# Contents

## A. Preface
- Purpose ........................................................................... 5
- Intended Audience ....................................................... 5
- Document Reference .................................................... 5
- Feedback ........................................................................ 5

## B. Overview
- Introduction .................................................................... 6
- Key Features .................................................................... 6
- What’s new in ESD 4.13 .................................................. 6
- Known Issues & Limitations ............................................ 6
- Terms & Description ..................................................... 7
- Credits ........................................................................... 8

## C. Setup & Installation
- System Requirements ................................................... 8
- Hardware Requirements ............................................... 9
- Dependencies / Pre-Requisites ...................................... 9
- Installing ESD 4.13 Package .......................................... 11
- Installation Folder ....................................................... 16

## D. Working with ESD 4.13
- Major features ........................................................... 18
  - Support full BT81X features ........................................... 18
  - Support for ASTC bitmap format .................................... 19
  - Support extended font format and Unicode font ............... 20
  - ASTC Encoder ............................................................ 21
  - Video Converter .......................................................... 21
  - Generate Flash file and write Flash file to device .............. 21
  - Restructured EVE HAL Library ..................................... 21
  - UI enhancements ......................................................... 22
- Impacts of ESD 4.13 updated features ......................... 22
- Migrating from ESD 3.0 to ESD 4.X project ..................... 23
- Setting Platform Specific Properties ............................... 23
  - Migrating existing projects to new platform ................... 24
- Advanced User Settings ............................................... 25
- Application Settings .................................................... 25

## E. Getting Started
- The Graphical User Interface ........................................ 27
  - Menu bar ..................................................................... 27
  - Toolbar ....................................................................... 28
  - Touch IC configuration .................................................. 30
  - Project Explorer .......................................................... 31
  - Screen Layout Editor .................................................... 31
  - Logic Node Editor ...................................................... 33
  - Library Browser .......................................................... 34
  - Error List .................................................................... 35
  - Output ......................................................................... 36
  - Property Editor ........................................................... 36
  - Status bar .................................................................... 37
- Application Project Structure ........................................ 38
Local Method ................................................................. 90
Binary Operator ............................................................. 91
Ternary Operator ........................................................... 92
Unary Operator ............................................................... 92
Sequence .................................................................... 93
Switch Node ................................................................ 93
Switch Value .................................................................. 94
Set Variable .................................................................... 94
Watch Variables ............................................................ 95
Logic Interface .............................................................. 95
Input ............................................................................. 96
Output ............................................................................. 96
Signal ........................................................................... 97
Slot ................................................................................ 97
Variable ......................................................................... 98
Writer ............................................................................ 99
Widget Interface ............................................................ 99
Property Editor .................................................................. 101
Common Properties ......................................................... 101
Name ............................................................................. 102
Depth Sort ...................................................................... 102
Active ............................................................................ 103
Programming Features ......................................................... 103
Macros .......................................................................... 103
ESD_TYPE .................................................................. 103
ESD_ENUM .................................................................. 104
ESD_FUNCTION .......................................................... 105
ESD_METHOD ............................................................. 106
ESD_VARIABLE .......................................................... 107
ESD_GLOBAL .............................................................. 108
ESD_INPUT ................................................................. 108
ESD_OUTPUT ............................................................... 108
ESD_UPDATE ............................................................... 108
Pre-compiler options ....................................................... 108
ESD_SIMULATION ....................................................... 109
FT900_PLATFORM ......................................................... 109
Add User Functions ......................................................... 109
Creating Source File ....................................................... 109
Editing the Source File .................................................... 109
F. Appendix A – List of Figures ........................................... 111
G. Appendix B – List of Tables ......................................... 113
H. Appendix C – Revision History ..................................... 114
A. Preface

Purpose

This document describes the functionality and procedures involved in using the EVE Screen Designer (ESD)

Intended Audience

The intended audience shall be any GUI application developer working with EVE products.

Document Reference

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Document Type</th>
<th>Document Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT81x Series Programmers Guide</td>
<td>Programming Guide</td>
<td>PDF</td>
</tr>
<tr>
<td>FT81x Datasheet</td>
<td>Datasheet</td>
<td>PDF</td>
</tr>
<tr>
<td>FT9xx Toolchain Installation Guide</td>
<td>Installation Guide</td>
<td>PDF</td>
</tr>
<tr>
<td>BT81X Series Programming Guide</td>
<td>Programming Guide</td>
<td>PDF</td>
</tr>
<tr>
<td>BT815/6 Datasheet</td>
<td>Datasheet</td>
<td>PDF</td>
</tr>
<tr>
<td>BT817/8 Datasheet</td>
<td>Datasheet</td>
<td>PDF</td>
</tr>
</tbody>
</table>

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/.
B. Overview

Introduction

**EVE Screen Designer (ESD)** is the next generation of 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 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 to experience the UI before compiling and downloading the generated source code. Furthermore, ESD has the capability to work seamlessly with Bridgetek FT9XX tool chain. Users can compile, link the generated source code with FT9XX tool chain and upload it to the targeted platform without leaving ESD. From 4.13 onward, ESD has the same capability as FT90X platform to work with Raspberry Pi Pico\(^2\).

A layout mechanism is introduced to manage widgets and pages in a more generic way. The layout mechanism will enable users to create more dynamic UI much easier than before. In addition, ESD 4.13 dramatically enhances the functionality of the logic nodes editor, layout editor and project browser, for better user experience.

Key Features

The following are some of the key features of **EVE Screen Designer**:

- **WYSIWYG GUI**
  - High level widgets
  - No EVE display list knowledge required
  - Widget based GUI construction
  - Drag and drop widget to create screen layout
  - Inter 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 FT9XX/pico Toolchain is installed)

What’s new in ESD 4.13

**New Features:**

- Add new host platform support: Raspberry Pi Pico
- Add feature to export for Raspberry Pi Pico platform
- Add SD card support for Raspberry Pi Pico platform
- Add new Eve module support: Gameduino 3X dazzler

Known Issues & Limitations

The following are some known issues and limitations of ESD:

- Only the BT81X/FT81X series EVE is supported. FT80X Series EVE is **NOT** supported.
- The C code editor in ESD does not support all the features of a code editor.
- If the project file path is too long (i.e. more than 512 bytes), ESD may have a problem opening it. The typical error message is shown below:

  “Unable to generate output files, check directory permission at: C:\Users\xxxx.xxxx\Project\ESD\......”

  Where

  “C:\Users\xxxx.xxxx\Project\ESD\......” refers to the project folder.

- Logic node editor background goes white after system hibernates.
- In some unusual 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.

- In few cases, users may need to click **Recompile** at the toolbar to update the frozen simulation result.
- For Gameduino 3X dazzler platform, game controller is not supported yet, and there is no way for user to interactive with ESD project, and SD card is a must.

---

**Terms & Description**

<table>
<thead>
<tr>
<th>Abbreviations/Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 | EVE Screen Designer
EVE Emulator | Bridgetek behaviour-modelling software for EVE Series chip
HAL | Hardware Abstraction Layer is a software subsystem providing hardware abstraction.
IDE | Integrated Development Environment
Layout type widget | One type of widget which has no visual appearance rendered by EVE, but manages the associated widget
Logic Node | The node expressing certain logic. (Also referred to simply as “node” in this document)
Logic Node Editor | The place where you create logic by connecting the logic node
Page | One single screen in design
Simulation | Preview the project or page by running the generated C code on PC
Widget | One type of logic node which has visual appearance rendered by EVE

Credits

Open Source Software

- Errorlist module: [https://github.com/kaetemi/errorlist](https://github.com/kaetemi/errorlist) available under MIT license
- QScintilla: part of PyQt available under PyQt commercial license [https://www.riverbankcomputing.com/commercial/license-faq](https://www.riverbankcomputing.com/commercial/license-faq)

Icons Copyright

Some of the icons used in ESD 4.13 are from: [http://p.yusukekamiyamane.com/icons/search/fugue/](http://p.yusukekamiyamane.com/icons/search/fugue/)

They are used in compliance with the Creative Commons Attribution 3.0 License.

C. Setup & Installation

System Requirements
To install ESD 4.13 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 1080x800 pixels or higher
- “Write” permission to the installation folder

### Hardware Requirements

The exported C source code from ESD is targeted at **EVE based** platform, which typically consists of an Eve module and a host (MCU or PC based module).

The supported platforms are listed as below:

<table>
<thead>
<tr>
<th>Platform Name</th>
<th>Compatible EVE Module</th>
<th>Compatible MCU Module Name</th>
<th>Compatible PC based Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME812A WH50R</td>
<td>ME810A-WH70R(800x480)</td>
<td>MM900EV-LITE</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>ME811A-WH70R(800x480)</td>
<td>MM900EV1A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ME812A-WH50R(800x480)</td>
<td>MM900EV2A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ME813A-WH50C(800x480)</td>
<td>MM900EV3A</td>
<td></td>
</tr>
<tr>
<td>ME810A HV35R</td>
<td>ME810A-HV35R(320X480)</td>
<td>Raspberry Pi Pico</td>
<td>NA</td>
</tr>
<tr>
<td>VM816CU50A</td>
<td>VM816C (800x480)</td>
<td>NA</td>
<td>FT4222 (libFT4222.dll)</td>
</tr>
<tr>
<td>ME813AU WH50C</td>
<td>ME813AU WH50R(800x480)</td>
<td>NA</td>
<td>FT4222 (libFT4222.dll)</td>
</tr>
<tr>
<td>VM816C50A</td>
<td>VM816C (800x480)</td>
<td>NA</td>
<td>MPSSE (libMPSSE.dll)</td>
</tr>
<tr>
<td>ME817EV WH70C</td>
<td>ME817EV (1024X600)</td>
<td>MM900EV-LITE</td>
<td>FT4222 (libFT4222.dll)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MM900EV1A</td>
<td></td>
</tr>
<tr>
<td>ME817EV WH10C</td>
<td>ME817EV (1280X800)</td>
<td>MM900EV2A</td>
<td>FT4222 (libFT4222.dll)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MM900EV3A</td>
<td></td>
</tr>
<tr>
<td>GD3X_Dazzler</td>
<td>GameDuino 3X Dazzler (HDMI output 1280x7020)</td>
<td>Raspberry Pi Pico</td>
<td>NA</td>
</tr>
</tbody>
</table>

⚠️ **While working on hardware platform (FT90X modules or Raspberry Pi Pico), users must ensure that an SD card is inserted into the SD Slot since ESD will try to detect the presence of an SD card. If the SD card cannot be detected, then the application may fail.**

### 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:
- **Windows 10 Universe C Runtime**

ESD 4.13 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](https://www.microsoft.com/en-us/download/details.aspx?id=48234) and install on your PC should the following problem be encountered:

![Screen Designer - System Error](image)

*Figure 1 - Screen Designer - System Error*

- **FT9XX Tool Chain Version 2.4.0 or later**

To compile and build projects, the FT9XX Tool Chain 2.4.0 must be installed on the PC. It is downloadable from [http://brtchip.com/ft9xx-toolchain/](http://brtchip.com/ft9xx-toolchain/).

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

Users are advised to check the known issues and limitations (of FT9XX Toolchain) while building the ESD 4.13 project with FT9XX Toolchain. The respective FT9XX Toolchain package version release note contains the list of known issues and limitations.

For ESD 4.13, we recommend users to install FT9XX Tool Chain version 2.5.0 for the best results.

- **Pico Tool Chain**

To compile and build Pico projects, several tool chains must be installed on the PC. They are: pico-sdk, python 3.8, Microsoft visual studio/visual code and cmake. They can be downloaded/cloned from the following:

- Pico-sdk - [https://github.com/raspberrypi/pico-sdk](https://github.com/raspberrypi/pico-sdk)
- Python 3.8 - [https://www.python.org/downloads/](https://www.python.org/downloads/)
- Visual Code - [https://code.visualstudio.com/](https://code.visualstudio.com/)
- C make - [https://cmake.org/download/](https://cmake.org/download/)

ESD will check system environment to find these tool chains, and it also provides a user dialog to configure the tool chain path under **Tools** menu:
Installing ESD 4.13 Package

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


ii. When prompted with a download dialog box. Click [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 4.13.exe

v. The EVE Screen Designer 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. In the Select Components window, check “Install ESD essential Microsoft VC++ redistributables (internet required)” boxes, to install runtime components of Visual C++ Libraries. User can select the custom installation to skip the installation of the Visual C++ library redistributable. Internet connection is not needed for custom installation. Click [Next] to prepare for the installation.
ix. In the **Select Additional Tasks** window, check “**Create a desktop / Create Quick Launch shortcut**” boxes, to have the ESD 4.13 icon and Quick Launch shortcut displayed on the desktop if required. 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. First page progress bar indicates the progress of the download of the MSVC C++ redistributable and the second page progress bar indicates that the installation is in progress.
xii. In the **Information** page, Click **[Next]** to proceed ahead.

xiii. Upon successful installation, click **[Finish]**.
Installation Folder

The following table provides a list of folders that can be found under the installation path:

<table>
<thead>
<tr>
<th>Folder Name</th>
<th>Description</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples</td>
<td>The example projects created by ESD</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Imageformats</td>
<td>Qt run-time DLLs for image format supporting</td>
<td>Read-Only</td>
</tr>
<tr>
<td>Styles</td>
<td>Qt run-time DLLs for UI styles</td>
<td>Read-Only</td>
</tr>
<tr>
<td>Libraries</td>
<td>Widget library, application framework and Hardware abstraction layer</td>
<td>Read-Only</td>
</tr>
<tr>
<td>Log</td>
<td>Stores the runtime logs for debug purpose</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Manual</td>
<td>Contains this document</td>
<td>Read-Only</td>
</tr>
<tr>
<td>Platforms</td>
<td>Qt platform specific run-time DLLs</td>
<td>Read-Only</td>
</tr>
<tr>
<td>Settings</td>
<td>Configuration files for third-party utilities and tool chains as well as ESD settings. The files are in XML format.</td>
<td>Read-Only. Reserved for advanced users to change.</td>
</tr>
<tr>
<td>Templates</td>
<td>The template files used by ESD</td>
<td>Read-Only</td>
</tr>
<tr>
<td>TinyCC</td>
<td>TinyCC run-time used for ESD simulation purpose.</td>
<td>Read-Only</td>
</tr>
<tr>
<td>Tools</td>
<td>The utilities used in ESD for bitmap and font data generation purpose</td>
<td>Read-Only</td>
</tr>
</tbody>
</table>

**Table 1 - Installation Folder**
D. Working with ESD 4.13

Major features

Support full BT81X features

Users can refer to EVChargePoint examples in the path `<ESD Folder>/Examples/Advanced/EVChargePoint`. Opening it may take more than 5 minutes because of ASTC, bitmap and font conversion. This example contains all of the new BT81X features. Here are the details:

1. Apply ASTC Bitmap Format for images and fonts.

![Figure 2 - ASTC applied for images and fonts](image)

2. Resources are stored and loaded from flash.

![Figure 3 - Load resources from flash](image)
3. Use Unicode font with ASTC format and predefined character set

Figure 4 - Use Unicode font with pre-defined character set

Figure 5 - Chinese font used in application

Support for ASTC bitmap format

In ESD 4.13, images can be converted to ASTC bitmap format to save significantly on storage space while image quality is still retained.

