**Emobiix Application Development Guide**

For PEEK INC Only

Table of Contents

# Introduction

This document outlines the general architecture of application developed for the emobiix platform along with specifics on aspects of applications that are independent of architecture design.

# Application XML Elements

Emobiix applications contain an XML component to describe the visual characteristics, UI, scripts, data usage, and policy resident on the client side. This section defines the XML structure and how elements relate to their client side aspect.

## Understanding This Section

This chapter outlines various XML tags, fields, and contents. Understanding the following sections involves understanding how relevant information about each entity is presented. The format of following sections is as specified.

XML tag names are represented in bold text. For example the **label** tag would appear in bold in text.

XML field names are represented in italic text. For example the *id* field would appear italic.

Field names will be presented in a table for each tag type. The first column in the table indicates the field name, the second indicates the possible values, the third provides a description. Possible values could be blank to indicate that the possible values are not reasonably finite or useful to document. In XML tag field lists, mandatory field will be indicated with a blue background whereas optional fields will have a yellow background as shown. In the example below, headings are present for each column to aid in documentation, however these headings will not appear in actual definition lists

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Possible Types** | **Description** |
| *name* |  | Identifying |
| *canfocus* | “0” or “1” | Marks widget as focusable or not |

A definition of an entity may contain an example. Such an example will be shown in the form below, always proceeded by a description of the example:

|  |
| --- |
| <tagname field=”value1” field2=”value2”/> |

Using a **tagname** to hold values for *field1* and *field2* -

## Common Fields

Emobiix widgets all derive from the same basic widget type. This basic type offers some characteristics that are relevant for most widget types and thus considered common fields. These fields largely control layout, interaction, style, and function.

Although common fields apply to many widget types, not all widgets can make use of all common fields, therefore common fields will be outlined in each widgets’ representing XML node section.

In addition to fields appearing on the common field list, they may be repeated in a node’s section if the definition is slightly different or serves a different purpose.

|  |  |  |
| --- | --- | --- |
| *id* | string | Style identifier of node element. Controls visual aspect. Default implies default style |
| *name* | string | Unique identifier used to locate element |
| *width* | number or  percentage | Requested width of widget. May be a number to specify pixels, a percentage to specify width relative to parent, or omitted to imply based on children |
| *height* | number or  percentage | Requested height of widget. May be a number to specify pixels, a percentage to specify height relative to parent, or omitted to imply based on children |
| *alignment* | “left”, “center”, or “right” | Positioning of widget relative to parent. For vertical alignment, values can be taken literally. For horizontal alignment, “left” represents “top” and “right” represents bottom. Default is “left” |
| *packing* | “horizontal” or “vertical” | Positioning applied to children widgets relative to self. Horizontal packing places widgets next-to-each-other, vertical packing places them on-top-of-each-other |
| *script* | Lua script | Specifies a script context to be used for this and child widgets |
| *canfocus* | “0” or “1” | Specifies if a widget can be focused |
| *onreturn* | Lua script | Script executed when the widget is focused and the return input is actioned |
| *accesskey* | Character | Specifies that when an unhandled key press matching the character is received by the application, the onreturn script for this widget should be executed |

## Application

**application** is the root XML node for any emobiix application and defines meta-information and application-wide properties.

|  |  |  |
| --- | --- | --- |
| *name* | string | Name used when presenting application in any UI objects, eg application lists, task switching, etc |
| *startupview* | name | Object name of view in application to be used on application launch. Initial view focused |
| *description* | string | Brief explanation of application purpose |
| *Icon* | name | Object name of icon for application. Name must resolve to an image element within the application. |
| *onload* | Lua script | Script to execute when application is loaded |
| *onfocus* | Lua script | Script to execute when application becomes focused |
| *onunfocus* | Lua script | Script to execute when application looses focus |

Common Fields: *script*

|  |
| --- |
| <application name=”Hello World” startupview=”mainview” icon=”appicon”>  <image name=”appicon” src=”hello\_world.png”>  <view name=”mainview”>  <label>Hello World</label>  </view>  </application> |

Hello world **application** with an *icon* reference and a single **view** -

### Image

An **image** defines a standard graphics file component. The contents of the image is specified in the *src* field.

Supported image file formats are PNG, and JPG. MNG, GIF, and GIF-GCE may also be supported largely depending on patent licensing and platform capabilities.

|  |  |  |
| --- | --- | --- |
| *src* | filename | Filename of supported image file |
| *transparency* | “none”, “stencil”, “full” | If “none”, pixels from image are presented to display unaltered. If “stencil”, color 0 (black) is considered completely transparent, every other color is considered completely opaque. If “full”, RGBA images will have alpha map to opacity, black and white images will have intensity map to opacity. Default is “full” |
| *color* | Hex string | RGBA color to be applied to a black and white image to interpret as intensity of color. Alpha component is ignored. Default is white |

Common fields: *id*, *name*, *packing*, *alignment, script*, *canfocus*, *onreturn*, *accesskey*

|  |
| --- |
| <image name=”childindicator” src=”leaf\_mask.png” transparency=”stencil” color=”8F000000”/> |

Stencil **image** displayed in red -

### View

A view is a top-level widget that contains child widgets to make up what is seen on the client. A view can be considered the root visible node. An application is typically made up from many views, switching between them to control application flow.

A view itself has no visual representation other than it is a container for other widgets. It can however, effect layout of child widgets since it is a container.

Common fields: *id, packing, name, alignment, script, width, height*

|  |
| --- |
| <image name=”childindicator” src=”leaf\_mask.png” transparency=”stencil” color=”8F0000”/> |

Stencil **image** displayed in red -

### Box

A **box** is a basic layout element used to vary packing and alignment styles. Optionally, the box may also have a visual representation if so defined by its assigned style.

Label1

Label2

Label3

Label4 Label5 Label6 Label7

Box showing horizontal layout (dotted), and one showing visual style (label8) -

Label8

An example usage of a box is where you have a view of vertically packed elements where you would like to add some additional elements that are horizontally packed. You can use a box without visual representation to achieve this. Another use would be if you wish to create a visual box around some elements, you could insert them into a box with a defined visual style.

Common fields: *id, name, packing, alignment, script, width, height, canfocus, onreturn, accesskey*

|  |
| --- |
| <box name=”labelbox” width=”90%” packing=”horizontal”>  …  </box> |

Grouping box to align children horizontally in 90% of parent width -

### Label

A label is a widget that can represent a single or multi-line body of text. It is non-interactive and often static but can be dynamic where needed.

The displayed text of a label is controlled by the contents of the XML node.

### Button

### Stack

### Array

### Set

### Menu

### Dialog

Using a tagname to hold values for field1 and field2 - 1 4