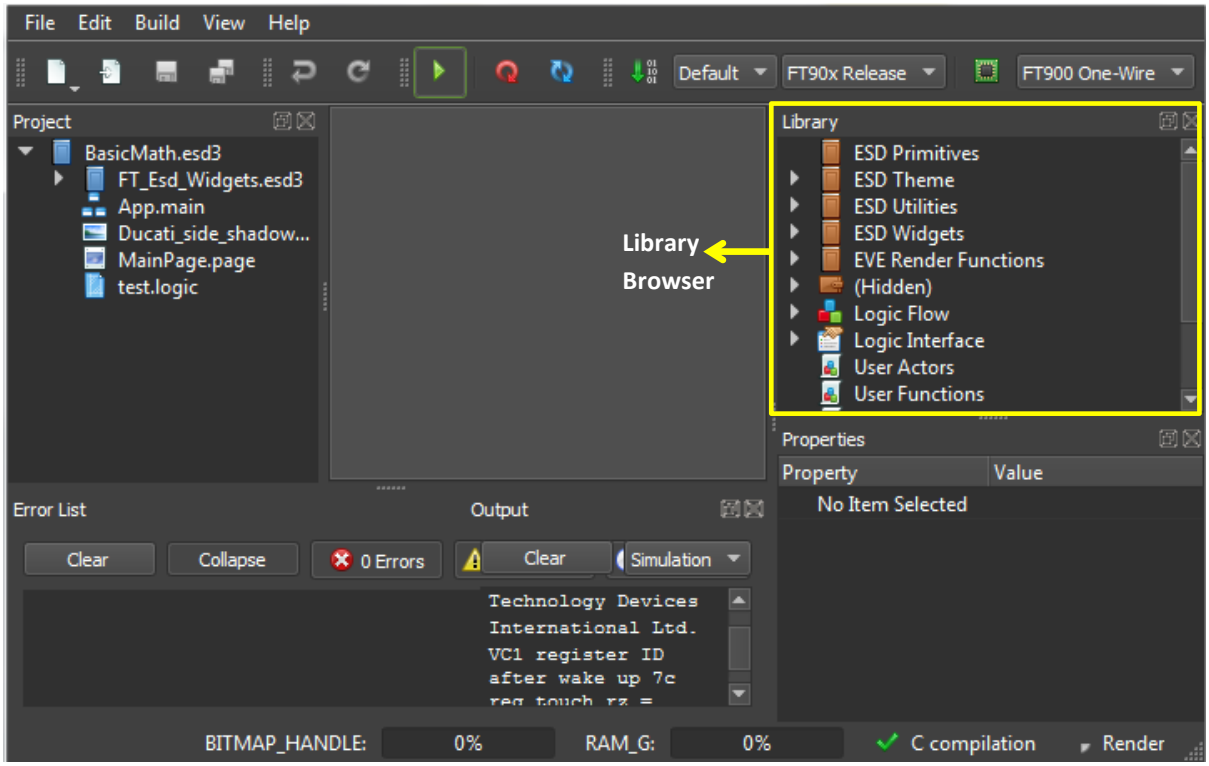


Figure 34 - Library Browser

The forthcoming sections provide information about the different components of the library browser.

ESD Theme

A *Theme* is a collection of colours which can be used across the application in order to maintain a consistent style. The library browser provides the necessary logic nodes to make most use of the theme function.

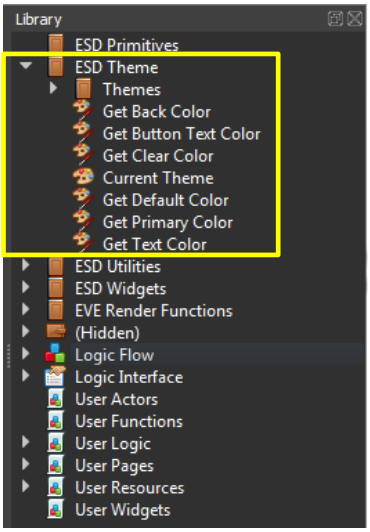


Figure 35 - Library Browser - ESD Theme

Built In Themes

ESD 3.0 provides two *Built-in Themes* for users to configure the colour scheme of project. They are: *Ft_Esd_Theme_LightBlue* theme and *Ft_Esd_Theme_DarkOrange* theme.

Property	Value
(Linked File Proxy)	
File Name	Ft_Esd_Theme_LightBlue.theme
Name	Theme
Category	EsdTheme_Themes
ClearColor	[235, 235, 235] (255)
BackColor	[255, 255, 255] (255)
TextColor	[0, 0, 0] (255)
ButtonTextColor	[250, 250, 250] (255)
DefaultColor	[113, 113, 113] (255)
PrimaryColor	[34, 119, 199] (255)

Property	Value
(Linked File Proxy)	
File Name	Ft_Esd_Theme_DarkOrange.theme
Name	Theme
Category	EsdTheme_Themes
ClearColor	[33, 33, 33] (255)
BackColor	[21, 21, 21] (255)
TextColor	[255, 255, 255] (255)
ButtonTextColor	[255, 255, 255] (255)
DefaultColor	[107, 107, 107] (255)
PrimaryColor	[255, 127, 63] (255)

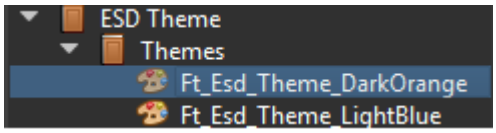


Figure 36 - ESD 3.0 Built-in Themes

The following tables provide information about the built-in themes property -

Property Name	Type	Value (Default)	Description
Name	String	Theme	Ft_Esd_Theme_LightBlue
ClearColor	RGBA	235,235,235(255)	Clear Color
BackColor	RGBA	255,255,255(255)	Background Color
TextColor	RGBA	0,0,0(255)	Text Color
ButtonTextColor	RGBA	250,250,250(255)	Button Text Color
DefaultColor	RGBA	113,113,113(255)	Default Color
PrimaryColor	RGBA	34,119,199(255)	Primary Color

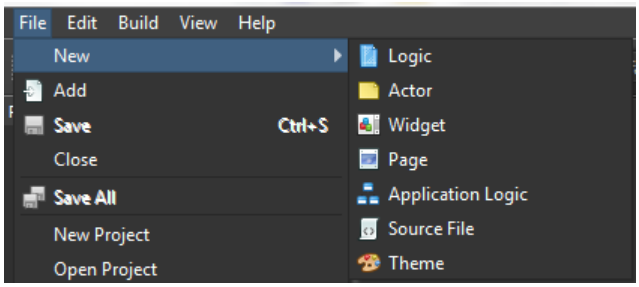
Table 7 - Ft_Esd_Theme_LightBlue Theme

Property Name	Type	Value (Default)	Description
Name	String	Theme	Ft_Esd_Theme_DarkOrange
ClearColor	RGBA	33,33,33(255)	Clear Color
BackColor	RGBA	21,21,21(255)	Background Color
TextColor	RGBA	255,255,255(255)	Text Color
ButtonTextColor	RGBA	255,255,255(255)	Button Text Color
DefaultColor	RGBA	107,107,107(255)	Default Color
PrimaryColor	RGBA	255,127,63(255)	Primary Color

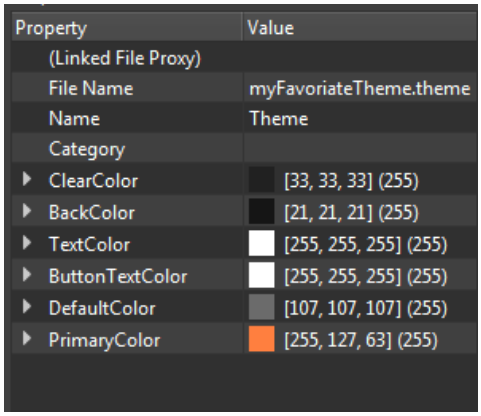
Table 8 - Ft_Esd_Theme_DarkOrange Theme

To create a new theme –

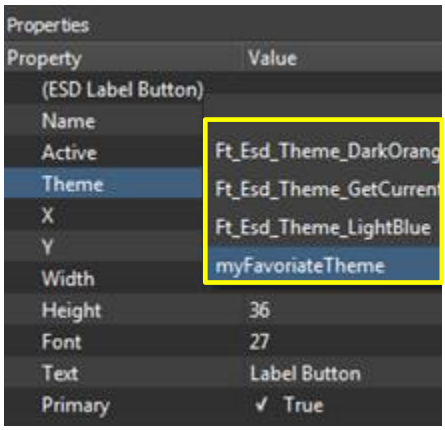
1. Click and select **"File → New → Theme"**.



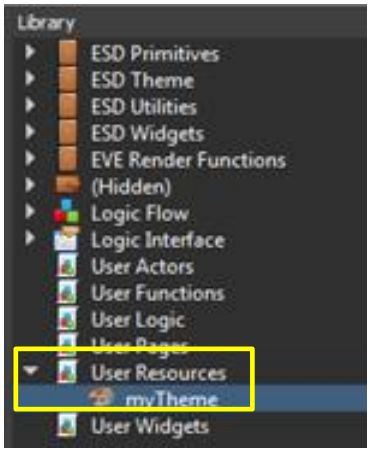
2. A new "*.*theme*" file is created within the Project folder. Using the Property Editor, users can configure it as per their requirement.



3. The newly-created theme will be available across the project and may be applied it to other widgets with the theme property.



4. The newly created theme will also be available under **User Resources** in the *Library Browser*. Users can drag and drop the theme to the logic editor.



ESD Primitives

A *Primitive* is a special type of logic node which is a wrapper for a rendering function called by Render slot. It has no return value and its properties cannot be output to other logic nodes. Similar to widgets, it appears on the screen, but is unable to handle user input.

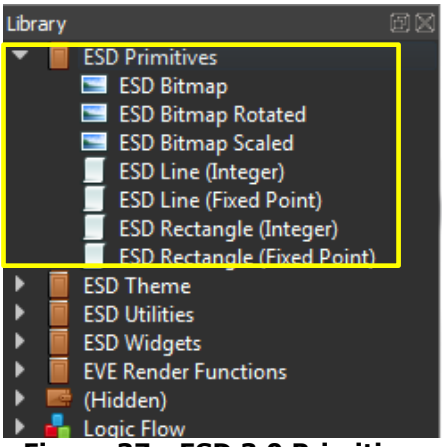


Figure 37 - ESD 3.0 Primitives

ESD Bitmap

The *ESD Bitmap* Widget allows users to display a bitmap resource. A bitmap resource is generated from an image file by adding it into a project. A bitmap resource is treated as a single bitmap cell by default with the name "(Bitmap Resource)_0".

For example, if the bitmap resource is named "photo.jpg", the default bitmap resource will be named as "photo_0". Users can define how many bitmap cells they want by specifying the *"CellHeight"* property.

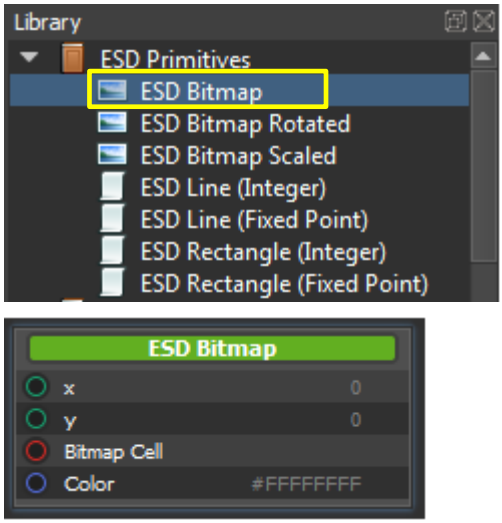


Figure 38 - ESD Bitmap Primitive

Property Name	Description
x	x coordinate of Bitmap, top-left, in pixels
y	Y coordinate of Bitmap, top-left, in pixels
Bitmap Cell	Bitmap cell to display
Color	Color to be applied to bitmap

Table 9 - ESD Bitmap Properties

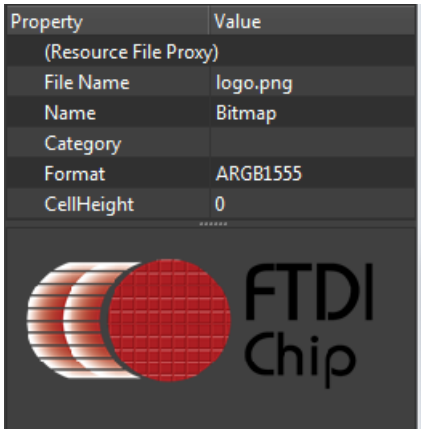
To add a bitmap

Click and select **"File → Add"** or click from the toolbar. A bitmap cell is then assigned. ESD 3.0 supports the following image file formats as input – *".png", ".jpg"* and *".jpeg"*. Bitmap resources can contain a single or multiple cells based on the *CellHeight* value.

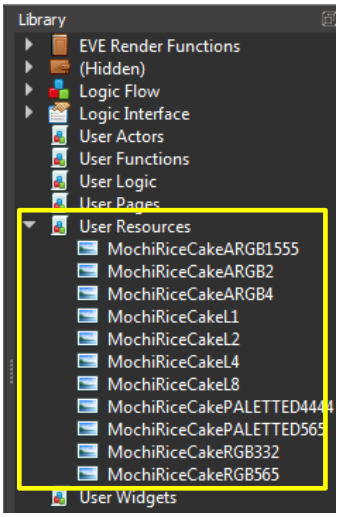
If the value of the *"CellHeight"* is zero, then the number of cells is one.

If the value of the *"CellHeight"* is non-zero, then the number of cells value is calculated as shown below –

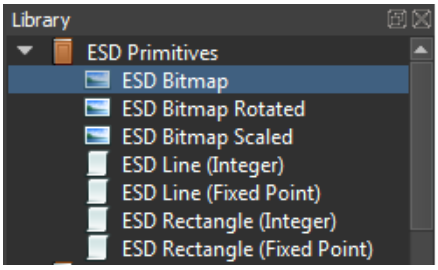
`Number of Cell = Bitmap Height / CellHeight`



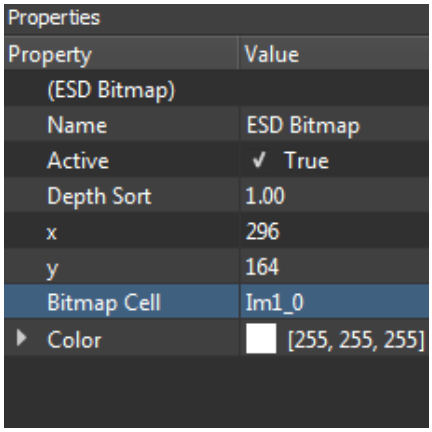
Upon adding a Bitmap, the new bitmap resource will be added into the Library under the category – *"User Resources"*.



To use an *ESD Bitmap*, drag and drop the ESD Bitmap Node into the *Layout Editor* or *Logic Node Editor*.



Choose a Bitmap Cell to display from the dropdown list.



The list of Bitmap Cell values is updated automatically after a new image file is added into a project.