Figure 6 - ASTC Format
Support extended for font format and Unicode font

Unicode font file can be added to a project by clicking **File -> Add** from the menu bar and in the **Property** window, input font size and character set as desired.

![Figure 7 - Font size and Character set](image)

Once the font resource is ready, add a widget, such as ESD Label, ESD Button to project. In **Property** window, select appropriate **Font Resource**.

![Figure 8 - Select Unicode Font Resource](image)

![Figure 9 - Unicode font applied for Label and Button](image)
ASTC Encoder

In ESD 4.13, the ASTC encoder is upgraded to version 2.0.

This is a new version of ASTC encoder, which is responsible for the image conversion process to ASTC format.

It is from ARM and claims 30% faster performance than ASTC encoder 1.0.

Video Converter

ESD enables users to add video format into project and converts it to Eve specific format by employing ffmpeg.

It eases the procedure to add the video playback functionality in ESD project.

In ESD examples, a demo project PlayVideoMp4 is added to play an mp4 video.

Generate Flash file and write Flash file to device

Resources can be loaded from Eve connected flash or RAM_G. ESD can generate Flash file for BT81X projects, then uploads the Flash file to the device by pressing the [Build and Upload to Hardware] button.

Restructured EVE HAL Library

EVE HAL Library is simplified. It contains C source files, and a sub-folder “Hdr” that comprises of the C header files.
UI enhancements

The following User Interface related enhancements has been added to ESD 4.13 –

- Filter for Library Browser and Project Browser window
- More comprehensive toolbar buttons for platform description
- New menu to clean up unused/generated files in project
- Simplify the process of adding new resources into project
- Added context menu for document tab

Impacts of ESD 4.13 updated features

Since the new/updated features of ESD 4.13 may cause API changes, existing ESD 3.0 projects are not fully compatible with ESD 4.13. Although ESD 4.13 has the utility inside to detect and migrate opening ESD 3.0 project, it may still not work as expected.

- **ESD Layouts**: A new category of widgets “ESD Layout” are introduced into the Library Browser, which enables the layout feature.

- **ESD Widgets**: A new set of widgets are introduced into “ESD Widgets” category in the library. These set of widgets are to be used in constructing the pages. The nodes under the “ESD Render Functions” are not supposed to construct page as it has no widget interface and the layout widget is unable to manage them.
Migrating from ESD 3.0 to ESD 4.X project

When ESD 4.X opens an existing ESD 3.0 project, it will prompt users to migrate to the latest version. If user chooses not to migrate, ESD 4.X will not open ESD 3.0 project.

MIGRATION NOTES

1. Please back up your project if you are not sure about the migration. The migration process will **overwrite the project and it cannot be reversed.**
2. The migrated project file will be renamed from “*.esd3” to “*.esd”.
3. The target module of the opened project will be redirected to two built-in platforms after migration. So if the target module is in 320 x 480 resolutions, users may select the platform “ME810A HV35R” from “Build Target” combo box from the toolbar and make a platform switch.

In some cases, if an existing ESD 3.0 project cannot be migrated to ESD 4.X, then users are required to migrate manually.

- The variable “Parent” of logic is renamed to “Owner” in ESD 4.X.
- *X, Y, Width, Height* properties in widgets is now accessible through the widget variable.
- Instead of pointer, struct “Ft_Esd_BitmapCell” shall be used by value in ESD 4.X.
- In order to restore the default behaviour of application logic "App.main", users may need to manually move the existing pages into layout widget "Switch Page" in the Project Browser.

Setting Platform Specific Properties

ESD users are able to create one project supporting multiple platforms with various screen size. Therefore, on different platforms, the same widget may have a different property value. For instance, for a platform with screen size 480 x 320, the button size may shrink to a smaller size from the original value for a platform with screen size 800 x 480.

Users can make a property value specific to the current target through the context menu of the property browser. This can be done by right clicking the selected property of the current widget. This is illustrated in the picture given below –
User can remove value specific to current target through context menu in property browser by right clicking the selected property of current widget. This is illustrated in the picture given below -

Figure 13 - Removing Target Specific Value

Users can refer to the “ScreenResolution” project under the “Installation/Example” folder.

**Migrating existing projects to new platform**

All the existing projects may have been developed using a specific platform. Users can choose to migrate their projects to a different platform by selecting the appropriate “Build Target” combo box. If the desired target is not available in the “Build Target” Combo box, then the user can configure the individual specifications (“Eve Platform”, “Display Resolution” and “Host Platform”) of the target by selecting the appropriate combo boxes.
Advanced User Settings

Advanced users are allowed to configure the ESD settings to better assist them in their GUI application development with EVE products. The Advanced User Settings allow the user to turn on/off specific features and functions in the ESD. This section shall provide a brief explanation on the usage and effect on the ESD. As it involves modifying the application configuration file, it should only be carried out by someone with adequate knowledge. It is recommended to leave any advanced settings at their default settings.

Application Settings

The “Main.config” file can be found in the “Settings” folder under the installation path. This file contains application wide settings which may affect the interface and behaviour of the ESD. Users are allowed to modify to suit their application needs. After the modification has been done, a restart on the application is required for it to take effect.

Below is the list of settings that can be found in the file.

- **PortableSettings** - Enabling/Disabling loading of Host Device Environment settings from the from Screen Designer config upon startup. The loaded settings will be reflected in “Config Host Device”.

- **SimplifiedExport** - Enabling/disabling of the simplified export menu actions. An “Export” option will be displayed under the menu bar if it is disabled. The “Export” option will be shown under file menu if it is enabled.

- **Hiddentypes** – Hide types from library and menu interface. The listed parameters under the “HiddenTypes” will be hidden.
  - **_GroupHidden** – A new set of commands will be made available to user when it is removed from the hidden state. These controls are mainly to be used by advanced users.
  - **_WidgetDocument** – Allows the user to create new customised widget.
- **_LogicIncludeHeader** – This allows the user to use “include header” logic node under the library which helps to add a header file into project.

- **_LogicWatcher** – This allows the user to access “Watch Variables” logic node which helps to watch multiple variables for change.

- **_SourceFile** – This allows the user to add new empty .c file to the project.
  - **EnableInspector** – Enabling of Inspector window which is useful for debugging.
  - **ShowBitmapHandleUsage** – Displaying of bitmap handle usage in status bar.
  - **Keywords** – List of keywords to be avoided when naming ESD file object. An error display message box will be shown if there is a conflict with the reserved keyword.
E. Getting Started

The Graphical User Interface

ESD 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/Inspector
8. Output window
9. Property Editor
10. Status Bar

Figure 14 - EVE Screen Designer User Interface Components
Menu bar

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

![Menu bar image](image-url)

**Figure 15 - Menu bar**

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

<table>
<thead>
<tr>
<th>Menu</th>
<th>Submenu</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>New</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Actor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Widget</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Theme</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source Files</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Logic Source Files</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td></td>
<td>To add existing file(s)/resource (such as bitmap, C source file etc.) to the project</td>
</tr>
<tr>
<td>Save</td>
<td></td>
<td>To save the current file</td>
</tr>
<tr>
<td>Close</td>
<td></td>
<td>To close the current file</td>
</tr>
<tr>
<td>Save All</td>
<td></td>
<td>To save all the files in the current project</td>
</tr>
<tr>
<td>New Project</td>
<td></td>
<td>To open a new project</td>
</tr>
<tr>
<td>Open Project</td>
<td></td>
<td>To open an existing project</td>
</tr>
<tr>
<td>Close Project</td>
<td></td>
<td>To close a current project</td>
</tr>
<tr>
<td>Clean And Close Project</td>
<td></td>
<td>To clean current project then close it</td>
</tr>
<tr>
<td>Export as Eclipse Project</td>
<td>To export as Eclipse project. This will only be shown when supported platform is selected</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Export as MSVC Project</td>
<td>To export as MSVC project. This will only be shown when supported platform is selected</td>
<td></td>
</tr>
<tr>
<td>Export as Pico Project</td>
<td>To export as Pico project. This will only be shown when supported platform is selected</td>
<td></td>
</tr>
<tr>
<td>1:&lt;Project Path&gt;</td>
<td>To open recent projects, up to 10 projects</td>
<td></td>
</tr>
<tr>
<td>Exit</td>
<td>To close the ESD application</td>
<td></td>
</tr>
<tr>
<td>Edit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undo</td>
<td>To reverse the action of an recently performed action</td>
<td></td>
</tr>
<tr>
<td>Redo</td>
<td>To revert the effects of the undo action</td>
<td></td>
</tr>
<tr>
<td>Cut</td>
<td>Cut the selected node(s) to clipboard</td>
<td></td>
</tr>
<tr>
<td>Copy</td>
<td>Copy the selected node(s) to clipboard</td>
<td></td>
</tr>
<tr>
<td>Paste</td>
<td>Paste the node(s) from clipboard</td>
<td></td>
</tr>
<tr>
<td>Find</td>
<td>Find a specific text string in source editor</td>
<td></td>
</tr>
<tr>
<td>Replace</td>
<td>Replace a specific text string in source editor</td>
<td></td>
</tr>
<tr>
<td>Build</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build Executable</td>
<td>To generate an executable file</td>
<td></td>
</tr>
<tr>
<td>Build and Upload to Hardware</td>
<td>To generate an executable file and upload to hardware</td>
<td></td>
</tr>
<tr>
<td>Browse to Executable</td>
<td>To navigate to the folder under which the executable file is located</td>
<td></td>
</tr>
<tr>
<td>Tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate Class ID</td>
<td>To generate a Class ID for user widget or layout. It is a hash from the widget name and generally useful when user prefers to write code in “C” language.</td>
<td></td>
</tr>
<tr>
<td>Clean Unused Generated Files</td>
<td>To clean the generated files which are not in use</td>
<td></td>
</tr>
<tr>
<td>Configure cross-compilation Environment</td>
<td>To show the current tool chain path for the host device, and user can change the paths according to their own configuration.</td>
<td></td>
</tr>
<tr>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The **Toolbar** provides easy access to common functions (in the form of icons) such as *new file, save file, undo, redo* etc.

<table>
<thead>
<tr>
<th>Toolbar Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>New</strong></td>
<td>To open new Logic/Actor/Widget/Page/Theme/Source Files/Logic Source Files</td>
</tr>
<tr>
<td><strong>Add</strong></td>
<td>To add existing resource (such as bitmap, C source file etc.) to the project</td>
</tr>
<tr>
<td><strong>Save</strong></td>
<td>To save the currently open file</td>
</tr>
<tr>
<td><strong>Save All</strong></td>
<td>To save all the files in the current project</td>
</tr>
<tr>
<td><strong>Undo</strong></td>
<td>To reverse the action of a recently performed action</td>
</tr>
<tr>
<td><strong>Redo</strong></td>
<td>To revert the effects of the undo action</td>
</tr>
<tr>
<td><strong>Simulation</strong></td>
<td>A toggle button which starts or stops the simulation mode.</td>
</tr>
</tbody>
</table>

*This state indicates that the ESD 4.13 is in simulation mode.*
Clicking this button stops the simulation.

- This state indicates that the ESD 4.13 is out of simulation mode. Users can drag/drop widgets or edit them safely.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restart</td>
<td>To automatically restart the EVE emulator. Clicking this button will force</td>
</tr>
<tr>
<td></td>
<td>the simulated screens to be re-drawn.</td>
</tr>
<tr>
<td>Recompile</td>
<td>To recompile the whole project’s source code using ESD built-in TinyCC</td>
</tr>
<tr>
<td></td>
<td>compiler. It is a mandatory procedure for simulation.</td>
</tr>
<tr>
<td>Platform Target</td>
<td>Select the hardware platform as source code building target.</td>
</tr>
<tr>
<td>EVE Module Type</td>
<td>To select EVE module type</td>
</tr>
<tr>
<td>Screen Size</td>
<td>To set screen size</td>
</tr>
<tr>
<td>MCU Platform Type</td>
<td>Select the building configuration when toolchain is invoked to build</td>
</tr>
<tr>
<td>Flash Chip Type</td>
<td>To select flash chip for BT81x based series platform</td>
</tr>
<tr>
<td>Touch IC</td>
<td>To select capacitive touch controller type</td>
</tr>
<tr>
<td>Build Executable</td>
<td>Build the project files</td>
</tr>
<tr>
<td>Toolchain</td>
<td>Select toolchain to build</td>
</tr>
<tr>
<td>Build and Upload</td>
<td>Upload Flash file to device and/or run executable file</td>
</tr>
<tr>
<td>to Hardware</td>
<td></td>
</tr>
<tr>
<td>Programmer</td>
<td>Select executable mode</td>
</tr>
</tbody>
</table>

**Table 3 - Toolbar & Description**

The red exclamation mark in the image above means those MCU types are not applicable to the currently selected platform.

**Touch IC configuration**

FT811, BT815 & BT817 work with capacitive touch controllers. Currently, only the following capacitive touch controllers are supported by EVE:

- **FocalTech**: FT5xx6 / FT6xx6 series IC
- **Goodix**: GT911/GT9271 series IC.

ESD users can configure the touch IC controller by using the touch IC configuration combo box as shown below.

More information on the touch controllers can be found [here](#).
Figure 19: CTP configuration

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 20 - Project Explorer window

In case of many items existing in a project, user can use filter textbox in order to get a sort list of items. Text that matches the filter is highlighted. Regular expressions are supported.

Figure 21 - Project Explorer Filter
Screen Layout Editor

The Screen Layout Editor 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.

