Antmicro

Antmicro grvl

2026-01-26

CONTENTS

CHAPTER

ONE

WHAT IS GRVL?

Grvl (pronounced gravel) is Antmicro's open source library for creating rich, modern, animated graphical user interfaces, developed for use on constrained devices like MCUs.

Grvl is designed to be portable, currently supporting standalone Simple DirectMedia Layer (SDL) based applications and Zephyr RTOS.

With its collection of built-in widgets, XML-based config and support for reconfiguration in runtime, grvl was created with ease of use for both designers and developers in mind.

CHAPTER

TWO

QUICKSTART

2.1 Linking (CMAKE)

2.1.1 Standalone

Pull grvl and link it in your project:

```
add_subdirectory(GRVL_SOURCE_DIR)
target_link_libraries(app PRIVATE grvl)
```

2.1.2 Zephyr

```
set(GRVL_ZEPHYR ON)
add_subdirectory(GRVL_SOURCE_DIR)
target_link_libraries(app PRIVATE grvl)
```

2.2 Minimal example

```
#include <grvl.h>
#include <Manager.h>
...
int main(){
    ...
    // initialize callbacks
    gui_callbacks_t callbacks;
    callbacks.malloc = malloc;
    callbacks.free = free;
    callbacks.dma_operation = dma_operation;
    callbacks.dma_operation_clt = dma_operation_clt;
    callbacks.dma_fill = dma_fill;
    callbacks.wait_for_vsync = wait_for_vsync;
    callbacks.flipping_completed = flipping_completed;
    callbacks.set_layer_pointer = set_layer_poiner;
    callbacks.gui_printf = gui_printf;
    callbacks.mutex_create = mutex_create;
    callbacks.mutex_lock = mutex_lock;
    callbacks.mutex_unlock = mutex_unlock;
    callbacks.mutex_destroy = mutex_destroy;
```

(continues on next page)

(continued from previous page)

```
    callbacks.get_timestamp = get_timestamp;
    grvl::Init(&callbacks);

    // initalize manager
    Manager::Initialize(WIDTH, HEIGHT, BPP, IS_FLIPPED);

    // fill the gui media
    Manager::GetInstance()
        .AddFontToFontContainer(FONT_NAME_USED_IN_XML, new Font(FONT_PATH))
        .AddImageContentToContainer(IMAGE_NAME_USED_IN_XML, new␣
↪ImageContent(ImageContent::FromPNG(IMAGE_PATH)))
        .AddCallbackToContainer(CALLBACK_NAME_USED_IN_XML,␣
↪(grvl::Event::CallbackPointer) callback_function)
        .BuildFromXML(XML_PATH)
        .InitializationFinished();

    while(running){
        // update gui
        Manager::GetInstance().MainLoopIteration();
        ...
        platform_specific_wait(); // e.g. SDL_WaitEventTimeout or k_msleep
        ...

        // update specific components
        ProgressBar* progressBar = dynamic_cast<ProgressBar*>(displayManager->
↪GetActiveScreen()->GetElement("progress_bar"));
        progressBar->SetProgressValue(counter % 100);
        ...
        // handle events
        Manager::GetInstance().ProcessTouchPoint(state, x, y);
    }
}
```

For more in-detail examples see *the Examples chapter*.

CHAPTER

THREE

EXAMPLES

3.1 Animating values

First, grab a component by ID:

```
ProgressBar* progressBar = dynamic_cast<ProgressBar*>(displayManager->
↪GetActiveScreen()->GetElement("progress_bar"));
```

Then execute a setter in the main loop:

```
progressBar->SetProgressValue((SDL_GetTicks() / 100) % 100);
```

3.2 Callbacks

There are two ways to add callbacks to GUI components. You can write them in either C++ or JavaScript.

3.2.1 C++

Create the callbacks:

```
void ButtonCallback(Button *Sender, const Event::ArgVector& Args) {
    printf("Button clicked! (%s)\n", Sender->GetID());
}

void SwitchCallback(SwitchButton *Sender, const Event::ArgVector& Args) {
    switch (Sender->GetSwitchState()) {
        case true:
            printf("Switch is ON! (%s)\n", Sender->GetID());
            break;
        default:
            printf("Switch is OFF! (%s)\n", Sender->GetID());
    }
}
```

Add them to the callbacks container:

```
displayManager->AddCallbackToContainer("ButtonCallback",␣
↪(grvl::Event::CallbackPointer)ButtonCallback);
displayManager->AddCallbackToContainer("SwitchCallback",␣
```

(continues on next page)

(continued from previous page)

```
↪(grvl::Event::CallbackPointer)SwitchCallback);
                                        /*^^^^ choose a callback name */
```

3.2.2 JavaScript

Create a file with JavaScript callbacks, e.g.:

```
// callbacks.js

function ButtonCallback(caller) {
    const buttonId = caller.name
    Print("Button clicked! (" + buttonId + ")")
}

function SwitchCallback(caller) {
    const callerName = caller.name
    if (caller.switchState) {
        Print("Switch is ON! (" + callerName + ")")
    } else {
        Print("Switch is OFF! (" + callerName + ")")
    }
}
```

Then include it in your application's XML layout:

```
<script src="callbacks.js"></script>
```

You can specify a non-default working directory for JavaScript files with:

```
JSEngine::SetSourceCodeWorkingDirectory("Scripts/JavaScript/");
```

See *JavaScript documentation* for further reference.