ESD Line

The *ESD Line* allows the users to draw the lines on the screen.

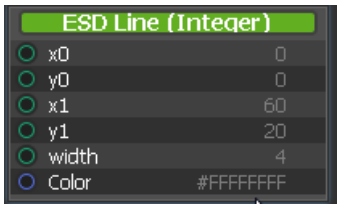


Figure 39 - ESD Line

Property Name	Description
x0	x – coordinate of the start point, in pixels
Y0	y – coordinate of the start point, in pixels
x1	x – coordinate of the end point, in pixels
y1	y – coordinate of the end point, in pixels
width	Line width, in pixel
Color	Line color, in RGBA format

Table 10 - ESD Line Properties

Fixed Point Variant

ESD Line (Fixed point) expresses the properties with pixel attribute in fixed point format. It provides more control to lines.

ESD Rectangle

The *ESD Rectangle* allows users to draw rectangle on the screen.

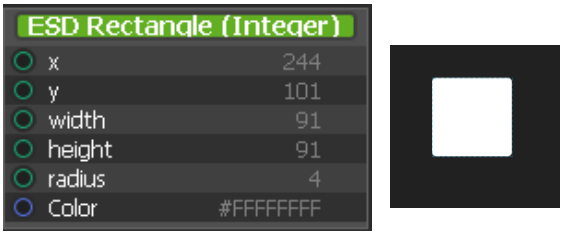


Figure 40 - ESD Rectangle

Property Name	Description
x	x coordinate of Bitmap, top-left, in pixels
y	Y coordinate of Bitmap, top-left, in pixels
width	width, in pixels
height	height, in pixels
radius	radius of the round corner, in pixels
Color	color of the rectangle

Table 11 - ESD Rectangle Properties

Fixed Point Variant

ESD Rectangle (Fixed point) expresses the properties with pixel attribute in fixed point format. It provides more control to rectangles.

Widgets

A *widget* is a kind of logic node which has a visual appearance rendered by the Embedded Video Engine (EVE). The widgets can be found in the library browser area. There are two types of widgets – *ESD Widgets* (built-in widgets provided by ESD 3.0) and *User Widgets* (defined by users). Users can browse for all the available widgets easily from the library browser.

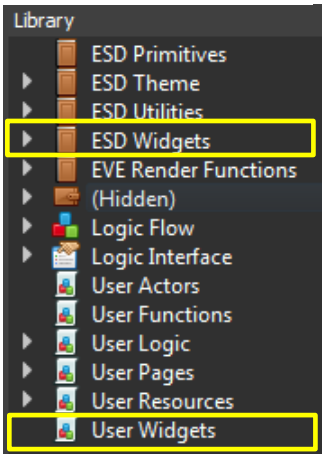


Figure 41 - Widgets

Elements

The *ESD Elements* are both simple and basic widgets, usually used to construct higher level and more complex widgets. Normally, they do not handle any touch input.

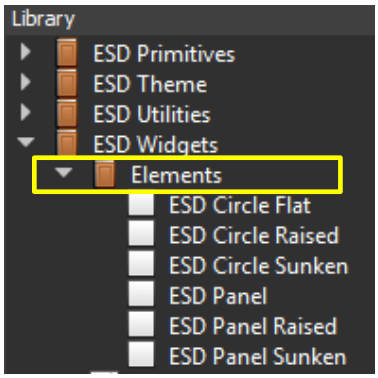


Figure 42 - ESD Elements

ESD Circle

The *ESD Circle* allows the user to draw circles on the screen. The following styles are available – *ESD Circle Flat; ESD Circle Raised* and *ESD Circle Sunken*.

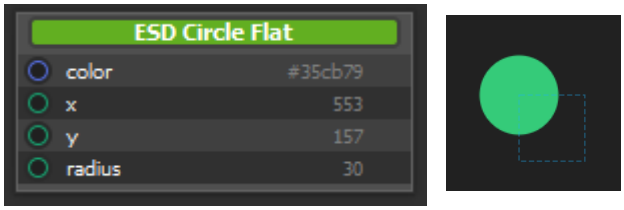


Figure 43 - ESD Circle Element

Property Name	Description
color	RGB value to be rendered inside the circle
x	x coordinate of central point, in pixels
y	Y coordinate of central point, in pixels
radius	radius of the point

Table 12 - ESD Circle Element Properties

ESD Panel

The *ESD Panel* is an adjustable rectangular widget that defines a coloured area. The colour can be specified by a theme or user's selection. Internally, it is constructed by EVE primitives RECTS and SCISSORS. Two styles of ESD Panel are available namely – *ESD Panel Raised* and *ESD Panel Sunken*.

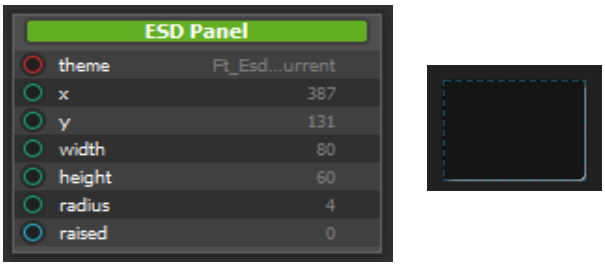


Figure 44 - ESD Panel Element

Property Name	Description
Theme	Theme to be applied on the panel
X	x coordinate of the top-left point, in pixels
Y	Y coordinate of the top-left point, in pixels
Width	Panel width
Height	Panel Height
Radius	radius of the corners
Raised	Set to true to make panel in raised style

Table 13 - ESD Panel Element Properties

ESD Clock

The *ESD Clock* is a basic widget based on an EVE built-in widget. It can be accessed from the library browser under the ESD Widgets folder.

Figure 45 - ESD Clock Widget

Property Name	Description
Active	Enable or Disable displaying clock
Theme	Theme to be applied on the widget
X	x coordinate of the top-left of the widget, in pixels
Y	Y coordinate of the top-left of the widget, in pixels
Width	Widget width
Height	Widget Height
Alpha	Adjust the transparency
Hours	The hour hand position
Minutes	The minute hand position
Seconds	The seconds hand position
Milliseconds	The time expressed in milliseconds unit
~~Interactive~~	~~Currently not in use~~

Table 14 - ESD Clock Widget Properties

Users can connect the ESD clock with other widgets, such as ESD toggle or ESD timer via hours/minutes/seconds or milliseconds property. The sample pictures given below shows how a clock is started or stopped by an ESD Toggle Widget and the corresponding logic node connection.

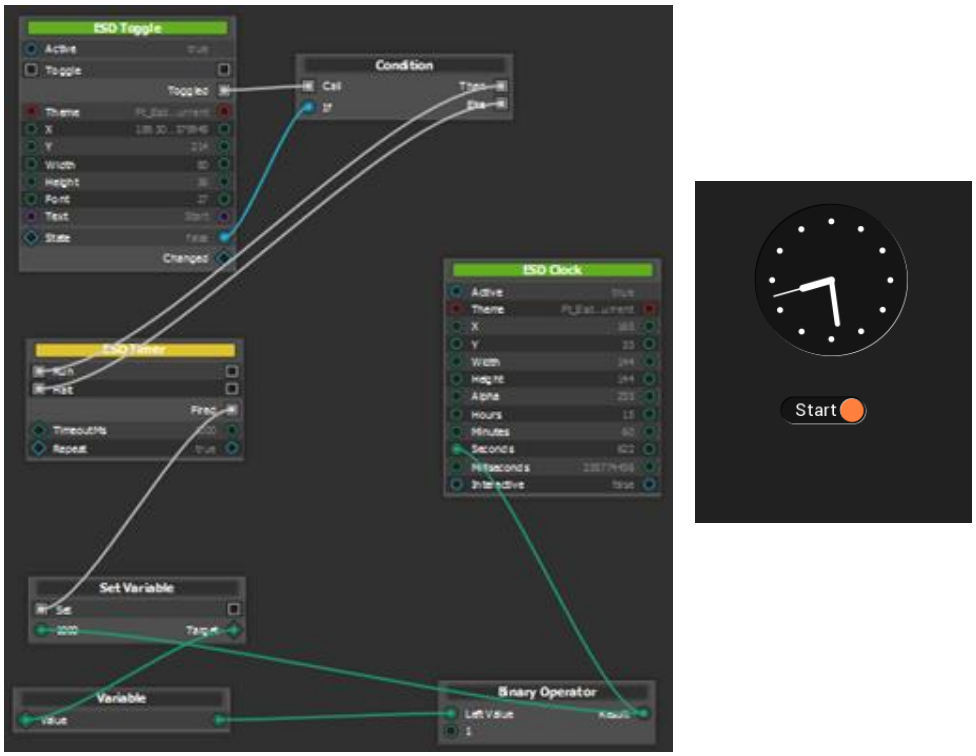


Figure 46 - ESD Clock Widget Example

ESD Check Box

The *ESD Check Box* is a widget which has two states and toggles its own state based on user touch input.

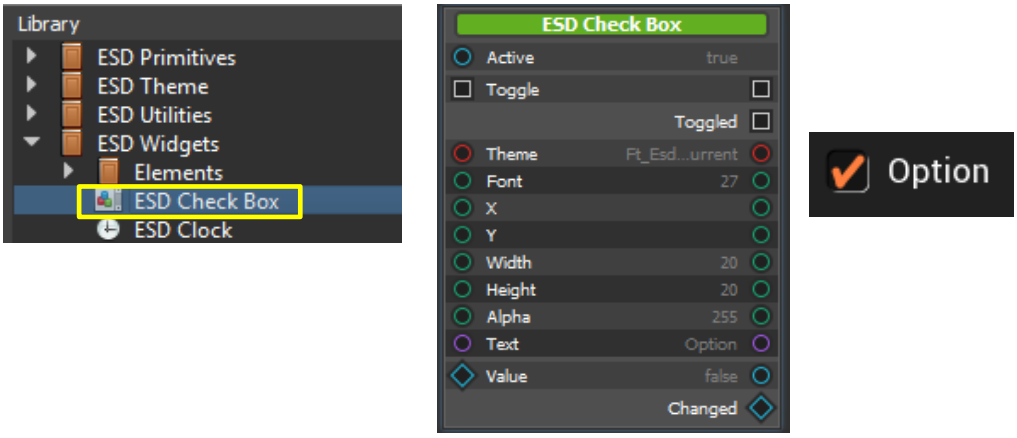


Figure 47 - ESD Check Box Widget

Property Name	Description
Active	Enable displaying this widget
Theme	Get theme (background/text/default/…color)
X	Absolution position of the horizon
Y	Absolution position of the vertical
Width	Widget width
Height	Widget Height
Alpha	Adjust the transparency
Text	The display label behind the check box
Value	Checked/Unchecked

Table 15 - ESD Check Box Widget Properties

Users can connect the *ESD Check Box* with other widgets in order to get user's input via a signal mechanism.

ESD Gauge

The *ESD Gauge* is a circular widget which is based on the EVE built-in widget. It can be connected with other widgets to display the value by needle. This widget does not have the capability of interacting with the user's touch input.

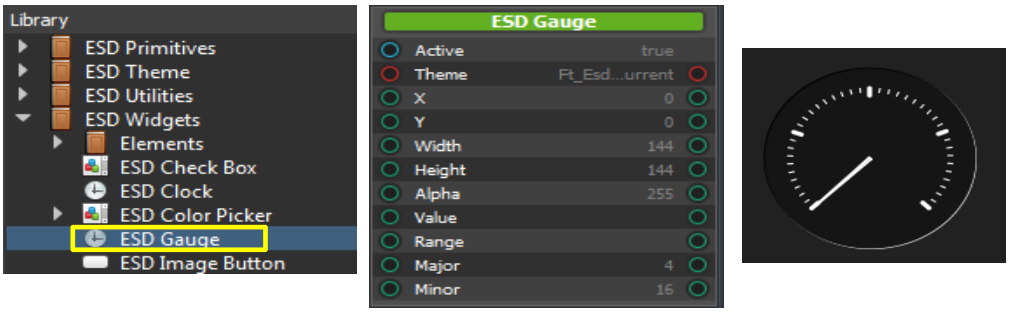


Figure 48 - ESD Gauge Widget

Property Name	Description
Active	Enable or disable displaying this widget
Theme	Theme to be applied to this widget
X	Absolution position of the horizon

Y	Absolution position of the vertical
Width	Widget width
Height	Widget Height
Alpha	Adjust the transparency
Value	Current value that the needle is pointing to
Range	Value range
Major	Major Division
Minor	Minor Division

Table 16 - ESD Gauge Widget Properties

An ESD Gauge can display the temperature, speed, pressure etc. The sample picture given below illustrates how an ESD Gauge can display the current pressure inside a tire. Users can control a pump by increasing/decreasing pressure on the screen.

Figure 49 - ESD Gauge Example

The corresponding logic node connection in the logic note editor is shown below –

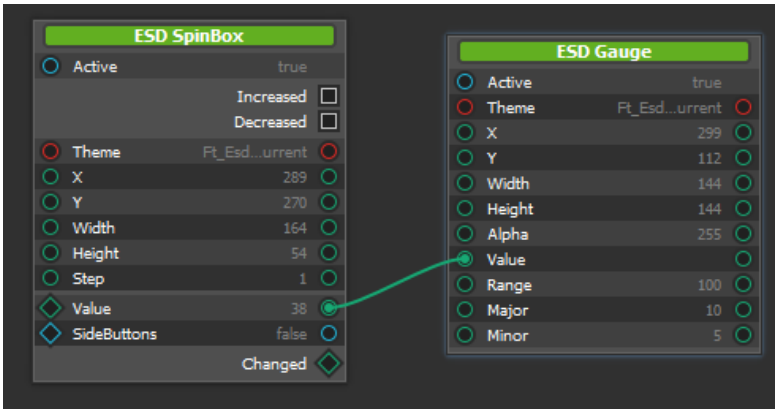


Figure 50 - ESD Gauge Logic Node Connection Example

ESD Slider

The *ESD Slider* is used to adjust the values by dragging a slider.

