Amulet

GEMstudio Pro

User's guide
2018

Table of Contents

Welcome

Congratulations on your Amulet purchase! Please take a few minutes to read this file which contains the latest information about setting up and using your Amulet product. This documentation is valid for GEMstudio Pro software version 3.4.0.0

Overview

The Amulet method of displaying graphics on an LCD is totally different from traditional methods. The Amulet GEM Graphical OS Chip handles all the LCD and touchpanel functions so your microcontroller doesn't have to. Hardware wise, the only requirement is that your microcontroller needs a UART or USB in order to use the Amulet system. On the software side, you need to create an Amulet serial protocol handler. Generally, the only thing that is being sent via the serial link is data.

Here's the Amulet system in a nutshell.

1. GUI authoring tool called GEMstudio Pro is used to create a Graphical User Interface(GUI).
2. GEMstudio Pro is used to compile your GUI to a small binary file which is then downloaded into the Amulet module.
3. The Amulet module displays the GUI and handles all touchpanel interaction.
4. The Amulet module receives input data from your host microcontroller via the serial link and also sends command messages back to your microcontroller based upon timer-based or touchpanel events.

There are two types of variables in the Amulet system. External variables (byte, word, color and string) which reside on your microcontroller's side and InternalRAM variables (byte, word, color and string) which reside on the Amulet module. InternalRAM is a quasi-dual port RAM that can be read from and written to by the Amulet chip through commands inserted in the GEMstudio Pro code. Your microcontroller interfaces to InternalRAM through the serial link. There are specific UART and USB commands that can read from and write to InternalRAM.

There are four major types of serial messages that will be sent between the Amulet module and your microcontroller.

1. A request of a variable (byte, word, color or string)
2. A setting of a variable (byte, word, color or string)
3. A Remote Procedure Call, which is a completely generic message which allows the Amulet module to inform your microcontroller of a certain event. You can have up to 256 unique RPC's. What those RPC's signify is entirely up to you.
4. A raw byte, or group of bytes, can be sent from the Amulet module to your microcontroller. This option is not part of the Amulet serial protocol, but rather it gives you the flexibility to have the Amulet module send you small commands that do not need to be answered. In addition, a scripting language gives access to the serial receive byte interrupt and a custom handler can be written to define your own serial protocol.

Please see the Amulet UART Protocol documentation for more details.

Your LCD's user interface is created using GEMstudio Pro. Amulet has created a number of I/O objects, referred to as Amulet Widgets. There are two types of Amulet Widgets, Control Widgets and View Widgets. Control Widgets are input objects, like function buttons, sliders, radio buttons, etc... Control Widgets have a function, or a set of functions, that can be applied to them. For instance, a function button can be set to send a Remote Procedure Call #5 out the UART or USB every time it is pressed. View Widgets are output objects, like bargraphs, numeric fields, string fields, etc... View Widgets call a function which returns the data used as the input for that particular widget. For instance, a bargraph can have a function which requests external byte variable #3 every 100ms over the UART or USB. When the Host microcontroller replies to that message, the bargraph automatically redraws itself with the new value.

The complexity of the serial protocol handler depends upon the type of communication you will be using in your system. You can set up your project so that the Amulet module is the master, requesting data at given update rates and sending command messages asynchronously. Or your microcontroller can be the master, sending data to the Amulet module unsolicited. And you can also use a dual master setup, where the Amulet module is sending asynchronous command messages to your microcontroller, yet your microcontroller is also sending unsolicited data to the Amulet.

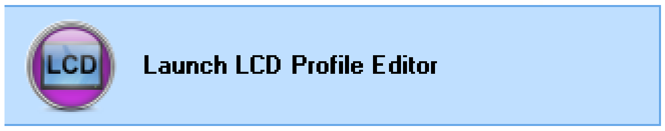
GEMstudio IDE Features

Overview

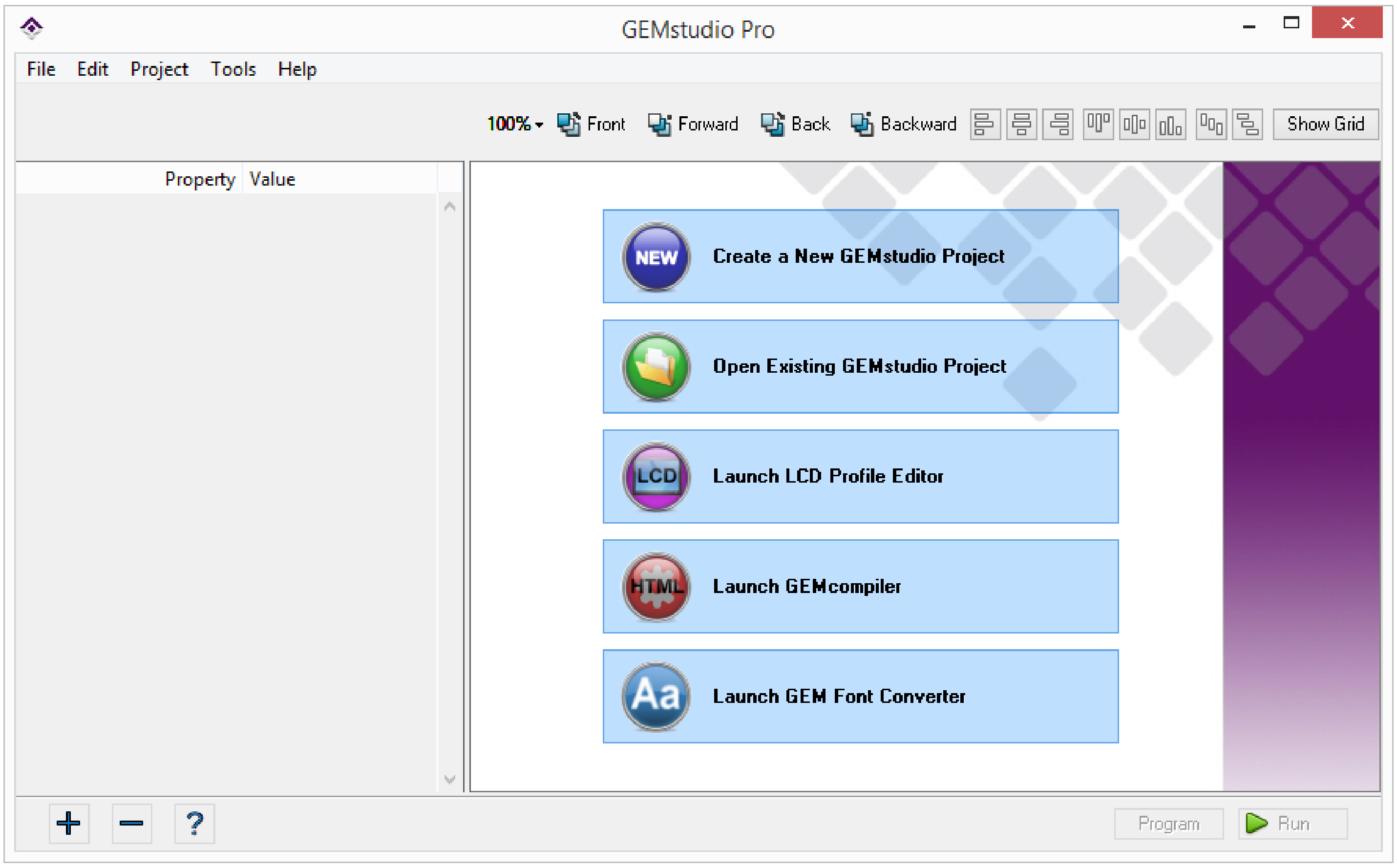
There are many several different dialog boxes accessible through the various menus and other types of popups. This section will give an overview of each dialog box and how they are used.

Create LCD Configuration

If there is no matching LCD selection available, the user can customize their own LCD setttings and save as a specific LCD profile.
On the main window of GEMstudio, click on

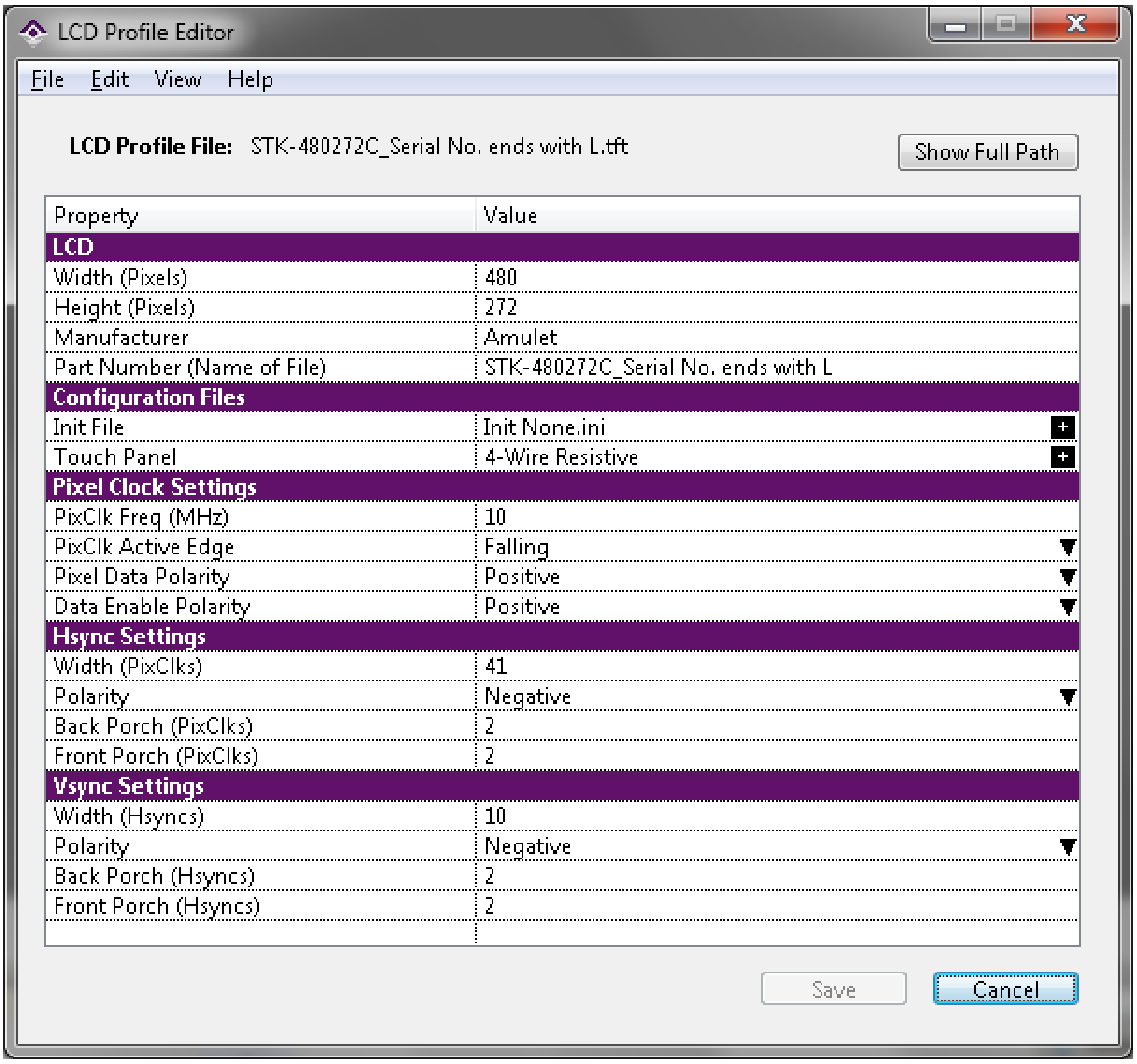


button.



A new window will pop up for user to enter their customizations.

The LCD Profile Editor window enables user to select the manufacturer, width and height, part number and color depth of the LCD display. This is also where initialization files can be specified. The pixel clock, horizontal sync, and vertical sync settings can all be configured based on the requirements of the specific display.



LCD Profile File: STK-480272C_Serial No. ends with L.tft

Property	Value
LCD	
Width (Pixels)	480
Height (Pixels)	272
Manufacturer	Amulet
Part Number (Name of File)	STK-480272C_Serial No. ends with L
Configuration Files	
Init File	Init None.ini
Touch Panel	4-Wire Resistive
Pixel Clock Settings	
PixClk Freq (MHz)	10
PixClk Active Edge	Falling
Pixel Data Polarity	Positive
Data Enable Polarity	Positive
Hsync Settings	
Width (PixClks)	41
Polarity	Negative
Back Porch (PixClks)	2
Front Porch (PixClks)	2
Vsync Settings	
Width (Hsyncs)	10
Polarity	Negative
Back Porch (Hsyncs)	2
Front Porch (Hsyncs)	2

Removing Touchpanel Calibration

The default mode of resistive touchpanel configurations is to boot to a touchpanel calibration upon first boot/reinstallation of OS. This is done for multiple reasons:

1) The touchpanel configuration file is overwritten when programming Amulet OS.
2) Changes in the unit's environment (temperature, humidity) can affect resistive touch performance.

If desired, it is possible to remove touchpanel calibration from the process and hard code the calibration constants into the OS. To remove calibration, the first step is to determine your hard coded calibration constants. To do this, first set up a test page with the following HREF in a functionButton:

Amulet:InternalRAM.word(z).calArraySet()

This sets the calibration constants into successive Internal RAM word variables, starting with z.
InternalRAM.word(z)= mx128
InternalRAM.word(z+1) = bx
InternalRAM.word(z+2) = my128

InternalRAM.word(z+3) = by

The calibration constants can be read by four Numeric Field widgets on the same test page that displays these four Internal RAM word variables. They can then be entered into a touch panel configuration file. In Windows Vista or later, these calibration files are located at:

C:\ProgramData\AmuletTech\Global\Configuration\TouchPanel

If "tpCalibration=0x00" and the four constants added to a touch panel configuration file, touch panel calibration will not be required upon loading a new OS.

The four variables are referenced in the touch panel configuration file as:
tpMx128=
tpBx=
tpMy128=
tpBy=

See STK-CY-043.ini touch panel configuration file in ProgramData/AmuletTech/Global/Configuration/TouchPanel for an example of a configuration that skips calibration. After saving these values into your Touchpanel Calibration file and, if you saved with a new file name, making sure that your LCD configuration is using that calibration file, GEMstudio will detect the change in your setup and automatically recompile the OS. The next time you program any project, it will not require calibration.

Note that these can still be changed at runtime by calling the Amulet:calibrate() command or by resetting/booting with the touchpanel calibration signal (TPCAL) switched to High.

LCD/Board Chooser

The LCD/Board Chooser can be opened from a drop down menu (Tools --> LCD/Board Chooser) on the top of the project window as shown in Figure 1 or the LCD tab of Project Properties when the user creates a new GEMstudio project as shown in Figure 2.