3.2.3 Binding callbacks

Bind callbacks to GUI components in XML by referencing the callback name:

```
<Button id="test_button" x="0" y="0" width="100" height="100" onClick=
↪"ButtonCallback" text="TEST" />
<SwitchButton id="test_switch" x="500" y="500" width="150" height="100"␣
↪onSwitchON="SwitchCallback" onSwitchOFF="SwitchCallback" />
```

3.3 Popups

Adding an element in XML with a callback to show a popup:

```
<Button id="btn" x="0" y="0" width="150" height="100" onClick="ShowPopup('example_
↪popup')" text="Show Popup" />
```

Adding a popup in XML (with an element closing it on callback):

```
<Popup id="example_popup" x="0" y="0" width="300" height="200">
    <button id="close_popup_btn" x="100" y="10" width="100" height="50" onClick=
↪"ClosePopup" text="Close" />
    <label id="popup_label" font="roboto-medium" x="0" y="100" width="300" height=
↪"50" text="This is a Popup." />
</Popup>
```

3.4 Adding images and fonts

```
displayManager->AddFontToFontContainer("my_font", new Font(path_to_font));
displayManager->AddImageContentToContainer("my_image", new␣
↪ImageContent(ImageContent::FromPNG(path_to_image)));
```

CHAPTER

FOUR

GENERAL API REFERENCE

class **Manager**

grvl manager class.

This class is used as an entry point for all operations invoked on the library.

Public Functions

uint32_t **GetWidth**() const

Returns
Width of the display in pixels.

uint32_t **GetHeight**() const

Returns
Height of the display in pixels.

Manager &**SetTransparency**(float *value*, uint32_t *milliseconds*)

Sets transparency. TODO - Adjust the description

> **Remark**
>
> Calling this method will invoke registered callback called 'set_layer_transparency' with provided argument. It does not change internal state of a library.

Parameters

float value
Transparency value.

Manager &**SetActiveScreen**(const char **activeScreenId*, int8_t *direction*)

Changes currently displayed screen with optional animation.

Parameters

const char *activeScreenId
Identifier of a screen to display.

int8_t direction
Direction of animation (-1 = to the left, 1 = to the right, 0 = no animation)

Manager &**SetLoadingImage**(const *Image* &*image*)

Sets an image that is displayed while the application is loading its resources.

> **Remark**
>
> In order to switch from loading to normal state, application should call InitializationFinished method.

Parameters

const *Image* &image
Reference to image object that should be displayed when loading.

Manager &**SetBackgroundImage**(const *Image* &*image*)

Sets image that is displayed below screens.

Parameters

const *Image* &image
Reference to the image to display.

Manager &**SetCollectionImage**(const *Image* &*image*)

Sets image that is displayed as screen collection indicator.

Parameters

const *Image* &image
Reference to the image to display.

Manager &**SetBackgroundColor**(uint32_t *color*)

Sets background color for underlying layer.

> **Remark**
>
> Calling this method will invoke registered callback called 'set_background_color' with provided argument. It does not change internal state of a library.

Parameters

uint32_t color
Background color in format acceptable by the provided callback.

void **ProcessTouchPoint**(bool *touched*, uint32_t *touchX*, uint32_t *touchY*)

Handles touch event.

Parameters

bool touched
Indicates if the display is touched.

uint32_t touchX
Position in axis X where the touch event was detected.

uint32_t touchY
Position in axis Y where the touch event was detected.

void **ShowPopup**(const char *_Style_)

Displays pop-up window.

Parameters

const char *Style
Name of the style defined in XML document.

void **ShowKeyboard**(TextInput *_destinationInput_)

Displays keyboard pop-up window if defined.

void **SwitchKeyboardKeys**()

Switches keyboard pop-up keys.

void **ClosePopup**()

Closes last pop-up window.

void **InitializationFinished**()

Switches library to normal state when screens are displayed.

> **Remark**
>
> Before calling this method loading image (if defined) is displayed.

AbstractView *__GetScreen__(const char *_id_)

Gets screen based on its name.

Parameters

const char *id
Name of the screen.

Returns
Pointer to a screen or NULL if not found.

AbstractView *__GetScreen__(int _id_)

Gets screen based on its location.

Parameters

int id
Numerical identifier of a screen (zero-based).

Returns
Pointer to the screen or NULL if not found.

Manager &**AddImageContentToContainer**(string *name*, *ImageContent* **image*)

Registers image content in content manager.

The image content will be accessible for other components by the provided name.

> **Remark**
>
> It is advised to use content manager when image content is shared by different components, as it will save memory.

Parameters

string name
Image content's identifier.

***ImageContent* *image**
Image content object.

Manager &**BindImageContentToImage**(const string &*contentName*, *Image* **image*)

Binds registered image content to an image object.

Parameters

const string &contentName
Identifier of the content.

***Image* *image**
Image object.

Manager &**AddCallbackToContainer**(const string &*name*, Event::CallbackPointer *Callback*)

Registers callback method for an event.

> **Remark**
>
> This method filters out empty callbacks (i.e., NULLs).

Parameters

const string &name
Identifier of an event that should invoke the callback.

Event::CallbackPointer Callback
Callback that should be called on event.

Event **GetOrCreateCallback**(const string &*callbackFunctionName*, const Event::ArgVector &*callbackArgs*)

Tries to search if there is callback defined with C/C++ code (added by AddCallbackToContainer), if not then it creates new one that will call JavaScript code with provided constant args.