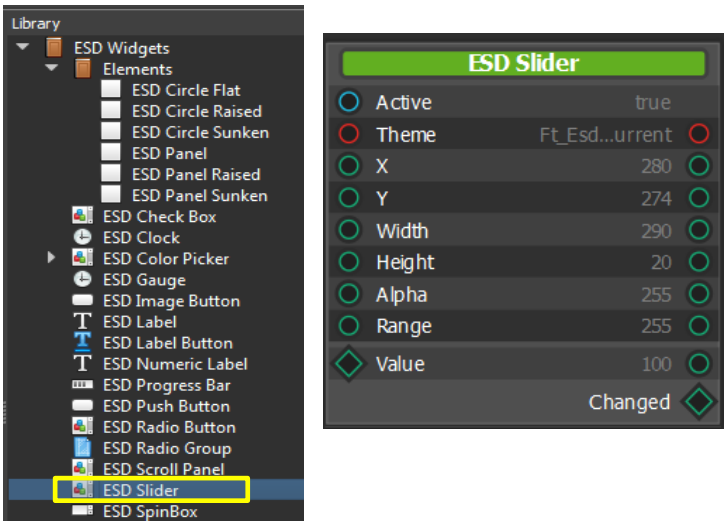


Figure 51 - ESD Slider Widget

Property Name	Description
Active	Enable or disable displaying this widget
Theme	Theme to be applied to this widget
X	x coordinate of the top-left of the widget, in pixels
Y	Y coordinate of the top-left of the widget, in pixels
Width	Widget width
Height	Widget Height
Alpha	Adjust the transparency
Range	Value range

Table 17 - ESD Slider Widget Properties

The slider modifies the alpha value and combines it with the RGB color. The combined value creates the ESD Rectangle colour. It will set the connected variable to the new value once the value is changed. This is illustrated in the pictures given below –

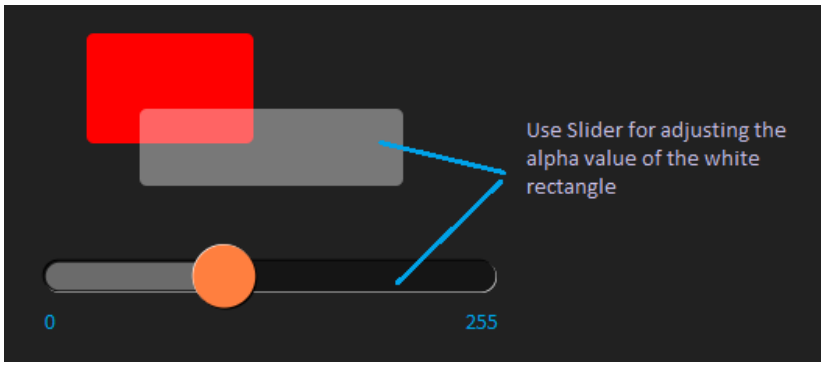


Figure 52 - ESD Slider Example

The corresponding logic node connection in the logic note editor is shown below –

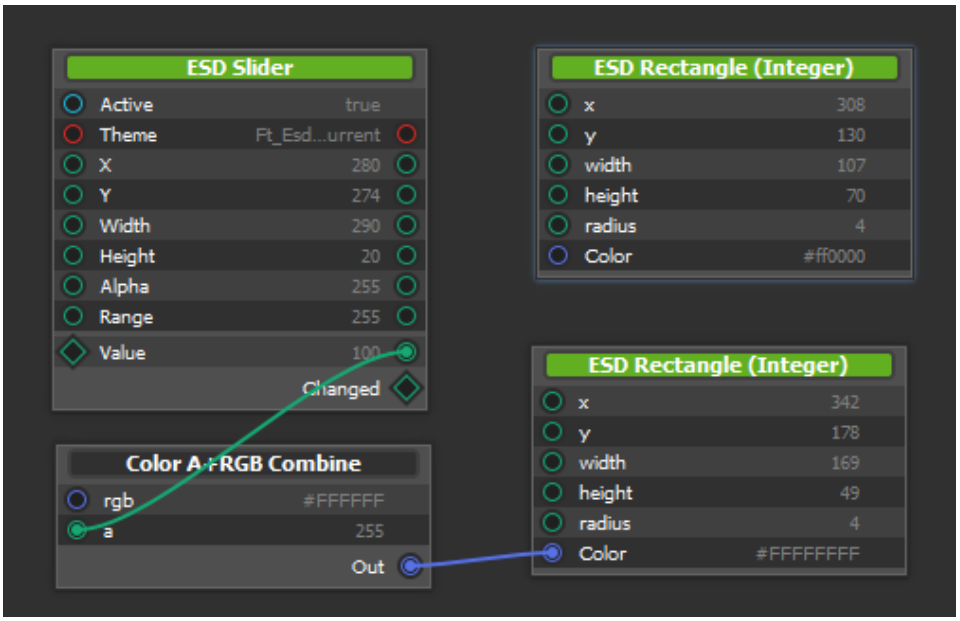


Figure 53 - ESD Slider Logic Node Connection Example

ESD Radio Button and ESD Radio Group

The *ESD Radio Button* is used to choose options. The *ESD Radio Button Group* is a utility widget which is not rendered to display. It enables multiple radio buttons to form a single group; only one radio button can be selected at a time.

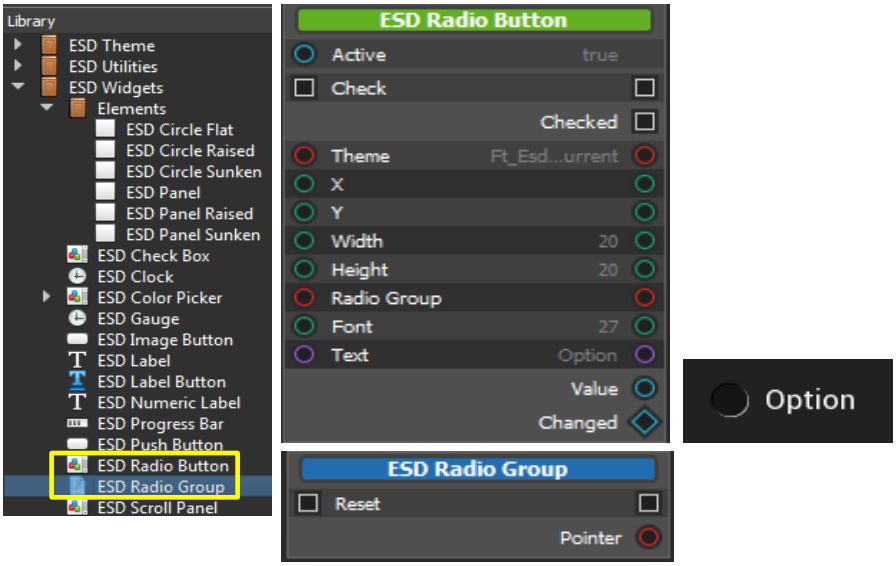
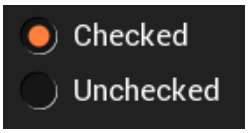


Figure 54 - ESD Radio Button & ESD Radio Group

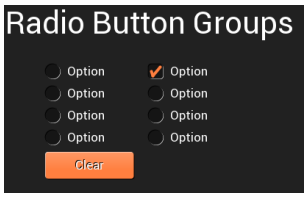
Property Name	Description
Active	Enable or disable displaying this widget
Check	Selected or not selected
Theme	Theme to be applied to this widget
X	Absolute position of the horizon
Y	Absolute position of the vertical
Width	Widget width
Height	Widget Height
Radio Group	Pointer to a radio group
Font	Font Size
Text	The display label beside the radio button

Table 18 - ESD Radio button Properties

An ESD Radio Button has 2 states: *Checked* or *Unchecked*. *Checked* state is selected by clicking on an empty box, or by an external signal from other sources (Push Button, Image Button, Checkbox, etc.).



Each Radio Button has a pointer to an ESD Radio Group; it shares the same context. Only one Radio Button can be checked at a time. When an ESD Radio Group receives a Reset signal, it will reset all states of its children Radio Buttons. Refer to the sample pictures.



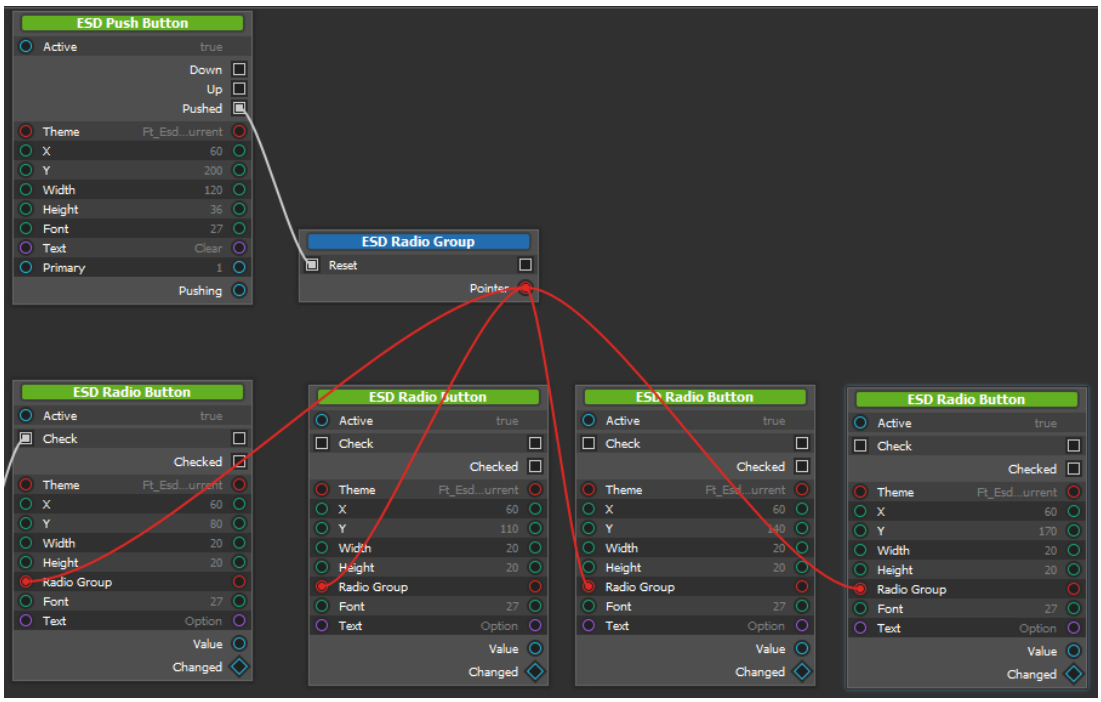


Figure 55 - ESD Radio Button & ESD Radio Group Example

ESD Push Button

The *ESD Push Button* allows the users to add a 3D effect rectangle button with customized size and text label.

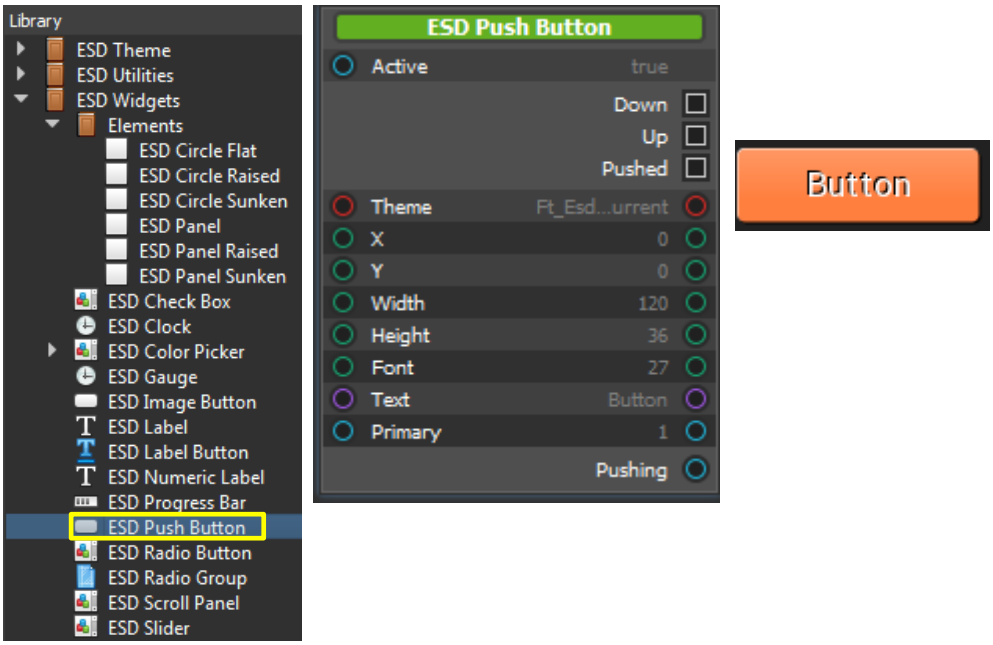


Figure 56 - ESD Push Button

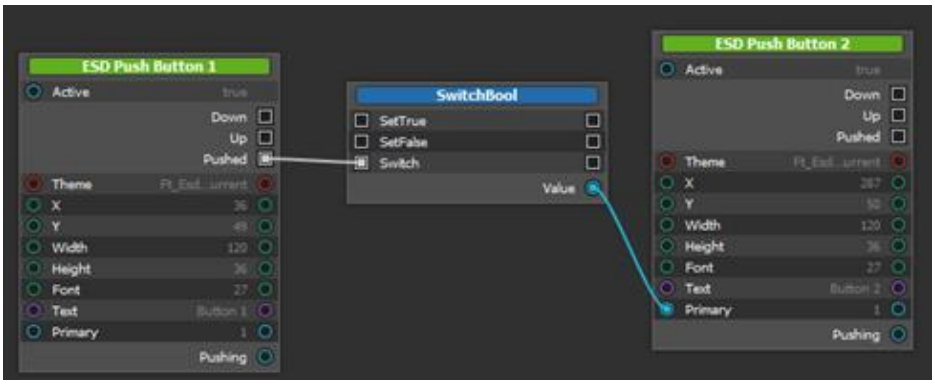
Property Name	Description
Active	Active state of the button, set to true to appear on the screen
Theme	Theme applied for the button
X	Coordinate of button, top-left, in pixels
Y	Coordinate of button, top-left, in pixels
Width	Button width, in pixels
Height	Button height, in pixels
Font	Fonts used in the label
Text	The label displayed on the button
Primary	Primary state of the button – Set to True to use the Primary color from theme Set to False to use the default color from theme

Table 19 - ESD Push Button Properties

Output / Signal	Description
Down / Up / Pushed	Output signal when the push button is Down/Up or Pushed state
Pushed	Output signal when the push button is in pushed state

Table 20 - ESD Push Button Output/Signal

The logic node connection in the sample picture below illustrates how a Push Button ("Button1") is created to toggle the primary state of another Push Button ("Button 2").



Here is the output –

Normal State

Pushed State

Figure 57 - ESD Push Button Example

ESD Label

The *ESD Label* allows the users to add a Text Label with customized size and text.

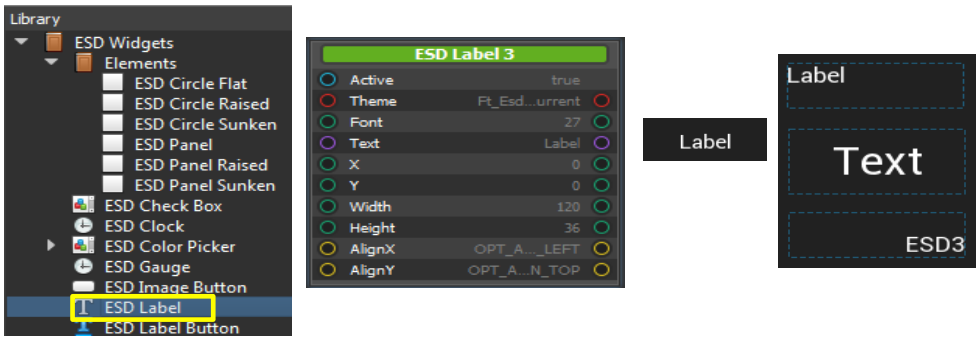


Figure 58 - ESD Label

Property Name	Description
Active	Active state of the label, set to true to appear on the screen
Theme	Theme applied for the label
Font	Fonts used in the label. Same as bitmap handle defined in EVE
Text	The text content of the label. By default, "Label"
X	X coordinate of label, top-left, in pixels
Y	Y coordinate of label, top-left, in pixels
Width	Label width, in pixels
Height	Label height, in pixels
Align X	Horizontal alignment of text *OPT_ALIGN_LEFT: Left,* *OPT_ALIGN_CENTER: Center,* *OPT_ALIGN_RIGHT: RIGHT*
Align Y	Vertical alignment of text *OPT_ALIGN_TOP: Top,* *OPT_ALIGN_CENTER: Center,* *OPT_ALIGN_BOTTOM: Bottom*

Table 21 - ESD Label Properties

ESD Label Button

The *ESD Label Button* allows the users to add a button in the form of a label.