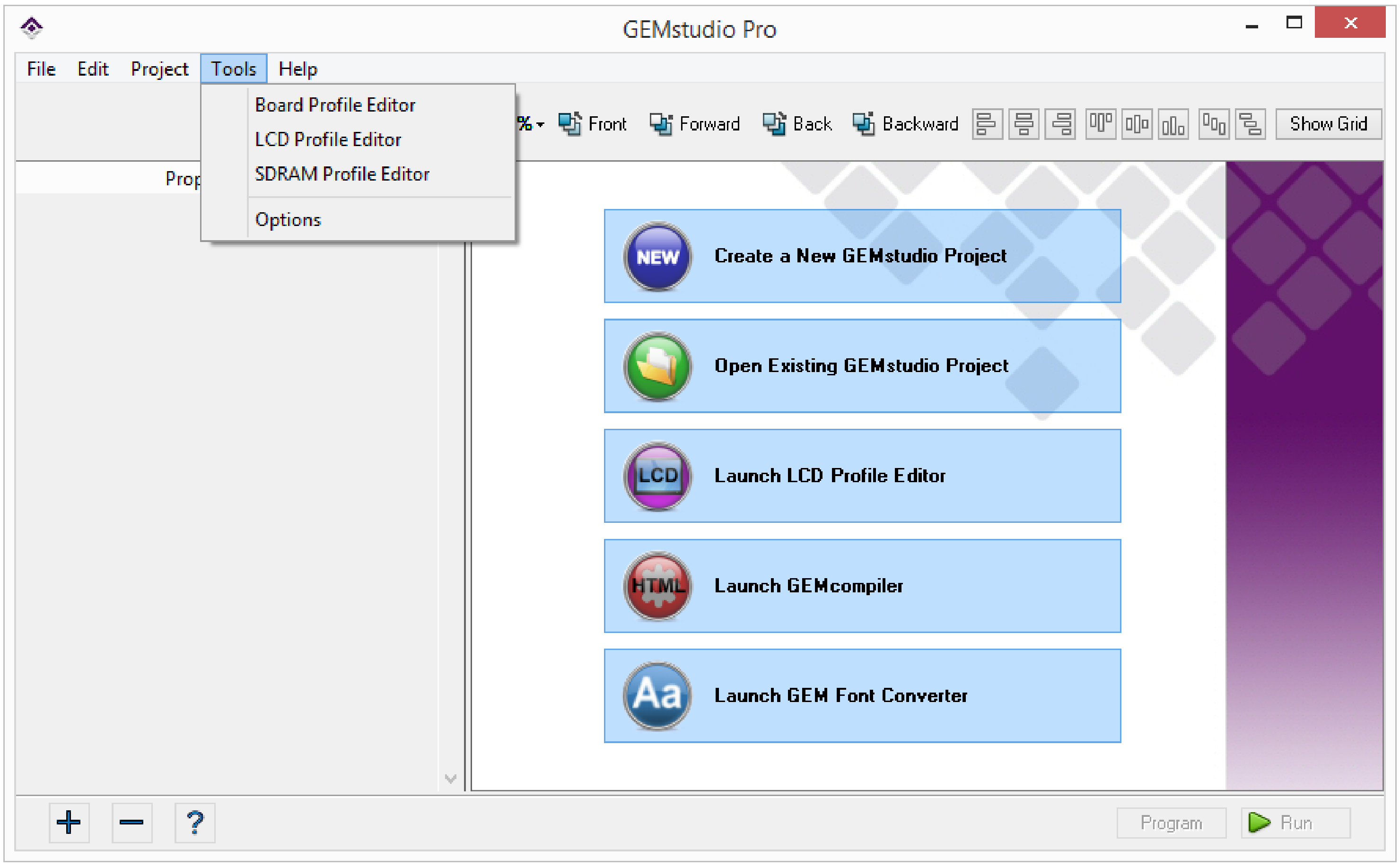
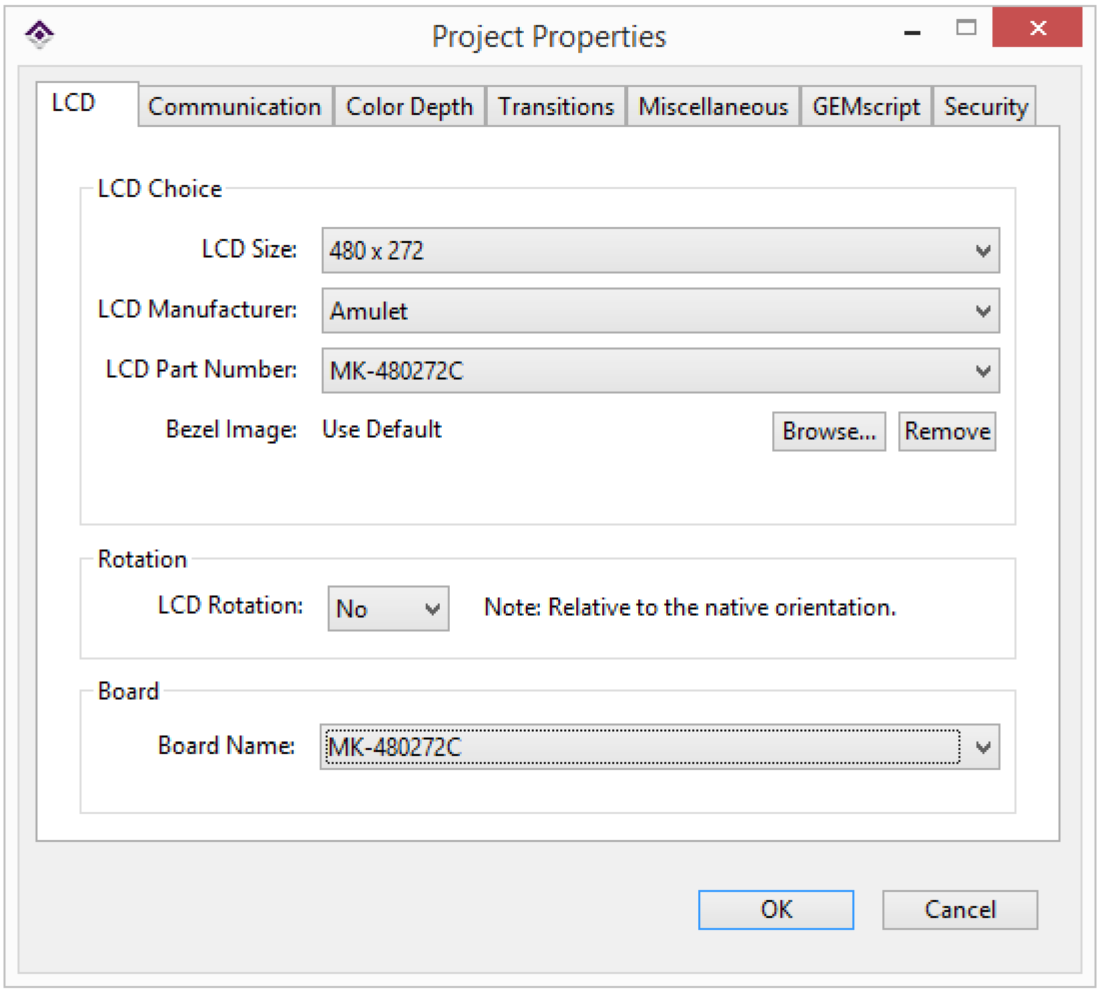


Figure 1: Main GEMstudio Window Showing Tools Drop Down Menu


Project Properties

LCD | Communication | Color Depth | Transitions | Miscellaneous | GEMscript | Security

LCD Choice

LCD Size: 480 x 272

LCD Manufacturer: Amulet

LCD Part Number: MK-480272C

Bezel Image: Use Default

Browse... Remove

Rotation

LCD Rotation: No

Note: Relative to the native orientation.

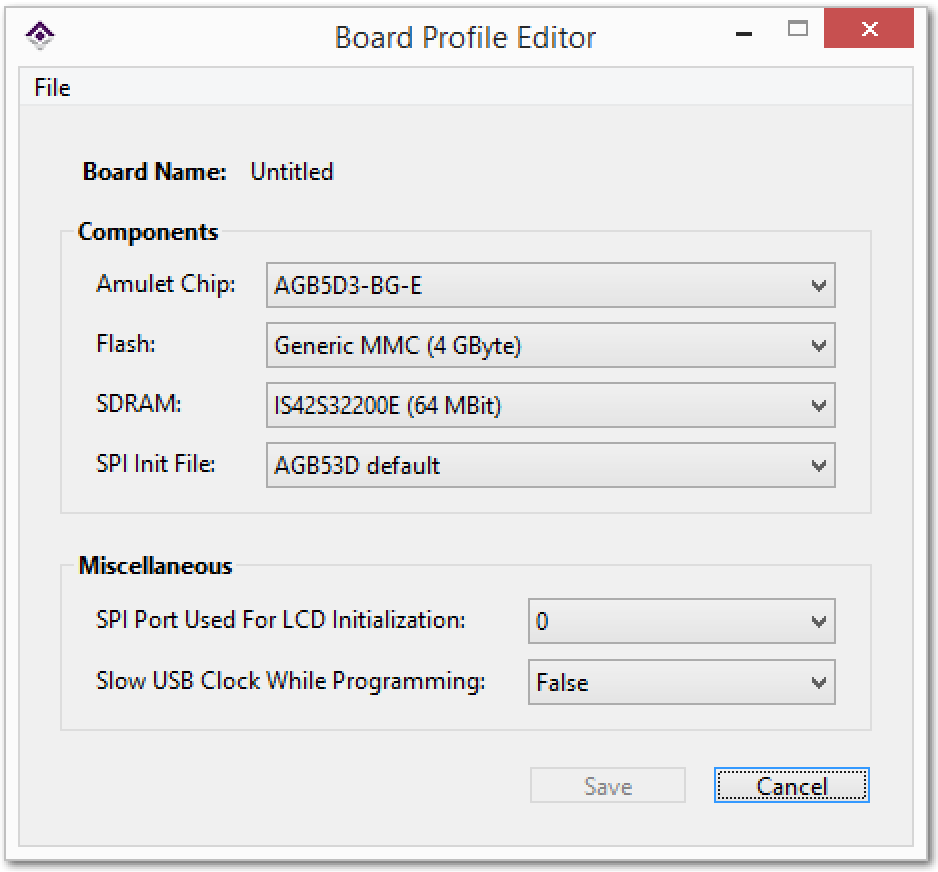
Board

Board Name: MK-480272C

OK Cancel

Figure 2: Project Properties LCD Tab

The LCD/Board Chooser provides the user with multiple LCD and Board choices as shown in Figure 2 and Figure 3. For LCD choices, the options includ LCD Size, LCD Manufacturer, and LCD Part Number. For the Board choices, select a board name. If the board profile is not available, please go to the drop down menu (Tools --> Board Profile Editor) to make a new board profile.



Board Name: Untitled

Components

Amulet Chip: AGB5D3-BG-E

Flash: Generic MMC (4 GByte)

SDRAM: IS42S32200E (64 MBit)

SPI Init File: AGB53D default

Miscellaneous

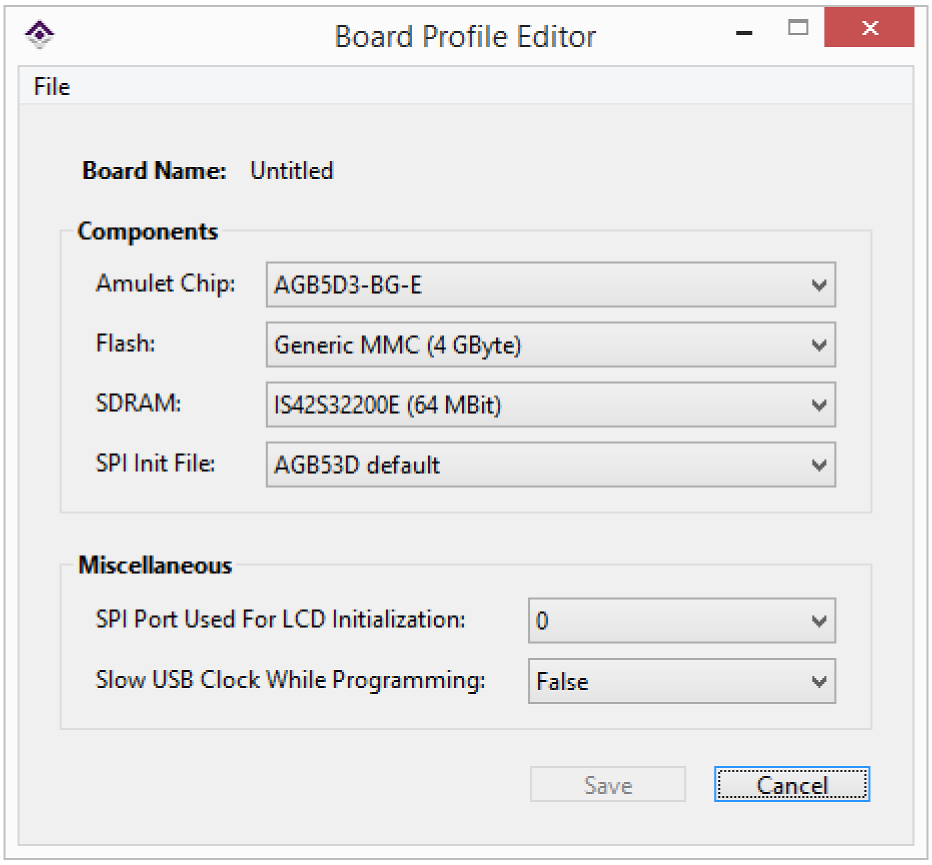
SPI Port Used For LCD Initialization: 0

Slow USB Clock While Programming: False

Figure 3: LCD/Board Chooser Window

Board Profile Editor

Board Profile Editor enables the user to customize the components and board programming on the user's board. The editor has two sections: supported components and miscellaneous. To name the board, click "Save" at the bottom of the window.



Board Name: Untitled

Components

Amulet Chip: AGB5D3-BG-E

Flash: Generic MMC (4 GByte)

SDRAM: IS42S32200E (64 MBit)

SPI Init File: AGB53D default

Miscellaneous

SPI Port Used For LCD Initialization: 0

Slow USB Clock While Programming: False

Components

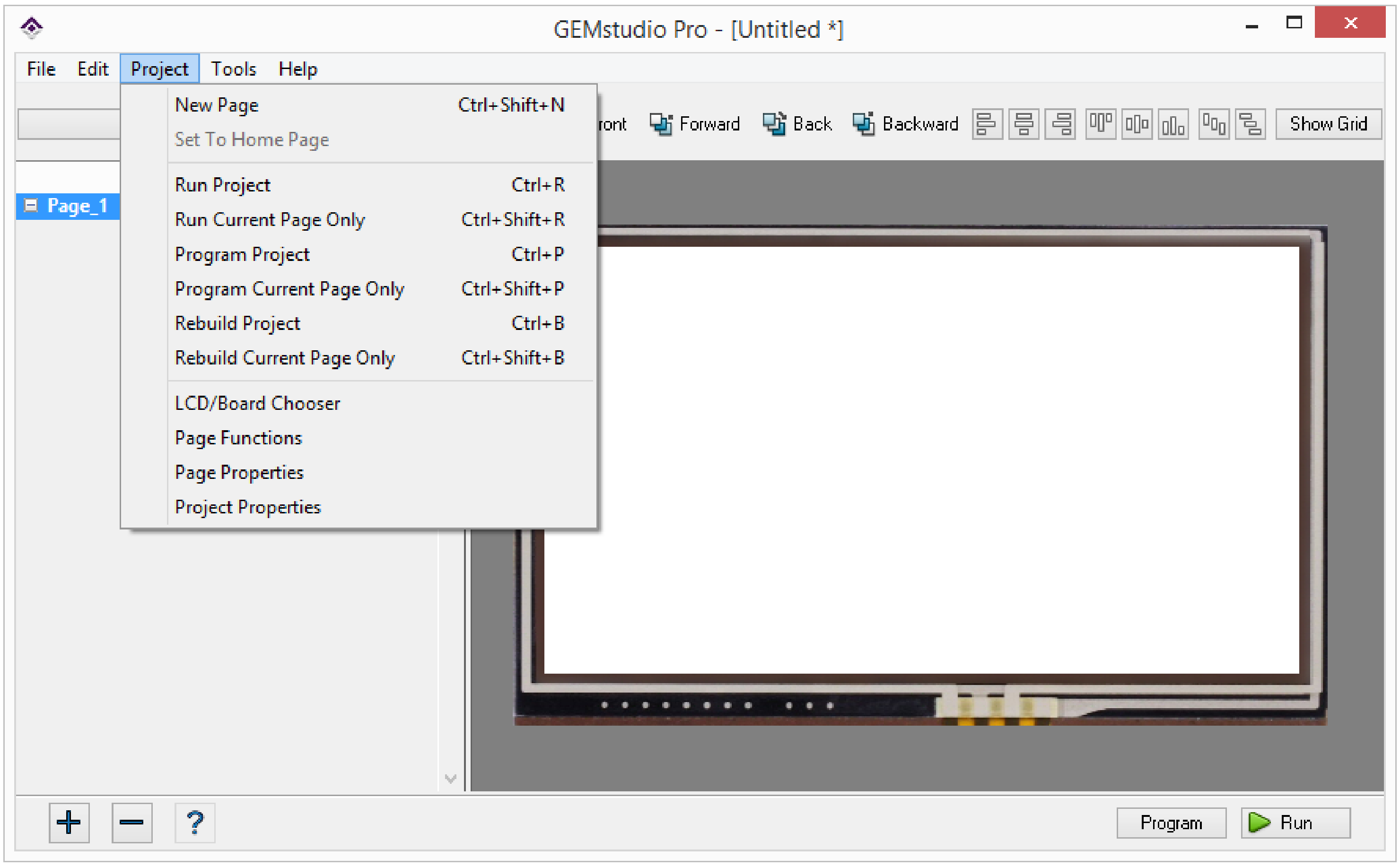
- Amulet Chip:AGB75LC04-BG-E: 225-pin BGA packaged GEM Graphical OS Chip
- AGB75LC04-BG-E: 206-pin QFP packaged GEM Graphical OS Chip
- Flash:AT45DB08: 8 MBit Atmel Flash
- AT45DB16: 16 MBit Atmel Flash
- AT45DB32: 32 MBit Atmel Flash
- AT45DB64: 64 MBit Atmel Flash
- SDRAM:IS42S32200E: 64 MBit ISSI SDRAM
- IS42S32800D: 256 MBit ISSI SDRAM
- SPI Init File:AGB75L default: default SPI init file

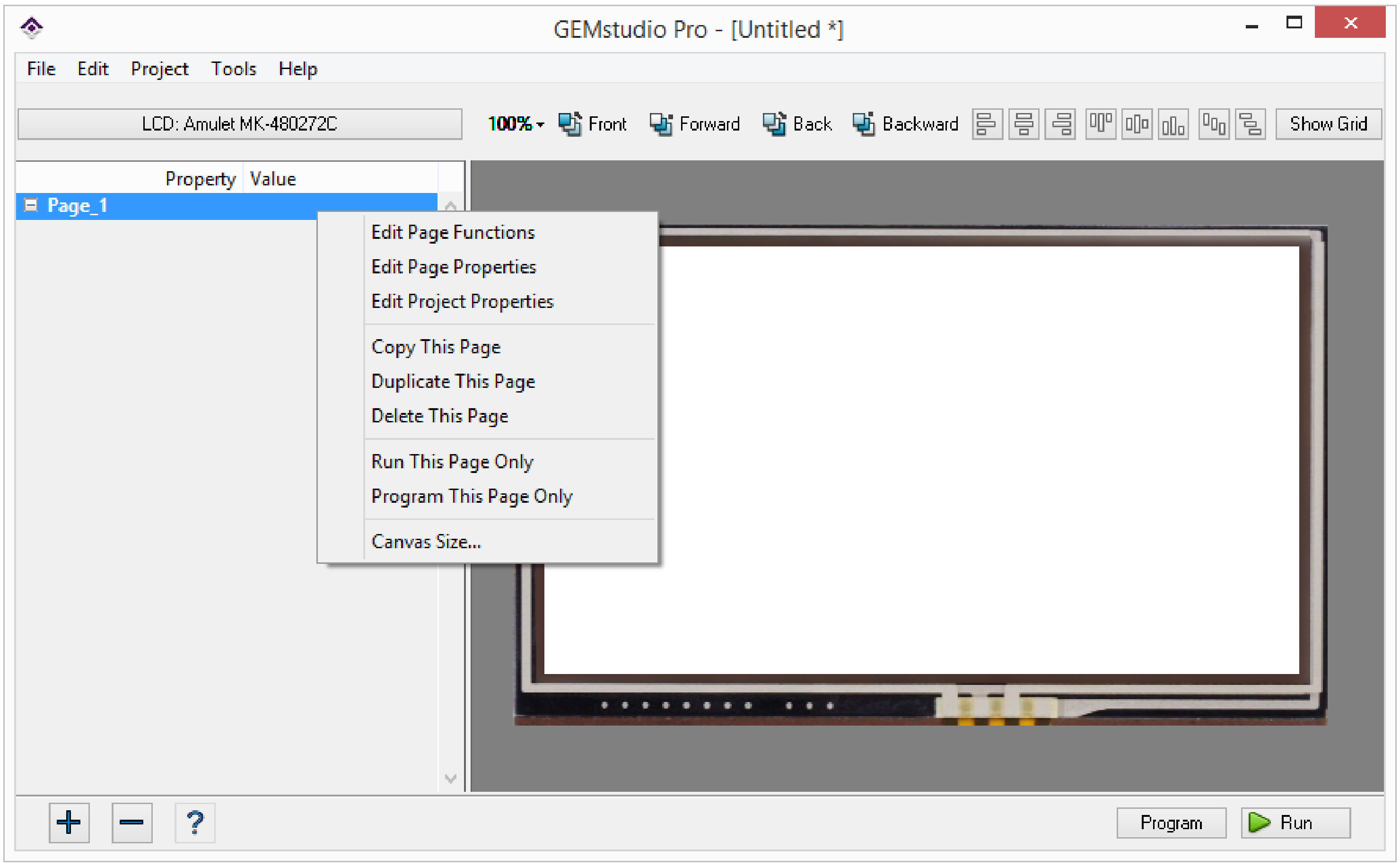
Miscellaneous

- SPI Port Used for LCD Initialization (if LCD requires SPI Initialization):1: SPI Slave Select 1
- 2: SPI Slave Select 2
- 3: SPI Slave Select 3
- Slow USB Clock While ProgrammingFalse: Select False to use the normal clock for USB programming.
- True: Select True if user is having problems programming via USB. The result of this selection is slightly slower programming speed and the LCD will have a flicker during actual programming. Once the programming is done, it goes back to full speed.