Parameters

const string &callbackFunctionName
Identifier of a function invoked by the callback.

const Event::ArgVector &callbackArgs
Constant args that will be passed as callback function arguments

Manager &**AddFontToFontContainer**(const string &*name*, Font **font*)

Register font in content manager.

The font will be accessible for other components by the provided name.

> **Remark**
>
> It is advised to use content manager when font is used by multiple components, as it will save memory.

Parameters

const string &name
Font's identifier.

Font *font
Font object.

int32_t **BuildFromXML**(const char **filename*)

Loads screens from XML file.

Parameters

const char *filename
Path to the file with screens definition.

Returns
Result of parsing the file (0 = OK, -1 = there was an error)

void **Draw**()

Redraws content of a display based on current state of components.

void **MainLoopIteration**()

Executes an iteration of processing loop.

This method handles pop-up windows, processes events and redraw screen if needed.

Manager &**SetExternalContentRequestCallback**(ContentManager::ContentCallback *requestCallback*)

Registers method that is called when request for external content is issued by the library.

Parameters

ContentManager::ContentCallback requestCallback
Pointer to the method to call.

class **KeyData**

class **ImageContent**

Represents content of an image loaded into memory.

struct **FromJPEG** : public grvl::*ImageContent*::*FromPNG*

struct **FromPNG**

Subclassed by *grvl::ImageContent::FromJPEG*, *grvl::ImageContent::FromRAW*

struct **FromRAW** : public grvl::*ImageContent*::*FromPNG*

CHAPTER

FIVE

XML REFERENCE

5.1 XML validity

A schema is available in grvl along with an example that it validates. To check the validity, use an external tool, for example `xmllint`.

```
$ xmllint --schema <path_to_schema.xsd> <path_to_my_xml_grvl_file.xml>
```

5.2 How to construct a grvl XML document?

First define the root of the document.

```
<doc>
    <!-- screens/special components -->
</doc>
```

Then include the screens/special components in the document.

5.3 Generic component/container attributes

Each component has the following attributes:

- id
- x (absolute, in pixels)
- y (absolute, in pixels)
- width
- height
- visible (true/false)
- foregroundColor
- activeForegroundColor
- backgroundColor
- activeBackgroundColor
- borderColor
- activeBorderColor

- borderType (one of: none, box, top, right, bottom, left)
- borderArcRadius
- onClick (callback)
- onPress (callback)
- onRelease (callback)

Containers can also be specified as `selection`, which makes it a single-choice container, meaning that only one child component can be active and will remain active until another component from that container is activated.

5.4 Generic screen attributes

Each screen is a container, so it has all of the attributes from the *previous section* as well as:

- onSlideToLeft (callback)
- onSlideToRight (callback)
- onLongPress (callback)
- onLongPressRepeat(callback)
- collection (string)
- globalPanelVisible (true/false)

It can also contain `<key>` that can have the following attributes:

- id
- onPress
- onLongPress (callback)
- onLongPressRepeat(callback)
- onRelease

5.5 Special attributes

5.5.1 Colors

Uses the `#aarrggbb` format color.

5.5.2 Alignment

One of: `Center`, `Left`, `Right`.

5.5.3 Border type

One of: `none`, `box`, `top`, `right`, `bottom`, `left`.

5.6 List of components/containers

5.6.1 Label

Attributes

- text
- font
- textColor
- alignment

Example

```
<label id="popup_label" font="roboto-medium" x="0" y="100" width="300" height="50
↪" text="This is a label." />
```

5.6.2 Button

Attributes

- text
- font
- image
- textColor
- activeTextColor
- icoChar
- icoFont
- icoColor
- onLongPress
- onLongPressRepeat

Example

```
<button id="close_popup_btn" x="100" y="10" width="100" height="50"␣
↪backgroundColor="#ffffffff" textColor="#ff0000ff" activeTextColor="#ffff00ff"␣
↪onClick="ClosePopup" text="Close" font="roboto-medium" />
```

5.6.3 Clock

Attributes

- font
- alignment
- onRelease (callback)
- foregroundColor

- seconds (true/false)

Example

```
<clock id="test_clock" x="150" y="10" width="100" height="100" font="roboto-medium
↪" foregroundColor="#fffff000" seconds="true" />
```

5.6.4 Slider

Attributes

- frameColor
- selectedFrameColor
- barColor
- activeBarColor
- activeScrollColor
- maxValue
- minValue
- keepBoundaries
- isDiscrete
- division
- font
- image
- onValueChange

Example

```
<slider id="vertical_slider" x="300" y="10" width="20" height="100" scrollColor="
↪#ffff0000" activeBarColor="#ff00ff00" />
```

5.6.5 ListItem

Attributes

- text
- font
- description
- descriptionFont
- ActiveTextColor
- descriptionColor
- activeDescriptionColor
- image

- additionalImage
- roundingImage
- onLongPress
- onLongPressRepeat
- type (one of: StdListField, LeftArrowField, RightArrowField, EmptyField, Dots, DoubleImageField, AlarmField)

Example

```
<ListItem backgroundColor="#ffff0000" id="item_1" height="50" type="StdListField"␣
↪text="first" textColor="#ffffffff" font="roboto-medium" />
```