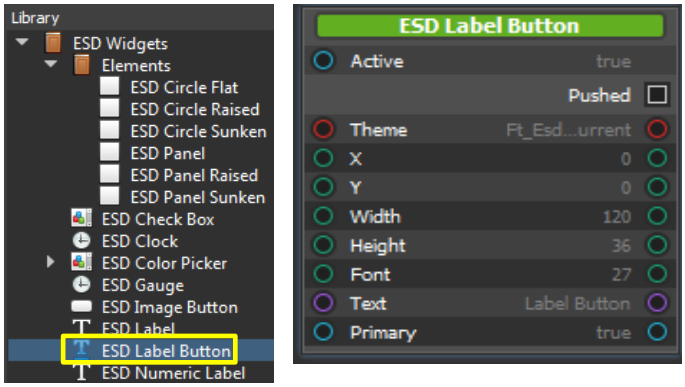


Figure 59 - ESD Label Button

Property Name	Description
Active	Active state of the label button, set to true to appear on the screen
Theme	Theme applied for the label button
X	X coordinate of label button, top-left, in pixels
Y	Y coordinate of label button, top-left, in pixels
Width	Label button width, in pixels
Height	Label button height, in pixels
Font	Font used in the label button. Same as bitmap handle defined in EVE
Text	The text content of the label button. By default, "Label"
Primary	Primary state of the label button – Set to True to use the Primary colour from theme Set to False to use the default colour from theme

Table 22 - ESD Label Button Properties

The logic node connection in the sample picture illustrates how a toggle changes the state upon pushing the label button. When label button is pushed, the corresponding output signal is **"Pushed"**.

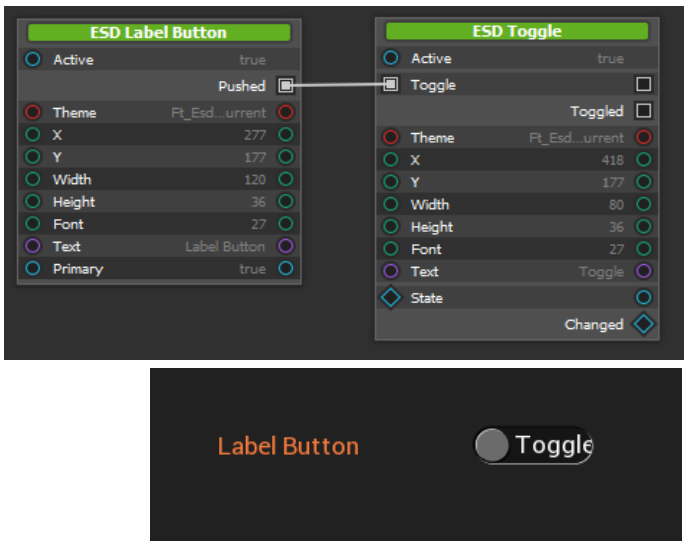


Figure 60 - ESD Label Button Example

ESD Image Button

The *ESD Image Button* allows the users to add a button in the form of a bitmap.

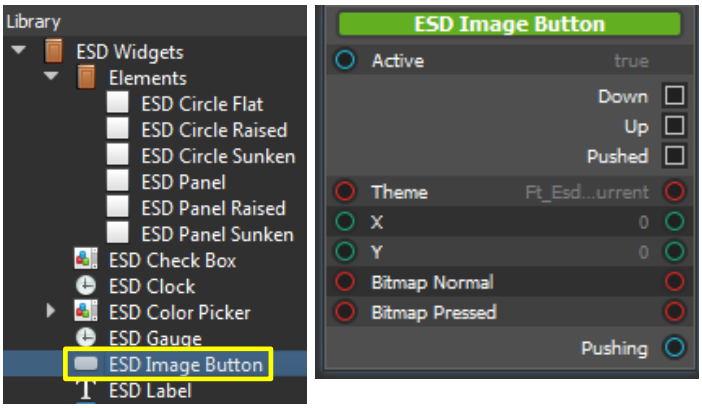


Figure 61 - ESD Image Button

Property Name	Description
Active	Active state of the image button, set to true to appear on the screen
Theme	Theme to be applied on the image button
X	X coordinate of the image button, top-left, in pixels
Y	Y coordinate of the image button, top-left, in pixels
Width	Image button width, in pixels
Height	Image button height, in pixels
Bitmap Normal	Bitmap cell to display in the normal state
Bitmap Pressed	Bitmap cell to display in the pressed state

Table 23 - ESD Image Button Properties

Output / Signal	Description
Down / Up / Pushed	Output signal when image button is Down/Up or Pushed state
Pushed	Output value indicated (true) if the image button is in pressed state

Table 24 - ESD Image Button Output/Signal

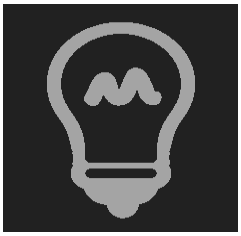
The following example illustrates how to add / use an image button –

Add new image buttons and assign the bitmaps - *Bulb Off Image* and *Bulb On Image* (refer to the bitmap pictures) to *Bitmap Normal* and *Bitmap Pressed*.

Bulb Off image (bitmap cell name is bulbOff1_0)

Bulb On image (bitmap cell name is bulb1On_0)

Property	Value
(ESD Image Button)	
Name	ESD Image Button
Active	✓ True
Theme	Ft_Esd_Theme_GetCurrent
X	400
Y	200
Bitmap Normal	bulb1Off_0
Bitmap Pressed	bulb1On_0



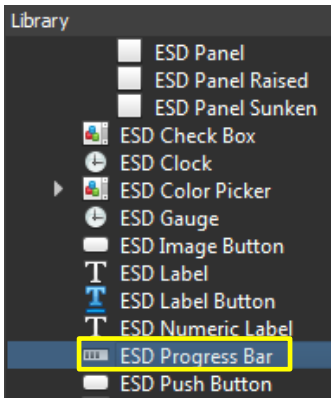
Normal State

Pressed State

Figure 62 - ESD Image Button Example

ESD Progress Bar

The *ESD Progress Bar* widget allows the users to add and use a progress bar.



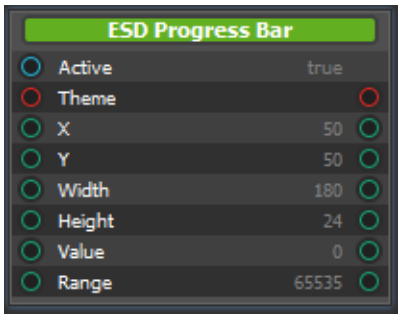


Figure 63 - ESD Progress Bar

Property Name	Description
Active	Active state of the progress bar, set to true to appear on the screen
Theme	Theme to be applied on the progress bar
X	X coordinate of the progress bar, top-left, in pixels

Y	Y coordinate of the progress bar, top-left, in pixels
Width	Progress bar width, in pixels
Height	Progress bar height, in pixels
Value	Indicates the progress level and displayed as the filled portion of the progress bar
Range	Progress bar values range

Table 25 - ESD Progress Bar Properties

The logic node connection in the sample picture illustrates the creation of a continuous progress bar with a range from 0 to 1000, taking its input from the built-in "GetMilliseconds" function node.

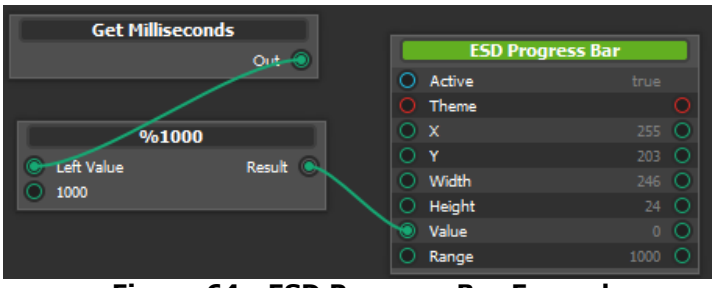


Figure 64 - ESD Progress Bar Example

ESD Spin Box

The *ESD Spin Box* Widget allows the users to create a spin box with customized style and size.

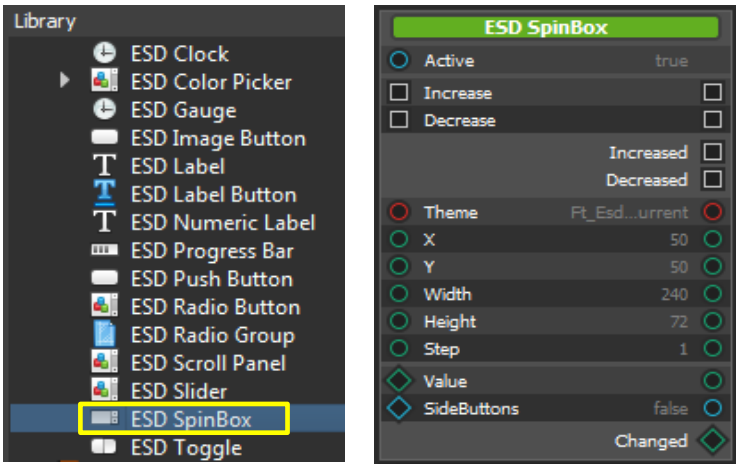


Figure 65 - ESD Spin Box

Property Name	Description
Active	Active state of the spin box, set to true to appear on the screen
Increase	Input slot to trigger the increase (up) event
Decrease	Input slot to trigger the decrease (down) event
Theme	Theme to be applied on the spin box
X	X coordinate of the spin box, top-left, in pixels
Y	Y coordinate of the spin box, top-left, in pixels
Width	Spin box width, in pixels
Height	Spin box height, in pixels
Step	When arrows are used to change the spin box's value, the value will be incremented/decremented by the amount of step
Value	Value of the spin box
SideButtons	Denotes the Spin box style  False (Default) True

Table 26 - ESD Spin Box Properties

Output / Signal	Description
Increased/Decreased	Output signal when the spin box is in Up (Increased)/ Down (Decreased) state
Changed	Output signal, the changed value of spin box is written out

Table 27 - ESD Spin Box Output/Signal

The logic node connection in the sample picture illustrates interconnection of two different styles of ESD Spin box widgets that will increase or decrease simultaneously.

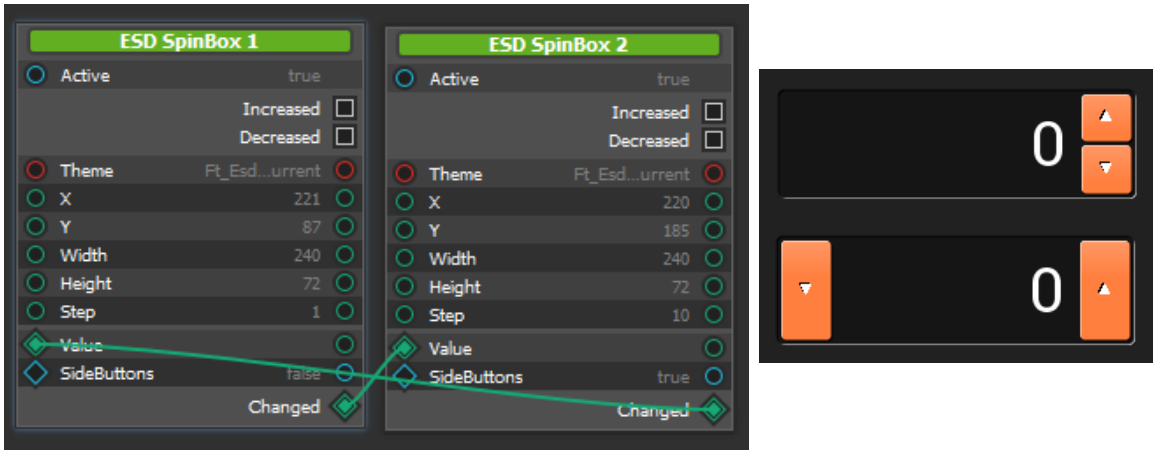


Figure 66 - ESD Spin Box Example

The *ESD Scroll Panel* provides a scrollable area to display contents.

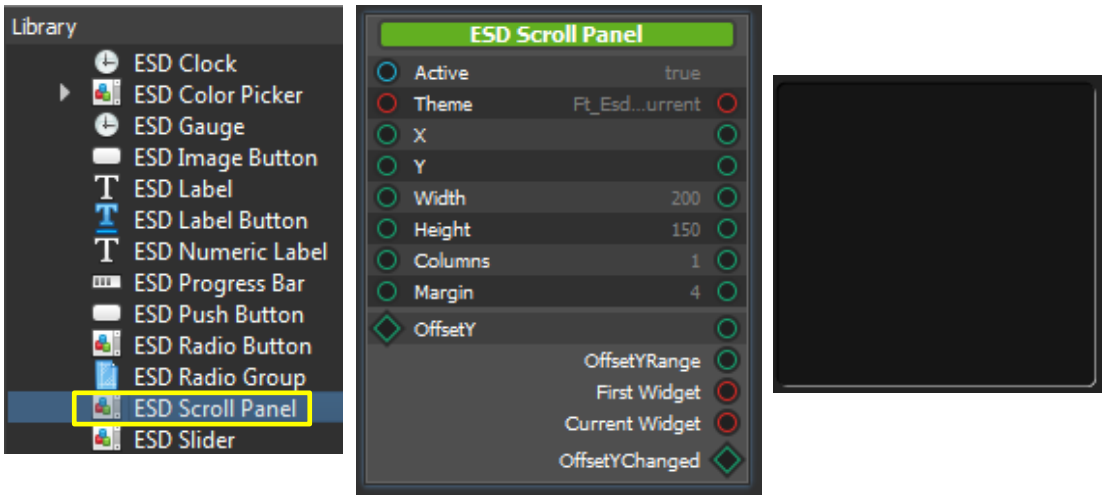


Figure 67 - ESD Scroll Panel

Property Name	Description
Active	Enable or disable displaying this widget
Theme	Theme to be applied to this widget
X	X coordinate of the top-left, in pixels
Y	Y coordinate of the top-left, in pixels
Width	Panel width
Height	Panel height
~~Columns~~	~~Currently not in use~~
~~Margin~~	~~Currently not in use~~
OffsetY	The distance changed from the default position upon moving the mouse up/down inside the scroll panel. *OffsetYRange*: The length of the contents inside the scroll panel. • Returns the total height of the content, if it is greater than the height of the scroll panel (scroll panel cannot display the whole content). • Else returns 0. *First Widget:* Pointer to the first object inside the scroll panel. *Current Widget:* Pointer to the last object inside the scroll panel. *OffsetYChanged:* Writer of OffsetY

Table 28 - ESD Scroll Panel Properties

Ft_Esd_ScrollPanel

Ft_Esd_ScrollPanel (Scroll Panel) is an object inside a page. It has a pointer to the *Ft_Esd_LayoutChild* (Layout Child), which is an abstract layer. These LayoutChild layers are containers; they hold widgets that users use to place inside the Scroll Panel. The first pointer points to the first LayoutChild node in a linked list, while the last pointer points to the last node.