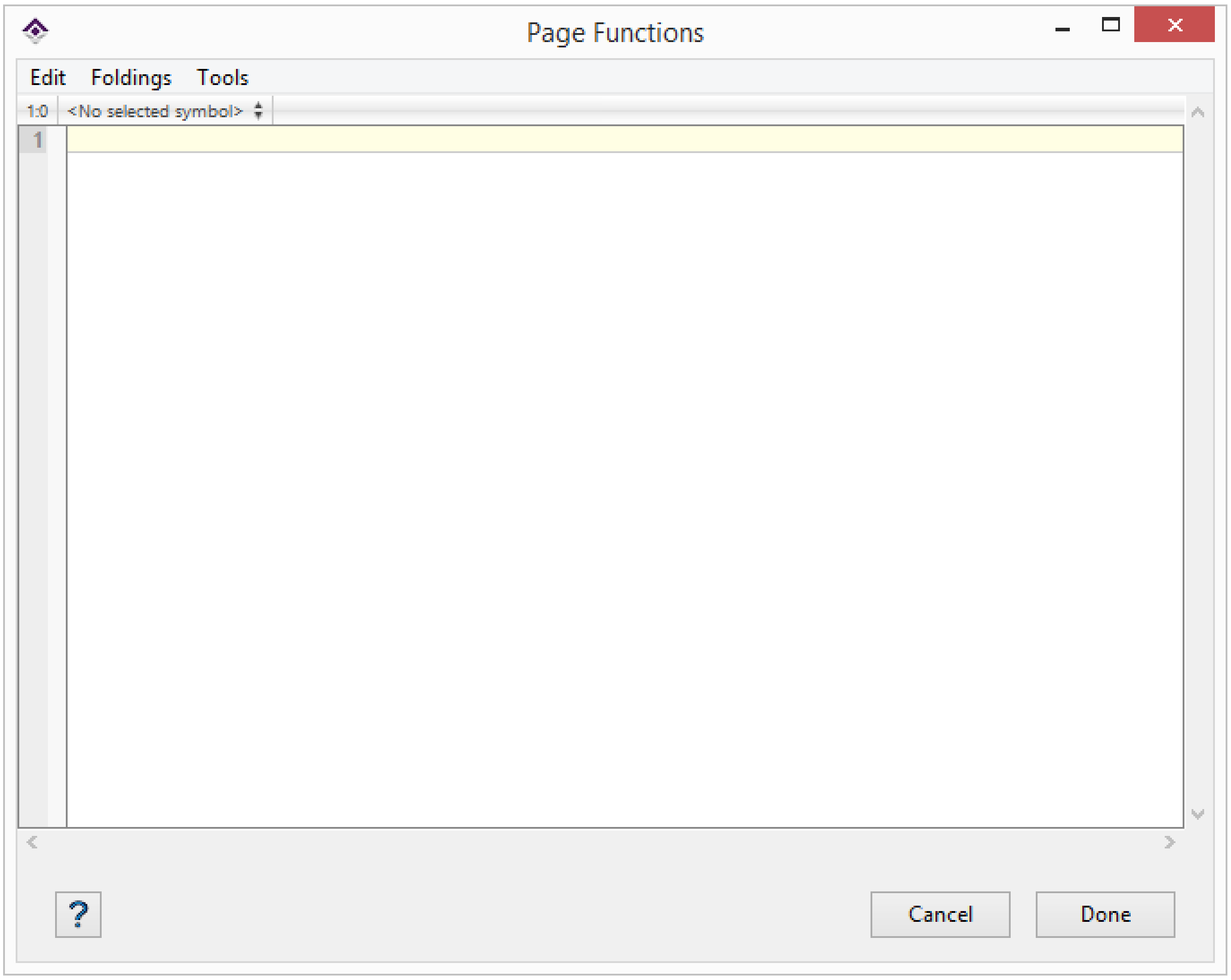
Page Functions Editor

There are 2 ways to open a “Page Functions” window to include META REFRESH functions or to add GEMscript to the page. First, select “Page Functions” from the Project drop down menu at the top of the project window as shown in the first figure below. The second method is to select “Edit Page Functions” from a pop up menu that appears when the user right clicks on the page name as shown in the second figure below.

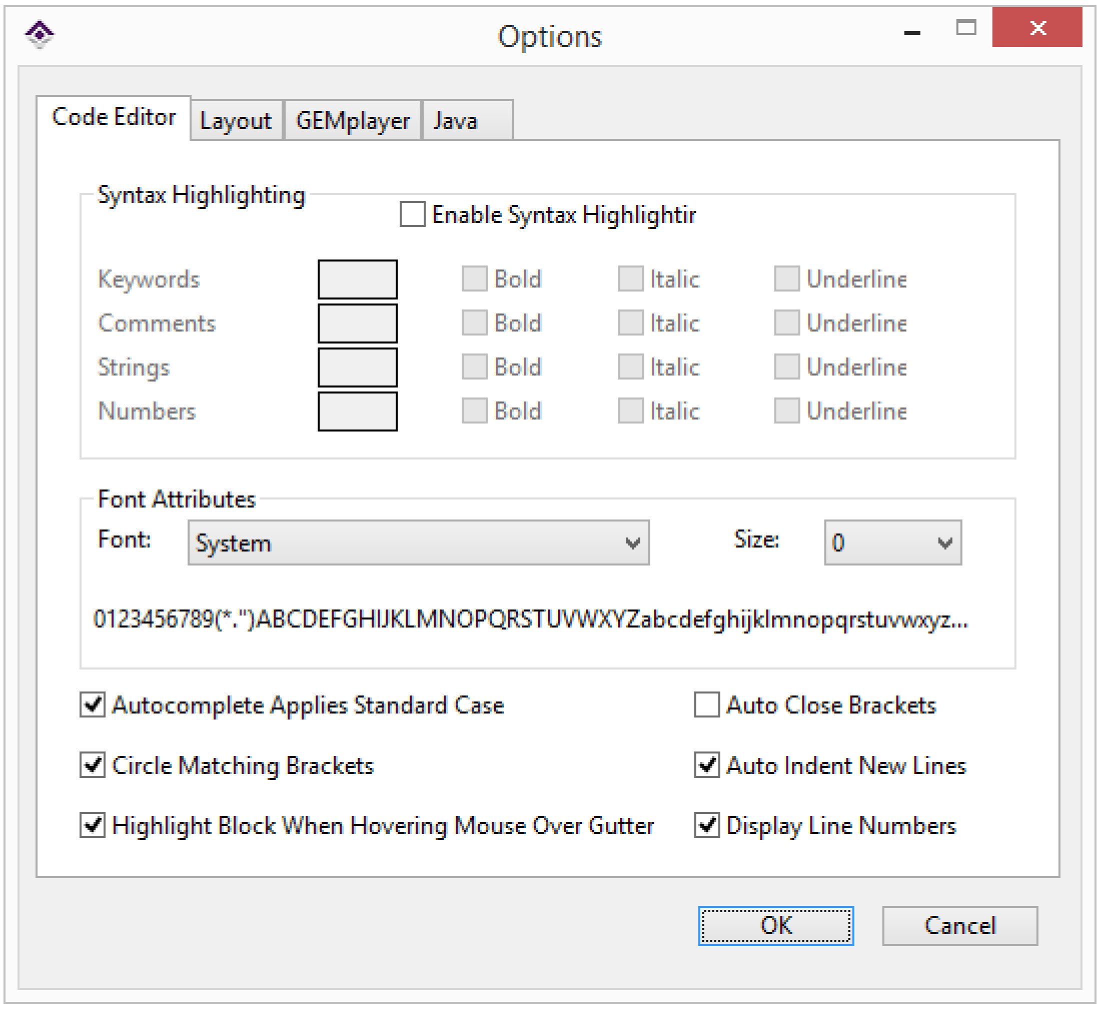




A new Page Functions window will pop up. An example of the Page Functions window is shown below.



To customize the look and feel of the Page Functions Editor and HREF Editor, select “options” from the drop down Tools menu on the top of the Page Functions window.


Options

Code Editor | Layout | GEMplayer | Java

Syntax Highlighting

☐ Enable Syntax Highlightir

Keywords	☐	☐ Bold	☐ Italic	☐ Underline
Comments	☐	☐ Bold	☐ Italic	☐ Underline
Strings	☐	☐ Bold	☐ Italic	☐ Underline
Numbers	☐	☐ Bold	☐ Italic	☐ Underline

Font Attributes

Font: System Size: 0

0123456789(*.")ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz...

☑ Autocomplete Applies Standard Case ☐ Auto Close Brackets

☑ Circle Matching Brackets ☑ Auto Indent New Lines

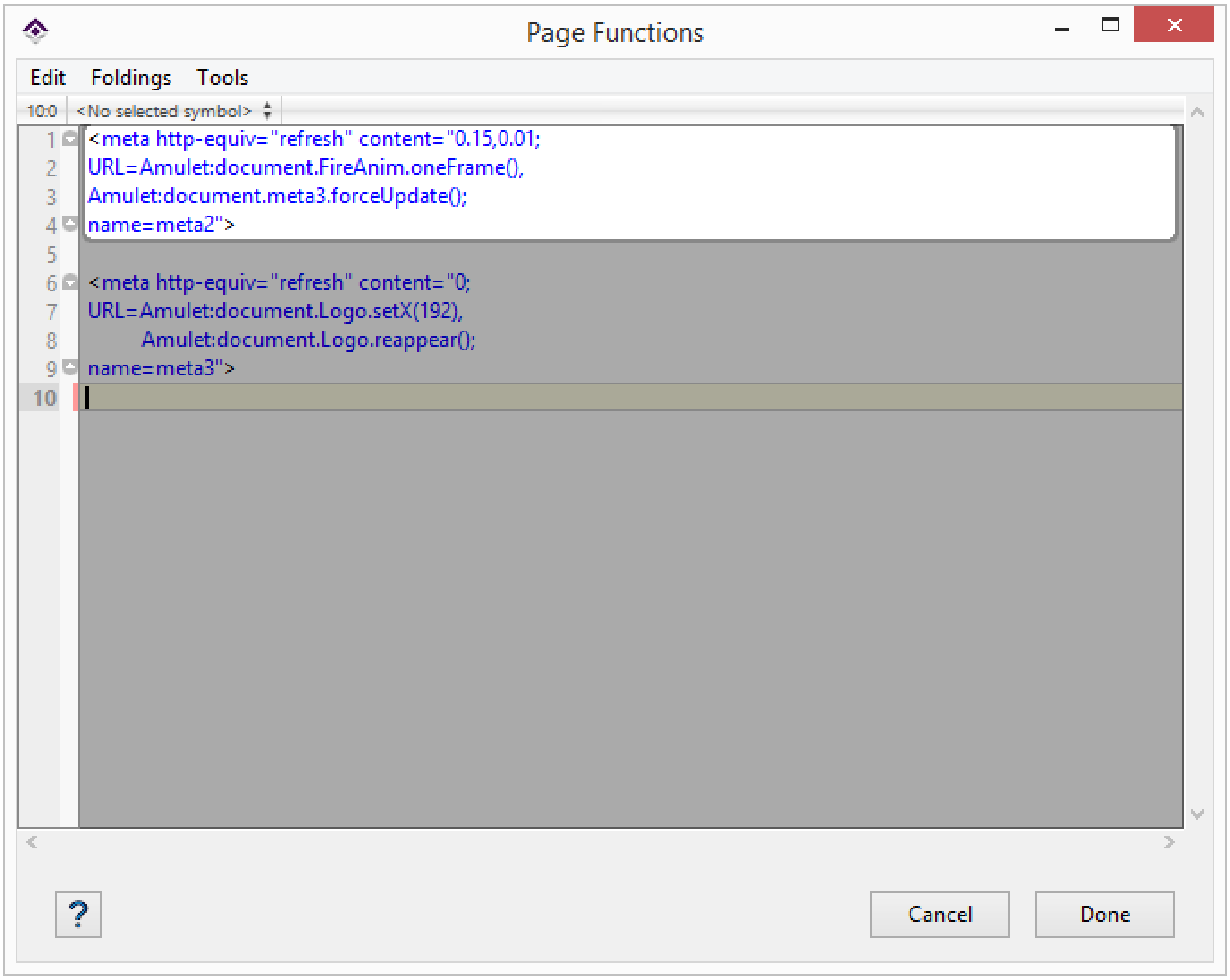
☑ Highlight Block When Hovering Mouse Over Gutter ☑ Display Line Numbers

The user can customize Code Editor with the following options:
Syntax Highlighting

1. Keywords: customize the color from a color picker that pops up when you click on the color selection, and font style (bold, italic and/or underline).
2. Comments: customize the color from a color picker that pops up when you click on the color selection, and font style (bold, italic and/or underline).
3. Strings: customize the color from a color picker that pops up when you click on the color selection, and font style (bold, italic and/or underline).
4. Numbers: customize the color from a color picker that pops up when you click on the color selection, and font style (bold, italic and/or underline).

Other Code Editor Options:

1. Autocomplete Applies Standard Case: CHECKED or UNCHECKED. If CHECKED, then GEMstudio will autocomplete the METAs using the font case from the GEMstudio libraries. If UNCHECKED, then GEMstudio will autocomplete using the same case font the user was typing with.
2. Auto Close Brackets: CHECKED or UNCHECKED. If CHECKED, GEMstudio will automatically provide a closing bracket.
3. Auto Indent New Lines: CHECKED or UNCHECKED. If CHECKED, GEMstudio will automatically indent new lines of code.
4. Highlight Block When Hovering Mouse Over Gutter: If CHECKED, GEMstudio will highlight a block of code when the mouse hovers over the gutter as shown in the figure below.

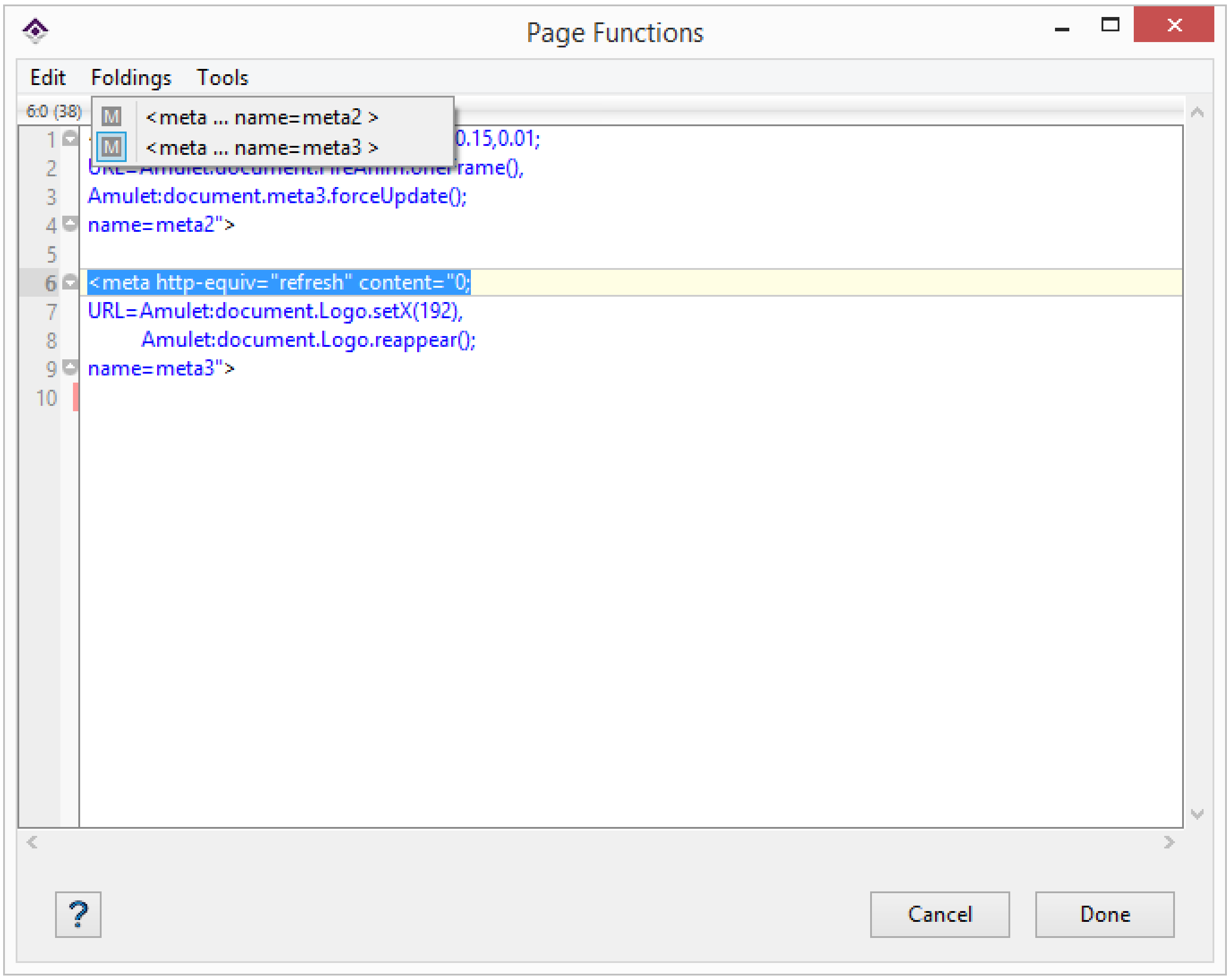


```
<meta http-equiv="refresh" content="0.15,0.01;
URL=Amulet:document.FireAnim.oneFrame(),
Amulet:document.meta3.forceUpdate();
name=meta2">

<meta http-equiv="refresh" content="0;
URL=Amulet:document.Logo.setX(192),
        Amulet:document.Logo.reappear();
name=meta3">
```

1. Circle Matching Brackets: CHECKED or UNCHECKED. If CHECKED, GEMstudio will circle the matching brackets as the user closes the bracket.
2. Display Line Numbers: CHECKED or UNCHECKED. If CHECKED, GEMstudio will display line number in a gutter on the left of the Page Functions Editor.