5.6.6 ProgressBar

Attributes

- progressBarColor

Example

```
<ProgressBar id="progress_bar" x="90" y="60" width="50" height="10"␣
↪progressBarColor="#ffffff00" />
```

5.6.7 CircleProgressBar

Attributes

- progressBarColor
- startColor
- endColor
- startAngle
- endAngle
- radius
- thickness
- staticGradient

Example

```
<CircleProgressBar id="circle_progress_bar" x="175" y="120" width="50" height="50
↪" radius="20" thickness="5" startColor="#fffcba03" endColor="#ffa903fc" />
```

5.6.8 SwitchButton

Attributes

Derives all attributes from `Button` with the addition of:

- onSwitchON - callback executed when switching to active state
- onSwitchOFF - callback executed when switching to inactive state
- stateIndicatorWidth/stateIndicatorHeight - size dimensions of state indicator
- stateIndicatorArcRadius - arc radius of state indicator

Example

```
<SwitchButton id="test_switch" x="0" y="60" width="80" height="50"␣
↪stateIndicatorArcRadius="5" stateIndicatorWidth="20" stateIndicatorHeight="20"␣
↪foregroundColor="#fffcba03" backgroundColor="#ffffffff" font="roboto"␣
↪onSwitchON="SwitchCallback" onSwitchOFF="SwitchCallback" />
```

5.6.9 Checkbox

Attributes

Derives all attributes from *SwitchButton*, but renders its state indicator from width and height size dimensions, instead of state indicator parameters.

Example

```
<Checkbox id="active" x="0" y="0" width="18" height="18" backgroundColor="
↪#FFAAAAAA" activeBackgroundColor="#FFFFFFFF" onClick="NotifyAboutStateChange" />
```

5.6.10 Image

Attributes

- contentId

Example

```
<image contentId="minus" x="253" y="131" />
```

5.6.11 GridRow

Example

```
<GridRow id="row_1" backgroundColor="#ffff0000">
        <button id="row_1_btn_1" text="1" font="roboto-medium" width="50" height=
↪"50" backgroundColor="#ff00ffff" />
        <button id="row_1_btn_2" text="2" font="roboto-medium" width="50" height=
↪"50" backgroundColor="#ffff00ff" />
</GridRow>
```

5.6.12 Division

Component designed to be used as a container for other components, with the purpose of making XML layout clean and well-organized. It has a similar purpose to HTML's `div`.

Example

```
<Division x="0" y="0" width="128" height="32" backgroundColor="#FF0E0F10" >
    <Button id="first" text="First" x="0" y="0" width="32" height="32" font=
↪"roboto" backgroundColor="#FF0E0F10" textColor="#FF9EA2A6" />
    <Button id="second" text="Second" x="42" y="0" width="32" height="32" font=
↪"roboto" backgroundColor="#FF0E0F10" textColor="#FF9EA2A6" />
    <Button id="third" text="Third" x="86" y="0" width="32" height="32" font=
↪"roboto" backgroundColor="#FF0E0F10" textColor="#FF9EA2A6" />
</Division>
```

5.6.13 Separator

Used for horizontal separation of content.

Example

```
<Separator id="separator" x="0" y="20" width="128" foregroundColor="#FF2E2E2E" />
```

5.7 List of screens

5.7.1 GridView

Arranges its chilren elements in a grid layout.

Attributes

- overscrollEnabled
- overscrollHeight
- scrollingEnabled
- overscrollColor
- elementWidth
- elementHeight
- verticalOffset

Example

```
<GridView id="grid" x="0" y="0" width="140" height="140" backgroundColor="
↪#FF0A0A0A" elementWidth="40" elementHeight="40" horizontalOffset="5"␣
↪verticalOffset="5" selection="true" >

    <GridRow id="monthdays1" >
        <button id="0" text="0" font="mona14" backgroundColor="#FF0A0A0A"␣
↪activeBackgroundColor="#FF0059EC" textColor="#FFECEDEE" onClick="SomeCallback" /
↪>
        <button id="1" text="1" font="mona14" backgroundColor="#FF0A0A0A"␣
↪activeBackgroundColor="#FF0059EC" textColor="#FFECEDEE" onClick="SomeCallback" /
↪>
```

(continues on next page)

(continued from previous page)

```
        <button id="2" text="2" font="mona14" backgroundColor="#FF0A0A0A"␣
↪activeBackgroundColor="#FF0059EC" textColor="#FFECEDEE" onClick="SomeCallback" /
↪>
    </GridRow>

    <GridRow id="monthdays2" >
        <button id="3" text="3" font="mona14" backgroundColor="#FF0A0A0A"␣
↪activeBackgroundColor="#FF0059EC" textColor="#FFECEDEE" onClick="SomeCallback" /
↪>
        <button id="4" text="4" font="mona14" backgroundColor="#FF0A0A0A"␣
↪activeBackgroundColor="#FF0059EC" textColor="#FFECEDEE" onClick="SomeCallback" /
↪>
        <button id="5" text="5" font="mona14" backgroundColor="#FF0A0A0A"␣
↪activeBackgroundColor="#FF0059EC" textColor="#FFECEDEE" onClick="SomeCallback" /
↪>
    </GridRow>

    <GridRow id="monthdays3" >
        <button id="6" text="6" font="mona14" backgroundColor="#FF0A0A0A"␣
↪activeBackgroundColor="#FF0059EC" textColor="#FFECEDEE" onClick="SomeCallback" /
↪>
        <button id="7" text="7" font="mona14" backgroundColor="#FF0A0A0A"␣
↪activeBackgroundColor="#FF0059EC" textColor="#FFECEDEE" onClick="SomeCallback" /
↪>
        <button id="8" text="8" font="mona14" backgroundColor="#FF0A0A0A"␣
↪activeBackgroundColor="#FF0059EC" textColor="#FFECEDEE" onClick="SomeCallback" /
↪>
    </GridRow>

</GridView>
```

5.7.2 ListView

Arranges its chilren elements into a list.