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”. For the other kinds of logic nodes, if the EVE rendering process is not defined, screen layout editor does not allow them to be dropped in.
Users can right click on tab bar to active a context menu. Those menu items are described in table below.

<table>
<thead>
<tr>
<th>Menu</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close tab</td>
<td>Close the current tab</td>
</tr>
<tr>
<td>Close other tabs</td>
<td>Close all tabs except the current tab</td>
</tr>
<tr>
<td>Close all tabs</td>
<td>Close all tabs that were opened</td>
</tr>
</tbody>
</table>

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

![Library Browser](image)

**Figure 26 - Library Browser**

The following table provides the ESD library name and its description –

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESD Layouts</td>
<td>ESD built-in layout widgets and relevant utilities</td>
</tr>
<tr>
<td>ESD Render Functions</td>
<td>ESD built-in Render functions</td>
</tr>
<tr>
<td>ESD Theme</td>
<td>ESD built-in basic theme manipulation functions</td>
</tr>
<tr>
<td>ESD Utilities</td>
<td>ESD built-in utilities</td>
</tr>
<tr>
<td>ESD Widgets</td>
<td>ESD built-in widgets</td>
</tr>
<tr>
<td>Logic Flow</td>
<td>ESD built-in logic node for control flow</td>
</tr>
<tr>
<td>Logic Interface</td>
<td>ESD built-in logic node for interface</td>
</tr>
<tr>
<td>User Actors</td>
<td>User defined actor logic node. It is empty by default.</td>
</tr>
<tr>
<td>User Functions</td>
<td>User defined functions. It is empty by default.</td>
</tr>
<tr>
<td>User Layouts</td>
<td>User defined layouts. It is empty by default.</td>
</tr>
<tr>
<td>User Logic</td>
<td>User defined logics. It is empty by default.</td>
</tr>
<tr>
<td>User Pages</td>
<td>Pages added by user. It is empty by default if no page is created.</td>
</tr>
<tr>
<td>User Resources</td>
<td>Resources added by users (For example: bitmap). It is empty by default.</td>
</tr>
<tr>
<td>User Widgets</td>
<td>Widgets created and added by users. It is empty by default.</td>
</tr>
</tbody>
</table>

**Table 4 - ESD Libraries**
Error List

The Error List is a dock window which shows the message output from ESD 4.13, 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.

![Error List Window](image)

Figure 27 - Error List Window

Output

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

![Output Window](image)

Figure 28 - 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.
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.

![Property Editor (ESD Push Button)](image)

**Figure 29 - Property Editor (ESD Push Button)**

Status bar

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

![Status bar](image)

- The percentage of bitmap handles used by the current page
- The percentage of RAM_G memory used by the current page
- Simulation status (whether in progress or not)
- The compilation status of the generated “C” code. Project cannot be simulated if the compilation fails.

**Figure 30 - Status bar**
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 the 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 the widgets and pages is therefore required to be routed through the hierarchical chain. (Only node connections exist between the directly connected nodes in the hierarchy.)

![Application Project Structure Diagram]

**Figure 31 - Application Project Structure**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.esd</td>
<td>Project file - This XML file describes what files are included in this project. This file is unique for the whole project.</td>
</tr>
<tr>
<td>*.main</td>
<td>Logic file - This XML file describes the application level logic. By default, it is named as “App.main” and is unique project wide.</td>
</tr>
<tr>
<td>*.page</td>
<td>Page Node Definition file - This XML file describes what widgets are contained and connected in each page. By default: “AppScreen.Page” is added, where a switch page may be included for control on page transition. A page can contain other pages, but switch page is required for page transition control.</td>
</tr>
<tr>
<td>*.widget</td>
<td>Widget Node Definition file. This XML file describes a widget detail. A widget can contain other nodes, which may include widget node, actor node and logic node.</td>
</tr>
<tr>
<td>*.actor</td>
<td>Actor Node Definition file. This XML file describes the actor node details.</td>
</tr>
<tr>
<td>*.logic</td>
<td>Logic Object Definition file. This XML file describes the logic node details.</td>
</tr>
<tr>
<td>*.theme</td>
<td>Application Theme Definition file. This XML file describes the theme detail.</td>
</tr>
</tbody>
</table>
### Application Workflow

The workflow given below illustrates how to create an EVE based GUI application using ESD 4.13.

![Application Workflow Diagram](image)

**Figure 32 - Application Workflow**

The 5 main modules that comprise the application workflow are the *Screen Layout Editor*, *Logic Node Editor*, *Source Generator*, *JIT Compiler* and *EVE Emulator*. Among these, *Source Generator*, *JIT Compiler* and *EVE Emulator* are the back-end modules which have no UI exposed to end user. The compiler is a JIT style C compiler, i.e. TinyCC compiler, which compiles and links the generated code into executable. The executable is invoked by the emulator module for simulation.

---

**Note:** The file names listed above shall **NOT** be conflicting with the C standard library function as well as windows header files. Here are some examples: `fopen`, `windows`, `winbase`, `math`, `printf`, `realloc`. ESD 4.13 simulation process may give errors and fail, if user gives the file name of their widgets, logics, images, or any resources with these reserved names. Therefore, ESD 4.13 enforces the ‘Usr_’ prefix to all the new/renamed widgets, pages, functions, actors, as well as source file, theme, logic source file etc, in order to ensure that the simulation process succeeds without error.
The application workflow starts by design layout (1a) or design logic (1b); Upon user saving the changes, it will trigger C code generation (2), then the compiler will try to build the generated C code; if any error encountered during building, then user needs to continue on design layout (4b) or design logic (4c); or if no error found, then the emulator will start simulating (4a) the project. Users will be able to upload (5b) the binary to the EVE chips or export (5c) the project as eclipse C project.

**Design 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 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 how they look like together.

**Custom Widget**

ESD 4.13 provides a set of built-in widgets for users to construct screen layouts. Additionally, if the built-in widgets does 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).*

![Create New Project](image1)

b. Click *File → Add* from the menu.

![Add Bitmap Resource](image2)
c. Browse and select the image file(s) from the file explorer window.

Upon adding the file, users can drag and drop the ESD Image widget to the desired page and select the bitmap cell (which is default at cell 0) 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.

![Property Editor](image)

*Figure 33 - Property Editor*
<table>
<thead>
<tr>
<th>Property</th>
<th>Value/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>Name of the image file</td>
</tr>
<tr>
<td>Name</td>
<td>Resource Type Name. In this case, the selected resource type is Bitmap.</td>
</tr>
<tr>
<td>Category</td>
<td>Category of the selected object</td>
</tr>
<tr>
<td>Format</td>
<td>Image Format – The target format of bitmap can be –</td>
</tr>
<tr>
<td></td>
<td>• L1/L2/L4/L8</td>
</tr>
<tr>
<td></td>
<td>• RGB332/RGB565</td>
</tr>
<tr>
<td></td>
<td>• ARGB1555/ARGB2/ARGB4</td>
</tr>
<tr>
<td></td>
<td>• PALETTE4444/PALETTE565/PALETTE8</td>
</tr>
<tr>
<td></td>
<td>• DXT1</td>
</tr>
<tr>
<td></td>
<td>• JPEG (Decoded into RGB565 format only)</td>
</tr>
<tr>
<td></td>
<td>• PNG (Decoded into RGB565 format only)</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_4x4_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_5x4_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_5x5_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_6x5_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_6x6_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_8x5_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_8x6_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_8x8_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_10x5_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_10x6_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_10x8_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_10x10_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_12x10_KHR</td>
</tr>
<tr>
<td></td>
<td>• COMPRESSED_RGBA_ASTC_12x12_KHR</td>
</tr>
<tr>
<td>Compressed</td>
<td>Set bitmap compression on/off</td>
</tr>
<tr>
<td>Flash</td>
<td>Set this to store the resource in the Flash</td>
</tr>
<tr>
<td>CellHeight</td>
<td>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. Cell Height default at 0 which means there will be only one cell no matter how large the bitmap it is.</td>
</tr>
<tr>
<td>CellNames</td>
<td>This field will enable system to create a list of n number of user defined cell names instead system generated which is suffixed with “_x”.</td>
</tr>
<tr>
<td>Persistent</td>
<td>Keeps the memory persistent even if the bitmap is not displayed when it is true.</td>
</tr>
<tr>
<td>Width</td>
<td>The width of bitmap image in pixels</td>
</tr>
<tr>
<td>Height</td>
<td>The height of bitmap image in pixels</td>
</tr>
<tr>
<td>Prefer RAM</td>
<td>For ASTC images, resources can be rendered directly from Flash without the need to store it to RAM_G. Set Prefer RAM (given Flash flag is set) to load ASTC image from Flash directly. If it is not set, Resources will be rendered from RAM_G. Prefer RAM feature is exclusively available for ASTC format alone.</td>
</tr>
</tbody>
</table>

**Table 5 - Bitmap Resource Properties**
Design Screen Logic using Logic Node Editor

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

During this stage, users can decide the behaviour mode of both inner-page 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 "AppScreen.page". Users are required 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.

Simulation

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 (AppScreen.page or App.main) is selected, users can simulate the whole project by clicking [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 the application directly.

To validate the behaviour of single page, users need select the page file and open it with logic node editor before clicking the [Simulation] button on the toolbar. Similarly, users can simulate the behaviour of other logic node, such as widget, actor by opening it with logic node editor and clicking [Simulation] button on the toolbar.

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

<table>
<thead>
<tr>
<th>Simulation State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>ON State - Indicates that the simulation is in progress. Clicking on it will switch off the simulation. In this state, users will not be able to edit the page or project.</td>
</tr>
<tr>
<td>[ ]</td>
<td>OFF State - Indicates that currently simulation is not in progress. Clicking on it will switch on the simulation.</td>
</tr>
</tbody>
</table>

Target selection

ESD 4.13 provides support for the following built-in build targets -

1. ME812A WH50R
2. ME810A HV35R
3. ME813AU WH50C
4. VM816CU50A
5. VM816C50A
6. ME817EV-WH10C
7. ME817EV-WH70C
8. GD3X Dazzler
These build targets typically come with standard configuration which the user can modify within certain limits as per their requirements.

ESD 4.13 also provides a mechanism to create new custom targets for new ESD projects. For example, if the user wants to create a new build target “PC_BT815EMU_WVGA” with PC as Host platform, BT815 as EVE platform, Screen Resolution as WVGA (800 X 480) and Flash set to 16MB MX25L16. Refer to the screenshots given below –

![New Project](image)

*Figure 34: New Target button*
The newly created target can then be set via build target. Refer to the screenshot given below:

```
Figure 35 : Configure custom target
```

Build & Upload

```
Figure 36 : Select custom build target
```

“Build Executable” button builds a project.
“Build and Upload to Hardware” button for a FT90X toolchain will invoke the compiler and programmer, so that the current project can be built and uploaded into the target hardware. If user selects x64 toolchain, this button will build and generate an `.exe` application which can be run on a PC as host environment.

For pico RP2040 platform, ESD will set up the build environment and trigger cmake and nmake to build pico project. A Uf2 image file will be generated and flashed to the pico MCU, so user needs to enter pico flash mode before triggering this option. If user has failed to do this, a dialog will pop up and requesting user to enter flash mode.

![Image showing a dialog for Pico drive not connected alert](image)

**Figure 38 – Pico drive not connected alert**

The respective output files generated by “Build Executable” for different build configuration and host platforms are shown in Table 6.

<table>
<thead>
<tr>
<th>Build Target</th>
<th>Host Platform</th>
<th>Output files</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME812A WH50R</td>
<td>FT90x toolchain</td>
<td>*.bin; *.elf</td>
</tr>
<tr>
<td></td>
<td>X64 toolchain (FT4222, MPSSE)</td>
<td>*.exe</td>
</tr>
<tr>
<td>ME810A HV35R</td>
<td>FT90x toolchain</td>
<td>*.bin; *.elf</td>
</tr>
<tr>
<td></td>
<td>X64 toolchain (FT4222, MPSSE)</td>
<td>*.exe</td>
</tr>
<tr>
<td>ME813AU WH50C</td>
<td>FT90x toolchain</td>
<td>*.bin; *.elf</td>
</tr>
<tr>
<td></td>
<td>X64 toolchain (FT4222, MPSSE)</td>
<td>*.exe</td>
</tr>
<tr>
<td>VM816CU50A</td>
<td>FT90x toolchain</td>
<td>*.bin; *.elf</td>
</tr>
<tr>
<td></td>
<td>X64 toolchain (FT4222, MPSSE)</td>
<td>*.exe</td>
</tr>
<tr>
<td>VM816C50A</td>
<td>FT90x toolchain</td>
<td>*.bin; *.elf</td>
</tr>
<tr>
<td></td>
<td>X64 toolchain (FT4222, MPSSE)</td>
<td>*.exe</td>
</tr>
<tr>
<td>ME817EV-WH10C</td>
<td>FT90x toolchain</td>
<td>*.bin; *.elf</td>
</tr>
<tr>
<td></td>
<td>X64 toolchain (FT4222, MPSSE)</td>
<td>*.exe</td>
</tr>
<tr>
<td>ME817EV-WH70C</td>
<td>FT90x toolchain</td>
<td>*.bin; *.elf</td>
</tr>
<tr>
<td></td>
<td>X64 toolchain (FT4222, MPSSE)</td>
<td>*.exe</td>
</tr>
</tbody>
</table>

**Table 6 - Output files for different Build Configuration & Host Platforms**

### Export

Upon completing the screen design, it’s time to move the project to the next phase of the workflow – export project. There are three ways of exporting project. They are “Export as Eclipse Project”, “Export as MSVC Project” and “Export as Pico Project”. This 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.

On exporting, ESD copies the generated “C” source code, the necessary bitmap resources and the ESD 4.13 application framework code into a user defined folder. In ESD, all the generated files are named as “*_generated.c". For eclipse export projects, “.cproject“ and “.project” files are generated so that the users can open the project in FT90X eclipse IDE\(^3\). For MSVC export projects, “Project” folder is generated containing the MSVC compatible

\(^3\) [http://brtchip.com/ft9xx-toolchain/](http://brtchip.com/ft9xx-toolchain/)
project. For Pico export projects, “CMakeLists.txt” and “pico_sdk_import.cmake” are generated for user to build the project. Additionally, an optional batch file “pico_build.cmd” is also generated. This allows the user to build Pico export project in a single step. Table 7 gives more information on the Build Targets, Host platform and the corresponding export projects that can be created.