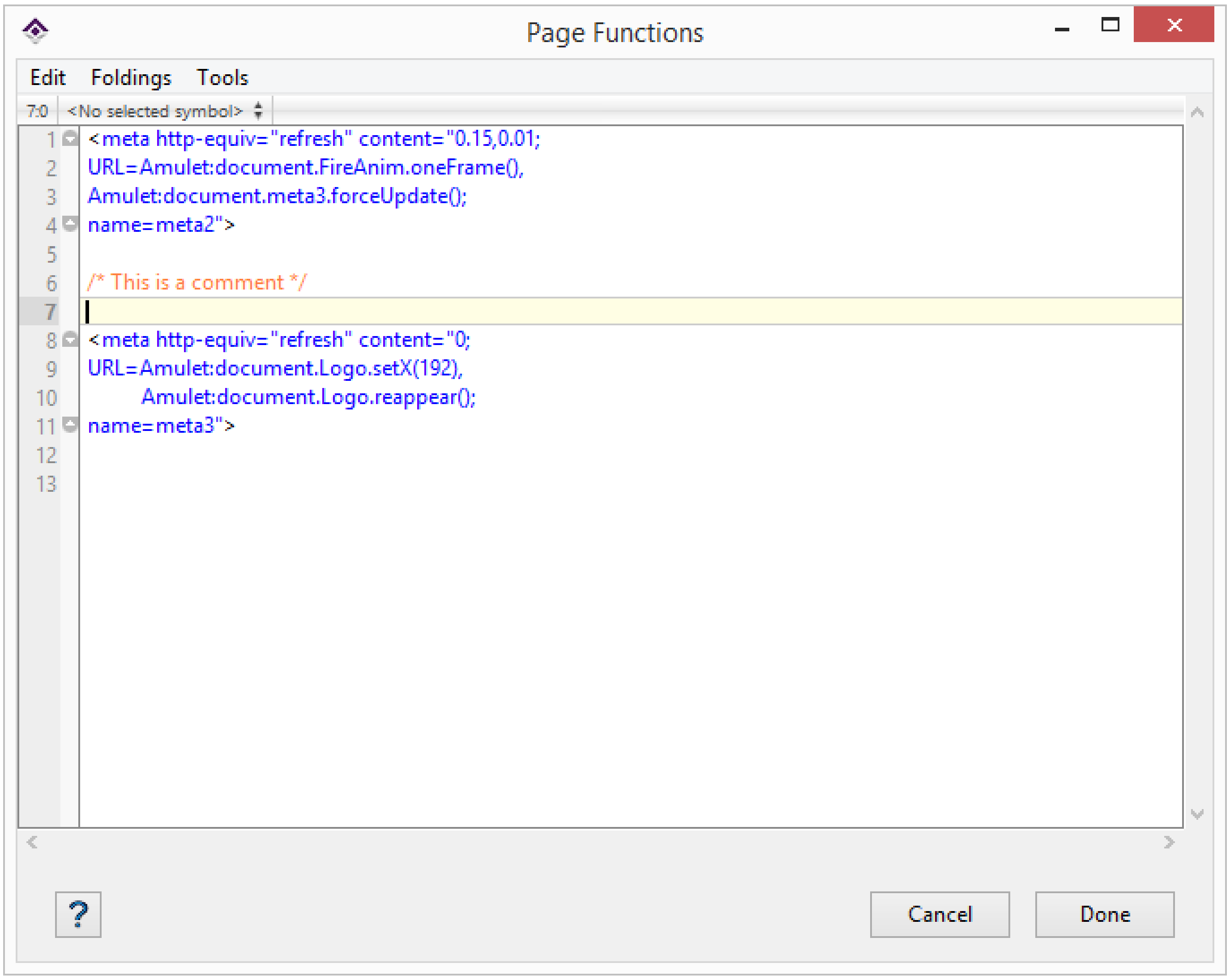
Page Functions Tips:

1. Block navigation is available on the top of the Page Functions Editor window. Click on the up down arrows and a small window will pop up listing the different blocks of code as shown in the Figure below. Select a block and the block will be highlighted in the Page Functions Editor.



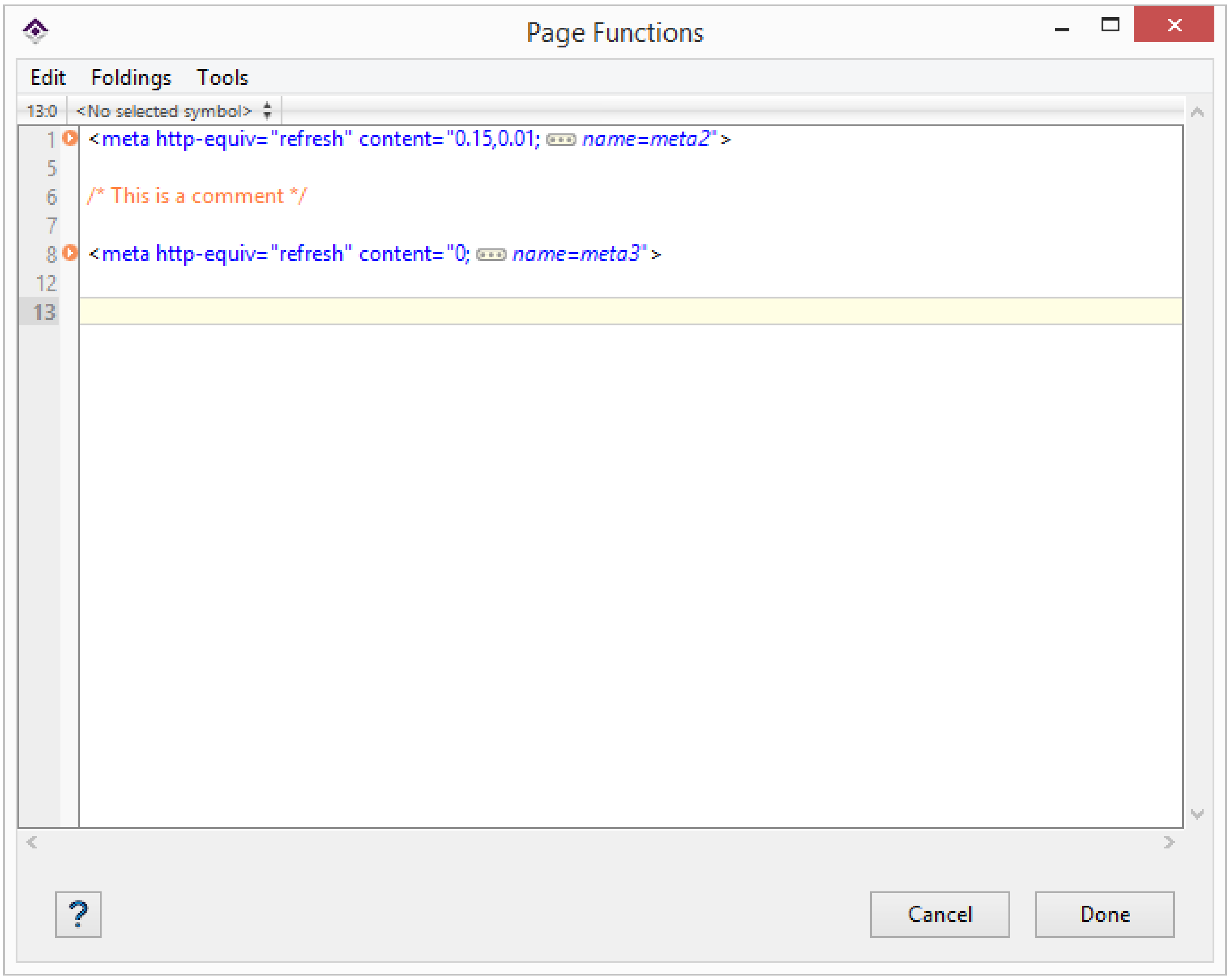
```
[illegible] 0.15,0.01;
URL=Amulet:document.FireAnim.onerFrame(),
Amulet:document.meta3.forceUpdate();
name=meta2">

<meta http-equiv="refresh" content="0;
URL=Amulet:document.Logo.setX(192),
        Amulet:document.Logo.reappear();
name=meta3">
```

1. Auto complete is activated with the TAB key. If the cursor is not at the end of a line, auto complete will not show visual gray. The user must hit TAB to show the available options. If the cursor is at the end of a line and there are multiple options available, the user must hit TAB to show options. However, if there is only one option available, hitting TAB will auto complete the visually grayed function.
2. C style comments (/* ... */) are supported as shown in the Figure below. Auto complete is not available inside comments.



```
<meta http-equiv="refresh" content="0.15,0.01;
URL=Amulet:document.FireAnim.oneFrame(),
Amulet:document.meta3.forceUpdate();
name=meta2">

/* This is a comment */

<meta http-equiv="refresh" content="0;
URL=Amulet:document.Logo.setX(192),
        Amulet:document.Logo.reappear();
name=meta3">
```

1. The user can fold and unfold blocks of code for easier reading and navigation. The user can access these commands from the Folding drop down menu. Arrows shown in the left gutter of the editor can also be used to collapse or expand blocks of code. An example is shown below:



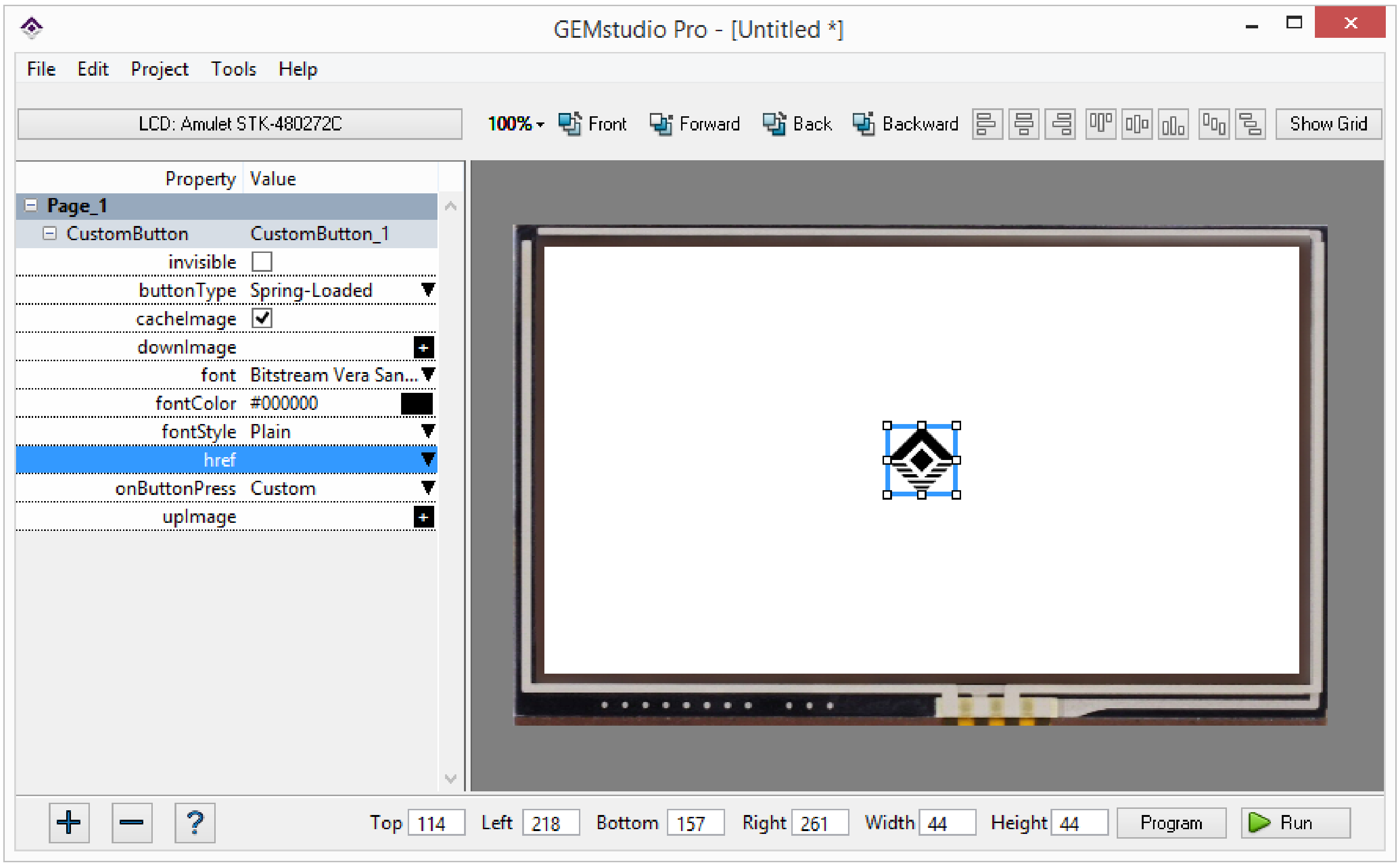
```
<meta http-equiv="refresh" content="0.15,0.01; ... name=meta2">

/* This is a comment */

<meta http-equiv="refresh" content="0; ... name=meta3">

```

For more information on the usage of page functions, please go to Amulet Function Calls and META Refresh Object. For the available functions, please check Appendix B.

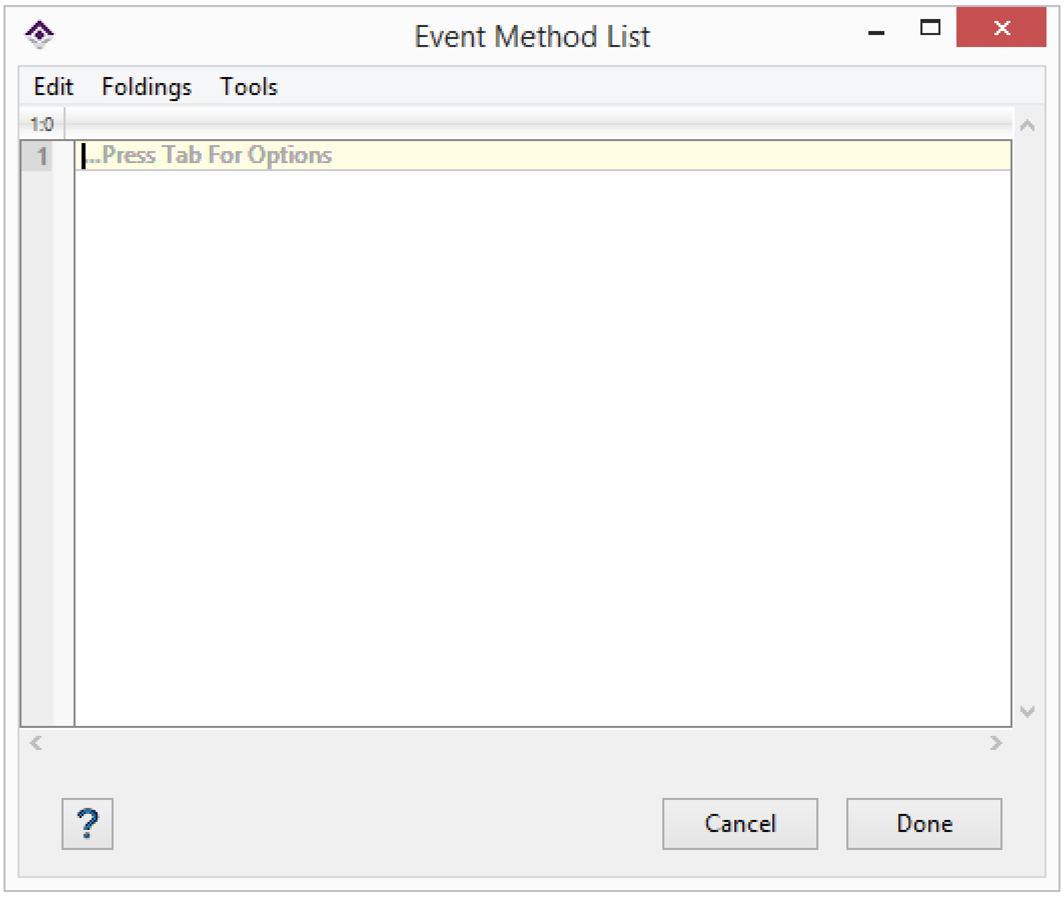
NOTE: If at any time, you need further explanation on widget or parameter in your project, select the widget or parameter and click on the ? button on the lower left hand corner of the project window or press "F1". A help window will pop up and explain the selected widget or parameter.