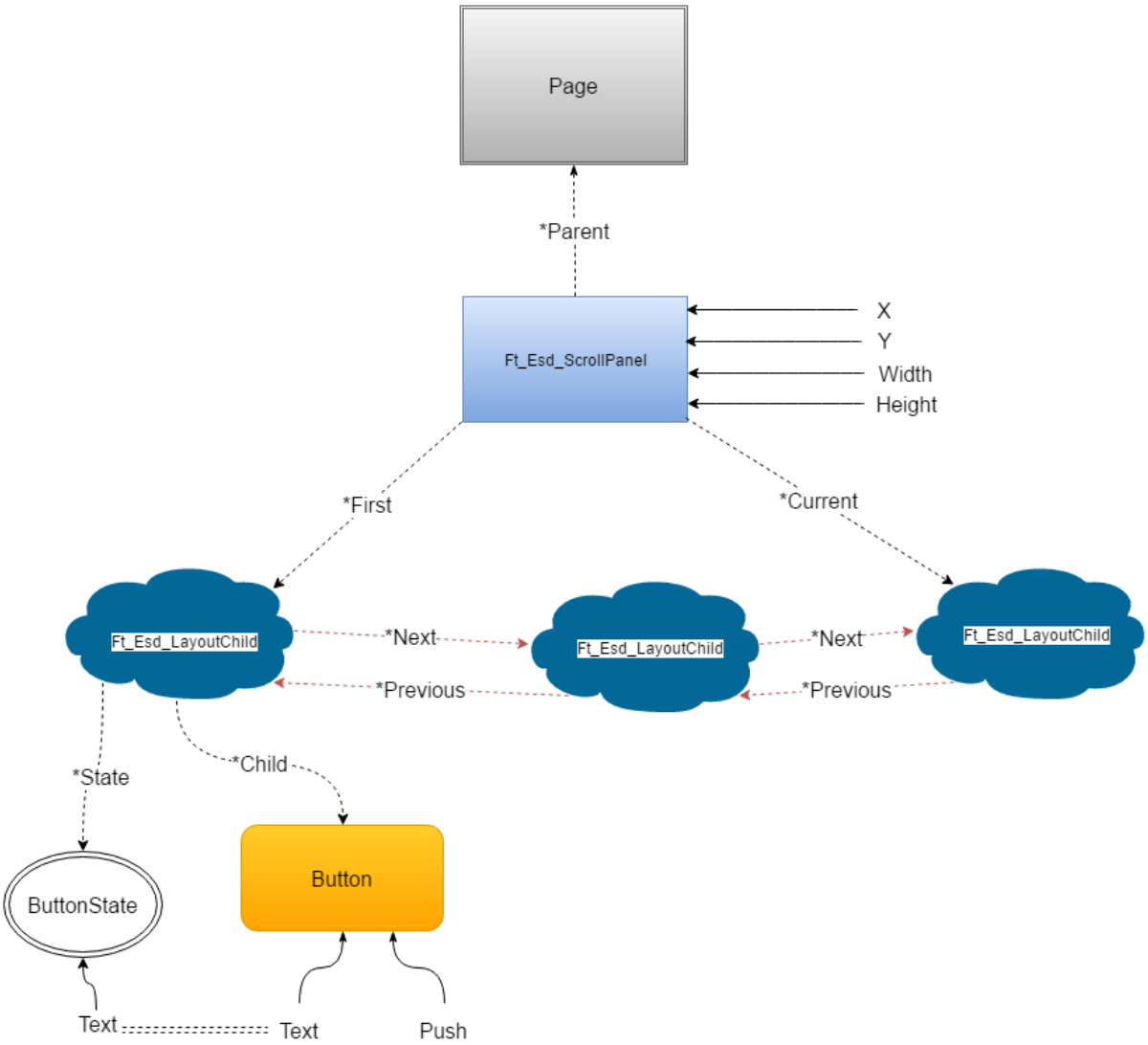


Figure 68 - Ft_Esd_ScrollPanel - Use Case Diagram

Each LayoutChild holds 2 pointers: *Child* and *State*; *Child* links to a user widget (in this case a button). The *State* stores the user state for that widget.

Ft_Esd_ScrollPanel_Add()

Users need to call *Ft_Esd_ScrollPanel_Add()* function to add the widget into a scroll panel. However, the target widget has to be encapsulated into "Ft_Esd_LayoutChild" type.

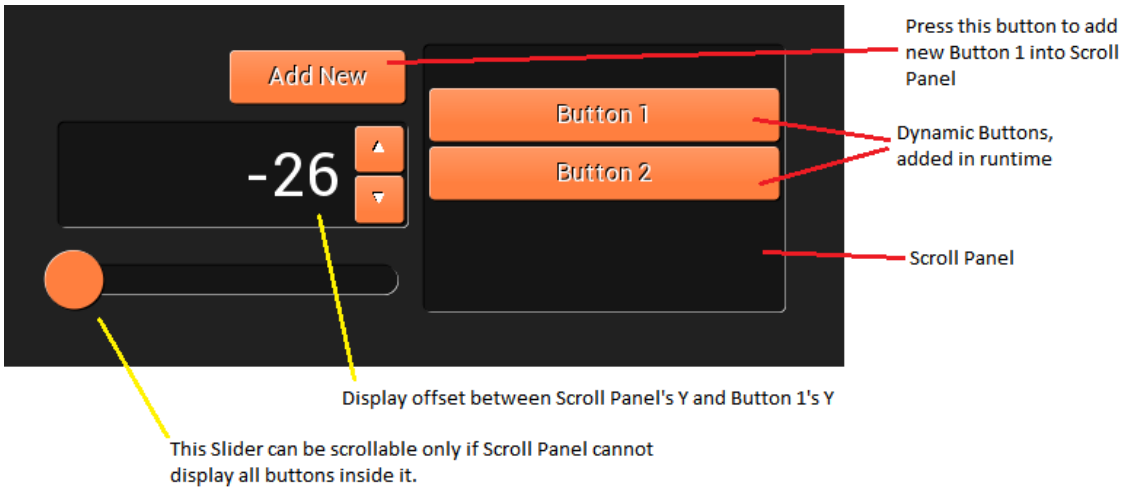
```
void Ft_Esd_ScrollPanel_Add(Ft_Esd_ScrollPanel *context, Ft_Esd_LayoutChild
*child)
```

Ft_Esd_ScrollPanel_Remove()

Users need to call *Ft_Esd_ScrollPanel_Remove()* function to remove the widget from a scroll panel. However, the target widget has to be encapsulated into "Ft_Esd_LayoutChild" type.

```
void Ft_Esd_ScrollPanel_Remove(Ft_Esd_ScrollPanel *context,
Ft_Esd_LayoutChild *child)
```

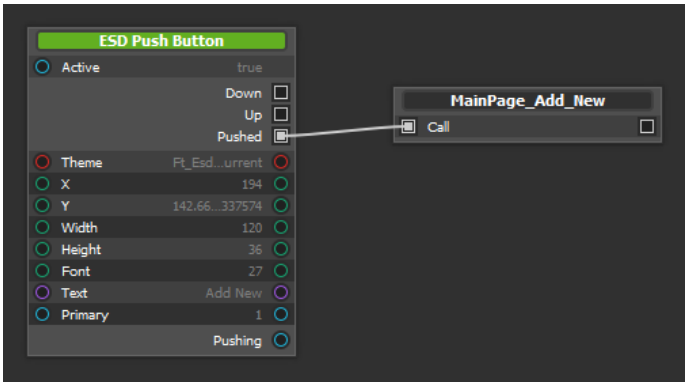
The following example illustrates the usage of ESD Scroll Panel widget.



"Add New" button is use to add a new button into the Scroll Panel.

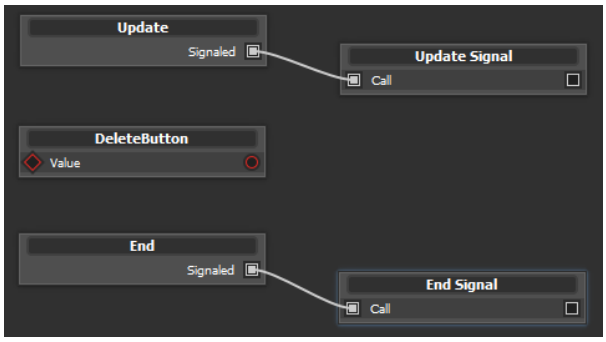
Content inside the scroll panel can be moved by dragging up or down.

Create a method in *MainPage.c* and place it in the logic node. This can be done by pressing the *"Add New"* button. (Refer to the code under the Create Widget section).



Use the *"Update signal"* to delete a pending deleting button. Upon pressing the dynamic button inside the scroll panel, a delete request is sent. This request will be handled by the next Update application call.

If users switch to another page, all buttons will be deleted. The *"End Application"* call handles this state.



Create Widget

The MainPage_Add_New method creates dynamic objects (Button, LayoutChild, and ButtonState) and adds them to the scroll panel. The MainPage_Add_New method is called by pressing the *"Add New"* button.

```
// Add a button to the scroll panel

ESD_METHOD(MainPage_Add_New, Context = MainPage)
void MainPage_Add_New(MainPage *context)
{
    …
}
```

Inside the following method, the widget is created and allocated dynamically in runtime.

```
// Create the widget

Ft_Esd_PushButton *widget = malloc(sizeof(Ft_Esd_PushButton));
Ft_Esd_PushButton__Initializer(widget);
```

Users can bind signals, or properties of the widget, to user methods.

```
// Set user bindings

widget->Pushed = Button_Pushed;
widget->Text = Button_Text;
```

The bind assignments depend on specific use cases. Users can customise bindings.

- Create a LayoutChild container for a new button.

```
// Create an entry for the layout mechanism

Ft_Esd_LayoutChild *entry = malloc(sizeof(Ft_Esd_LayoutChild));
memset(entry, 0, sizeof(Ft_Esd_LayoutChild));
entry->Child = widget;
entry->State = state;
```

- Users can bind properties of a new Button to a LayoutChild and/or Scroll Panel. This step arranges the layout correctly.

```
// Bind entry
```

```
entry->Width = widget->Width;
entry->Height = widget->Height;
entry->Update = Ft_Esd_PushButton_Update;
entry->Render = Ft_Esd_PushButton_Render;

// Bind scrollpanel

widget->Parent = &context->ScrollPanel;
widget->X = Ft_Esd_ScrollPanel_ChildX;
widget->Y = Ft_Esd_ScrollPanel_ChildY;
widget->Width = Ft_Esd_ScrollPanel_ChildWidth;
widget->Height = Ft_Esd_ScrollPanel_ChildHeight;
```

Finally, attach the LayoutChild into ScrollPanel.

```
// Start widget

Ft_Esd_PushButton_Start(widget);
Ft_Esd_ScrollPanel_Add(&context->ScrollPanel, entry);
```

Delete Widget

Users should delete objects inside the Scroll Panel before switching to other pages. This action helps the system to free the unused resource, and reduces memory consumption.

```
ESD_METHOD(MainPage_End_Signal, Context = MainPage)

void MainPage_End_Signal(MainPage *context)
{
        while (context->ScrollPanel.First)
        {
                Free_Button(context, context->ScrollPanel.First);
        }
}

// Utility to free the button state

void Free_Button(MainPage *context, Ft_Esd_LayoutChild *entry)
{
        // End widget
        Ft_Esd_PushButton *widget = entry->Child;
        Ft_Esd_PushButton_End(widget);
        Ft_Esd_ScrollPanel_Remove(&context->ScrollPanel, entry);

       // Free all state
        ButtonState *state = entry->State;
        free(state->Text);
        free(state);
        free(widget);
        free(entry);
}
```

ESD Toggle

The *ESD Toggle* widget provides the toggle switch functionality with user touch enabled.

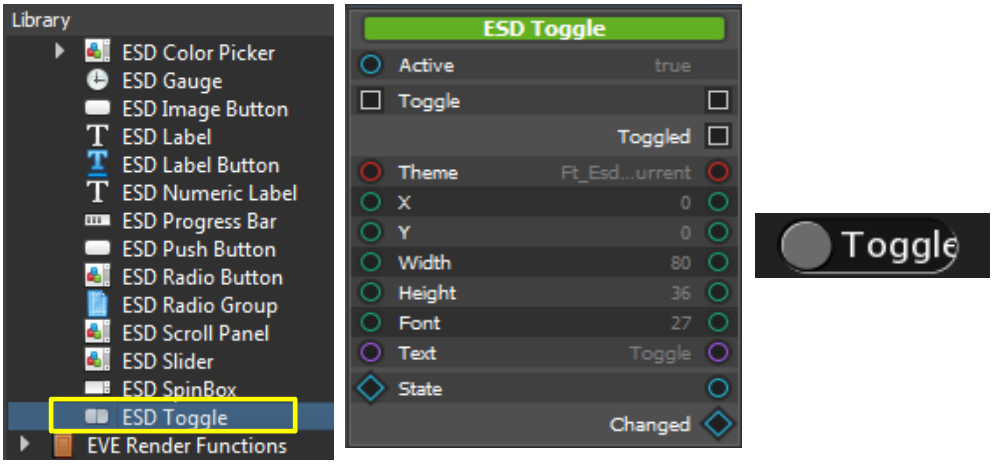


Figure 69 - ESD Toggle

Property Name	Description
Active	Enable or disable displaying this widget
Theme	Theme to be applied to this widget
X	X coordinate of the top-left, in pixels
Y	Y coordinate of the top-left, in pixels
Width	Toggle widget width
Height	Toggle widget height
Font	The font handle used by label inside the widget
Text	Toggle label
State	The current state of the toggle widget

Table 29 - ESD Toggle Widget Properties

Output / Signal	Description
Toggled	Output signal triggered by toggle action
Changed	Output value of the changed state

Table 30 - ESD Toggle Widget Output/Signal

The sample pictures below illustrate the creation of two toggle widgets, *"Toggle1"* and *"Toggle2"* which is connected with the same state.