<table>
<thead>
<tr>
<th>Build Target</th>
<th>Host Platform</th>
<th>Export Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME812A WH50R</td>
<td>Emulator, FT4222*, MPSSE*</td>
<td>MSVC Project</td>
</tr>
<tr>
<td></td>
<td>MM900EV1A, MM900EV1B, MM900EV2A, MM900EV3A, MM900EV-Lite</td>
<td>Eclipse Project</td>
</tr>
<tr>
<td>ME810A HV35R</td>
<td>Emulator, FT4222*, MPSSE*</td>
<td>MSVC Project</td>
</tr>
<tr>
<td></td>
<td>MM900EV1A, MM900EV1B, MM900EV2A, MM900EV3A, MM900EV-Lite</td>
<td>Eclipse Project</td>
</tr>
<tr>
<td>ME813A WH50C</td>
<td>Emulator, FT4222*, MPSSE*</td>
<td>MSVC Project</td>
</tr>
<tr>
<td></td>
<td>MM900EV1A, MM900EV1B, MM900EV2A, MM900EV3A, MM900EV-Lite</td>
<td>Eclipse Project</td>
</tr>
<tr>
<td>ME813AU WH50C</td>
<td>Emulator, FT4222</td>
<td>MSVC Project</td>
</tr>
<tr>
<td>VM816CU50A</td>
<td>Emulator, FT4222</td>
<td>MSVC Project</td>
</tr>
<tr>
<td>VM816C50A</td>
<td>Emulator, FT4222*, MPSSE*</td>
<td>MSVC Project</td>
</tr>
<tr>
<td></td>
<td>MM900EV1A, MM900EV1B, MM900EV2A, MM900EV3A, MM900EV-Lite</td>
<td>Eclipse Project</td>
</tr>
<tr>
<td>ME817EV-WH10C</td>
<td>Emulator, FT4222, MPSSE*</td>
<td>Eclipse Project</td>
</tr>
<tr>
<td></td>
<td>MM900EV1A, MM900EV1B, MM900EV2A, MM900EV3A, MM900EV-Lite</td>
<td>MSVC Project</td>
</tr>
<tr>
<td>ME817EV-WH70C</td>
<td>Emulator, FT4222, MPSSE*</td>
<td>Eclipse Project</td>
</tr>
<tr>
<td></td>
<td>MM900EV1A, MM900EV1B, MM900EV2A, MM900EV3A, MM900EV-Lite</td>
<td>MSVC Project</td>
</tr>
<tr>
<td>Gameduino 3X dazzler</td>
<td>MM2040EV(Raspberry pi Pico)</td>
<td>Pico Project</td>
</tr>
<tr>
<td>Any</td>
<td>MM2040EV(Raspberry pi Pico)</td>
<td>Pico Project</td>
</tr>
</tbody>
</table>

Table 7 - Build Targets, Host platform and the corresponding export projects

(* Requires separate MPSSE adapter or FT4222H circuit *)

**Exported Folder Structure**

Table 8 provides the list of folders and files that are created upon exporting the project:

<table>
<thead>
<tr>
<th>Folder / Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esd_Core</td>
<td>Contains the application core files. <strong>It is recommended not to make any changes to this folder/files</strong></td>
</tr>
<tr>
<td>FT_Esd_Framework</td>
<td>Contains the application framework files. <strong>It is recommended not to make any changes to this folder/files</strong></td>
</tr>
<tr>
<td>FT_Esd_Widgets</td>
<td>Contains the widget files. <strong>It is recommended not to make any changes to this folder/files</strong></td>
</tr>
<tr>
<td>FT_Esd_Hal</td>
<td>Contains the hardware abstraction layer files. These files may be changed to support a different MCU.</td>
</tr>
<tr>
<td>Data</td>
<td>Contains the converted bitmap resource data and the flash data which is used by the current project</td>
</tr>
<tr>
<td>$(ProjectName)</td>
<td>Contains the application specific source code (Eclipse export files)</td>
</tr>
</tbody>
</table>
### Bitmap Resource

The bitmap resources used in an ESD 4.13 project can be saved in a SD card or to flash based on the selection of flash flag configuration in the ESD 4.13.

#### SD card

If Bitmap resources meant for SD card are used in a project, it is converted to *.astc (for ASTC file format)/*.raw (for other file formats) or *.bin file format (if compressed flag is selected). Users are required 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 located at:

\$\{ExportFolder\}/data

The bitmap resource file with .bin extension is called to decompress using the EVE engine command CMD_INFLATE.

As per the Hardware Platform requirements, the SD card must be formatted as a FAT32 file system.

#### Flash

If bitmap resources meant for flash are used in a project, it is converted to a Flash.bin file. Users are required to download the converted bitmap resource into flash memory of the hardware module.

The converted bitmap resource is located at:

\$\{ExportFolder\}/data

The bitmap resource file with .bin extension is called to decompress using the EVE engine command CMD_INFLATE.

### Layout Editor
The Layout Editor allows users to preview a single page as well as the whole project. ESD 4.13 employs the EVE emulator to render an EVE display onto the 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 ESD layout editor:

- **Application logic file** - generally with "main" extension
- **Page file** - generally with "page" extension

### Page File

Page stands for a single screen which will be rendered. One application consists of at least one page. Page may have its own life cycle and manages all the widgets on it. Only widgets can be dragged and dropped into pages at Page layout editor. 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.

### Switch Page

Users can define multiple pages within the project. "Switch Page" ESD layout should add to manage the pages. User can drag and drop the pages into this switch page from project browser. The picture below displays how the logic editor view looks like for the switch page logic.

![Switch Page](image.png)

*Figure 39 – Switch Page*
Page Persistence

To allow a page to remain in memory with its current state even when it is made inactive, set the allocation mode to “Lazy Persistent” or “Static” from the application logic in the page's Properties. This feature can help users to reserve the information of a page even when it is not visible.

The page allocation modes supported in ESD 4.13 are listed in the table given below:

<table>
<thead>
<tr>
<th>Allocation Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>Page is always persistent in memory after application starts</td>
</tr>
<tr>
<td>Lazy</td>
<td>Page is created when it is in “active” state and destroyed when it is inactive</td>
</tr>
<tr>
<td>Lazy Persistent</td>
<td>Page is created when it is in “active” state but persistent in memory when it is “inactive” until application is closed.</td>
</tr>
</tbody>
</table>
**User Defined Function for Page File**

Users can write their own code to handle the “Pushed” signal of the “ESD Push Button” widget. Users are required 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 can be located at `$(PageFileName).c` file.

![Figure 42 - Sample Source File](image1)

**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 sample picture given below:

![Figure 43 - Zoom In & Out](image2)
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.

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.
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.

![Simple C Editor](image)

**Figure 46 - 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.

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 4.13 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.

![Basic Logic Node](image)

**Figure 47 – 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".
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. Refer to the picture given below -

![Composite Logic Node](image)

**Figure 49 - Composite Logic Node**

The following are the different types of composite logic nodes defined in ESD 4.13:

- *Page Node*
- *Widget Node*
- *Layout Widget*
- *Actor Node*
- *Logic Object*
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 50 - 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* colour title as shown in the sample picture below -

**Figure 51 - Logic Node Editor – Widget Node**

The highlighted 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”. Figure 52 provides a sample of the “X” property.

Rendering a 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 display 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:

```c
int x = context->X(context->Owner);
```
This calls the `get()` function assigned by the parent to the X property, passing the owner’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:

```c
Ft_Esd_Theme *theme = context->Theme(context->Owner);
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.

**Layout Type Widget**

Layout widget is a layout container for widgets. Unlike widget nodes which have visual appearance, a layout widget manages the layout specification for its child widgets. Layout widgets are introduced since ESD 4.0. Figure 53 & Figure 54 displays the various layout widgets supported in ESD 4.13.
**Actor Node**

Actor type is an extended type of logic node which functions similar 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 colour within the logic node editor.

![Actor Node](image)

**Figure 55 - Actor Node**

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

Refer to the example project “ActorAnimator.esd” under the “EVE Screen Designer → Examples → Basic → ActorAnimators” 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 grey colour.

![Logic Object](image)

**Figure 56 - 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.

<table>
<thead>
<tr>
<th>Connection Line End Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event slots and Signals</td>
<td>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.</td>
</tr>
<tr>
<td>Data property bindings</td>
<td>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.</td>
</tr>
<tr>
<td>Variable write</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Table 9 - 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 “X”/”Y” coordinate (within layout widget) determines the rendering order. The page node with the lowest value of the “X”/”Y” coordinate is rendered first (as lower depth value).

![Figure 57 - Rendering Order Example](image)

Logic Note Editor – User Interaction

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

“Panning” the logic node editor screen helps reveals the parts of the logic node editor which could not have been fit in a single screen. Panning can be done by 2 ways:

(i) Press Alt Key + Left mouse button and move the mouse
(ii) Press the middle mouse button and move the mouse.

User can select multiple node items in the logic node editor by pressing Control key and selecting the desired objects by left mouse button click. Single/multiple objects can thus be moved if needed by moving the mouse while pressing the Left mouse button.
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.
Library Browser

The Library Browser consists of the basic components which are predefined and built in as part of ESD 4.13, 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.

In the Library Browser, items can be filtered by entering the search text in the textbox. Regular expression is supported. The matching text is highlighted as shown in the sample picture given below -

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

ESD 4.13 Layouts contains the collection of ESD 4.13 layout widgets and layout utilities functions.

<table>
<thead>
<tr>
<th>Layout</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Switch Page</td>
<td>This Layout is used to achieve a fade-in/fade-out animation from one page to another page</td>
</tr>
<tr>
<td>Fill Layout</td>
<td>This layout is used to fill up the whole widget display region</td>
</tr>
<tr>
<td>Fixed Positioning</td>
<td>This layout is for fixed coordinate layouts</td>
</tr>
<tr>
<td>Linear Layout</td>
<td>This layout supports both horizontal and vertical layouts</td>
</tr>
<tr>
<td>Scroll Layout</td>
<td>Specific layout for scrollable panel</td>
</tr>
<tr>
<td>Stretch</td>
<td>This layout supports both horizontal and vertical layouts</td>
</tr>
<tr>
<td>Switch Page</td>
<td>To manage page switching logic</td>
</tr>
<tr>
<td>Scroll Switch Page</td>
<td>This layout supports scroll between pages according to the swipe direction.</td>
</tr>
<tr>
<td>Signal Switch Page</td>
<td>This layout support switch between pages via signals, can be set as vertical switch or horizontal switch.</td>
</tr>
</tbody>
</table>

There are also layout utilities and advanced layout features provided in ESD Layouts collection. These sections are for advanced user only.

Layout utilities are a set of helping functions to ease the development of layout. Users can build their own layout widget by making use of them. There are several items defined for sorting widgets purpose.

**Utilities**

ESD 4.13 provides various Layout utility functions while designing layouts.
Get Horizontal Align

The *Get Horizontal Align* utility is used to align the widget horizontally.

The function can be called as shown below:

```c
ESD_FUNCTION(ESD_ALIGN_HORIZONTAL, Type = Esd_AlignHorizontal, DisplayName = "Get Horizontal Align", Category = EsdLayoutUtilities, Macro)
ESD_PARAMETER(align, Type = Esd_Align)
#define ESD_ALIGN_HORIZONTAL(align) (align & 3)
```

Get Vertical Align

The *Get Vertical Align* utility is used to align the widget vertically.

The function can be called like below:

```c
ESD_FUNCTION(ESD_ALIGN_VERTICAL, Type = Esd_AlignVertical, DisplayName = "Get Vertical Align", Category = EsdLayoutUtilities, Macro)
ESD_PARAMETER(align, Type = Esd_Align)
#define ESD_ALIGN_VERTICAL(align) (align & 12)
```

Detach

The *Detach* utility is used to detach the widget from the current parent layout.

The function can be called like below:

```c
ESD_FUNCTION(Ft_Esd_Widget_Detach, DisplayName = "Detach", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
void Ft_Esd_Widget_Detach(Ft_Esd_Widget *context);
```

Get Global Height

The *Get Global Height* utility is used to get the global X coordinate of widget.

The function can be called like below:

```c
ESD_FUNCTION(Ft_Esd_Widget_GetGlobalHeight, Type = ft_int16_t, DisplayName = "Get Global Height", Category = EsdLayoutUtilities, Macro)
ESD_PARAMETER(context, Type = Ft_Esd_Widget *)
#define Ft_Esd_Widget_GetGlobalHeight(context) (((Ft_Esd_Widget *)context)->GlobalHeight)
```
Get Global Width

The *Get Global Width* utility is used to get the width of the widget.

The function can be called like below:

```c
ESD_FUNCTION(Ft_Esd_Widget_GetGlobalWidth, Type = ft_int16_t, DisplayName = "Get Global Width", Category = EsdLayoutUtilities, Macro)
ESD_PARAMETER(context, Type = Ft_Esd_Widget *)
#define Ft_Esd_Widget_GetGlobalWidth(context) (((Ft_Esd_Widget *)context)->GlobalWidth)
```

Get Global X

The *Get Global X* utility is used to get the global X coordinate of the widget.

The function can be called like below:

```c
ESD_FUNCTION(Ft_Esd_Widget_GetGlobalX, Type = ft_int16_t, DisplayName = "Get Global X", Category = EsdLayoutUtilities, Macro)
ESD_PARAMETER(context, Type = Ft_Esd_Widget *)
#define Ft_Esd_Widget_GetGlobalX(context) (((Ft_Esd_Widget *)context)->GlobalX)
```

Get Global Y

The *Get Global Y* utility is used to get the global Y coordinate of the widget.

The function can be called like below:

```c
ESD_FUNCTION(Ft_Esd_Widget_GetGlobalY, Type = ft_int16_t, DisplayName = "Get Global Y", Category = EsdLayoutUtilities, Macro)
ESD_PARAMETER(context, Type = Ft_Esd_Widget *)
#define Ft_Esd_Widget_GetGlobalY(context) (((Ft_Esd_Widget *)context)->GlobalY)
```

Insert Above

The *Insert at Above* utility is used to insert a widget above the specified sibling widgets in the sibling widgets parent layout.