Href Editor

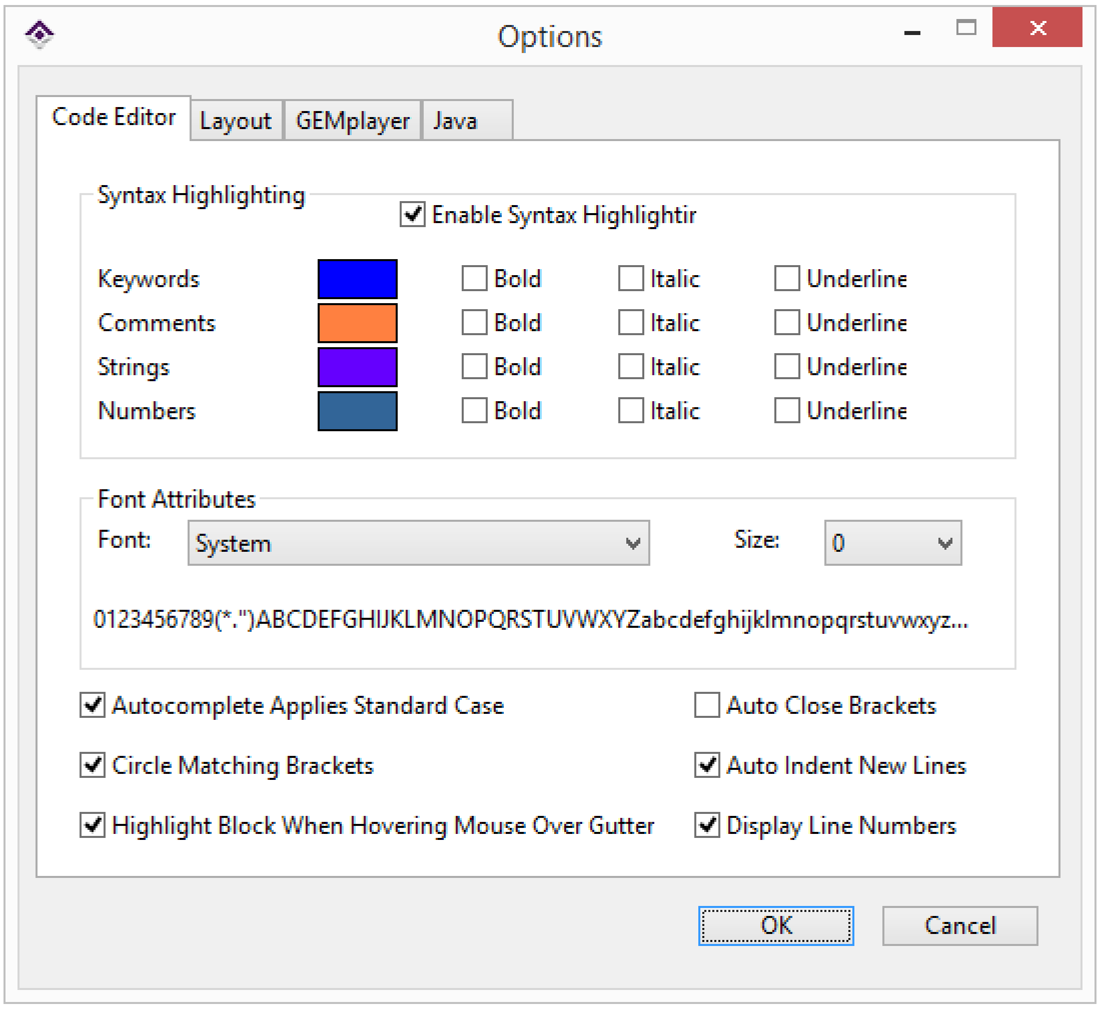
To add function to the widget, the user can include href functions when modifying the widget. The user just has to click on the "Value" next to the Href Property for that particular widget as shown below.



A new Href Editor window will pop up. An example of the Href Editor window is shown below.



To customize the look and feel of the Href Editor, select “options” from the drop down Tools menu on the top of the Project window.



Syntax Highlighting

☑ Enable Syntax Highlightir

Keywords	🟦	☐ Bold	☐ Italic	☐ Underline
Comments	🟧	☐ Bold	☐ Italic	☐ Underline
Strings	🟪	☐ Bold	☐ Italic	☐ Underline
Numbers	🟦	☐ Bold	☐ Italic	☐ Underline

Font Attributes

Font: System

Size: 0

0123456789(*.")ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz...

☑ Autocomplete Applies Standard Case

☐ Auto Close Brackets

☑ Circle Matching Brackets

☑ Auto Indent New Lines

☑ Highlight Block When Hovering Mouse Over Gutter

☑ Display Line Numbers

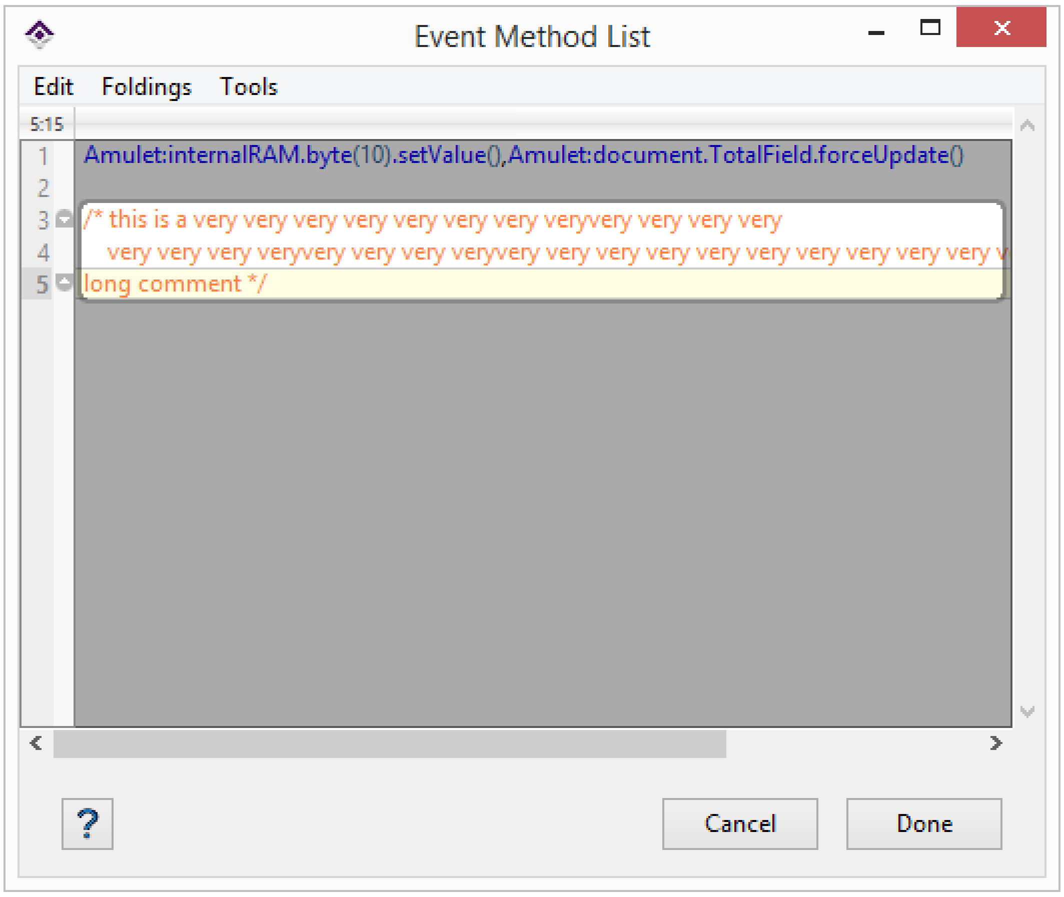
The user can customize the Editor with the following options:

Syntax Highlighting

1. Keywords: customize the color from a color picker that pops up when you click on the color selection, and font style (bold, italic and/or underline).
2. Comments: customize the color from a color picker that pops up when you click on the color selection, and font style (bold, italic and/or underline).
3. Strings: customize the color from a color picker that pops up when you click on the color selection, and font style (bold, italic and/or underline).
4. Numbers: customize the color from a color picker that pops up when you click on the color selection, and font style (bold, italic and/or underline).

Other Code Editor Options:

1. Autocomplete Applies Standard Case: CHECKED or UNCHECKED. If CHECKED, then GEMstudio will autocomplete the METAs using the font case from the GEMstudio libraries. If UNCHECKED, then GEMstudio will autocomplete using the same case font the user was typing with.
2. Auto Close Brackets: CHECKED or UNCHECKED. If CHECKED, GEMstudio will automatically provide a closing bracket.
3. Auto Indent New Lines: CHECKED or UNCHECKED. If CHECKED, GEMstudio will automatically indent new lines of code.
4. Highlight Block When Hovering Mouse Over Gutter: If CHECKED, GEMstudio will highlight a block of code when the mouse hovers over the gutter as shown in the figure below.

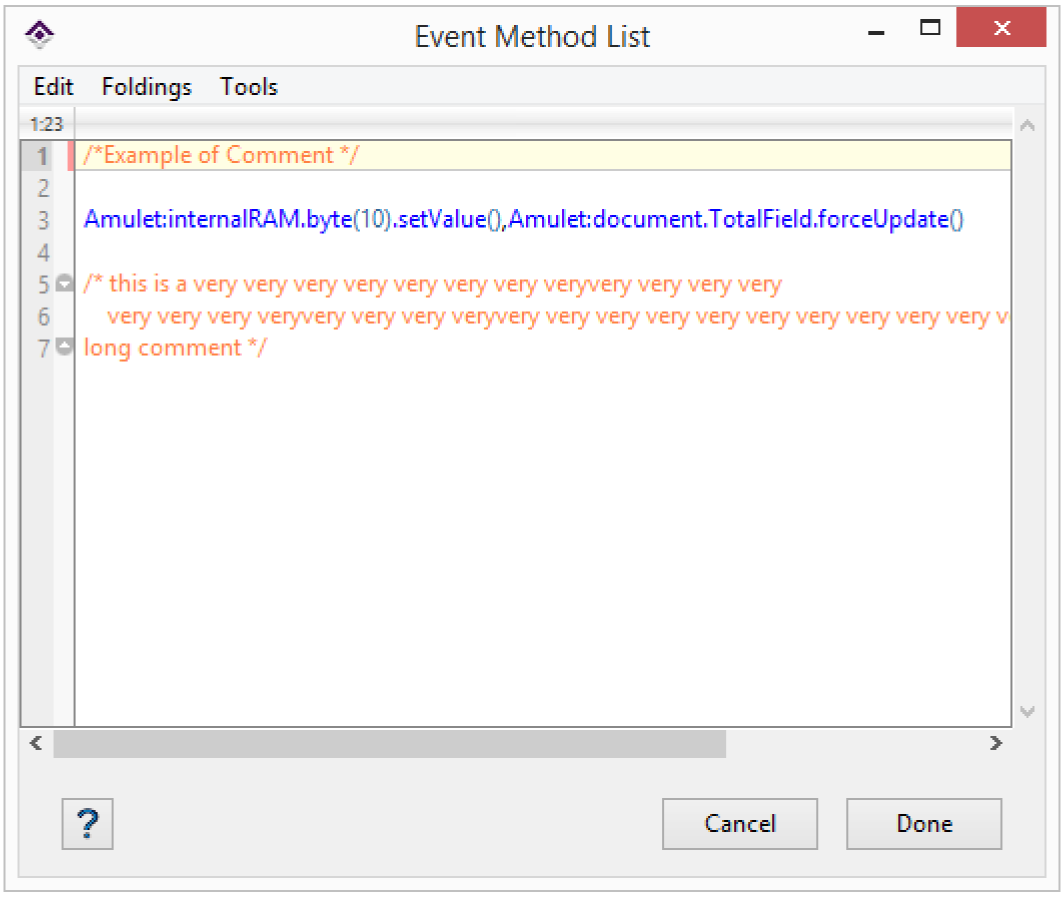


```
Amulet:internalRAM.byte(10).setValue(),Amulet:document.TotalField.forceUpdate()

/* this is a very very very very very very very veryvery very very very
   very very very veryvery very very veryvery very very very very very very very very very v
long comment */
```

1. Circle Matching Brackets: CHECKED or UNCHECKED. If CHECKED, GEMstudio will circle the matching brackets as the user closes the bracket.
2. Display Line Numbers: CHECKED or UNCHECKED. If CHECKED, GEMstudio will display line number in a gutter on the left of the Page Functions Editor.

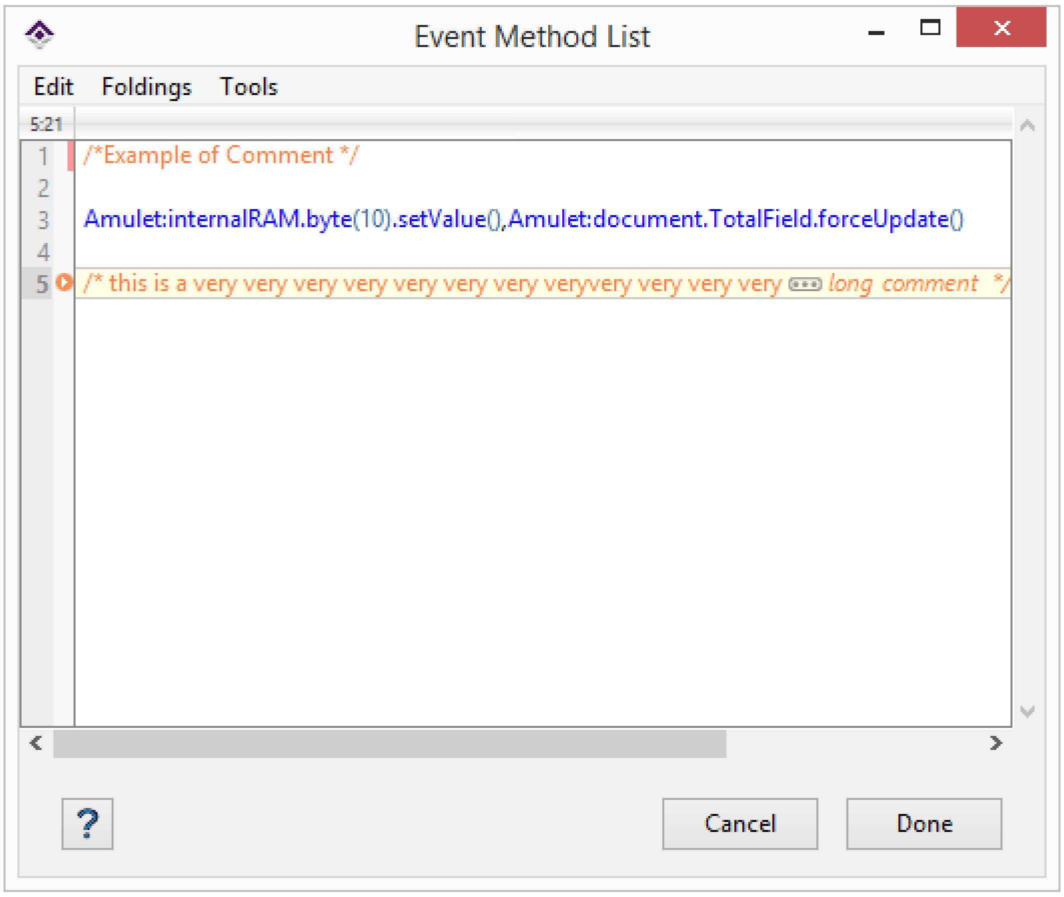
Href Editor Tips:

1. Auto complete is activated with the TAB key. If the cursor is not at the end of a line, auto complete will not show visual gray. The user must hit TAB to show the available options. If the cursor is at the end of a line and there are multiple options available, the user must hit TAB to show options. However, if there is only one option available, hitting TAB will auto complete the visually grayed function.
2. C style comments (/* … */) are supported as shown in the Figure below. Auto complete is not available inside comments.



```
/*Example of Comment */

Amulet:internalRAM.byte(10).setValue(),Amulet:document.TotalField.forceUpdate()

/* this is a very very very very very very very veryvery very very very
   very very very veryvery very very veryvery very very very very very very very very very v
long comment */
```

1. The user can fold and unfold blocks of comments (unlike the Page Functions Editor) for easier reading and navigation. The user can access these commands from the Folding drop down menu. Arrows shown in the left gutter of the editor can also be used to collapse or expand blocks of code. An example is shown below:



```
/*Example of Comment */

Amulet:internalRAM.byte(10).setValue(),Amulet:document.TotalField.forceUpdate()

/* this is a very very very very very very very veryvery very very very ••• long comment */
```

For more information on the usage of href editor functions, please go to Inter-Widget Communications and Appendix C.
NOTE: If at any time, you need further explanation on widget or parameter in your project, select the widget or parameter and click on the ? button on the lower left hand corner of the project window or press "F1". A help window will pop up and explain the selected widget or parameter.

Amulet GEM Font Converter

The Amulet GEM Font Converter converts any installed Windows font into an Amulet font file that can be used in any Amulet project. Fonts can be converted either into a 1-bit bitmap format(.auf) or an 8-bit anti-aliased format (.aauf). Both types of font files can be used in any GEMstudio project.

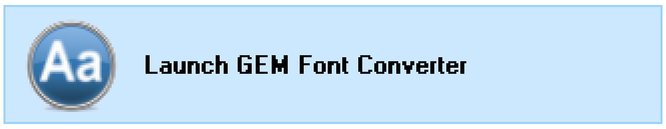
Fonts and the Amulet GEM Font Converter

Any font used in a GEMstudio project will need to be converted into an Amulet Unicode Font file (.auf) or Amulet Anti-Aliased Unicode Font file (.aauf) using the Amulet GEM Font Converter. The subsequent .auf/.aauf file should be placed in one of three acceptable folders, either the root directory of the project you are compiling, the root\Fonts directory, or the ProgramData\AmuletTech\Global\Configuration\Fonts folder. GEMstudio gives priority to font files in the project root directory, then the root\Fonts directory, and finally the ProgramData\AmuletTech\Global\Configuration\Fonts directory.