Attributes

- overscrollEnabled
- overscrollHeight
- splitLineColor
- overscrollColor
- scrollIndicatorColor

Example

```
<ListView id="list" x="0" y="0" width="200" height="200">

    <ListItem backgroundColor="#ffff0000" id="item1" height="50" type=
```

(continues on next page)

(continued from previous page)

```
↪"StdListField" text="first" textColor="#ffffffff" font="mona12" />
    <ListItem backgroundColor="#ff00ff00" id="item2" height="50" type=
↪"StdListField" text="second" textColor="#ffffffff" font="mona12" />
    <ListItem backgroundColor="#ff0000ff" id="item3" height="50" type=
↪"StdListField" text="third" textColor="#ffffffff" font="mona12" />

</ListView>
```

5.7.3 ScrollPanel

Allows to arrange its children elements inside the container in a custom way, defined by each component's x and y.

Attributes

- overscrollEnabled
- overscrollHeight
- splitLineColor
- overscrollColor
- scrollIndicatorColor

Example

```
<ScrollPanel id="scroll" x="0" y="0" width="100" height="100" overscrollBarColor="
↪#FF0E0F10" >

    <Button id="add" text="Add" font="mona12" x="5" y="5" width="40" height="40"␣
↪textColor="#FFFF575E" onClick="SomeCallback" />
    <Button id="new" text="New" font="mona12" x="55" y="5" width="40" height="40"␣
↪textColor="#FFFF575E" onClick="SomeCallback" />
    <Button id="cancel" text="Cancel" font="mona12" x="5" y="50" width="90"␣
↪height="40" textColor="#FFFF575E" onClick="SomeCallback" />

</ScrollPanel>
```

5.7.4 Other screens

Screens with no special attributes

CustomView

```
<customView id="start" backgroundColor="#FF4287F5">
        <button id="test_button" image="button_image" x="10" y="10" width="100"␣
↪height="100" textColor="#ff00ff00" activeTextColor="#ffff00ff" onClick=
↪"ButtonCallback" text="TEST" font="roboto-medium" />
    <clock id="test_clock" x="150" y="10" width="100" height="100" font="roboto-
↪medium" foregroundColor="#fffff000" seconds="true" />
```

(continues on next page)

(continued from previous page)

```
    <slider id="vertical_slider" x="300" y="10" width="20" height="100"␣
↪scrollColor="#ffff0000" activeBarColor="#ff00ff00" />
    <slider id="horizontal_slider" x="340" y="10" width="100" height="20"␣
↪scrollColor="#fffcba03" activeBarColor="#ffa903fc" />
</customView>
```

Header

```
<header height="50" backgroundColor="#ff000000">
    <label id="h_label" text="Header (Label)" font="roboto-medium" alignment=
↪"Center" x="0" y="0" width="800" height="50" />
</header>
```

Popup

```
<popup id="test_popup" backgroundColor="#ff000000" x="250" y="200" width="300"␣
↪height="200">
        <button id="close_popup_btn" x="100" y="10" width="100" height="50"␣
↪backgroundColor="#ffffffff" textColor="#ff0000ff" activeTextColor="#ffff00ff"␣
↪onClick="ClosePopup" text="Close" font="roboto-medium" />
        <label id="popup_label" font="roboto-medium" x="0" y="100" width="300"␣
↪height="50" text="This is a Popup." />
</popup>
```

5.8 List of special components

5.8.1 guiConfig

Attributes

- touchRegionModificator
- dotColor
- dotActiveColor
- dotDistance
- dotRadius
- dotYPos
- debugDot

Example

```
<guiConfig dotColor="#ffffffff"></guiConfig>
```

5.8.2 stylesheet (UNIMPLEMENTED)

The current version of grvl only implements parsing, the stylesheet is not applied.

Example

```
<stylesheet>
    label {
        bgColor: "red"
    }
</stylesheet>
```

5.8.3 keypadMapping

Contains key elements with the following attributes:

- id
- code (int)
- repeat (int)

Example

```
<keypadMapping>
    <key id="space" code="456" repeat="200"/>
    <key id="enter" code="789" repeat="100"/>
</keypadMapping>
```

CHAPTER

SIX

JAVASCRIPT REFERENCE

6.1 JavaScript feature support

The JavaScript engine used in grvl is duktape. For more information about its capabilities, see duktape docs.