Figure 70 - ESD Toggle Widget Example

ESD Color Picker

ESD Color Picker provides a circle style Colour picker which is associated with a circle bitmap. It handles user touch and translates the touch point into a corresponding RGB Colour value as an output. Users can connect this widget with the bitmap *"circular_colorwheel.png"* under *"$(LIBRARY)\Ft_Esd_Widget"* after adding it into the current project. The bitmap can be changed but it needs to be in the same style, only with a different radius value, which can be adjusted in the Property Editor.

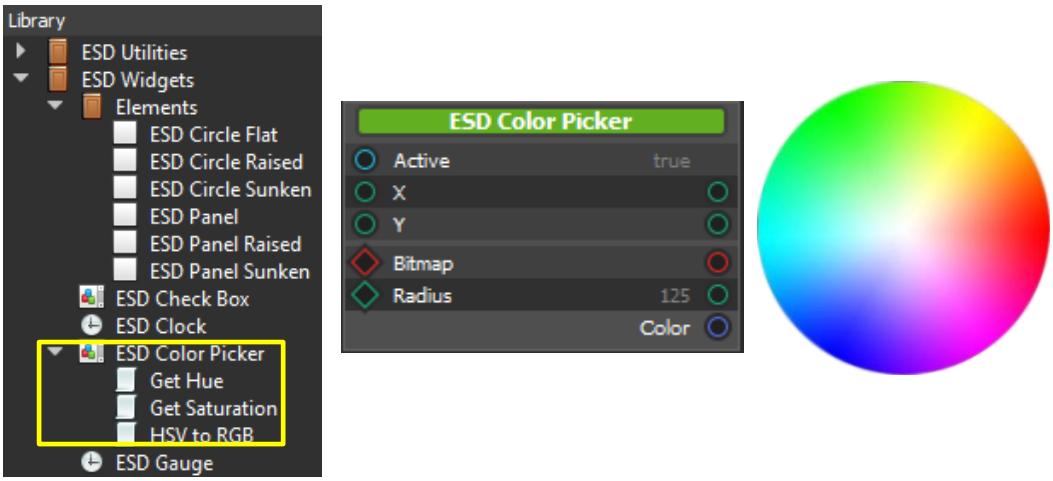


Figure 71 - ESD Color Picker Widget

Property Name	Description
Active	Enable or disable displaying this widget
X	X coordinate of the top-left, in pixels
Y	Y coordinate of the top-left, in pixels
Bitmap	The bitmap cell used in the colour picker
Radius	The radius of the circular image (in pixel)

Table 31 - ESD Color Picker Widget Properties

Output / Signal	Description
Color	The current colour based on the user's touch and selected bitmap

Table 32 - ESD Color Picker Widget Output/Signal

The logic node connection in the sample picture below illustrates connecting the output of the ESD Color Picker widget with the colour property of the other widget.

Figure 72 - ESD Color Picker Widget Example

Logic Flow

A *Logic Flow* is an ESD built-in logic node. It is used to define the logic or control flow and contains the following nodes:

- *Condition*
- *Include Header*
- *Binary Operator*
- *Ternary Operator*
- *Unary Operator*
- *Set Variable*

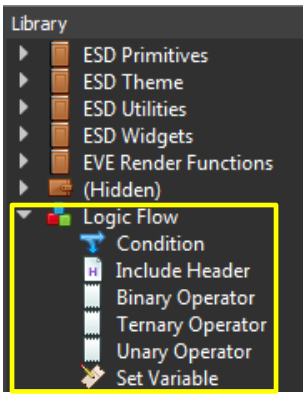


Figure 73 - Logic Flow

Condition

Condition node provides a set of rules to be performed if a certain condition is met. In other words, it allows applying decision points.

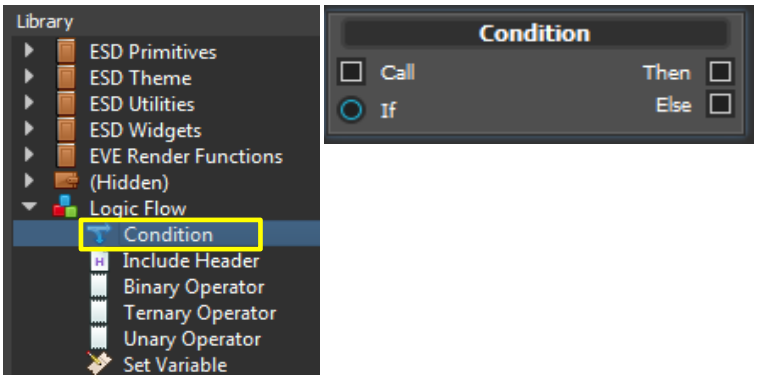


Figure 74 - Logic Flow – Condition Node

Whenever a condition node is called, a *"Then"* condition will be called when the *"If"* expression is True, otherwise an *"Else"* condition will be called.

Ensure that the input for the "If" connection is a Boolean variable (i.e. *True* or *False*).

Include Header

Include Header node is used to include a header file from the project folder. Users are required to change only the property to the name of the header file for it to be included into the project.

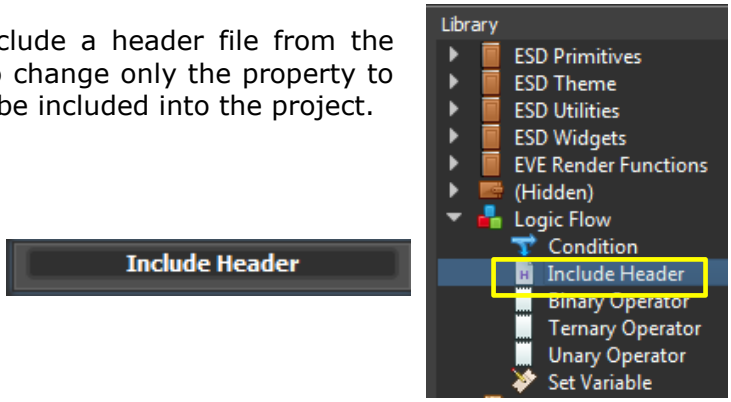


Figure 75 - Logic Flow - Include Header Node

The sample picture given below shows a header file – *"FT_DataTypes.h"* included into the project.

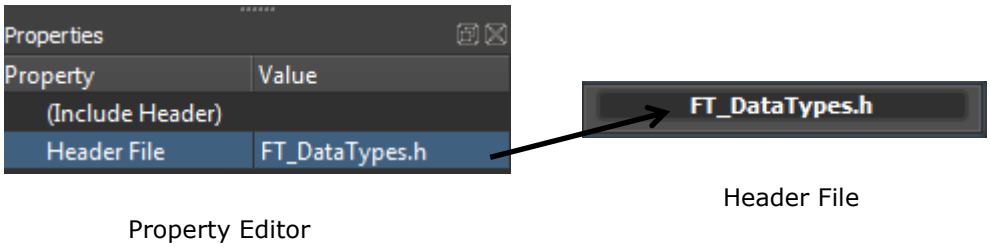


Figure 76 - Include Header Node Example

Binary Operator

This node contains *Binary Operator* that operates on two operands (inputs) and manipulates them to return an output. For example: *Result=Left Value * Right Value*.

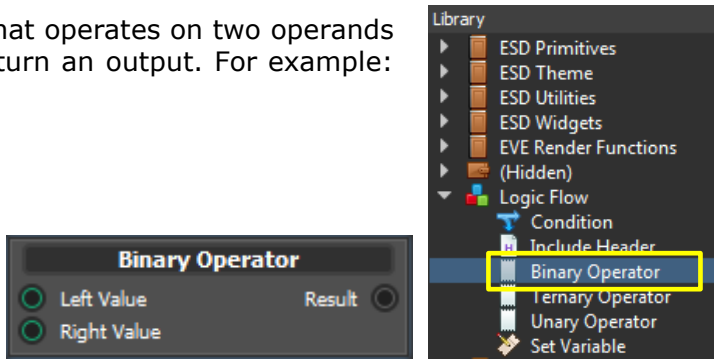


Figure 77 - Logic Flow - Binary Operator Node

The following table provides a list of binary operators supported by ESD 3.0.

Binary Operator	Description
==	Equal to comparison operator
!=	Not Equal to comparison operator
&&	Logical AND operator
&	Binary AND operator
||	Logical OR operator
|	Logical OR operator
*	Multiplication operator
/	Division operator
%	Modulus operator
^	Binary XOR operator
+	Addition operator
-	Subtraction operator
,	Comma operator
>	Greater than operator
<	Less than operator
>=	Greater than or equal to

<=	Less than or equal to
<<	Binary Left Shift Operator
>>	Binary Right Shift Operator

Table 33 - ESD Supported Binary Operators

Ternary Operator

This node contains *Ternary Operator* that operates on three arguments. First is the comparison argument; second is the result upon a true comparison and the third is the result upon a false comparison. For example: *Result= condition ? "Value if true" : "Value if false".*

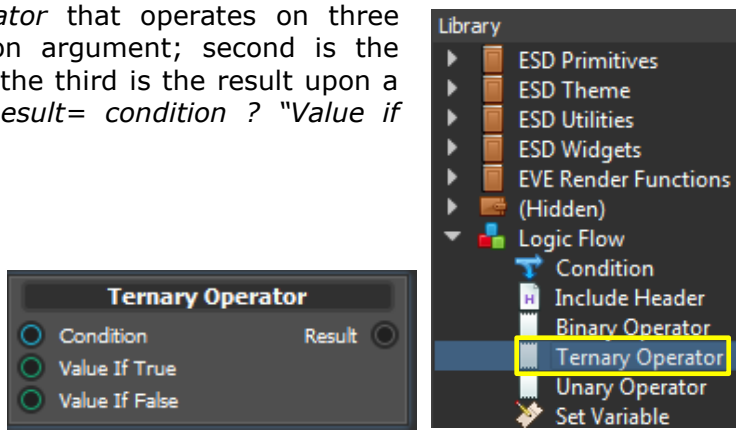


Figure 78 - Logic Flow - Ternary Operator Node

Unary Operator

This node contains *Unary Operator* that operates on a single operand (input). For example: *~.*

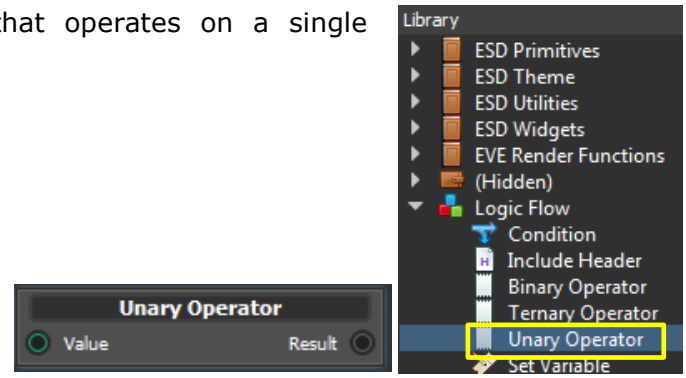


Figure 79 - Logic Flow - Unary Operator Node

The following table provides a list of unary operators supported by ESD 3.0.

Unary Operator	Description
!	Logic Negation

&	Address of
*	De-reference
+	Positive operator
-	Negative operator
~	One's complement
!!	Non-zero value converts to 1and other values converts to 0.

Table 34 - ESD Supported Unary Operators

Set Variable

The *Set Variable* node is used to set the values of a variable. It links an event call with a data binding to a Writer (output a functional call through "writer) or a *Variable* (sets a variable to the value).

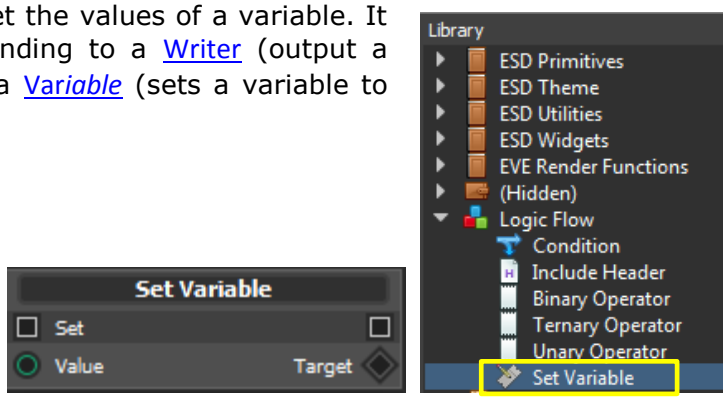


Figure 80 - Logic Flow - Set Variable Node

Logic Interface

A *Logic Interface* is an ESD built-in logic node. It is used to define an interface and contains the following nodes:

- *Input*
- *Output*
- *Signal*
- *Slot*
- *Variable*
- *Writer*

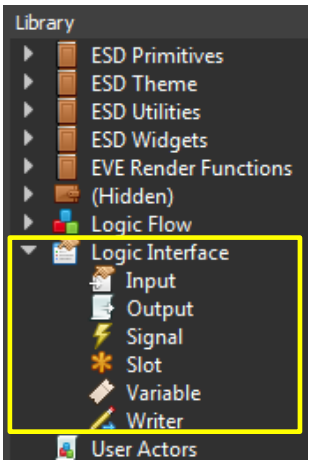
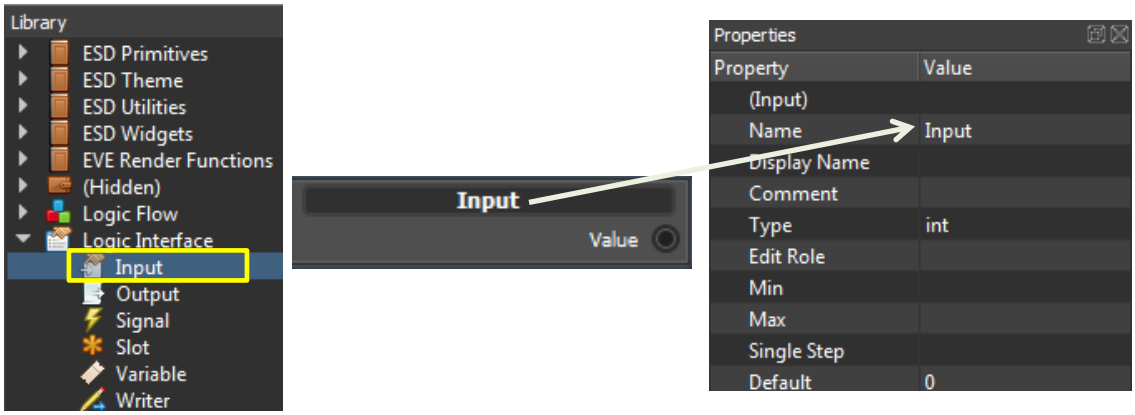


Figure 81 - Logic Interface

Input

The *Input* node is used to create a variable for incoming data binding. Users can configure its property through the Property Editor.