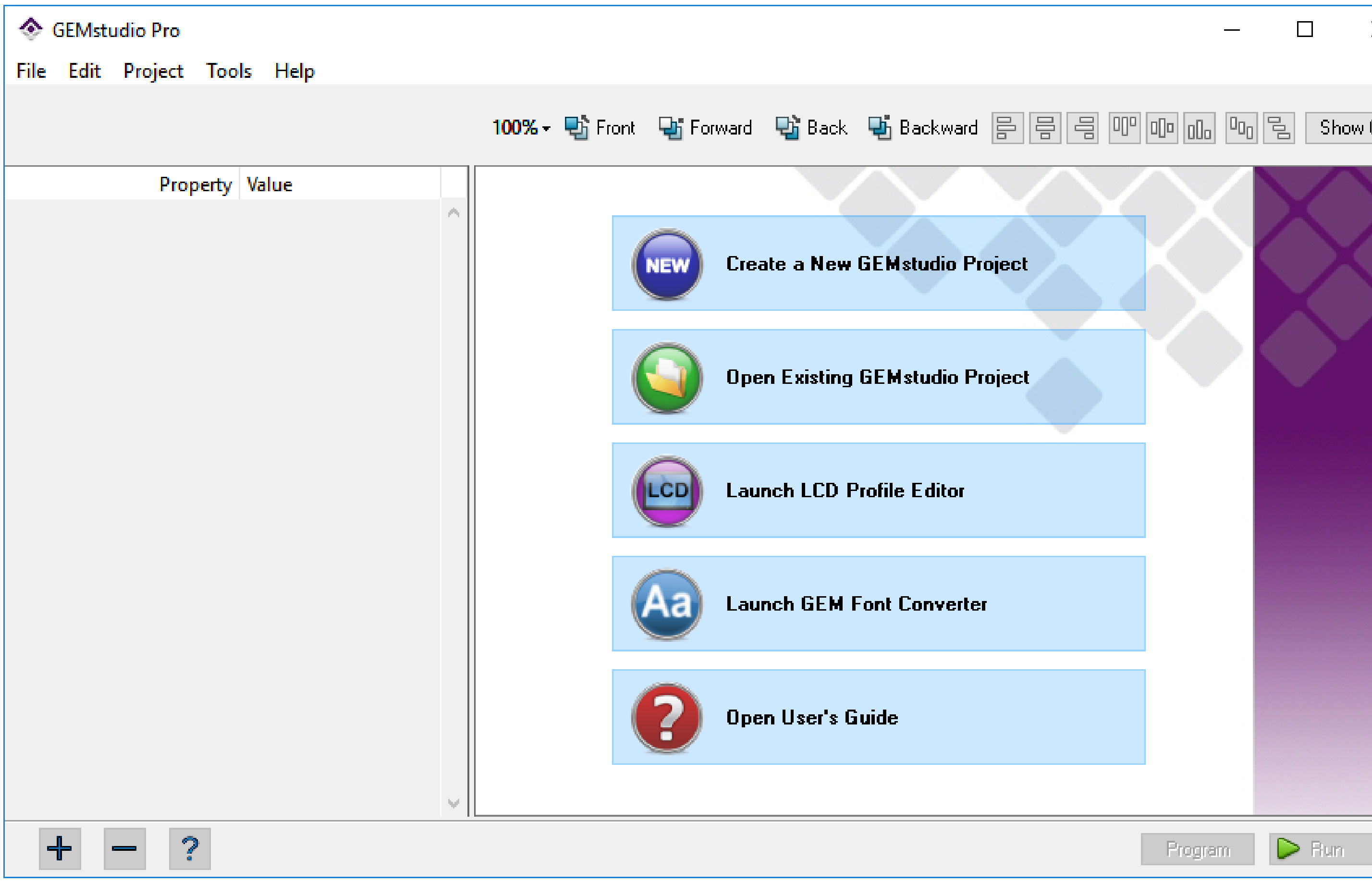
If the .auf/.aauf file is placed in the ProgramData\AmuletTech\Global\Configuration\Fonts folder, the font can be used in any GEMstudio project. If the file is placed in the project root or root\Fonts folder, the font can only be used in a project saved in that same root directory.

Starting the Amulet GEM Font Converter

The Amulet GEM Font Converter runs on Windows Vista and newer and may be invoked one of the following ways:

• From the main GEMstudio Pro window, click on the 

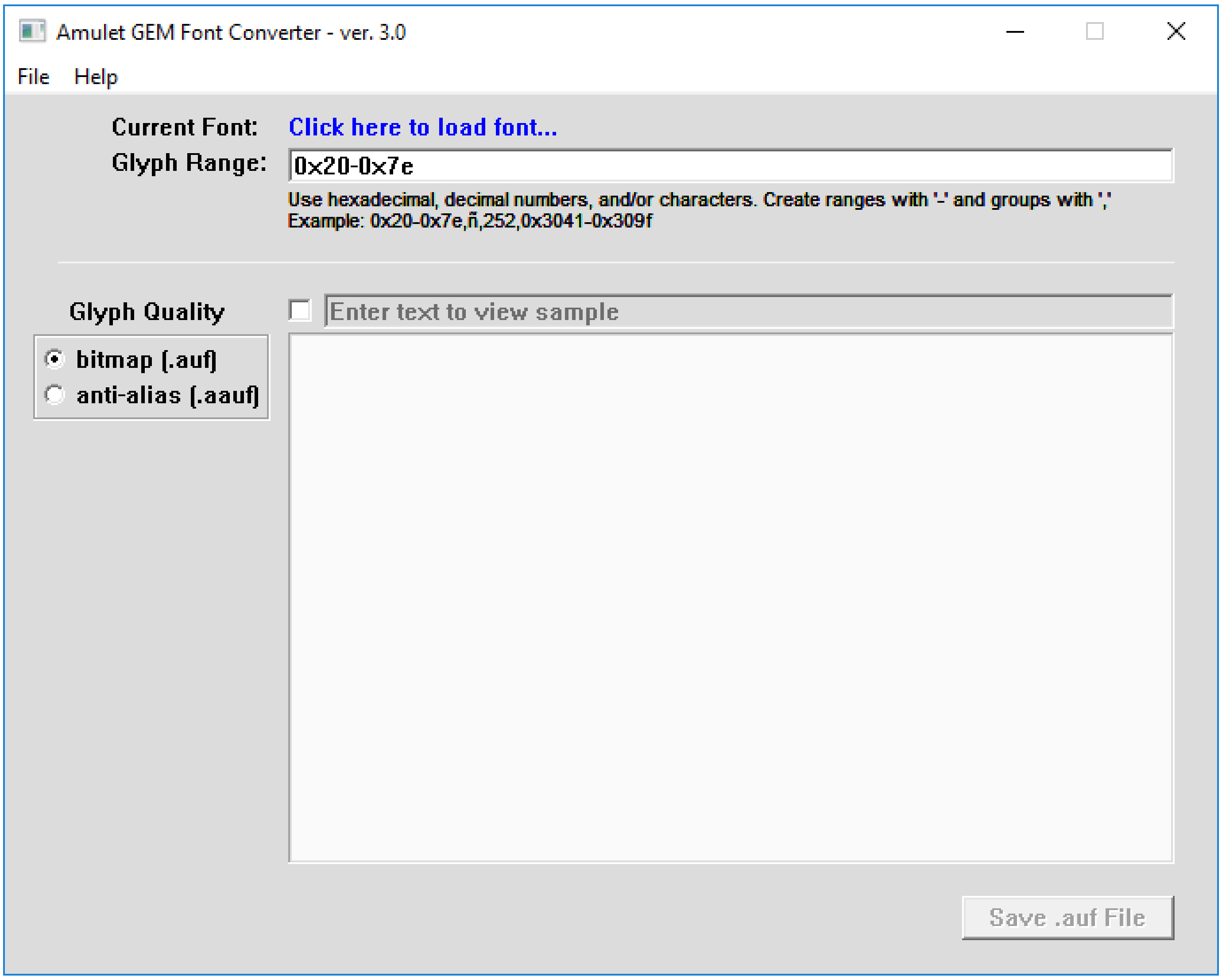
button .
• Click on Launch GEM font Converter from the File menu


GEMstudio Pro
File Edit Project Tools Help
100% Front Forward Back Backward Show
Property Value
Create a New GEMstudio Project
Open Existing GEMstudio Project
Launch LCD Profile Editor
Launch GEM Font Converter
Open User's Guide
Program Run

Using the Converter

To convert a Windows font into an Amulet Font file(.auf/.aauf), load and save the font with the Amulet GEM Font Converter.

To load the font, click on the blue "Click here to load font..." from the main window.



Current Font: Click here to load font...

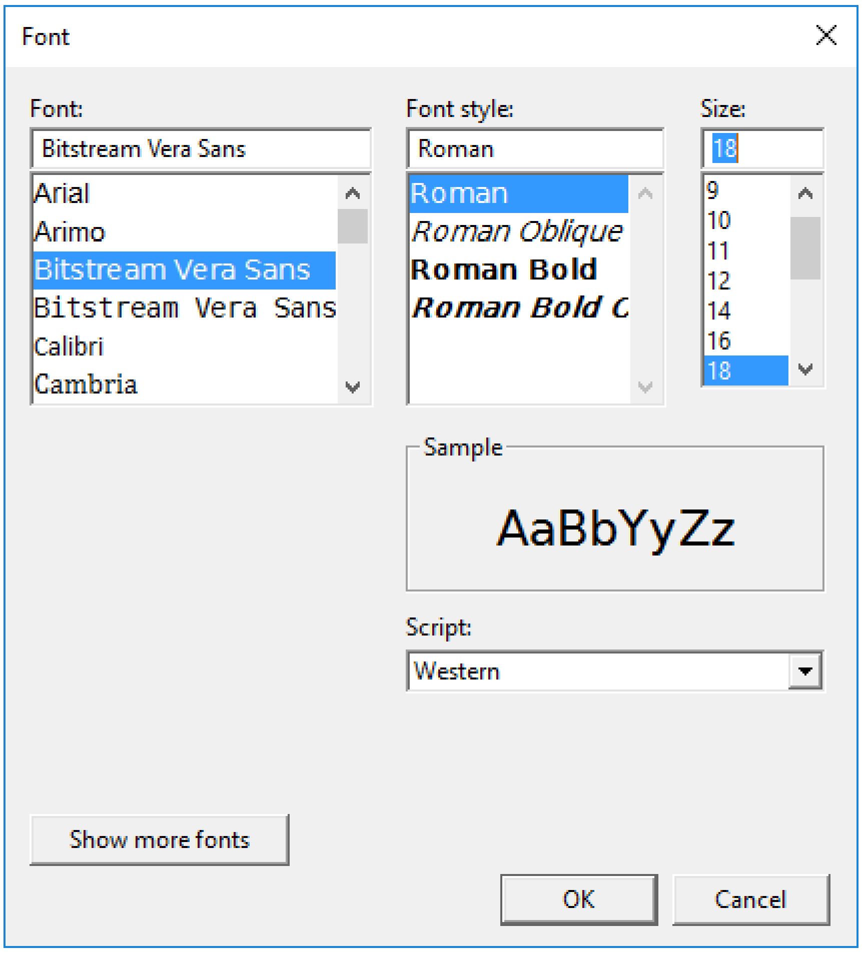
Glyph Range: 0x20-0x7e

Use hexadecimal, decimal numbers, and/or characters. Create ranges with '-' and groups with ','
Example: 0x20-0x7e,ñ,252,0x3041-0x309f

Glyph Quality
(● bitmap (.auf)) ○ anti-alias (.aauf)

☐ Enter text to view sample

Once the pop-up window appears, select the Font, Font style, and Size. Select OK when the correct font has been chosen.


Font

Font: Bitstream Vera Sans

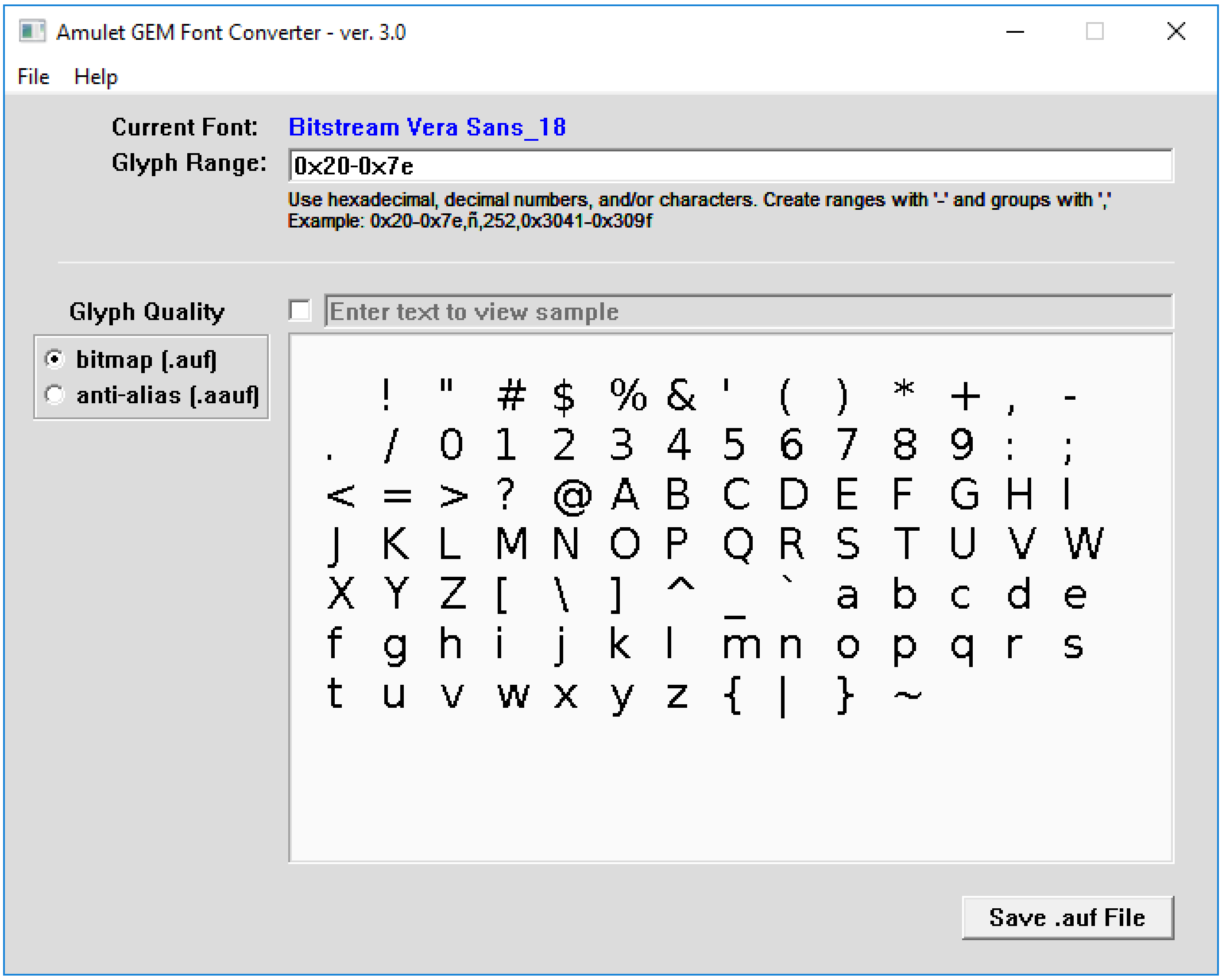
Font style: Roman

Size: 18

Sample: AaBbYyZz

Script: Western

Once the font is loaded, the main viewing box displays all the glyphs that will be converted into the Amulet font file in the quality as they will appear on the Amulet display. When first opened, the Amulet GEM Font Converter will convert the font using a 1-bit bitmap format. To see what the glyphs will look like using an 8-bit anti-aliased format, select the "anti-alias" Glyph Quality radio button.



Current Font: Bitstream Vera Sans_18

Glyph Range: 0x20-0x7e

Use hexadecimal, decimal numbers, and/or characters. Create ranges with '-' and groups with ','
Example: 0x20-0x7e,ñ,252,0x3041-0x309f

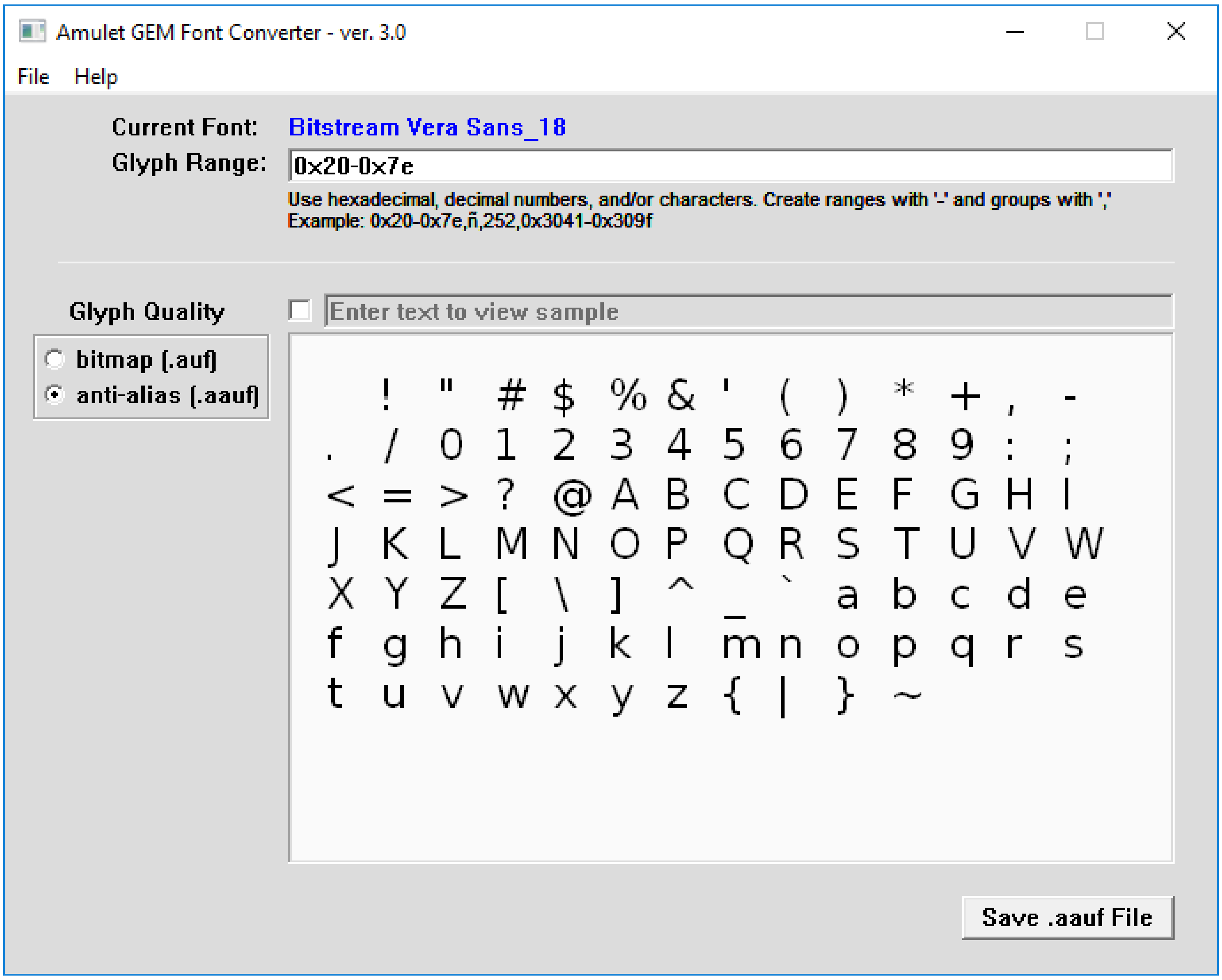
Glyph Quality

[✓ bitmap (.auf)] anti-alias (.aauf)

Enter text to view sample

```
  !  "  #  $  %  &  '  (  )  *  +  ,  -
.  /  0  1  2  3  4  5  6  7  8  9  :  ;
<  =  >  ?  @  A  B  C  D  E  F  G  H  I
J  K  L  M  N  O  P  Q  R  S  T  U  V  W
X  Y  Z  [  \  ]  ^  _  `  a  b  c  d  e
f  g  h  i  j  k  l  m  n  o  p  q  r  s
t  u  v  w  x  y  z  {  |  }  ~
```

Once the anti-alias radio button is selected, the main viewing box will show what the glyphs look like anti-aliased. NOTE: Some fonts at certain sizes show no visible difference between bitmap and anti-alias. For example, Arial size 13 and smaller look exactly the same whether bitmapped or anti-aliased. In those cases, only the bitmapped fonts should be used because they will use up less flash space, less RAM, and will render to the display quicker.