6.2 How to use JavaScript with grvl?

First, prepare your JavaScript source file:

```
// callbacks.js

var currentDate = null

function ButtonCallback(caller) {
    const buttonId = caller.name
    Print("Button clicked! (" + buttonId + ")")
}

function InitializeApplication() {
    currentDate = new Date()
}
```

Then, include it in your application's XML layout:

```
<script src="callbacks.js"></script>
```

You can specify a non-default working directory for JavaScript files with:

```
grvl::JSEngine::SetSourceCodeWorkingDirectory("Scripts/JavaScript/");
```

You can later bind JavaScript functions to GUI components by referencing a function name in callback parameters, e.g. `onClick`. The caller component will be passed in as the first argument:

```
<button id="button" x="10" y="140" width="200" height="80" onClick="ButtonCalback
↪" text="I am a button" font="roboto" />
```

JavaScript functions can be also called from any place in code without involving GUI components like this:

```
grvl::JSEngine::MakeJavaScriptFunctionCall("InitializeApplication");
```

6.3 grvl functions available in JavaScript

You can call functions exposed by grvl to control your application from JavaScript. They are divided into two types: globally available functions and GUI component accessors.

6.3.1 Globally available functions

These functions can be called from any place in JavaScript code:

- `GetElementById(componentID)` - returns component with given ID if found
- `Print(message)` - prints given message
- `ShowPopup(popupID)` - shows popup with given ID if available
- `ClosePopup()` - closes currently shown popup
- `SetActiveScreen(screenID)` - sets screen with given ID as active
- `GetTopPanel()` - returns top panel component
- `GetBottomPanel()` - returns bottom panel component
- `GetPrefabById(prefabID)` - returns prefab with given ID if available

6.3.2 Members

These functions can be used to access GUI component parameters or call their member functions.

Properties

The simple parameters of a GUI component can be modified with properties exposed by each component, e.g.:

```
Print(caller.name)
```

will print `caller`'s ID. There are other common properties such as:

- `x`, `y`, `width` and `height` for position and size dimensions respectively
- `foregroundColor` and `backgroundColor` for component's foreground and background color
- `visibility` which determines if a component is visible.

Metadata

Each component's metadata can be accessed by calling `AddMetadata(key, value)` or `GetMetadata(key)`:

```
button.AddMetadata("description", "grvl is very cool")
...
const description = button.GetMetadata("description")
```

Cloning

GUI components can be cloned inside JavaScript code by calling:

```
const clone = caller.Clone()
container.AddElement(clone)
```

CHAPTER

SEVEN

PREFABS

Prefabs are user-defined composites of already existing GUI components and can be used to instantiate complex structures at runtime. By using prefabs you can create new instances of hierarchical GUI components without delving into their internals.

Prefabs can be defined in application's XML layout:

```
<prefab id="event" width="95" height="60" borderType="left" >

    <label id="name" text="team planning" font="mona16" x="0" y="0" width="138"␣
↪height="35" horizontalOffset="8" textColor="#FF47A8FF" alignment="left" />
    <label id="description" font="mona14" x="0" y="25" width="138" height="35"␣
↪horizontalOffset="8" textColor="#FF47A8FF" alignment="left" visibility="false" /
↪>

</prefab>
```

and then later instantiated at runtime with:

```
const prefab = GetPrefabById("event")
const event = prefab.Clone()
events.AddElement(event)
```

CHAPTER

EIGHT

METADATA

Metadata gives an ability to store complex data and information for later use in GUI components. Metadata values can be accessed by calling components' member functions `AddMetadata(key, value)` and `GetMetadata(key)`:

```
button.AddMetadata("description", "grvl is very cool")
...
const description = button.GetMetadata("description")
```

CHAPTER

NINE

ON-SCREEN KEYBOARD

To provide an interactive experience for the user, you can add a keyboard, along with text inputs, to your grvl application. The keyboard can be specified in the application XML layout:

```
<Keyboard id="keyboard" backgroundColor="#FF0D0D0E" x="0" y="430" width="1024"␣
↪height="338">

    <KeyboardKey id="g" text="g" font="mona26" secondaryText="a"␣
↪secondaryTextFont="mona14" x="25" y="27" width="75" height="62" backgroundColor=
↪"#FF191A1C" textColor="#FFECEDEE" secondaryTextColor="#FF191A1C" onClick=
↪"AppendFromKeyboardKey" />
    <KeyboardKey id="r" text="r" font="mona26" secondaryText="n"␣
↪secondaryTextFont="mona14" x="115" y="27" width="75" height="62"␣
↪backgroundColor="#FF191A1C" textColor="#FFECEDEE" secondaryTextColor="#FF191A1C
↪" onClick="AppendFromKeyboardKey" />
    <KeyboardKey id="v" text="v" font="mona26" secondaryText="t"␣
↪secondaryTextFont="mona14" x="205" y="27" width="75" height="62"␣
↪backgroundColor="#FF191A1C" textColor="#FFECEDEE" secondaryTextColor="#FF191A1C
↪" onClick="AppendFromKeyboardKey" />
    <KeyboardKey id="l" text="l" font="mona26" secondaryText="m"␣
↪secondaryTextFont="mona14" x="295" y="27" width="75" height="62"␣
↪backgroundColor="#FF191A1C" textColor="#FFECEDEE" secondaryTextColor="#FF191A1C
↪" onClick="AppendFromKeyboardKey" />

</Keyboard>
```

You can specify any layout you require by defining individual keys. The keyboard will be shown whenever a text input is used.

```
<TextInput id="name" basicText="Name" font="mona26"
    x="0" y="0" width="426" height="34"
    borderType="box" borderArcRadius="12"
    backgroundColor="#FF191A1C" borderColor="#FF2C2E32" textColor="#FF9EA2A6" />
```

CHAPTER

TEN

WIDGET API REFERENCE

class **Label** : public grvl::Component

Widget displaying static text.