```c
ESD_FUNCTION(Ft_Esd_Widget_InsertAbove, DisplayName = "Insert Above", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
ESD_PARAMETER(sibling, DisplayName = "Sibling", Type = Ft_Esd_Widget *)
void Ft_Esd_Widget_InsertAbove(Ft_Esd_Widget *context, Ft_Esd_Widget *sibling);
```
The *Insert at Below* utility is used to insert a widget below the specified sibling widgets in the sibling widgets parent layout.

The function can be called like below:

```c
ESD_FUNCTION(Ft_Esd_Widget_InsertBelow, DisplayName = "Insert Below", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
ESD_PARAMETER(sibling, DisplayName = "Sibling", Type = Ft_Esd_Widget *)
void Ft_Esd_Widget_InsertBelow(Ft_Esd_Widget *context, Ft_Esd_Widget *sibling);
```

### Insert at Bottom

The *Insert at Bottom* utility is used to insert a widget in the bottom position of the specified parent layout.

The function can be called like below:

```c
ESD_FUNCTION(Ft_Esd_Widget_InsertBottom, DisplayName = "Insert at Bottom", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
ESD_PARAMETER(parent, DisplayName = "Parent", Type = Ft_Esd_Widget *)
void Ft_Esd_Widget_InsertBottom(Ft_Esd_Widget *context, Ft_Esd_Widget *parent);
```

### Insert at Top

The *Insert at Top* utility is used to insert a widget in the top position of the specified parent layout.

The function can be called like below:

```c
ESD_FUNCTION(Ft_Esd_Widget_InsertTop, DisplayName = "Insert at Top", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
ESD_PARAMETER(parent, DisplayName = "Parent", Type = Ft_Esd_Widget *)
void Ft_Esd_Widget_InsertTop(Ft_Esd_Widget *context, Ft_Esd_Widget *parent);
```

### Is Bottom

The *Is Bottom* utility returns FT_TRUE if the widget is the bottom widget within the current parent layout.

The function can be called like below:

```c
ESD_FUNCTION(Ft_Esd_Widget_IsBottom, Type = ft_bool_t, DisplayName = "Is Bottom", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
ft_bool_t Ft_Esd_Widget_IsBottom(Ft_Esd_Widget *context);
```
Is Top

This *Is Top* utility returns FT_TRUE if the widget is the topmost widget within the current parent layout.

The function can be called like below:

```
ESD_FUNCTION(Ft_Esd_Widget_IsTop, Type = ft_bool_t, DisplayName = "Is Top", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
ft_bool_t Ft_Esd_Widget_IsTop(Ft_Esd_Widget *context);
```

Is Visible

The *Is Visible* utility is used to check if the widget is within the current screen scissor area.

The function can be called like below:

```
ESD_FUNCTION(Ft_Esd_Widget_IsVisible, Type = ft_bool_t, DisplayName = "Is Visible", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, Type = Ft_Esd_Widget *)
ft_bool_t Ft_Esd_Widget_IsVisible(Ft_Esd_Widget *context);
```

Sort Higher

The *Sort Higher* utility is used to sort the widget above its current sibling within the current parent layout.

The function can be called like below:

```
ESD_FUNCTION(Ft_Esd_Widget_SortHigher, DisplayName = "Sort Higher", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
void Ft_Esd_Widget_SortHigher(Ft_Esd_Widget *context);
```

Sort Lower

The *Sort Lower* utility is used to sort the widget below its current sibling within the current parent layout.

The function can be called like below:

```
ESD_FUNCTION(Ft_Esd_Widget_SortLower, DisplayName = "Sort Lower", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
void Ft_Esd_Widget_SortLower(Ft_Esd_Widget *context);
```
The **Sort to Bottom** utility is used to sort the widget to the bottom within the current parent layout.

The function can be called like below:

```
ESD_FUNCTION(Ft_Esd_Widget_SortToBottom, DisplayName = "Sort to Bottom", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
void Ft_Esd_Widget_SortToBottom(Ft_Esd_Widget *context);
```

The **Sort to Top** utility is used to sort the widget to the top within the current parent layout.

The function can be called like below:

```
ESD_FUNCTION(Ft_Esd_Widget_SortToTop, DisplayName = "Sort to Top", Category = EsdLayoutUtilities)
ESD_PARAMETER(context, DisplayName = "Widget", Type = Ft_Esd_Widget *)
void Ft_Esd_Widget_SortToTop(Ft_Esd_Widget *context);
```

The **Switch Page** is a special layout which focuses on page/widget transitions. By default, there will be only one Switch Page layout in “AppScreen.Page” in every ESD project. However, this is optional and the users can remove or change the switch page layout in the project. Refer to the example project “EVE Screen Designer → Examples → Basic → MenuPage → MenuPage.esd → AppScreen.page” for a switch page demo.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>The pointer reference of the widget object</td>
</tr>
<tr>
<td>Active</td>
<td>Enable or disable displaying this widget</td>
</tr>
<tr>
<td>X</td>
<td>x coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>Y coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Width</td>
<td>Widget width</td>
</tr>
<tr>
<td>Height</td>
<td>Widget Height</td>
</tr>
<tr>
<td>Spinner</td>
<td>Set true to show spinner/loading icon when switching page. This is useful user indication when a page has heavy resource.</td>
</tr>
<tr>
<td>Scissor</td>
<td>Set true, to trim off any content is drawn outside the layout region.</td>
</tr>
<tr>
<td>AutoResize</td>
<td>Set true, to enable auto resize for this layout.</td>
</tr>
</tbody>
</table>

*Table 10 – Switch Page Layout Properties*
Switch Page Implementation

1. Set page/widget into Switch Page

Unlike widgets, Switch Page cannot be dragged on Screen Layout Editor or Logic Editor. In order to put a page under the control of the switch page layout, the user has to use Project Browser instead, drag the widget into the switch page layout.

2. Page Resource

Since ESD 4.0, a page resource allocation can be:

- Static – always load at the start
- Lazy – allocate when it is active, and free it when it is inactive.
- Lazy persistent – allocate when it is first time active, and it will remain in system until reset.

When a new page is activated, its parent object, the Switch Page will also mark the current page as inactive and free it from memory when it has “Lazy” mode of allocation.

Dynamic Switch Page

The Dynamic Switch Page is a special layout which focuses on page/widget transitions. The primary purpose of this widget is to achieve a fade-in/fade-out animation while transiting from one page to another page. Like Switch Page widget, only one of the children of Switch Page would be visible at any point of time. But unlike Switch Page widget which abruptly changes the visibility between the transiting pages, Dynamic Switch Page widget will fade-in and fade-out the new to-be-visible page and old not-to-visible page.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>The pointer reference of the widget object</td>
</tr>
<tr>
<td>Active</td>
<td>Enable or disable displaying this widget</td>
</tr>
<tr>
<td>X</td>
<td>x coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>y coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Width</td>
<td>Widget width</td>
</tr>
<tr>
<td>Height</td>
<td>Widget Height</td>
</tr>
<tr>
<td>AutoResize</td>
<td>Set true, to enable auto resize for this layout</td>
</tr>
<tr>
<td>Animation Duration</td>
<td>Sets the time within which the animation needs to be finished.</td>
</tr>
</tbody>
</table>

Table 11 – Dynamic Switch Page Layout Properties
Dynamic Switch Page Implementation

1. Set page/widget into Dynamic Switch Page

Unlike widgets, Dynamic Switch Page cannot be dragged on Screen Layout Editor or Logic Editor. In order to put a page under the control of the switch page layout, the user has to use Project Browser instead, drag the widget into the Dynamic switch page layout.

2. Page Resource

Since ESD 4.0, a page resource allocation can be:

- Static – always load at the start
- Lazy – allocate when it is active, and free it when it is inactive.
- Lazy persistent – allocate when it is first time active, and it will remain in system until reset.

When a new page is activated, its parent object, the Dynamic Switch Page will also mark the current page as inactive and free it from memory when it has “Lazy” mode of allocation.

Auto Switch Widget

The Auto Switch Widget is a layout which focuses on rapid transitions between its child widgets. The primary purpose of this widget is to rapidly activate its child widget one by one in the order of its depth. The rate of change is set by the user. This can be used to generate a gif like animation effect. Like Switch Page widget and Dynamic Switch Page widget, only one of its child is active at any given point of time. But unlike Switch Page and Dynamic Switch Page, the transition between child widget is automatic and not triggered. Use case of the widget can be found in Examples/Basic/AutoSwitchLayout.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>The pointer reference of the widget object</td>
</tr>
<tr>
<td>Active</td>
<td>Enable or disable displaying this widget</td>
</tr>
<tr>
<td>X</td>
<td>x coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>Y coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Width</td>
<td>Widget width</td>
</tr>
<tr>
<td>Height</td>
<td>Widget Height</td>
</tr>
<tr>
<td>Animation Frequency</td>
<td>Sets the rate of change of child widget activations</td>
</tr>
</tbody>
</table>

Table 12 – Auto Switch Page Layout Properties
Auto Switch Page Implementation

1. Set widgets into Dynamic Switch Page

Auto Switch widget can be dragged on Logic Editor. In order to put a widget under the control of the switch page layout, the user has to use Project Browser instead, drag the widget into the Auto Switch Widget layout.

2. Widget Rendering order

All child widgets are activated one by one and rendered in a sequence according to their depth orders. Thus to rearrange the order of activation, user needs to update the depth order of the children widgets.

Fixed Positioning

The Fixed Positioning is a layout widget which allows its child widgets to be fixed at design time. By default, every page starts with one fixed positioning layout, such that all widgets under the page are placed at design time. Fixed Positioning layout does not update its child widget's dimensions on runtime. Users should get what they see on design time, on runtime as well.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>The pointer reference of the widget object</td>
</tr>
<tr>
<td>Active</td>
<td>Enable or disable displaying this widget</td>
</tr>
<tr>
<td>X</td>
<td>x coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>Y coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Width</td>
<td>Widget width</td>
</tr>
<tr>
<td>Height</td>
<td>Widget Height</td>
</tr>
</tbody>
</table>

Table 13 – Fixed Positioning Layout Properties

Fixed Positioning Implementation

1. Set page/widget into Fixed Positioning Layout

Drag and drop a new widget into the Screen Layout Editor will be the highest depth sort widget in the layout by default. However, the depth sort values are configurable by the users. If a widget is not in fixed positioning layout, the Screen layout editor will not able to change it is position as it desired. Always check on project browser as picture on the right.
2. Change Child Widgets’ depth sort

Changing the depth sort value of child widgets will affect their rendering sequence. This functionality is same as in ESD 3.0.

**Fill Layout**

The *Fill Layout* is the base layout for every widget/page since in ESD 4.13. A widget always comes with a Fill Layout in it. However, a fill layout in a page has a fixed size due to the screen size defined from the hardware. All widgets in the Fill Layout will auto resize to fit the whole layout area.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>The pointer reference of the widget object</td>
</tr>
<tr>
<td>Active</td>
<td>Enable or disable displaying this widget</td>
</tr>
<tr>
<td>X</td>
<td>x coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>Y coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Width</td>
<td>Widget width</td>
</tr>
<tr>
<td>Height</td>
<td>Widget Height</td>
</tr>
</tbody>
</table>

*Table 14 – Fill Layout Properties*

**Fill Layout Implementation**

1. Change Widget sequence

The fill layout sequence is ordered in the sequence of widgets’ depth sort. Changing a particular widget to a higher depth sort value, will increase the priority widget's rendering. The example below shows a clock widget and an image widget under the same Fill Layout. Both widgets are filling up the full “Fill Layout” area. However, by setting the depth value of clock widget bigger than the image widget's depth sort, the clock widget will be rendered on top of the image widget.

**Linear Layout**

The *Linear Layout* supports both horizontal and vertical alignment in ESD 4.13. All child widgets will be distributed in the selected orientation. Users cannot change the sequence of widgets in the linear layout from Screen Layout Editor in ESD 4.13. However, changing their depth sort value from the property browser will change the sequence.
Property Name | Description
--- | ---
Pointer | The pointer reference of the widget object
Active | Enable or disable displaying 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
Spacing | Set the spacing gap between adjacent child widgets.
Orientation | Set Linear Layout orientation. It supports:
| ESD_ORIENTATION.HORIZONTAL
| ESD_ORIENTATION.VERTICAL
Alignment | Set the alignment setting for the child widgets in the layout
Overflow | Set child widget overflow mode. It supports:
| ESD_OVERFLOW_ALLOW
| ESD_OVERFLOW_CLIP
| ESD_OVERFLOW.FILL
Scissor | Set true to enable scissor to clip off overflow
ChildClipping | Set the flags when child clipping should check. It supports:
| ESD_CLIP.RENDER
| ESD_CLIP.UPDATE
| ESD_CLIP.IDLE
AutoResize | Set auto resize mode for this layout.

Table 15 – Linear Layout Properties

**Linear Layout Implementation**

1. Horizontal Layout

All child widgets of a linear layout with its orientation are set to "ESD_ORIENTATION.HORIZONTAL". They will be distributed horizontally with some spacing in between adjacent widgets. The layout used as in the example here has parameters:

- Orientation = ESD_ORIENTATION.HORIZONTAL
- Spacing = 4
- Align = ESD_ALIGN.TOPFILL
- Overflow = ESD_OVERFLOW_ALLOW
2. Vertical Layout

All child widgets of a linear layout with its orientation set to "ESD_ORIENTATION_VERTICAL". They will be distributed vertically with some spacing in between adjacent widgets. The layout used as in the example here has parameters:

- Orientation = ESD_ORIENTATION_VERTICAL
- Spacing = 4
- Align = ESD_ALIGN_TOPFILL
- Overflow = ESD_OVERFLOW_ALLOW

**Stretch**

The *Stretch Layout* is designed to work together with the linear layout. In the event some widgets need to have higher weightage in the layout, Stretch Layouts are used.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>The pointer reference of the widget object</td>
</tr>
<tr>
<td>Active</td>
<td>Enable or disable displaying this widget</td>
</tr>
<tr>
<td>X</td>
<td>x coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>Y coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Width</td>
<td>Widget width</td>
</tr>
<tr>
<td>Height</td>
<td>Widget Height</td>
</tr>
<tr>
<td>Weight</td>
<td>The weight factor when determine the stretch scale in the selected orientation</td>
</tr>
</tbody>
</table>

**Table 16 – Stretch Layout Properties**

**Stretch Layout Implementation**
1. Space Stretching

The example here demonstrates how to stretch unused space horizontally. A stretch layout has been added into the Linear Layout. The weight factor used here is 1.

![Stretch Layout Example]

2. Widget Weight Stretching

The example here demonstrates widget stretching. In this example, every clock widget has different stretch weight values. The result every widget gains their proportion of width as shown below -

![Widget Weight Stretching Example]

### Scroll Layout

The *Scroll Layout* is a support runtime resizable and scrollable layout in ESD 4.13.

![Scroll Layout]

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theme</td>
<td>Pointer</td>
</tr>
<tr>
<td>Active</td>
<td>true</td>
</tr>
<tr>
<td>X</td>
<td>351</td>
</tr>
<tr>
<td>Y</td>
<td>153</td>
</tr>
<tr>
<td>Width</td>
<td>249</td>
</tr>
<tr>
<td>Height</td>
<td>183</td>
</tr>
<tr>
<td>ScrollX</td>
<td></td>
</tr>
<tr>
<td>ScrollY</td>
<td></td>
</tr>
<tr>
<td>Align</td>
<td></td>
</tr>
<tr>
<td>ScrollBars</td>
<td></td>
</tr>
<tr>
<td>TouchScroll</td>
<td>true</td>
</tr>
<tr>
<td>Scroll Bar Size</td>
<td>24</td>
</tr>
</tbody>
</table>
### Scroll Layout Implementation

For the implementation of scroll layout, refer to the example project - "EVE Screen Designer → Examples → Intermediate → ScrollPanel.

### Scroll Switch Page Layout

The Scroll Switch Page layout consists of a switch page manager and a scroll panel for runtime scrollable support.

Scroll Switch Page layout requires 3 lazy page instances and a current index to start with, namely - the previous page instance, the current page instance and the next page instance.

Upon user touch swipe event, the page will switch according to the swipe direction. The behaviour of this scroll switch page layout are as below:

- On non-swiping runtime, it only renders the current page, but runs the update logic of all the 3 pages.
- On swiping event, it displays both the current page and the swiping to page (can be previous and next page instance).
- On touch up event, it will either recover to the original position if the swiped range has not reached the minimum threshold or switch to the new page. The touch scroll will be temporally disabled until it has recovered or switched to the target page.
- Upon switching page complete, the scroll switch page will fire “NewPage” event and requests a new page instance to be allocated and assigns it as the “New Page” input.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>The pointer reference of the widget object</td>
</tr>
<tr>
<td>Active</td>
<td>Enable or disable displaying this widget</td>
</tr>
<tr>
<td>X</td>
<td>x coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>y coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Width</td>
<td>Widget width</td>
</tr>
<tr>
<td>Height</td>
<td>Widget Height</td>
</tr>
<tr>
<td>ScrollX</td>
<td>Getter and Setter for scroll value along X-axis</td>
</tr>
<tr>
<td>ScrollY</td>
<td>Getter and Setter for scroll value along Y-axis</td>
</tr>
<tr>
<td>Align</td>
<td>Set alignment for its child widgets</td>
</tr>
<tr>
<td>Scrollbars</td>
<td>Set when scrollbar should be visible</td>
</tr>
<tr>
<td>TouchScroll</td>
<td>Set true to enable touch scroll</td>
</tr>
<tr>
<td>Scroll Bar Size</td>
<td>To set scroll bar size when it is visible</td>
</tr>
</tbody>
</table>

Table 17 – Scroll Layout Properties
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>The pointer reference of the widget object</td>
</tr>
<tr>
<td>Initialise</td>
<td>The &quot;initialise&quot; slot for initialising the scroll switch page layout</td>
</tr>
<tr>
<td>Initial Previous</td>
<td>Initial Previous page instance at start</td>
</tr>
<tr>
<td>Initial Current</td>
<td>Initial Current page instance at start</td>
</tr>
<tr>
<td>Initial Next</td>
<td>Initial Next page instance at start</td>
</tr>
<tr>
<td>Initial Index</td>
<td>Initial Current page index at start</td>
</tr>
<tr>
<td>Active Scroll</td>
<td>Set true to override child widgets touch tag event where tag is default(255)</td>
</tr>
<tr>
<td>Spinner</td>
<td>Set true to show spinner upon loading the new page instance.</td>
</tr>
<tr>
<td>New Page</td>
<td>New page instance input</td>
</tr>
<tr>
<td>Active</td>
<td>Enable or disable displaying this widget</td>
</tr>
<tr>
<td>X</td>
<td>x coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>Y coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Width</td>
<td>Widget width</td>
</tr>
<tr>
<td>Height</td>
<td>Widget Height</td>
</tr>
<tr>
<td>Loop Pages</td>
<td>Set true to loop the pages</td>
</tr>
<tr>
<td>Count</td>
<td>Set pages count/limit</td>
</tr>
<tr>
<td>Spacing</td>
<td>Defines the spacing between pages</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>Define the scroll switch sensitivity, range from: 0.05 to 20.00</td>
</tr>
<tr>
<td>Switch Speed</td>
<td>Define the switch speed during page transition</td>
</tr>
<tr>
<td>Previous Page</td>
<td>The output of previous page instance</td>
</tr>
<tr>
<td>Current Page</td>
<td>The output of current page instance</td>
</tr>
<tr>
<td>Next Page</td>
<td>The output of next page instance</td>
</tr>
<tr>
<td>Current Index</td>
<td>The output of current index</td>
</tr>
<tr>
<td>New Index</td>
<td>The output of new index</td>
</tr>
</tbody>
</table>

**Table 18 – Scroll Switch Page Layout Properties**

**Scroll Switch Page layout Implementation**

In typical configuration for a scroll switch page layout, the figure below demonstrates a sample setup. It should have the following rules:

- Connect start slot to the "Initialise" slot of scroll switch page layout instance.
- All switching pages should be configured as child widgets of Scroll Switch Page.
- All child page widgets should be lazy allocation and has same width as the Scroll Switch Page.
- Initial Previous Page, Initial Current Page and the Initial Next Page should be active by default while the rest of child pages should be inactive.
- Initial index should be the index of initial current page.
- Recommend using the “Switch” node to trigger the specific “Activate” signal of the target page.
- Recommend using the “Switch Value” node to get the specific target page instances from all the pages.
- Recommend adding header and footer pages which are Sibling widgets of the Scroll Switch Page Layout instance.
Figure 61 - Sample Scroll Switch Page Implementation

Signal Switch Page Layout

The Signal Switch Page Layout is a simplified layout of Scroll Switch Page Layout which designed for the user case which touch screen is not supported.

The same as Scroll Switch Page Layout, Signal Switch Page Layout requires 3 lazy page instances and a current index to start with, namely - the previous page instance, the current page instance and the next page instance.

2 slots, namely NextPageSignal and PreviousPageSignal are provided for the user to switch to next or previous page. The signals to trigger the switch page can be a button trigger, or another signals.

Figure 62 - Signal Switch Page Layout
Similar to *Scroll Switch Page Layout*, on swiping event (trigger by signal), it displays both the current page and the swiping to page (can be previous and next page instance).

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>The pointer reference of the widget object</td>
</tr>
<tr>
<td>Initialise</td>
<td>The &quot;initialise&quot; slot for initialising the scroll switch page layout</td>
</tr>
<tr>
<td>Initial Previous</td>
<td>Initial Previous page instance at start</td>
</tr>
<tr>
<td>Initial Current</td>
<td>Initial Current page instance at start</td>
</tr>
<tr>
<td>Initial Next</td>
<td>Initial Next page instance at start</td>
</tr>
<tr>
<td>Initial Index</td>
<td>Initial Current page index at start</td>
</tr>
<tr>
<td>New Page</td>
<td>New page instance input</td>
</tr>
<tr>
<td>Active</td>
<td>Enable or disable displaying this widget</td>
</tr>
<tr>
<td>X</td>
<td>x coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>Y coordinate of the top-left of the widget, in pixels</td>
</tr>
<tr>
<td>Width</td>
<td>Widget width</td>
</tr>
<tr>
<td>Height</td>
<td>Widget Height</td>
</tr>
<tr>
<td>Loop Pages</td>
<td>Set true to loop the pages</td>
</tr>
<tr>
<td>Count</td>
<td>Set pages count/limit</td>
</tr>
<tr>
<td>Spacing</td>
<td>Defines the spacing between pages</td>
</tr>
<tr>
<td>Switch Speed</td>
<td>Define the switch speed during page transition</td>
</tr>
<tr>
<td>Previous Page</td>
<td>The output of previous page instance</td>
</tr>
<tr>
<td>Current Page</td>
<td>The output of current page instance</td>
</tr>
<tr>
<td>Next Page</td>
<td>The output of next page instance</td>
</tr>
<tr>
<td>Current Index</td>
<td>The output of current index</td>
</tr>
<tr>
<td>New Index</td>
<td>The output of new index</td>
</tr>
<tr>
<td>Layout Direction</td>
<td>Choose between horizontally or vertically</td>
</tr>
<tr>
<td>Shifting</td>
<td>Boolean value to show if page is in the middle of shifting or not</td>
</tr>
</tbody>
</table>

**Table 19 – Signal Switch Page Layout Properties**

**ESD Render Functions**

All render functions are non-widget. They are not managed by layout in ESD 4.13. Non-widget functions are to be used only within a widget. These functions are provided as utilities for user to make their own custom widgets.

**Note:** In order to preview the render functions result in design time, render slot in the custom widget is required.

**Elements**

The *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.
ESD Circle

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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>RGB value to be rendered inside the circle</td>
</tr>
<tr>
<td>X</td>
<td>x coordinate of central point, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>Y coordinate of central point, in pixels</td>
</tr>
<tr>
<td>radius</td>
<td>Radius of the point</td>
</tr>
</tbody>
</table>

Table 20 - 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.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theme</td>
<td>Theme to be applied on the panel</td>
</tr>
<tr>
<td>x</td>
<td>x coordinate of the top-left point, in pixels</td>
</tr>
<tr>
<td>y</td>
<td>Y coordinate of the top-left point, in pixels</td>
</tr>
<tr>
<td>width</td>
<td>Panel width</td>
</tr>
<tr>
<td>height</td>
<td>Panel Height</td>
</tr>
<tr>
<td>radius</td>
<td>radius of the corners</td>
</tr>
<tr>
<td>raised</td>
<td>Set to true to make panel in raised style</td>
</tr>
</tbody>
</table>

Table 21 - ESD Panel Element Properties

ESD Gradient Panel

The Gradient Panel element is an adjustable rectangular widget that defines a gradient coloured area. The gradient colour can be specified by two colours (one at start and one at the end). The gradient direction is adjustable by changing the direction.

Figure 64 - ESD Circle Element

Figure 65 - ESD Panel Element

Figure 66 - ESD Gradient Panel Element
### 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 67 - Primitives](image.png)

### ESD Bitmap

The **ESD Bitmap** object 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 68 - Bitmap Primitive](image.png)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x coordinate of Bitmap, top-left, in pixels</td>
</tr>
<tr>
<td>y</td>
<td>Y coordinate of Bitmap, top-left, in pixels</td>
</tr>
<tr>
<td>Bitmap Cell</td>
<td>Bitmap cell to display</td>
</tr>
<tr>
<td>Color</td>
<td>Color to be applied to bitmap</td>
</tr>
</tbody>
</table>

**Table 23 - ESD Bitmap Properties**

---

![Image](image.png)
To add a bitmap

Click and select “File → Add” or click 📁 from the toolbar. A bitmap cell is then assigned. ESD 4.13 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.
Use ESD Image to display bitmap as ESD bitmap is non-widget, and used only as a rendering function in ESD 4.13. Users are required to upgrade ESD Bitmap to ESD Image from ESD 3.X.

ESD Line

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

Use ESD Widgets – Line Widget to display line as ESD line is non-widget, and used only as a rendering function in ESD 4.13. Users are required to upgrade ESD Line to Line Widget.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>x – coordinate of the start point, in pixels</td>
</tr>
<tr>
<td>y0</td>
<td>y – coordinate of the start point, in pixels</td>
</tr>
<tr>
<td>x1</td>
<td>x – coordinate of the end point, in pixels</td>
</tr>
<tr>
<td>y1</td>
<td>y – coordinate of the end point, in pixels</td>
</tr>
<tr>
<td>Width</td>
<td>Line width, in pixel</td>
</tr>
<tr>
<td>Color</td>
<td>Line color, in RGBA format</td>
</tr>
</tbody>
</table>

**Table 24 – 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.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>x coordinate of Bitmap, top-left, in pixels</td>
</tr>
<tr>
<td>Y</td>
<td>Y coordinate of Bitmap, top-left, in pixels</td>
</tr>
<tr>
<td>width</td>
<td>width, in pixels</td>
</tr>
<tr>
<td>height</td>
<td>height, in pixels</td>
</tr>
<tr>
<td>radius</td>
<td>radius of the round corner, in pixels</td>
</tr>
<tr>
<td>Color</td>
<td>color of the rectangle</td>
</tr>
</tbody>
</table>

**Table 25 - 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.

**Display List**

The Display List provides a collection of utilities functions related to EVE Display List.

**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.
ESD 4.13 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.

![ESD Theme](image)

**Table 26 - Ft_Esd_Theme_LightBlue Theme**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Value (RGBA)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Theme</td>
<td>Ft_Esd_Theme_LightBlue</td>
</tr>
<tr>
<td>ClearColor</td>
<td>RGBA</td>
<td>235,235,235(255)</td>
<td>Clear Color</td>
</tr>
<tr>
<td>BackColor</td>
<td>RGBA</td>
<td>255,255,255(255)</td>
<td>Background Color</td>
</tr>
<tr>
<td>TextColor</td>
<td>RGBA</td>
<td>0,0,0(255)</td>
<td>Text Color</td>
</tr>
<tr>
<td>ButtonTextColor</td>
<td>RGBA</td>
<td>250,250,250(255)</td>
<td>Button Text Color</td>
</tr>
<tr>
<td>DefaultColor</td>
<td>RGBA</td>
<td>113,113,113(255)</td>
<td>Default Color</td>
</tr>
<tr>
<td>PrimaryColor</td>
<td>RGBA</td>
<td>34,119,199(255)</td>
<td>Primary Color</td>
</tr>
</tbody>
</table>

**Table 27 - Ft_Esd_Theme_DarkOrange Theme**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Value (RGBA)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Theme</td>
<td>Ft_Esd_Theme_DarkOrange</td>
</tr>
<tr>
<td>ClearColor</td>
<td>RGBA</td>
<td>33,33,33(255)</td>
<td>Clear Color</td>
</tr>
<tr>
<td>BackColor</td>
<td>RGBA</td>
<td>21,21,21(255)</td>
<td>Background Color</td>
</tr>
<tr>
<td>TextColor</td>
<td>RGBA</td>
<td>255,255,255(255)</td>
<td>Text Color</td>
</tr>
<tr>
<td>ButtonTextColor</td>
<td>RGBA</td>
<td>255,255,255(255)</td>
<td>Button Text Color</td>
</tr>
<tr>
<td>DefaultColor</td>
<td>RGBA</td>
<td>107,107,107(255)</td>
<td>Default Color</td>
</tr>
<tr>
<td>PrimaryColor</td>
<td>RGBA</td>
<td>255,127,63(255)</td>
<td>Primary Color</td>
</tr>
</tbody>
</table>
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 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 Utilities contains the collection ESD utilities functions. The following table provides the list of ESD Utilities and its functionality.

<table>
<thead>
<tr>
<th>ESD Utility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESD BitmapCell GetInfo</td>
<td>Retrieves the BitmapCell information with given bitmap reference.</td>
</tr>
<tr>
<td>ESD BitmapInfo GetHeight</td>
<td>Returns the BitmapCell Height of given bitmap reference.</td>
</tr>
<tr>
<td>ESD BitmapInfo GetWidth</td>
<td>Retrieves the BitmapCell Width of given bitmap reference.</td>
</tr>
<tr>
<td>Bitmap Persist</td>
<td>To persist the given bitmap object</td>
</tr>
<tr>
<td>Color A+RGB Combine</td>
<td>Merge Alpha to RGB, to become RGBA colour</td>
</tr>
<tr>
<td>Clamp Float</td>
<td>Clamp float value with min and max limits</td>
</tr>
<tr>
<td>Get Delta Ms</td>
<td>Get Delta time difference in milliseconds since last frame update called</td>
</tr>
<tr>
<td>Get Font Height</td>
<td>Return Font Height</td>
</tr>
<tr>
<td>Get EVE Host</td>
<td>Get EVE Host</td>
</tr>
<tr>
<td>Get Milliseconds</td>
<td>Get time in milli-seconds for current frame</td>
</tr>
<tr>
<td>ESD Idle Checker</td>
<td>Check when to sleep and wake up based on the touch events.</td>
</tr>
<tr>
<td>Clamp Int16/UInt16</td>
<td>Clamp Int16/UInt16 value with min and max limits</td>
</tr>
<tr>
<td>Clamp Int32/UInt32</td>
<td>Clamp Int32/UInt32 value with min and max limits</td>
</tr>
<tr>
<td>Load Bitmap to RAM_G</td>
<td>Load Bitmap to RAM_G</td>
</tr>
<tr>
<td>Load Palette to RAM_G</td>
<td>Load Palette to RAM_G</td>
</tr>
<tr>
<td>Pop-up Spinner</td>
<td>Pop-up Spinner when the frame is rendered</td>
</tr>
<tr>
<td>Switch Bitmap Cell</td>
<td>Switch Bitmap Cell</td>
</tr>
<tr>
<td>ESD Timer</td>
<td>ESD Timer Actor</td>
</tr>
<tr>
<td>Touch Area</td>
<td>Touch Area Actor</td>
</tr>
<tr>
<td>Touch Scroll</td>
<td>Touch Scroll Actor</td>
</tr>
<tr>
<td>Touch Tag</td>
<td>Touch Tag Actor</td>
</tr>
<tr>
<td>Current Tag</td>
<td>Current Touch Tag</td>
</tr>
<tr>
<td>Suppress Current Tags</td>
<td>Suppress Current Tags</td>
</tr>
<tr>
<td>Touch X/Y</td>
<td>Touch X/Y coordinate</td>
</tr>
<tr>
<td>Touch X/Y Delta</td>
<td>Touch X/Y Delta</td>
</tr>
</tbody>
</table>

*Table 28 – ESD Utilities & Description*
ESD Idle Checker

ESD Idle Checker Actor provides an actor to check when the application should go to sleep mode or wake up from sleep mode. One of the main use cases of the idle checker actor is to check for idling and wake up event for displaying the screen saver page. The picture on the right demonstrated in "AppScreen.page" uses Idle Checker to switch between application pages and the idle page. The idle checker has an interval of 250 ms and count of 60 intervals. In other words, it may timeout when there is no touch event for 15 secs (60 * 250 = 15000 ms).

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
- Binary Condition
- Local Method
- Binary Operator
- Ternary Operator
- Unary Operator
- Sequence
- Switch
- Switch Value
- Set Variable
- Watch Variables
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.

![Condition Node](image)

**Figure 75 - 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*).

Binary Condition

*Binary Condition* node provides a set of rules to be performed based on the result of a binary operator on two input values. In other words, it allows applying decision points.

![Binary Condition Node](image)

**Figure 76 - Logic Flow – Binary Condition Node**

Whenever a binary condition node is called, a “Then” condition will be called when the result of binary operator on left and right value is True, otherwise an “Else” condition will be called.

The binary operator is specified by user and it will be evaluated when the node is called through “Do” input call.
Local Method

Local Method node is used to add a visual function node. It works as a place holder where user will provide input using parameters and put the function body in logic editor. It gives the option to specify the no. of parameters, parameter type and parameter name to the function as input. User can select the parameter count using scroll bar and add parameters (type and name) from the properties tab.

- "Parameter Type" – Type of input parameter to be selected from drop down menu i.e. int, float, char* or Ft_Esd_Theme* etc.
- "Parameter Name" – Name of the variable should be according to ANSI C coding standard.
- "Result" – Return value from this function which will be assigned to a variable or input to new widget or assigned to output directly.
**Binary Operator**

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

![Binary Operator Node](image)

**Figure 78 - Logic Flow - Binary Operator Node**

The following table provides a list of binary operators supported by ESD 4.13.

<table>
<thead>
<tr>
<th>Binary Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>Equal to comparison operator</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Not Equal to comparison operator</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>Logical AND operator</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>Binary AND operator</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Multiplication operator</td>
</tr>
<tr>
<td><code>/</code></td>
<td>Division operator</td>
</tr>
<tr>
<td><code>%</code></td>
<td>Modulus operator</td>
</tr>
<tr>
<td><code>^</code></td>
<td>Binary XOR operator</td>
</tr>
<tr>
<td><code>+</code></td>
<td>Addition operator</td>
</tr>
<tr>
<td><code>-</code></td>
<td>Subtraction operator</td>
</tr>
<tr>
<td><code>,</code></td>
<td>Comma operator</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than operator</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less than operator</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
</tr>
<tr>
<td><code>&lt;&lt;</code></td>
<td>Binary Left Shift Operator</td>
</tr>
<tr>
<td><code>&gt;&gt;</code></td>
<td>Binary Right Shift Operator</td>
</tr>
</tbody>
</table>

**Table 29 - Binary Operators Supported by ESD 4.13**
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: \( \text{Result} = \text{condition} \ ? \ "\text{Value if true}" : "\text{Value if false}". \]

![Figure 79 - Logic Flow - Ternary Operator Node](image)

Unary Operator

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

![Figure 80 - Logic Flow - Unary Operator Node](image)

The following table provides a list of unary operators supported by ESD 4.13.

<table>
<thead>
<tr>
<th>Unary Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Logic Negation</td>
</tr>
<tr>
<td>&amp;</td>
<td>Address of</td>
</tr>
<tr>
<td>*</td>
<td>De-reference</td>
</tr>
<tr>
<td>+</td>
<td>Positive operator</td>
</tr>
<tr>
<td>-</td>
<td>Negative operator</td>
</tr>
<tr>
<td>~</td>
<td>One’s complement</td>
</tr>
<tr>
<td>!!</td>
<td>Non-zero value converts to 1 and other values converts to 0.</td>
</tr>
</tbody>
</table>

**Table 30 – Unary Operators Supported by ESD 4.13**
**Sequence**

“Sequence” node allows users to define a series of calling actions. When the “Do” input call is invoked, the calls will be triggered sequentially. The number of calls is specified by users in property “Count”.

![Sequence Node](image)

Figure 81 - Logic Flow – Sequence Node

**Switch Node**

“Switch” node will call different branches based on the input value, like a “Switch-Case” statement in C language. Users can define any number of branches and the values to be compared with input value through “Cases” property. Figure 101 shows an example of 3 branches defined and called respectively if input value is equal to 1 or 2 or 3. Since, the input value is pre-set to 4; the default branch is called if no connection is made on it.

![Switch Node](image)

Figure 82 - Logic Flow – Switch Node
**Switch Value**

“Switch Value” is a node to select one output value from multiple predefined values in “Cases” property based on input value. The example in Figure 83 shows 2 “Cases” branch and property “Cases” 0 defines such logic: The “Result” value is 4 if input value is 1. Therefore, the result value is 5 when the input value is 3.

![Switch Value Diagram](image)

**Figure 83 - Logic Flow**

**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).

![Set Variable Diagram](image)

**Figure 84 - Logic Flow - Set Variable Node**
Watch Variables

The Watch variables node is used to monitor a set of variable updates. A single watch variable node can monitor ‘n’ number of variables which can be different (primitive) types. However, the more the input variables have connected, the more frequent the output “Changed” will be triggered. User can also connect input slots:

- “Check” – used to trigger an immediate check for the input variables against previously checked values.
- “Reset” – set base values as current input values, and monitors changes from there.

This watch variable could be very useful during the debugging, or multiple input trigger events. It simplifies nested-if conditional flows.

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
- Widget Interface
**Input**

The *Input* node is used to create an input interface for its parent logic node to bind incoming data. For instance, while creating a widget, an input node will define a property for the widget whose value depends on the connection. Users can configure its property through the Property Editor.

![Input Node](image1)

**Figure 87 - Logic Interface - Input Node**

**Output**

The *Output* node is used to create an output interface for its parent logic to bind the outgoing data. For instance, while creating a widget, an output node will define an interface for the widget to display values to the users. Users can configure its property through the Property Editor.

![Output Node](image2)

**Figure 88 - 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.

![Signal Node Diagram]

**Figure 89 - Logic Interface - Signal Node**

**Slot**

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

![Slot Node Diagram]

**Figure 90 - 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.

<table>
<thead>
<tr>
<th>Built-in Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Called when the node is created in memory</td>
</tr>
<tr>
<td>Render</td>
<td>Writing rendering calls to the co-processor command buffer. It is repeatedly called.</td>
</tr>
<tr>
<td>Update</td>
<td>Called once before the frame render. It is called frequently.</td>
</tr>
<tr>
<td>Idle</td>
<td>Called repeatedly while waiting for the frame to flip</td>
</tr>
<tr>
<td>End</td>
<td>Called when the node is removed from the memory</td>
</tr>
</tbody>
</table>

*Table 31 - Built-in Slots*
The following pseudo-code may help you understand how these slots are invoked:

```c
//Application is booting up and started
Start();

//Now UI is launched
while(UI is running)
{
    Update();
    Render();
    Idle();
}

//Application is ending
End();
```

### 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.
**Writer**

The **Writer** node is used to define a value writing interface for its parent node. Generally, it is used to setup the value for the external variable. For instance, while creating the ESD Spinbox widget, the “Changed” writer is defined to update the connected external variables of Spinbox widget once Spinbox value is changed.

![Writer Node](image1)

**Figure 92 - Logic Interface - Writer Node**

**Widget Interface**

The **Widget Interface** node shows the interface of base widget interface. Since ESD 4.0, all layouts, pages and widgets have been implemented from this widget interface. For advanced users, this widget interface node could be useful for designing user customized widget in order to use its base interface for accessing information from ESD widget framework.

![Widget Interface](image2)

**Figure 93 - Widget Interface**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>Provides the pointer of this widget, allow other widget/layout/page to refer</td>
</tr>
<tr>
<td>Container</td>
<td>Provides the pointer of this widget’s immediate container. Normally, a widget’s immediate container is a layout</td>
</tr>
<tr>
<td>Active</td>
<td>Set and get the current widget active</td>
</tr>
<tr>
<td>Local X</td>
<td>Set and get the current widget local X. Local X refers to the local x-coordinate with reference to its container</td>
</tr>
<tr>
<td>Local Y</td>
<td>Set and get the current widget local Y. Local Y refers to the local y-coordinate with reference to its container</td>
</tr>
<tr>
<td>Local Width</td>
<td>Set and get the current widget local width. Local width refers to the designed width with reference to its container</td>
</tr>
<tr>
<td>Local Height</td>
<td>Set and get the current widget local height. Local height refers to the designed height with reference to its container</td>
</tr>
<tr>
<td>Local Rect</td>
<td>Set and get the current widget local dimension with reference to its container</td>
</tr>
<tr>
<td>Global X</td>
<td>Set and get the current widget global X. Global X refers to the global x-coordinate with reference to screen</td>
</tr>
</tbody>
</table>
Global Y | Set and get the current widget global Y. Global Y refers to the global y-coordinate with reference to screen
---|---
Global Width | Set and get the current widget global width. Global width refers to the designed width with reference to screen
Global Height | Set and get the current widget global height. Global height refers to the designed height with reference to screen
Global Rect | Set and get the current widget global dimension with reference to its screen

**Table 32 – Widget Interface Properties**

- All Local properties are subjected to difference from the respective Global properties. As the global values are determined by the layers of the layouts as containers.
- Global properties are the values determined at the runtime. While Local properties are values determined at design time.
- Global properties are eventually stable in runtime, as layers of layouts may require multiple rendering cycle to stabilize.
Property Editor

The Property Editor provides users with a unified interface to edit properties. Each logic node in ESD 4.13, 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 given below shows an example of the ESD 4.13 widget “ESD Radio Button”.

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 4.13 application -

- **Name**
- **Depth Sort**
- **Active**
**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 for generating the source code.

![Figure 96 – Name Property](image)

**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 97 – Depth Sort Property](image)

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 98 – Right Click Menu](image)

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 99 – 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.

Programming Features

This section explains about the programming features. ESD 4.13 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 4.13 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 editor.

These macros are generally used together with the source code. It is recommended that the user defines all the definitions associated with macros immediately following the macros declarations where ever needed and in the same file. Not doing so, may result in unexpected behaviour. For e.g., ESD_FUNCTION macro immediately needs to be followed with function definition. Comments are acceptable between the macro and the definitions.

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

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td></td>
<td>Type used internally for formatting C literals and binary representations</td>
</tr>
<tr>
<td>Void</td>
<td></td>
<td>Not a type (default)</td>
</tr>
<tr>
<td>Int64, Int32, Int16, Int8, IntPtr</td>
<td></td>
<td>Signed integer</td>
</tr>
<tr>
<td>UInt64, UInt32, UInt16, UInt8, UIntPtr</td>
<td></td>
<td>Unsigned integer</td>
</tr>
<tr>
<td>Double</td>
<td></td>
<td>64bit floating point</td>
</tr>
<tr>
<td>Float</td>
<td></td>
<td>32bit floating point</td>
</tr>
<tr>
<td>Char</td>
<td></td>
<td>An 8bit text character</td>
</tr>
<tr>
<td>Bool</td>
<td></td>
<td>C99 Bool</td>
</tr>
<tr>
<td>Utf8</td>
<td></td>
<td>String with UTF-8 encoding</td>
</tr>
<tr>
<td>Latin1</td>
<td></td>
<td>String with Latin 1 (ISO8859-1) encoding</td>
</tr>
<tr>
<td>Pointer</td>
<td></td>
<td>Pointer type</td>
</tr>
<tr>
<td>Struct</td>
<td></td>
<td>Struct type (not fully supported yet)</td>
</tr>
<tr>
<td>Edit</td>
<td></td>
<td>Used by the Properties Editor to specify the control to edit the value and by code generation when using the fixed point formats</td>
</tr>
<tr>
<td>None</td>
<td></td>
<td>No editor control (default)</td>
</tr>
<tr>
<td>Int</td>
<td></td>
<td>Integer, spin-box integer control</td>
</tr>
<tr>
<td>Real</td>
<td></td>
<td>Floating point, spin-box floating point control</td>
</tr>
<tr>
<td>Boolean</td>
<td></td>
<td>One or Zero, checkbox</td>
</tr>
<tr>
<td>String</td>
<td></td>
<td>Text, single-line entry</td>
</tr>
<tr>
<td>Fixed1, Fixed2, ..., Fixed32</td>
<td></td>
<td>Fixed point, spin-box floating point control</td>
</tr>
<tr>
<td>ColorARGB</td>
<td></td>
<td>Color, rgb with alpha</td>
</tr>
<tr>
<td>ColorRGB</td>
<td></td>
<td>Color, rgb without alpha</td>
</tr>
<tr>
<td>Library</td>
<td></td>
<td>Select a function specified with ESD_FUNCTION to provide the value (useful for pointer values)</td>
</tr>
</tbody>
</table>

### 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.