Amulet GEM Font Converter - ver. 3.0

File Help

Current Font: Bitstream Vera Sans_18

Glyph Range: 0x20-0x7e

Use hexadecimal, decimal numbers, and/or characters. Create ranges with '-' and groups with ','
Example: 0x20-0x7e,ñ,252,0x3041-0x309f

Glyph Quality

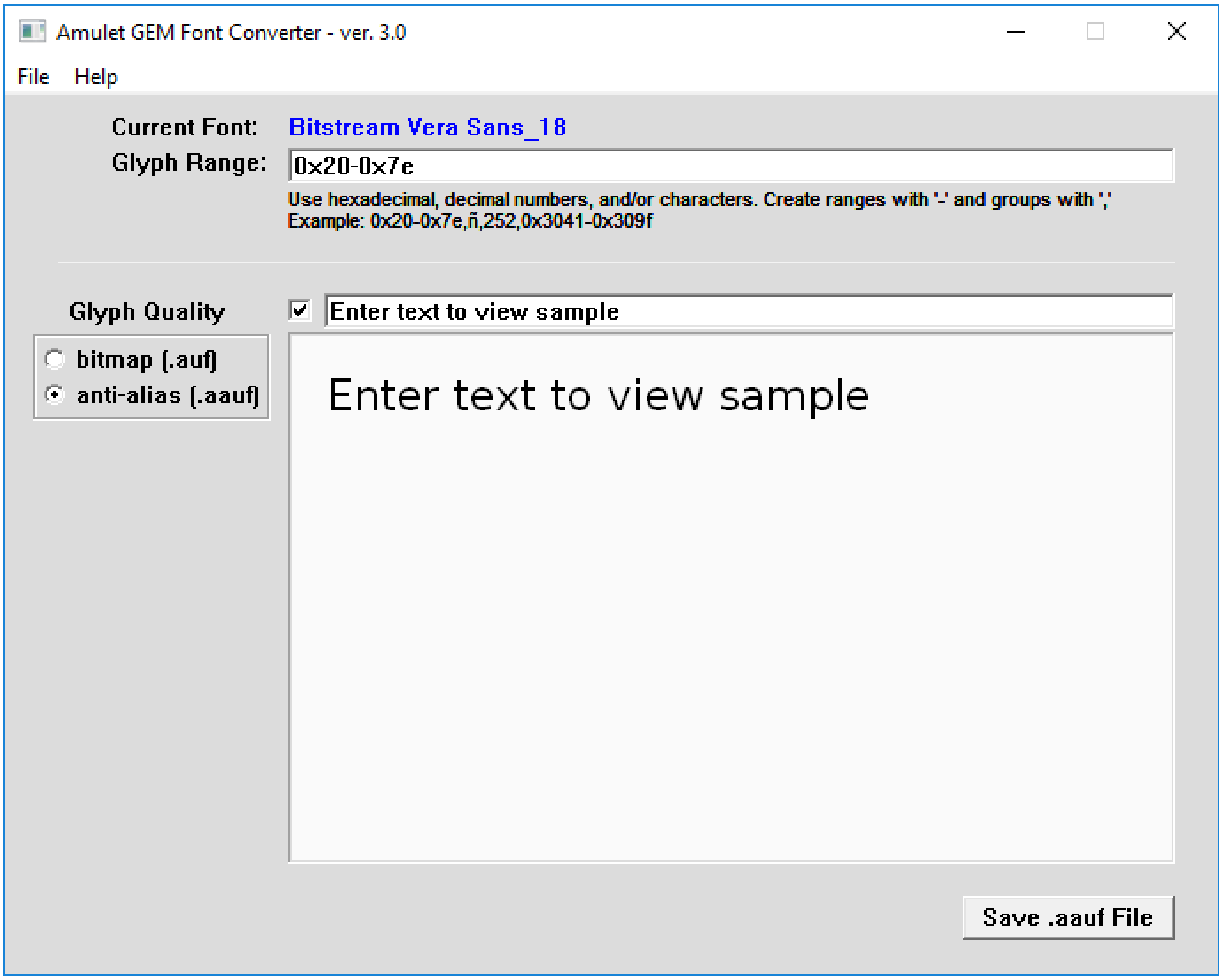
( ) bitmap (.auf)
(●) anti-alias (.aauf)

☐ Enter text to view sample

```
  !  "  #  $  %  &  '  (  )  *  +  ,  -
.  /  0  1  2  3  4  5  6  7  8  9  :  ;
<  =  >  ?  @  A  B  C  D  E  F  G  H  I
J  K  L  M  N  O  P  Q  R  S  T  U  V  W
X  Y  Z  [  \  ]  ^  _  `  a  b  c  d  e
f  g  h  i  j  k  l  m  n  o  p  q  r  s
t  u  v  w  x  y  z  {  |  }  ~
```

Save .aauf File

By default, the main viewing box shows all converted glyphs spaced out uniformly. To see a specific string using the true spacing for the glyphs, click the check box located to the left of the edit field that says "Enter text to view sample". Once checked, the edit field becomes active and anything that is typed in the edit field will show up in the main viewing box below using the currently selected font. NOTE: If any glyphs are entered into the edit field that are not part of the converted range, a blank space will show up in the main viewing box.



Current Font: Bitstream Vera Sans_18

Glyph Range: 0x20-0x7e

Use hexadecimal, decimal numbers, and/or characters. Create ranges with '-' and groups with ','
Example: 0x20-0x7e,ñ,252,0x3041-0x309f

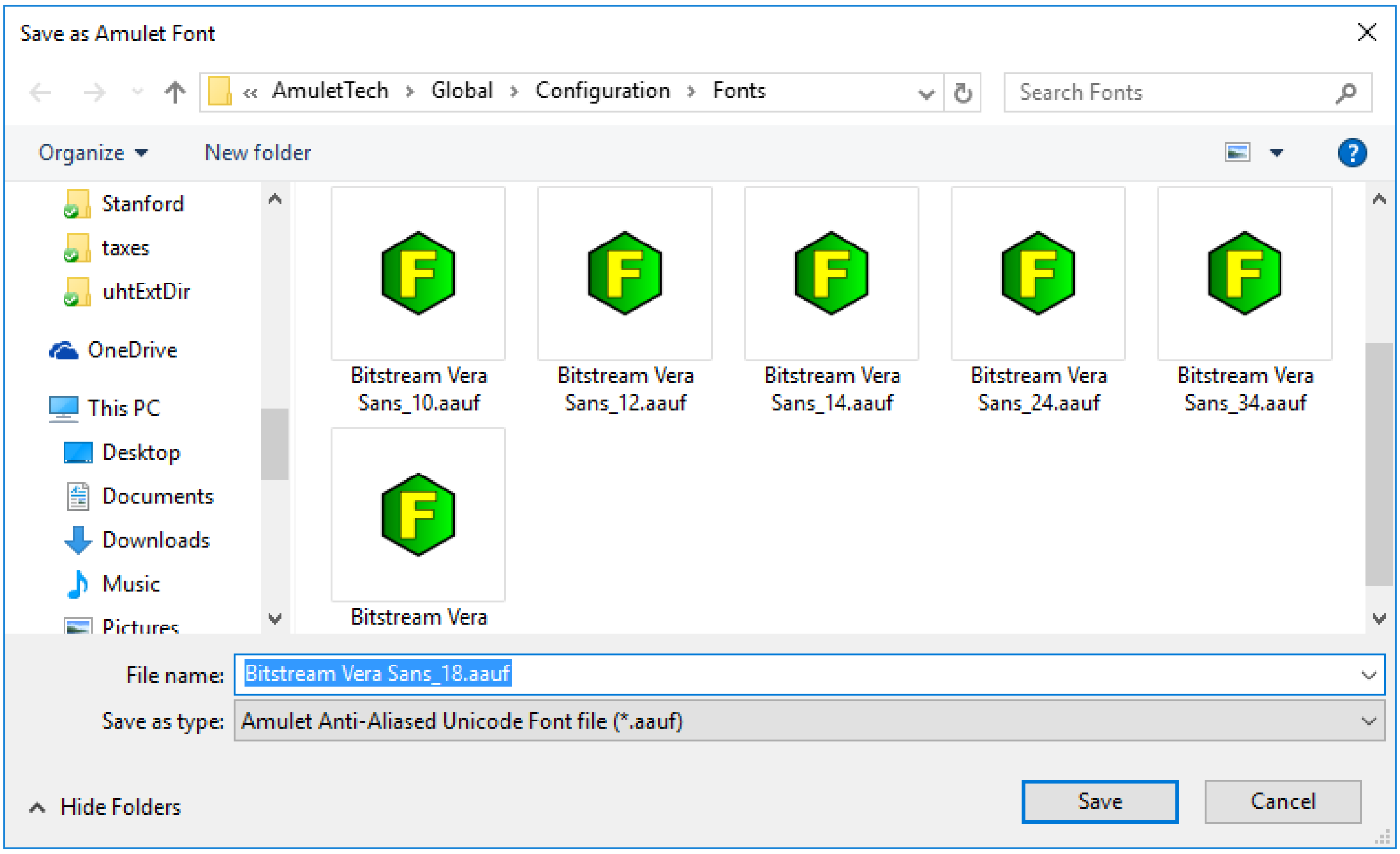
Glyph Quality
- bitmap (.auf)
- [✓ anti-alias (.aauf)]

✓ Enter text to view sample

Enter text to view sample

Depending upon the state of the Glyph Quality radio buttons, the save button will say either "Save .aauf File" or "Save .auf File". To save the converted font file into an Amulet font file, click on the save button. The font file will be saved in either the 1-bit bitmap format (.auf) or the 8-bit anti-aliased format (.aauf). It is possible to save the same font in both formats, but the state of the Glyph Quality radio buttons will determine in which format the font is saved. To save in both formats, the save button must be selected a second time after changing the Glyph Quality radio button.

By default, the Amulet GEM Font Converter will save the Amulet font file to the global ProgramData\AmuletTech \Global\Configuration\Fonts folder. The font will then be available to be used in any project within GEMstudio.


Save as Amulet Font
« AmuletTech › Global › Configuration › Fonts
Search Fonts
Organize
New folder
Stanford
taxes
uhtExtDir
OneDrive
This PC
Desktop
Documents
Downloads
Music
Pictures
Bitstream Vera Sans_10.aauf
Bitstream Vera Sans_12.aauf
Bitstream Vera Sans_14.aauf
Bitstream Vera Sans_24.aauf
Bitstream Vera Sans_34.aauf
Bitstream Vera
Hide Folders
Save
Cancel

File name: Bitstream Vera Sans_18.aauf

Save as type: Amulet Anti-Aliased Unicode Font file (*.aauf)

To make a project more portable, the font file can be saved directly in a project's root directory (i.e. the same folder where the .gemp file resides). The project's root directory can be navigated to in the "Save as Amulet Font" window, or once the font is saved in the ProgramData\AmuletTech\Global\Configuration\Fonts folder, it can then be copied over to the project's root directory.

The save file name is the font name with _xx appended to the end, where _xx is the corresponding point size. This is the format that MUST be used for GEMstudio to find and use the font file correctly. The program automatically uses the appropriate extension based on the Glyph Quality radio button. Do NOT change the extension in hopes of creating a different quality font file. 1-bit bitmapped font files must have the .auf extension and 8-bit anti-aliased font files must have the .aauf extension.

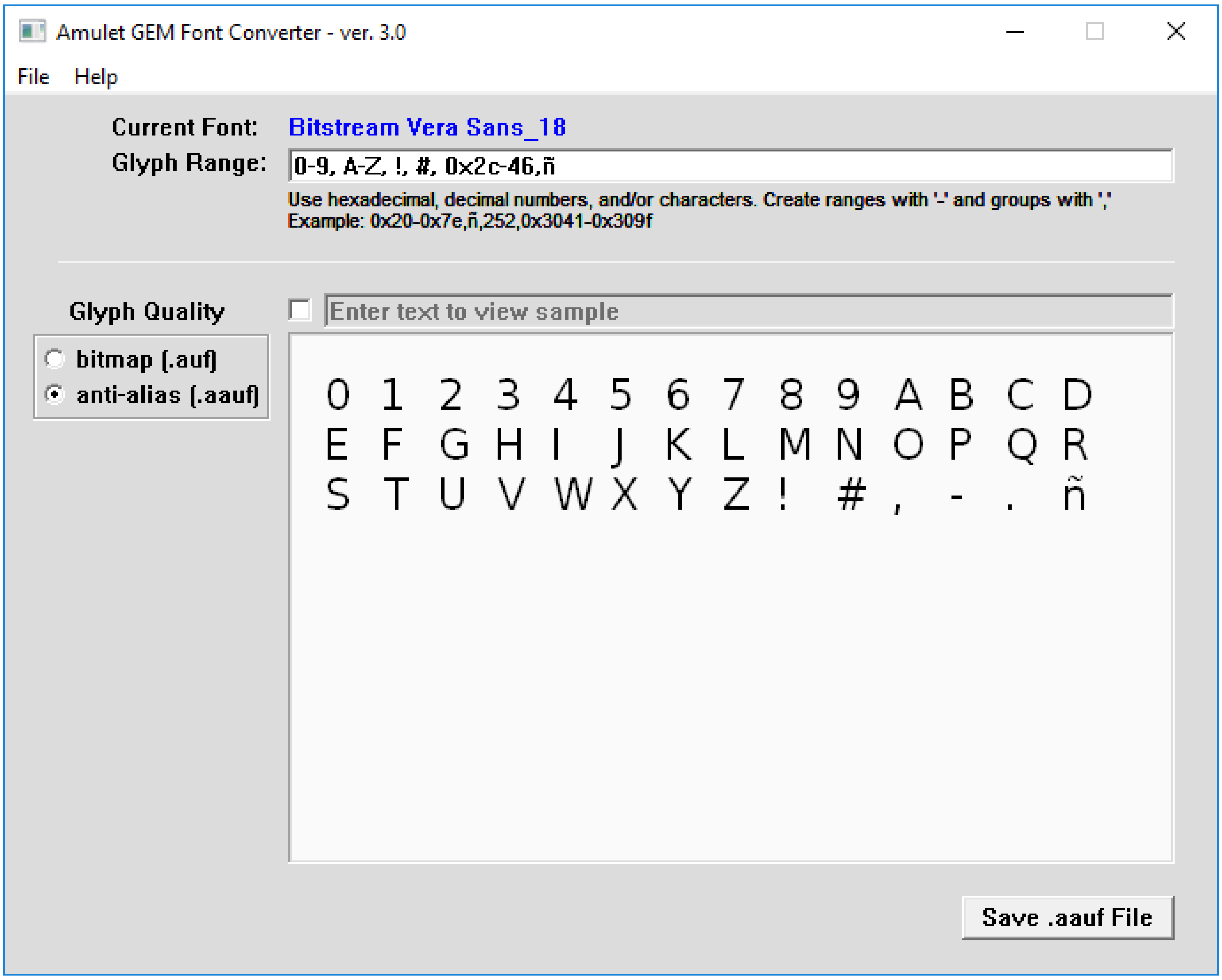
Selecting Ranges of Characters to Convert

The Amulet GEM Font Converter has the ability to convert all glyphs from 0x0000 to 0xFFFF. The Glyph Range edit box determines which glyphs to convert. If the edit box has not been modified, it will start with a default of the lower-ASCII characters, 0x20-0x7e. To convert and save either more or less than the default, edit the Glyph Range edit box. Ranges are specified by a - between the first and last character of the range. To specify the range, use either hexadecimal, decimal, or character format[*]. For instance, a capital A is 0x41 (or 65 if using decimal) in ASCII format, so entering A or 0x41 or 65 in the Glyph Range edit box will all result in the same capital A being used. The first character should be less than the last character. Multiple ranges can be selected by comma delimiting them. If a single character is desired, omit the - and just enter the single number or the character itself.

For example, to only convert the numbers 0 through 9, all the capital letters A through Z, the exclamation point, the number sign, the ASCII characters for , - and ., and the ñ(alt-164) symbol, the following string can be used in the Glyph Range edit box: 0-9,A-Z,!,#,0x2c-46,ñ

Notice that the actual characters were used in most of this edit box, but the range of ASCII characters for , - and . used a hexadecimal number(0x2c) to start the range and a decimal number(46) to end the range. The , and - are the only ASCII characters that cannot be used within the Glyph Range edit box. As in this example, if the comma or dash is to be entered in the edit box, use the hexadecimal or decimal representation of those two characters.