XML parameters:

- id - widget identifier
- x - widget position on x axis in pixels
- y - widget position on y axis in pixels
- width - widget width in pixels
- height - widget height in pixels
- visible - indicates if the widget is visible (default: true)
- text - text to display on the label (default: none)
- font - text font (default: normal)
- textColor - text color (default: black)
- backgroundColor - widget background color (default: transparent)
- frameColor - widget frame color (default: transparent)
- alignment - horizontal alignment of the text (default: center)

Subclassed by *grvl::Clock*

Public Functions

const char ***GetText**()

Returns
Label's text.

class **Button** : public grvl::AbstractButton

Represents rectangle button.

XML parameters:

- id - widget identifier
- x - widget position on x axis in pixels
- y - widget position on y axis in pixels

- width - widget width in pixels
- height - widget height in pixels
- visible - indicates if the widget is visible
- text - caption text to display on the widget (default: “”)
- font - caption text font (default: “normal”)
- textColor - caption text color (default: black)
- activeTextColor - caption text color while pressed (default: textColor)
- icoChar - single-character icon for the button (default: none)
- icoFont - single-character icon font (default: “normal”)
- icoColor - single-character icon color (default: textColor)
- activeIcoColor - single-character icon color while pressed (default: textColor)
- backgroundColor - button background color (default: transparent)
- activeBackgroundColor - button background color while pressed (default: backgroundColor)
- frameColor - button frame color (default: transparent)
- text_top_offset - caption text top position offset
- image - identifier of image content to display
- image_x - image position on x axis in pixels (setting to -1 will center the image)
- image_y - image position on y axis in pixels (setting to -1 will center the image)

XML events:

- onClick - event invoked when touch is released, but only when it has not left widget boundaries since pressing and long press was not reported
- onPress - event invoked when touch is detected within widget boundaries
- onRelease - event invoked when touch is released within widget boundaries or when it leaves the boundaries
- onLongPress - event invoked when the widget is pressed for longer than a second
- onLongPressRepeat - event invoked periodically (every half a second) while the widget is being pressed

Subclassed by grvl::KeyboardKey, grvl::TextInput

class **Clock** : public grvl::*Label*

Widget displaying current time.

XML parameters:

- id - widget identifier
- x - widget position on x axis in pixels
- y - widget position on y axis in pixels

- width - widget width in pixels
- height - widget height in pixels
- visible - indicates if the widget is visible
- foregroundColor - text color (default: black)
- backgroundColor - background color (default: transparent)
- font - text font (default: normal)
- alignment - horizontal text alignment (default: centered)

XML events:

- onClick - event invoked when touch is released, but only when it has not left widget boundaries since pressing
- onPress - event invoked when touch is detected within widget boundaries
- onRelease - event invoked when touch is released within widget boundaries or when it leaves the boundaries

class **`Image`** : public grvl::Component

Widget displaying an image.

XML parameters:

- id - widget identifier
- x - widget position on x axis in pixels
- y - widget position on y axis in pixels
- visible - indicates if the widget is visible
- contentId - identifier of image content to display (default: none)

> **Remark**
>
> Width and height of a widget are deduced from image content.

class **`ProgressBar`** : public grvl::Component

Rectangle progress bar widget.

XML parameters:

- id - widget identifier
- x - widget position on x axis in pixels
- y - widget position on y axis in pixels
- width - widget width in pixels
- height - widget height in pixels
- visible - indicates if the widget is visible

- progressBarColor - progress bar color (default: transparent)
- backgroundColor - background color (default: transparent)

Subclassed by *grvl::CircleProgressBar*

class **`CircleProgressBar`** : public grvl::*ProgressBar*

Represents circular progress bar widget.

XML parameters:

- id - widget identifier
- x - widget position on x axis in pixels
- y - widget position on y axis in pixels
- width - widget width in pixels
- height - widget height in pixels
- visible - indicates if the widget is visible
- startAngle - angle at which the ring starts in degrees (default: 0)
- endAngle - angle at which the ring ends in degrees (default: 360)
- radius - outer rim of the ring radius in pixels (default: 0)
- thickness - ring thickness in pixels (default: 0)
- staticGradient - indicates if static gradient method should be used instead of proportional one (default: true)
- startColor - gradient color at the beginning of the ring in ARGB8888 format (default: transparent)
- endColor - gradient color at the end of the ring in ARGB8888 format (default: transparent)
- backgroundColor - ring background color

class **`GridView`** : public grvl::VerticalScrollView

Grid view widget.

This is a layout widget which organizes the screen by displaying inner widgets on a grid.

XML parameters:

- id - widget identifier
- x - widget position on x axis in pixels
- y - widget position on y axis in pixels
- width - widget width in pixels
- height - widget height in pixels
- visible - indicates if the widget is visible
- scrollingEnabled - indicates if scrolling is enabled (default: false)

- overscrollEnabled - indicates if scrolling beyond top/bottom is possible (default: false)
- overscrollColor - overscrolled area color (default: light gray)
- overscrollHeight - overscrolled area height in pixels (default: 50)
- scrollIndicatorColor - scroll indicator color (default: transparent)
- elementWidth - grid element width in pixels (default: 0)
- elementHeight - grid element height in pixel (default: 0)
- verticalOffset - vertical offset of grid elements (default: 0)
- globalPanelVisible - indicates if global panel is visible (default: true)
- backgroundColor - widget background color (default: transparent)
- collection - collection of elements

XML events:

- onSlideLeft - event invoked when widget is scrolled to the left
- onSlideRight - event invoked when widget is scrolled to the right

Remark

XML node describing this widget can contain child nodes with components of type:

- gridRow
- panel (as a header)

Public Functions

virtual void **SetBackgroundColor**(uint32_t *color*)

Sets component's background color.