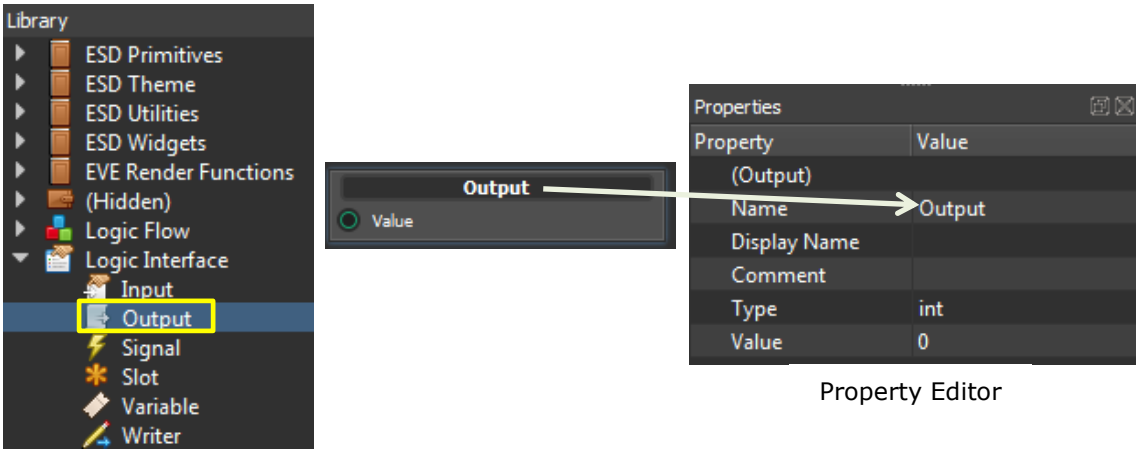


Property Editor

Figure 82 - Logic Interface - Input Node

Output

The *Output* node is used to create a variable for outgoing data binding. Users can configure its property through the Property Editor.



Property Editor

Figure 83 - Logic Interface - Output Node

Signal

The *Signal* node is used to create an outgoing event function call. Users can configure its property through the Property Editor.

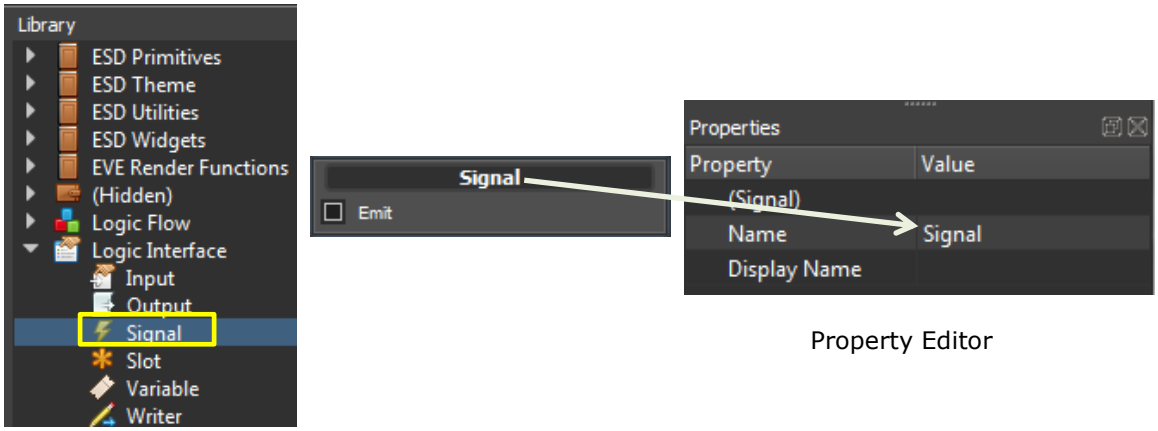


Figure 84 - Logic Interface - Signal Node

Slot

The *Slot* node is used to create an incoming event function call.

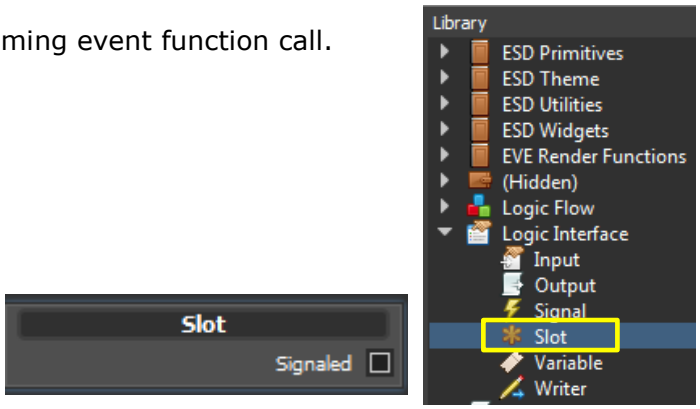


Figure 85 - Logic Interface - Slot Node

Built-in Slot

There are a number of *built-in slots* available for Widgets, Pages and Applications which are automatically called through the node hierarchy. The following table provides the list of built-in slots.

Built-in Slot	Description
Start	Called when the node is created in memory
Render	Writing rendering calls to the co-processor command buffer. It is repeatedly called.

Update	Called once before the frame render. It is called frequently.
Idle	Called repeatedly while waiting for the frame to flip
End	Called when the node is removed from the memory

Table 35 - Logic Interface – Slot – Built-in Slot

Variable

The *Variable* node is used to define a plain C variable. Users can drag and drop the Variable from the Library browser to the logic node editor and configure its property through the Property Editor.

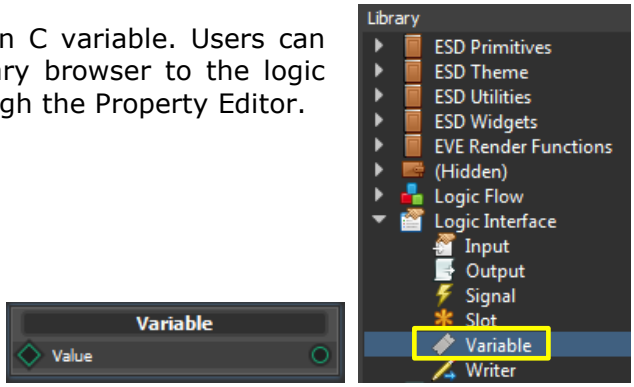
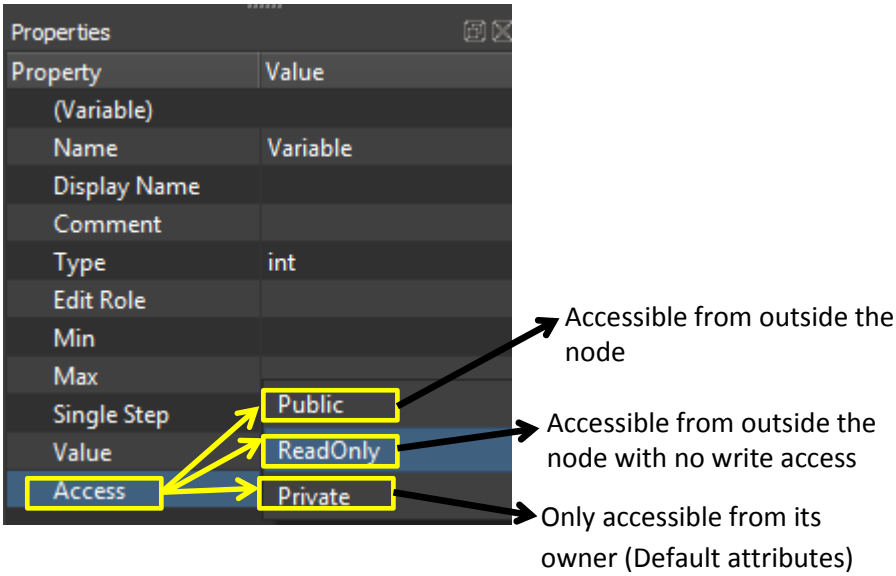


Figure 86 - Logic Interface - Variable Node



Property Editor

Writer

The *Writer* node is used to define an interface for the logic node. Generally, it is used to setup a value for a variable.

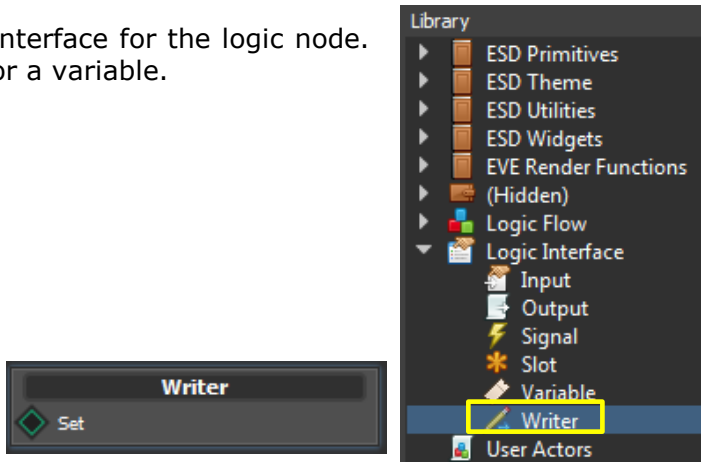


Figure 87 - Logic Interface - Writer Node

G. Property Editor

The *Property Editor* provides user a unified interface to edit properties. Each logic node in ESD 3.0, including a widget, has its own predefined properties. Users can edit the property values using the Property Editor to customize logic nodes.

The sample picture below shows an example of the ESD 3.0 widget "ESD Radio Button".

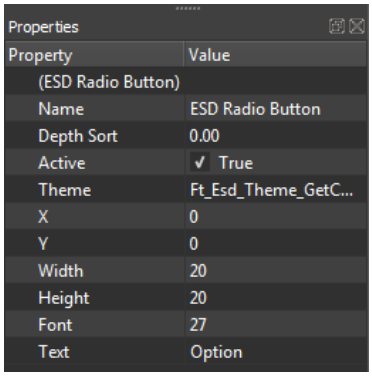


Figure 88 - Property Editor

Common Properties

While each widget has its own properties defined for its own specific features, the following are some of the properties that are commonly found in all the widgets of ESD 3.0 application -

- *Name*
- *Depth Sort*
- *Active*

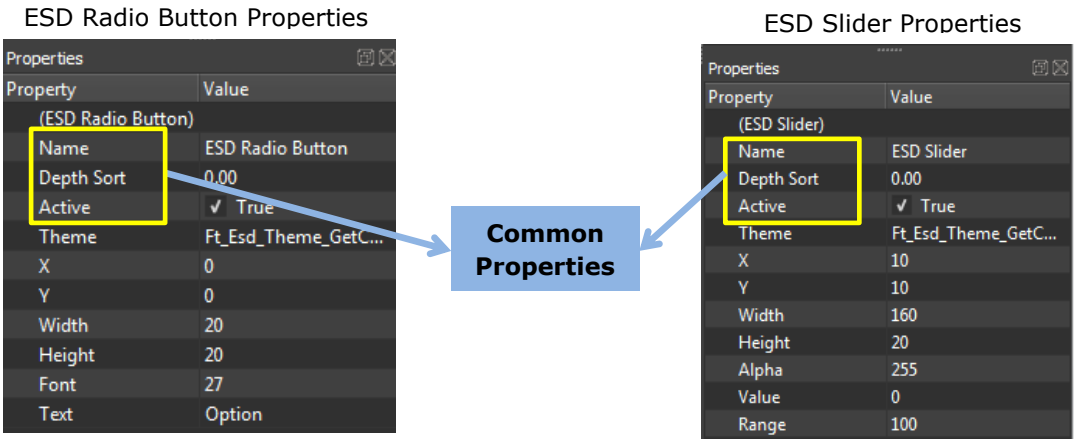


Figure 89 - ESD Common Properties

Name

Each node must have one unique *name* as an identifier. Users are required to define a name that is valid C identifier, since these names will be used by ESD 3.0 for generating the source code.

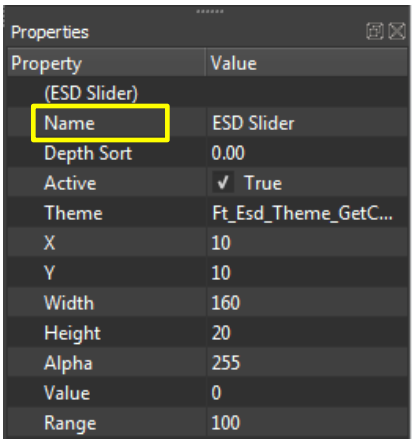


Figure 90 – Name Property

Depth Sort

Each widget and page has one common property called *Depth Sort*. It is of floating point type, and defines the sequence of rendering. The default value is "0.00" and users may change it accordingly to adjust the sequence of the rendering process.

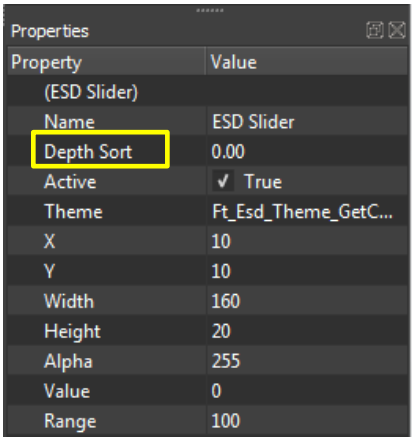


Figure 91 – Depth Sort Property

The context menu helps users adjust the rendering sequence without dealing with the values and can be accessed within the layout editor by selecting the widget or page and right-clicking it.

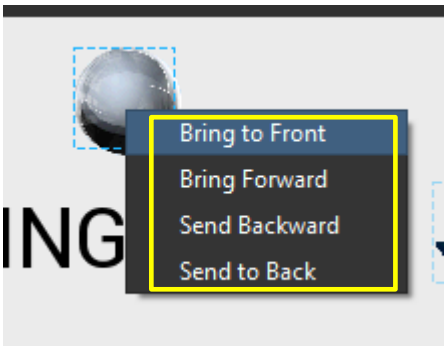


Figure 92 – Right Click Menu

The widget or page with the **greatest** positive "Depth Sort" value will be rendered **last** and appear on the **topmost** layer.

Active

Each widget has one *Active* property of Boolean type which controls whether or not it is rendered on the screen. Active property in false state does not mean that the widget is released from the memory. Instead, it means that the widget is not rendered on the screen, but still is part of the memory.

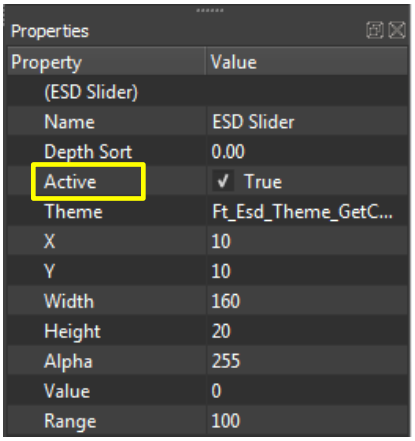


Figure 93 – Active Property

Users must ensure that the NAME property follows the standard C identifier syntax in order to make the generated source code more readable.

H. Programming Features

This section explains about the programming features. ESD 3.0 enables users to write C code and make this code available in the form of logic nodes so that the code can be re-used across multiple projects.

Macros

In order to provide the development environment with necessary information about the available functionality in a user's code, ESD 3.0 provides a set of macros. These macros are necessary and useful when users add their own code that needs to be used from the *logic node editor* or *layout designer*.

These macros are generally used together with the source code.