```plaintext
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 4.13, so that the application can add that function into the *User Functions* category node which is located in the Library browser. *ESD_FUNCTION* is not applicable for static functions even though ESD permits to make any function with *ESD_FUNCTION* macro.

![Figure 100 – User Functions Example](image)

**Syntax Example:**

The following syntax example is to register a function called “hsvToRgb” to ESD 4.13.

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

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The function’s return value type</td>
<td></td>
</tr>
</tbody>
</table>

**Table 34 - Macros – ESD_FUNCTION - Parameters**

- 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 101 – ESD_METHOD Example](image-url)
### Table 35 - Meta Macros - ESD_METHOD - Parameters

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

```c
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)
{
    // ...
}
```

- 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 -

```c
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_VARIABLE

**ESD_VARIABLE** is a built-in macro used to define a local variable of a logic node.

**Syntax Example:**

```c
ESD_VARIABLE(Margin, Type = ft_int16_t, Default = 0, Public)
ft_int16_t Margin;
```
**ESD_GLOBAL**

*ESD_GLOBAL* is a built-in macro used to define a global variable accessed across different pages of a logic node. Global variable declaration should present in the header file and definition should be in source file.

**Syntax Example:**

In header file:

```c
ESD_GLOBAL(Var, Type = int)   // type must be specified
Extern int Var;
```

In C file:

```c
Int Var = 0xFFFF;   // Assign initial value
```

**ESD_INPUT**

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

**Syntax Example:**

```c
ESD_INPUT(Margin, Type = int)
int(* Margin)(void *context);
```

**ESD_OUTPUT**

*ESD_OUTPUT* is a built-in macro used to define an output variable of a logic node.

**Syntax Example:**

```c
ESD_OUTPUT(FirstPos, Type = int)
int LayoutSplit_FirstPos(LayoutSplit *context);
```

**ESD_UPDATE**

*ESD_UPDATE* 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:**

```c
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 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 through the ESD simulation engine. If some part of the user’s C source code is not supposed to be run under the ESD simulation engine, for example, accessing some hardware specific registers, users may choose to skip the code by using this macro.

**FT900_PLATFORM**

FT900_PLATFORM macro is defined when the target platform is FT90X based. 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 Files**.

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

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

**Editing the Source File**

To make ESD responsive of the user functions, the ESD specific macro "ESD_FUNCTION or ESD_METHOD" needs 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) {
    //..
}
## F. Appendix A – List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Screen Designer - System Error</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2</td>
<td>ASTC applied for images and fonts</td>
<td>18</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Load resources from flash</td>
<td>18</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Use Unicode font with pre-defined character set</td>
<td>19</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Chinese font used in application</td>
<td>19</td>
</tr>
<tr>
<td>Figure 6</td>
<td>ASTC Format</td>
<td>19</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Font size and Character set</td>
<td>20</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Select Unicode Font Resource</td>
<td>20</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Unicode font applied for Label and Button</td>
<td>20</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Mark resource as storage in Flash</td>
<td>21</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Upload Flash file to device</td>
<td>21</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Making Value Specific to Current Target</td>
<td>24</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Removing Target Specific Value</td>
<td>24</td>
</tr>
<tr>
<td>Figure 14</td>
<td>EVE Screen Designer User Interface Components</td>
<td>27</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Menu bar</td>
<td>28</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Toolbar 1</td>
<td>30</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Toolbar 2</td>
<td>30</td>
</tr>
<tr>
<td>Figure 18</td>
<td>MCU Type is not applicable to selected platform</td>
<td>31</td>
</tr>
<tr>
<td>Figure 19</td>
<td>CTP configuration</td>
<td>32</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Project Explorer window</td>
<td>32</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Project Explorer Filter</td>
<td>32</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Screen Layout Editor – App.main</td>
<td>33</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Screen Layout Editor - MainPage.c (C Source Code)</td>
<td>33</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Context menu on Tab bar</td>
<td>34</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Logic Node Editor</td>
<td>34</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Library Browser</td>
<td>35</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Error List Window</td>
<td>36</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Output Window</td>
<td>36</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Property Editor (ESD Push Button)</td>
<td>37</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Status bar</td>
<td>37</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Application Project Structure</td>
<td>38</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Application Workflow</td>
<td>39</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Property Editor</td>
<td>41</td>
</tr>
<tr>
<td>Figure 34</td>
<td>New Target button</td>
<td>44</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Configure custom target</td>
<td>45</td>
</tr>
<tr>
<td>Figure 36</td>
<td>Select custom build target</td>
<td>45</td>
</tr>
<tr>
<td>Figure 37</td>
<td>Build &amp; Upload</td>
<td>45</td>
</tr>
<tr>
<td>Figure 38</td>
<td>Pico drive not connected alert</td>
<td>46</td>
</tr>
<tr>
<td>Figure 39</td>
<td>Switch Page</td>
<td>49</td>
</tr>
<tr>
<td>Figure 40</td>
<td>Sample Switch Page UI</td>
<td>50</td>
</tr>
<tr>
<td>Figure 41</td>
<td>Properties Editor – Page Allocation</td>
<td>50</td>
</tr>
<tr>
<td>Figure 42</td>
<td>Sample Source File</td>
<td>51</td>
</tr>
<tr>
<td>Figure 43</td>
<td>Zoom In &amp; Out</td>
<td>51</td>
</tr>
<tr>
<td>Figure 44</td>
<td>Main File</td>
<td>52</td>
</tr>
<tr>
<td>Figure 45</td>
<td>Sample Logic File</td>
<td>52</td>
</tr>
<tr>
<td>Figure 46</td>
<td>C File / C Editor</td>
<td>53</td>
</tr>
<tr>
<td>Figure 47</td>
<td>Logic Node Editor - Basic Logic Node</td>
<td>53</td>
</tr>
<tr>
<td>Figure 48</td>
<td>Library Browser - Basic Logic Nodes</td>
<td>54</td>
</tr>
<tr>
<td>Figure 49</td>
<td>Composite Logic Node</td>
<td>54</td>
</tr>
<tr>
<td>Figure 50</td>
<td>Composite Logic Node - Page Node</td>
<td>55</td>
</tr>
</tbody>
</table>
# G. Appendix B – List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Installation Folder</td>
<td>17</td>
</tr>
<tr>
<td>Table 2</td>
<td>Menu &amp; Description</td>
<td>30</td>
</tr>
<tr>
<td>Table 3</td>
<td>Toolbar &amp; Description</td>
<td>31</td>
</tr>
<tr>
<td>Table 4</td>
<td>ESD Libraries</td>
<td>35</td>
</tr>
<tr>
<td>Table 5</td>
<td>Bitmap Resource Properties</td>
<td>42</td>
</tr>
<tr>
<td>Table 6</td>
<td>Output files for different Build Configuration &amp; Host Platforms</td>
<td>46</td>
</tr>
<tr>
<td>Table 7</td>
<td>Build Targets, Host platform and the corresponding export projects</td>
<td>47</td>
</tr>
<tr>
<td>Table 8</td>
<td>List of Folder &amp; File created upon exporting project</td>
<td>48</td>
</tr>
<tr>
<td>Table 9</td>
<td>Connection End Types</td>
<td>59</td>
</tr>
<tr>
<td>Table 10</td>
<td>Switch Page Layout Properties</td>
<td>68</td>
</tr>
<tr>
<td>Table 11</td>
<td>Dynamic Switch Page Layout Properties</td>
<td>69</td>
</tr>
<tr>
<td>Table 12</td>
<td>Auto Switch Page Layout Properties</td>
<td>70</td>
</tr>
<tr>
<td>Table 13</td>
<td>Fixed Positioning Layout Properties</td>
<td>71</td>
</tr>
<tr>
<td>Table 14</td>
<td>Fill Layout Properties</td>
<td>72</td>
</tr>
<tr>
<td>Table 15</td>
<td>Linear Layout Properties</td>
<td>73</td>
</tr>
<tr>
<td>Table 16</td>
<td>Stretch Layout Properties</td>
<td>74</td>
</tr>
<tr>
<td>Table 17</td>
<td>Scroll Layout Properties</td>
<td>76</td>
</tr>
<tr>
<td>Table 18</td>
<td>Scroll Switch Page Layout Properties</td>
<td>77</td>
</tr>
<tr>
<td>Table 19</td>
<td>Signal Switch Page Layout Properties</td>
<td>79</td>
</tr>
<tr>
<td>Table 20</td>
<td>ESD Circle Element Properties</td>
<td>80</td>
</tr>
<tr>
<td>Table 21</td>
<td>ESD Panel Element Properties</td>
<td>80</td>
</tr>
<tr>
<td>Table 22</td>
<td>ESD Gradient Panel Element Properties</td>
<td>81</td>
</tr>
<tr>
<td>Table 23</td>
<td>ESD Bitmap Properties</td>
<td>81</td>
</tr>
<tr>
<td>Table 24</td>
<td>ESD Line Properties</td>
<td>83</td>
</tr>
<tr>
<td>Table 25</td>
<td>ESD Rectangle Properties</td>
<td>84</td>
</tr>
<tr>
<td>Table 26</td>
<td>Ft_Esd_Theme_LightBlue Theme</td>
<td>85</td>
</tr>
<tr>
<td>Table 27</td>
<td>Ft_Esd_Theme_DarkOrange Theme</td>
<td>85</td>
</tr>
<tr>
<td>Table 28</td>
<td>ESD Utilities &amp; Description</td>
<td>87</td>
</tr>
<tr>
<td>Table 29</td>
<td>Binary Operators Supported by ESD 4.13</td>
<td>91</td>
</tr>
<tr>
<td>Table 30</td>
<td>Unary Operators Supported by ESD 4.13</td>
<td>92</td>
</tr>
<tr>
<td>Table 31</td>
<td>Built-in Slots</td>
<td>97</td>
</tr>
<tr>
<td>Table 32</td>
<td>Widget Interface Properties</td>
<td>100</td>
</tr>
<tr>
<td>Table 33</td>
<td>Macros – ESD_TYPE - Parameters</td>
<td>104</td>
</tr>
<tr>
<td>Table 34</td>
<td>Macros – ESD_FUNCTION - Parameters</td>
<td>106</td>
</tr>
<tr>
<td>Table 35</td>
<td>Meta Macros – ESD_METHOD - Parameters</td>
<td>107</td>
</tr>
</tbody>
</table>
# H. Appendix C – Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Changes</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version Draft 0.3</td>
<td>Initial user guide release for ESD 4.5 version features &amp; enhancements</td>
<td>22-06-2018</td>
</tr>
<tr>
<td>Version Draft 0.6</td>
<td>Updated user guide as per ESD 4.8 version features and enhancements</td>
<td>27-09-2019</td>
</tr>
<tr>
<td>Version 1.0</td>
<td>User Guide updated as per the following changes for 4.10 –</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Overview/What’s new in ESD 4.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. ESD Workflow/Target Selection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. ESD Workflow/Build &amp; Upload</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. ESD Workflow/Export</td>
<td></td>
</tr>
<tr>
<td>Version 1.0</td>
<td>User Guide updated as per the following changes for 4.12 –</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Overview/What’s new in ESD 4.12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. ESD Layouts: add Signal Switch page layout guide</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. Add a section on video converter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. Other minor improvements</td>
<td>12-04-2021</td>
</tr>
<tr>
<td>Version 1.2</td>
<td>User Guide updated as per the following changes for 4.13 –</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Overview/What’s new in ESD 4.13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Modify Hardware Requirements session to add pico</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. Modify the outdated figures such as menu bar, installation wizard</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. Add new Tools menu description in Table 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. Add pico export description in Table 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. Added Gameduino 3X dazzler support</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7. Added Advanced User Setting Section</td>
<td>30-07-2021</td>
</tr>
</tbody>
</table>