Parameters

uint32_t color
Desired color in ARGB8888 format.

class **GridRow** : public grvl::Container

Widget representing single row of a grid displayed by *GridRow*. The widget can contain buttons only.

class **CustomView** : public grvl::AbstractView

Represents a screen type supporting any element placement except for listItems and gridRows.

class **ListView** : public grvl::VerticalScrollView

Vertical list view widget.

XML parameters:

- id - widget identifier
- x - widget position on x axis in pixels
- y - widget position on y axis in pixels
- width - widget width in pixels
- height - widget height in pixels
- visible - indicates if the widget is visible
- overscrollEnabled - indicates if scrolling beyond top/bottom is possible (default: false)
- overscrollColor - overscrolled area color (default: light gray)
- overscrollHeight - overscrolled area height in pixels (default: 50)
- scrollIndicatorColor - scroll indicator color (default: transparent)
- splitLineColor - color of the line drawn between list elements (default: transparent)
- globalPanelVisible - indicates if global panel is visible (default: true)
- backgroundColor - widget background color (default: transparent)
- collection - collection of elements

XML events:

- onSlideLeft - event invoked when widget is scrolled to the left
- onSlideRight - event invoked when widget is scrolled to the right

Remark

XML node describing this widget can contain child nodes with components of type:

- listItem
- panel (as a header)

class **`ListItem`** : public grvl::AbstractButton

Widget representing an element of a list displayed by *ListView*.

XML events:

- id - widget identifier
- height - widget height in pixels
- visible - indicates if the widget is visible
- type - list item type; available values: StdListField, LeftArrowListField, RightArrowListField, EmptyField, Dots, DoubleImageField
- text - caption text (default: none)

- description - description text (default: none)
- font - caption text font (default: “normal”)
- descriptionFont - description text font (default: “small”)
- textColor - caption text color (default: black)
- activeTextColor - caption text color when pressed (default: textColor)
- descriptionColor - description text color (default: black)
- activeDescriptionColor - description text color when pressed (default: descriptionColor)
- backgroundColor - background color (default: transparent)
- activeBackgroundColor - background color when pressed (default: backgroundColor)
- image - image content identifier
- additionalImage - second image content identifier

Public Functions

void **SetType**(ItemType *type*)

Sets type of the list item.

Parameters

ItemType type
New list item’s type.

Image ***GetAdditionalImagePointer**()

Returns
Pointer to the second image set for this item or NULL.

class **Panel** : public grvl::Container

Represents panel widget.

Panel is a fixed top part of a screen that displays widgets that should be always visible.

XML parameters:

- id - identifier of a widget
- height - widget height in pixels (default: 30)
- backgroundColor - background color (default: transparent)

Remark

XML node describing this widget can contain child nodes with components of type:

- guiClock
- image

- textView
- button
- switch
- scrollBar
- listItem
- progressBar

class **SwitchButton** : public grvl::AbstractButton

Represents switch (flip-flop) button.

XML parameters:

- id - widget identifier
- x - widget position on x axis in pixels
- y - widget position on y axis in pixels
- width - widget width in pixels
- height - widget height in pixels
- visible - indicates if the widget is visible

XML events:

- onClick - event invoked when touch is released, but only when it has not left the boundaries of widget since pressing and long press was not reported
- onPress - event invoked when touch is detected in the boundaries of the widget
- onRelease - event invoked when touch is released in the boundaries of the widget
- onLongPress - event invoked when the widget is pressed for longer than a second
- onLongPressRepeat - event invoked periodically (every half a second) while the widget is being pressed
- onSwitchON - event invoked when switch button is switched to ON mode
- onSwitchOFF - event invoked when switch button is switched to OFF mode

Subclassed by grvl::Checkbox

class **Slider** : public grvl::Component

Horizontal scroll bar widget.

XML parameters:

- id - widget identifier
- x - widget position on x axis in pixels
- y - widget position on y axis in pixels
- width - widget width in pixels

- height - widget height in pixels
- visible - indicates if the widget is visible (default: true)
- backgroundColor - background color (default: transparent)
- activeBackgroundColor - background color when pressed (default: backgroundColor)
- frameColor - frame color (default: transparent)
- selectedFrameColor - frame color when pressed (default: frameColor)
- barColor - scroll bar background color (default: white)
- activeBarColor - scroll bar background color when pressed (default: barColor)
- scrollColor - scrolling element color (default: blue)
- activeScrollColor - scrolling element color when pressed (default: scrollColor)
- minValue - minimal value (the one on the left) (default: 0)
- maxValue - maximal value (the one on the right) (default: 100)
- image - identifier of image content replacing standard scrolling element

XML events:

- onClick - event invoked when touch is released, but only when it has not left the boundaries of widget since pressing
- onPress - event invoked when touch is detected in the boundaries of the widget
- onRelease - event invoked when touch is released
- onValueChange - event invoked when the scrolling element is moved to another value

Public Functions

float **`GetValue`**`()` const

Returns
Scroll bar's current position.

INDEX

G