ESD_TYPE

ESD_TYPE provides information about a primitive type. All types are assumed to have a default value of 0.

Syntax Example:

```
ESD_TYPE(ft_uint32_t, Native = UInt32, Edit = Int)
ESD_TYPE(char *, Native = Latin1, Edit = String)
ESD_TYPE(Esd_BitmapCell *, Native = Pointer, Edit = Library)
```

Parameter Type	Parameters	Description
Native		Type used internally for formatting C literals and binary representations
	Void	Not a type (default)
	Int64, Int32, Int16, Int8,Int Ptr	Signed integer
	UInt64, UInt32, UInt16, UInt8, UIntPtr	Unsigned integer
	Double	64bit floating point
	Float	32bit floating point
	Char	An 8bit text character
	Bool	C99 Bool

	Utf8	String with UTF-8 encoding
	Latin1	String with Latin 1 (ISO8859-1) encoding
	Pointer	Pointer type
	Struct	Struct type (not fully supported yet)
Edit		Used by the Properties Editor to specify the control to edit the value and by code generation when using the fixed point formats
	None	No editor control (default)
	Int	Integer, spin-box integer control
	Real	Floating point, spin-box floating point control
	Boolean	One or Zero, checkbox
	String	Text, single-line entry
	Fixed1, Fixed2, ..., Fixed32	Fixed point, spin-box floating point control
	ColorARGB	Color, rgb with alpha
	ColorRGB	Color, rgb without alpha
	Library	Select a function specified with ESD_FUNCTION to provide the value (useful for pointer values)

Table 36 - Macros – ESD_TYPE - Parameters

ESD_ENUM

Enumeration types are usable as types from the logic editor. It will use the specified Type in the generated code. Internally, only the enumeration identifiers are parsed; values are ignored. Enumerations used in the properties are serialized as their identifier string. Enumerations are edited using a drop-down control in the *Properties Editor*.

Syntax Example:

Using pre-processor definitions, the symbol right after each *#define* statement is used.

```
ESD_ENUM(Ft_BitmapFormat, Type = ft_uint32_t)
#define PALETTED            8UL
#define PALETTED4444        15UL
#define PALETTED565         14UL
#define PALETTED8           16UL
ESD_END()
```

Parsing from an enum requires the identifiers to be on separate lines, the first identifier of each line is used.

```
ESD_ENUM(Ft_BitmapFormat)
enum Ft_BitmapFormat
{
    PALETTED,
    PALETTED565,
    PALETTED8
};
ESD_END()
```

In case the enumeration is defined in another file, a macro is available to manually specify the identifiers.

```
ESD_ENUM(Ft_BitmapFormat, Type = ft_uint32_t)
ESD_IDENTIFIER(PALETTED)
ESD_IDENTIFIER(PALETTED565)
ESD_END()
```

When the Flag value is specified, the enumeration can combine multiple values, and will show as a series of check-boxes in the Properties Editor instead of a drop-down menu.

```
ESD_ENUM(Ft_CoPro_Opt, Type = ft_uint16_t, Flags)
#define OPT_CENTERX          512UL
ESD_END()
```

To allow users to select 0 as a valid value, the Zero flag can be set. This flag has no effect when the Flag value is specified.

ESD_FUNCTION

ESD_FUNCTION is a macro to register a user defined function to ESD 3.0, so that the application can add that function into the *User Functions* category node which is located in the Library browser.

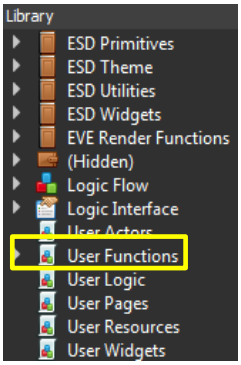


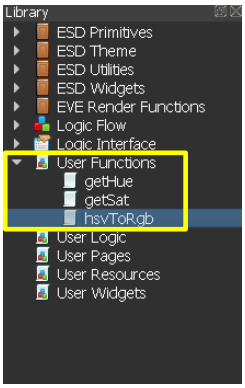
Figure 94 – User Functions Example

Syntax Example:

The following syntax example is to register a function called *"hsvToRgb"* to ESD 3.0.

```
ESD_FUNCTION(hsvToRgb, Type = ft_argb32_t )
ESD_PARAMETER(h, Type = double)
ESD_PARAMETER(s, Type = double)
ESD_PARAMETER(v, Type = double)
```

Upon registering the function, it is added to the *User Functions* category.



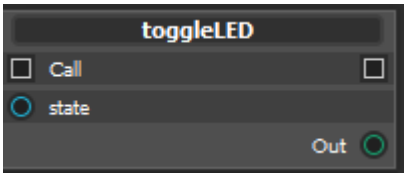
Parameters	Description
Type	The function's return value type

Table 37 - Macros – ESD_FUNCTION - Parameters

NOTE

- **Each ESD_PARAMETER defines only one function parameter. It must be kept in a single line for each parameter.**
- **"Buffered" flag can be attached after "Type". It will create the output of the function stored in a member variable, whenever the function is called. Without the Buffered flag, you'll have a function with just one output value which is called whenever the output value is used. Here is an example:**

```
ESD_FUNCTION(toggleLED, Type=int,Buffered)
ESD_PARAMETER(state,Type=ft_bool_t)
```



ESD_METHOD

ESD_METHOD works similar to ESD_FUNCTION with an enhanced feature. It takes a logic/actor/widget/page/app context pointer as first parameter. It can be used only within the logic editor of that logic/actor/widget/page/app.

Syntax Example:

The following syntax example defines one method to handle "Pushed" event for an ESD Push Button widget contained in "MainPage.page".

```
ESD_METHOD(MainPage_ESD_Push_Button_Pushed, Context = MainPage)
void MainPage ESD Push Button Pushed(MainPage *context)
{
        // ...
}
```

The logic node connection and the corresponding output is shown in the sample picture below –

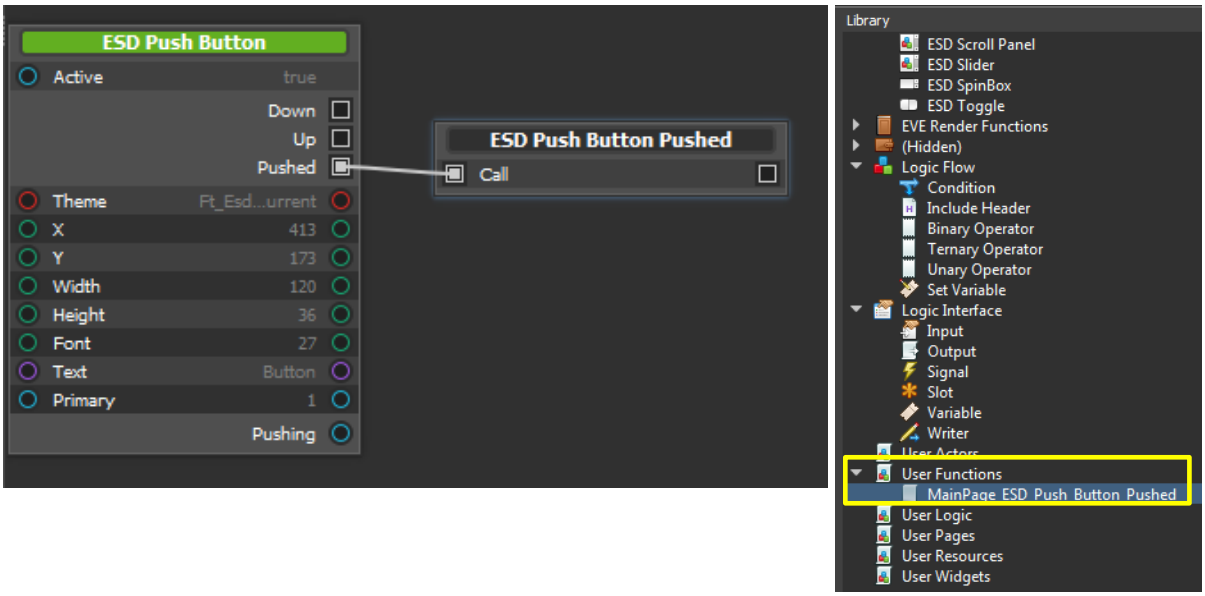


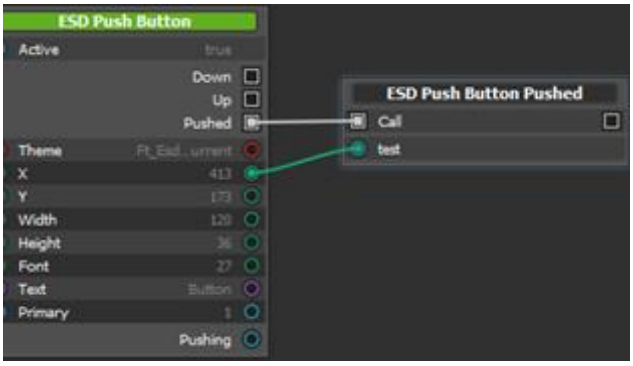
Figure 95 – ESD_METHOD Example

Parameters	Description
Context	Pointing to the caller's context

Table 38 - Meta Macros – ESD_METHOD - Parameters

NOTE

- **Additional parameters can be defined similar to ESD_FUNCTION. But these parameters must be added only after the Context parameter. Here is an example:**




```
ESD_METHOD(MainPage_ESD_Push_Button_Pushed, Context =
MainPage)
ESD_PARAMETER(test,Type = ft_bool_t)
void MainPage_ESD_Push_Button_Pushed(MainPage
*context,ft_bool_t test)
{
     // ...
}
```

NOTE

- **If the return value of method function is to be defined, "Buffered" keyword needs to be added after the"Type" definition. Refer to the syntax example -**

```
ESD_METHOD(MainPage_ESD_Push_Button_Pushed, Context =
MainPage,Type=int,Buffered)
ESD_PARAMETER(test,Type = ft_bool_t)
int MainPage_ESD_Push_Button_Pushed(MainPage
*context,ft_bool_t test)
{
     // ...
}
```

ESD_INPUT

ESD_INPUT is a built-in macro used to define an input variable of an ESD 3.0 logic node.

Syntax Example:

```
ESD_INPUT(Margin, Type = int, Default = 4)
```

ESD_OUTPUT

ESD_INPUT is a built-in macro used to define an output variable of an ESD 3.0 logic node.

Syntax Example:

```
ESD_OUTPUT(FirstPos, Type = int)
```

ESD_UPDATE

ESD_INPUT macro is used to register a function that is called repeatedly by built-in "Update" slot. This function shall not have any return value.

Syntax Example:

```
ESD_UPDATE(Ft_Esd_BitmapPersist)
ESD_PARAMETER(bitmapCell, Type = Ft_Esd_BitmapCell *)
void Ft_Esd_BitmapPersist(Ft_Esd_BitmapCell *bitmapCell);
```

Parameters:

Users can define their own parameter used by the registered function by using the ESD_PARAMETER.

Pre-compiler options

Within the ESD 3.0 C code simulation engine, the following macros are predefined in generated source code:

- *ESD_SIMULATION*
- *FT900_PLATFORM*

ESD_SIMULATION

ESD_SIMULATION macro is defined when the code is running inside the ESD 3.0 simulation engine. If some part of the user's C source code is not supposed to be run under the ESD 3.0 simulation engine, then users may choose to skip the code by using this macro.

FT900_PLATFORM

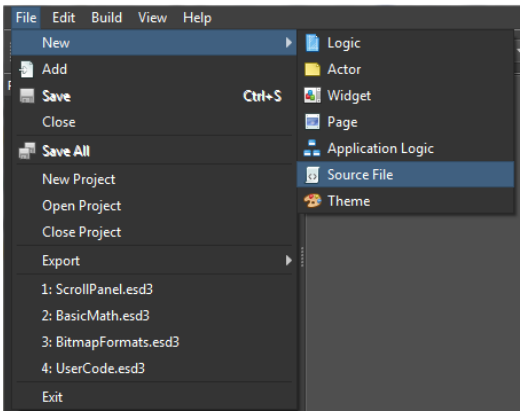
FT900_PLATFORM macro is defined when a FT900 tool-chain is selected. It can be set when the ESD_SIMULATION macro is defined.

Add User Functions

Users can define their own functions by writing the C source code directly. If these functions are defined by using the predefined "ESD_FUNCTION", they will be shown under the "User Functions" category in the library browser. Users can drag and drop these nodes into the logic node editor to use them.

Creating Source File

To add user functions, users need to create a C source file by clicking **File → New → Source File**.



Upon adding the source file, include the following header **(mandatory)** in the source file.

```
#include "Ft_Esd.h"
```

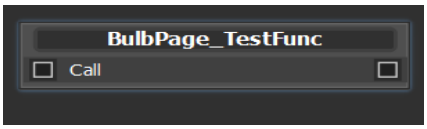
Editing the Source File

To make ESD 3.0 responsive of the user functions, the ESD 3.0 specific macro *"ESD_FUNCTION or ESD_METHOD"* need to be added before the function definition. Refer to the examples given below –

Syntax Example:

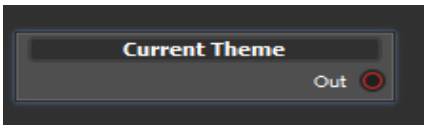
- Without parameter input and return value

```
ESD_FUNCTION(BulbPage_TestFunc)
void BulbPage_TestFunc(){}
```



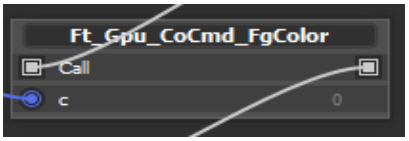
- Without parameter input, but with the return value

```
ESD_FUNCTION(Ft_Esd_Theme_GetCurrent, Type = Ft_Esd_Theme *)
Ft_Esd_Theme *Ft_Esd_Theme_GetCurrent();
```



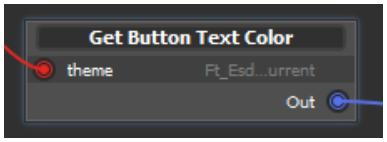
- Without return value defined, but with the parameter input

```
ESD_FUNCTION(Ft_Esd_Theme_GetButtonTextColor, Type = ft_rgb32_t)
ESD_PARAMETER(theme, Type = Ft_Esd_Theme *)
ft_rgb32_t Ft_Esd_Theme_GetButtonTextColor(Ft_Esd_Theme *theme) { //.. }
```



- Without return value and parameter input

```
ESD_FUNCTION(Ft_Esd_Theme_GetButtonTextColor, Type = ft_rgb32_t)
ESD_PARAMETER(theme, Type = Ft_Esd_Theme *)
ft_rgb32_t Ft_Esd_Theme_GetButtonTextColor(Ft_Esd_Theme *theme)
{
    //..
}
```



Appendix A – List of Figures

Appendix B – List of Tables

Appendix C – Revision History

Document Title: BRT_AN_005 EVE Screen Designer 3.0 User Guide

Document Reference No.: BRT_000081

Clearance No.: BRT#059

Product Page: http://brtchip.com/eve

Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	2017-02-09