*If dealing with glyphs greater than 0xff, use a hexadecimal or decimal representation of the glyph rather than entering the character itself.


Amulet GEM Font Converter - ver. 3.0

File Help

Current Font: Bitstream Vera Sans_18

Glyph Range: 0-9, A-Z, !, #, 0x2c-46,ñ

Use hexadecimal, decimal numbers, and/or characters. Create ranges with '-' and groups with ','
Example: 0x20-0x7e,ñ,252,0x3041-0x309f

Glyph Quality
- ( ) bitmap (.auf)
- (•) anti-alias (.aauf)

Enter text to view sample

0 1 2 3 4 5 6 7 8 9 A B C D
E F G H I J K L M N O P Q R
S T U V W X Y Z ! # , - . ñ

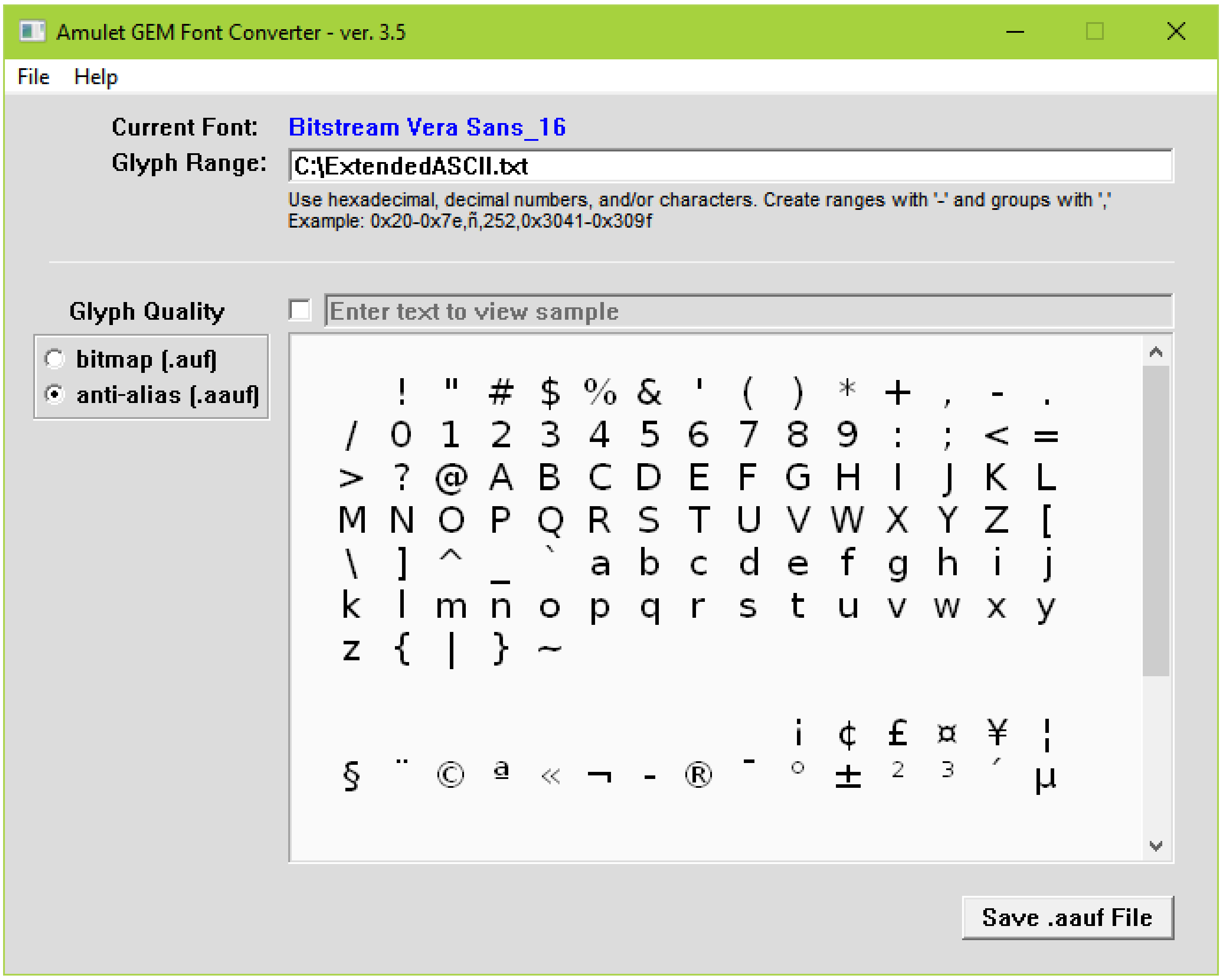
Save .aauf File

This allows for displaying font characters that are not part of the lower-ASCII range. The characters that can be found in the upper-ASCII and Unicode section (0x80 and above) are not always going to be the same from font to font, so ensure the glyph to be displayed is available in the selected font.

This option also allows for saving a smaller subset of a font if it is known that only a portion of a specific font is going to be used. For instance, if a large number font is to be used in a numeric field widget, it is possible to convert only the numbers 0 through 9. Being it is a large font, if all the lower ASCII characters were saved, it would result in a very large .auf/.aauf file which would eat up a large portion of the project's flash space. By selecting a range of 0-9, just the numbers 0 through 9 would be converted, resulting in a much smaller .auf/.aauf file, thus using much less flash space, and if the fonts are cached, much less RAM space. If the numeric field widget can display decimal or negative numbers then the period(.) and minus sign (0x2c) should be added to the Glyph Range.

Selecting Ranges of Characters to Convert with a File

Instead of typing your glyph ranges directly into the Font Converter's Glyph Range edit box, you can specify the location of a text file containing the glyph range information.


Amulet GEM Font Converter - ver. 3.5
File Help
Current Font: Bitstream Vera Sans_16
Glyph Range: C:\ExtendedASCII.txt
Use hexadecimal, decimal numbers, and/or characters. Create ranges with '-' and groups with ','
Example: 0x20-0x7e,ñ,252,0x3041-0x309f
Glyph Quality
bitmap (.auf)
anti-alias (.aauf)
Enter text to view sample
Save .aauf File

This feature can make it easier to switch between multiple ranges or specify a very long list of ranges.

The Glyph Range edit box value is persistent after closing down the Amulet GEM Font Converter no matter if a group of ranges or a file is specified.

File Menu

Load Font	Loads font for viewing. Brings up a font dialog box for choosing the font. Same as clicking the blue name of the font from the main window.
Exit	Exits the program.

Help Menu

Help	Opens this help document.
About	Brings up a pop-up window giving the version of the program.

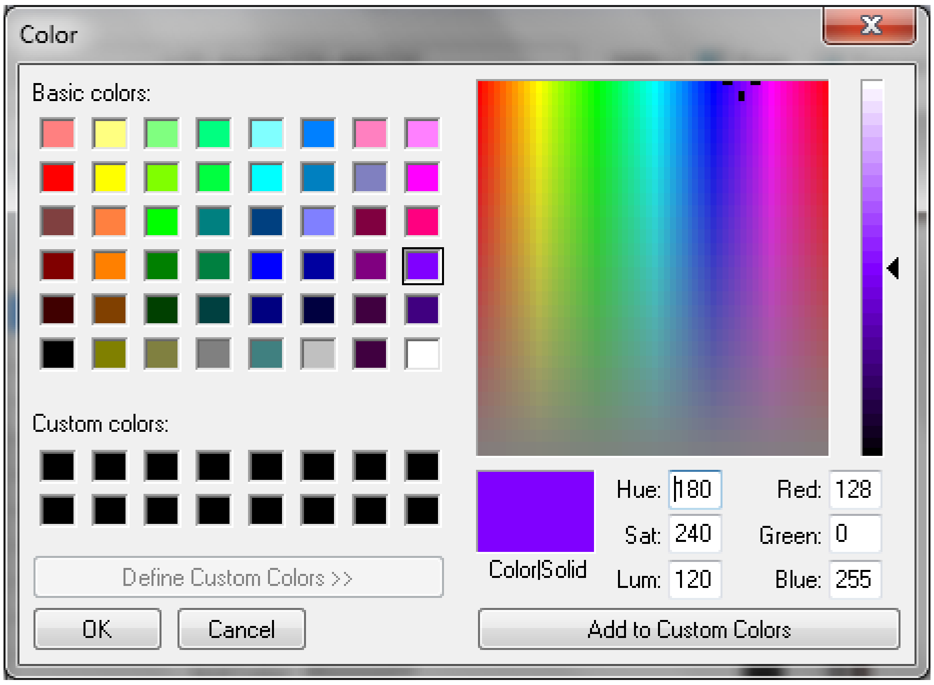
Disclaimer

Converting copyrighted TrueType and bitmap fonts for the purposes of resale, copyright infringement or licensing avoidance is strictly prohibited. Refer to the original font's licensing agreement for additional restrictions that may apply. Amulet Technologies will not be held liable for infringements made to a font's licensing agreement, nor will it take responsibility for the user's actions involving the use of the Amulet GEM Font Converter.

Colors

The color depth used in a project is determined by the Color Depth tab in the Project Properties menu option. The different color bit depth options are: 8 and 32. If using 8-bit color, a palette must be designated. If not designated, the AmuletDefault.pal color palette (found in the ProgramData/AmuletTech/Global/Configuration/Palettes directory) will be used. The AmuletDefault.pal color palette is the Web Safe palette with nine extra colors added. These nine extra colors correspond with the HTML standard color names that aren't normally found in the Web Safe palette.

When clicking on the color swatch rectangle of a color parameter, a standard color selection dialog appears. This dialog provides quick and easy access to configure your desired color, although it does not affect the alpha channel:



Hue: 180
Sat: 240
Lum: 120
Red: 128
Green: 0
Blue: 255

After selecting a color, the color value is displayed in the #rrggbbaa format within GEMstudio Pro and the alpha channel (aa) will not be changed. The one exception is the fontColor attribute, which uses the #rrggbb format because it does not support the alpha channel. By default, the alpha channel starts out at a value of FF, which means fully opaque. To change the alpha channel, click on the number instead of the color swatch rectangle and the value of any of the channels can be manually adjusted.

Instead of using the color picker, you can also use one of these methods below to specify a color.

1. Define a macro to represent your color. You can then change the value everywhere in your project with one central location, and you can define the macro as any of the methods below.

2. Use an InternalRAM Color variable. The full syntax is "Amulet:InternalRAM.color(x).value()", but you can use the shorthand "air.c(x).value()".

3. Use one of the 17 HTML standard color names:

#00FFFFFF	#000000FF	#0000FFFF	#FF00FFFF

Aqua	Black	Blue	Fuchsia
#808080FF Gray	#008000FF Green	#00FF00FF Lime	#800000FF Maroon
#000080FF Navy	#808000FF Olive	#FFA500FF Orange	#800080FF Purple
#FF0000FF Red	#C0C0C0FF Silver	#008080FF Teal	#FFFFFFFF White
#FFFF00FF Yellow			

HTML color names, along with the #rrggbbaa convention equivalent.

4. Specify the red, green, blue, and alpha values in hex using the following convention:#rrggbbaa, where rr is the 8-bit red value, gg is the 8-bit green value, bb is the 8-bit blue value, and aa is the 8-bit alpha (transparency) value. Each value can be a number from 00 to ff (in hex). The level of transparency is set by the alpha channel. The alpha channel is fully transparent with a value of 00 and completely opaque (no transparency) with a value of FF.

5. Specify the red, green, and blue values in decimal using the following convention:rgb(rrr,ggg,bbb), where rrr is the 8-bit red value, ggg is the 8-bit green value, and bbb is the 8-bit blue value. Each value can be a number from 0 to 255 (in decimal).

6. Specify the red, green, blue, and alpha values in decimal using the following convention:rgba(rrr,ggg,bbb,a.aa), where rrr is the 8-bit red value, ggg is the 8-bit green value, bbb is the 8-bit blue value, and a.a is the decimal value of alpha transparency. The color values can be a number from 0 to 255 (in decimal). The alpha value can range from 0.00 to 1.00. 0.00 is fully transparent and 1.00 is fully opaque(no transparency). 0.50 would be half transparency

7. Use an absolute hex number to specify an absolute index into an 8-bit color palette. This number should not be used to specify a 32-bit color! If you want to use decimal, precede with a 0, such as 0128.

Using Amulet Formatted Text

Amulet Formatted Text lets you change font attributes in the middle of a text string. The font attributes that can be modified are the font name and font size, the color of the font, and the four different style attributes of bold, italic, strikethrough and underline. The basic syntax used for the Amulet Formatted Text starts with a backslash followed by a control character and terminated with an underbar. Following the control character are modifiers that are specific to the control character. The Amulet Formatted Text can be used on any string used within the Amulet GEMstudio Pro system, including on strings sent via the UART and strings defined in GEMscript.

Style Attributes

The four style attribute modifiers all have a consistent syntax. To turn on a style attribute, use the backslash followed by the control character and terminated by the underbar. (i.e. to start using bold text, use: \b_) To turn off a style attribute, use the backslash followed by the control character and the number 0 and terminated by the underbar. (i.e. to stop using bold text, use: \b0_) Once a style attribute modifier is encountered in a string, all text to the right of the modifier takes on that attribute until either the end of the string or until the opposite style attribute modifier is encountered.

Style Attribute	Syntax
bold on	\b_
bold off	\b0_
italic on	\i_
italic off	\i0_

strikethrough on	\s_
strikethrough off	\s0_
underline on	\u_
underline off	\u0_

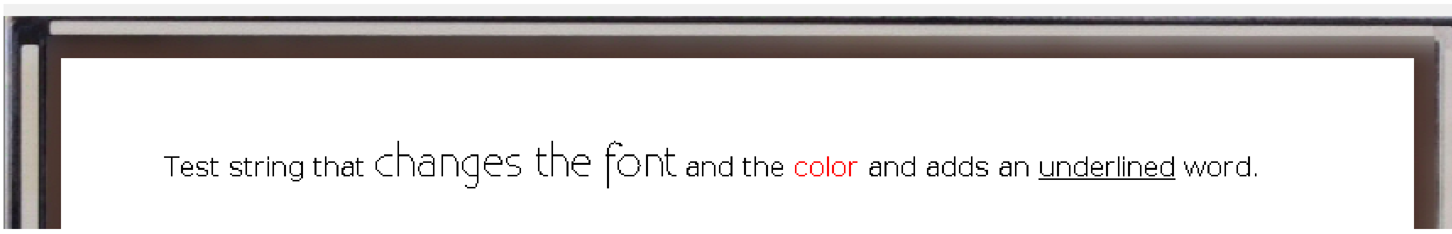
Font Attributes

Changing the color, size, or the font itself uses a similar syntax to the style attributes, but with different modifiers following the control character. Font changes require that the Amulet Font file exists in the project's font menu. The name and size of the font must match the name used in the project's font menu, including the three special font modifiers of (local), (anti-aliased), and (anti-aliased local).

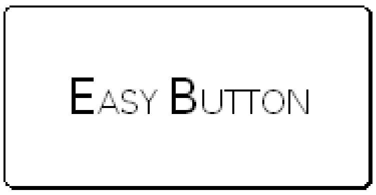
Font Attribute	Syntax	Notes
font change	\f{name, xxpt}_	where name is the name of the font and xx is the point size of the font
color change	\crrggbb_	where rr is the hex value of red, gg is the hex value of green, and bb is the hex valu
color change (8-bit)	\cxx_	where xx is the index into the current color palette using a hex number
color change (InternalRAM color)	\Cxx_	where xx is the index into the InternalRAM.color(xx) variable using a hex number

Examples:

1) In a static text that is set to use Bitstream Vera Sans, 12pt in black and the text attribute contains the following:
Test string that \f{Existence Light, 18pt}_changes the font\f{Bitstream Vera Sans, 12pt}_ and the \cff0000_color \c000000_ and adds an \u_underlined\u0_ word.


Test string that changes the font and the color and adds an underlined word.

2) In a function button with its font set to Bitstream Vera Sans, 18pt (anti-aliased) that has the following in its label attribute:
E\f{Bitstream Vera Sans, 12pt (anti-aliased)}_ASY \f{Bitstream Vera Sans, 18pt (anti-aliased)}_B\f{Bitstream Vera Sans, 12pt (anti-aliased)}_UTTON

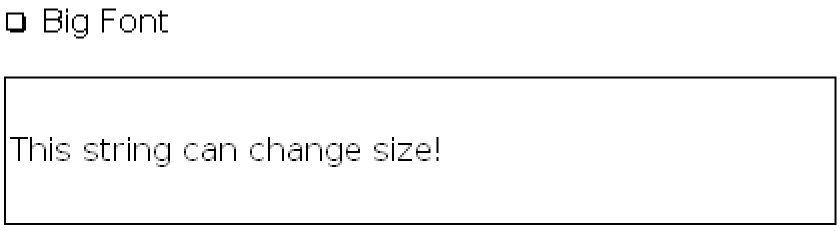


3) This example shows how you can use the font change modifier as a dynamic string to adjust the size of a font in a stringField widget using a lone checkBox and a GEMscript function.
In a stringField that has an href of InternalRAM.string(0) and the following in its printf:
%.50sThis string can change size!

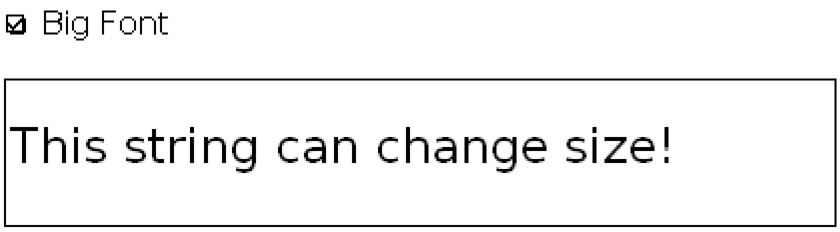
A lone checkBox called cbFontSize that can be checked or unchecked and calls GEMscript.setFontSize() in its href, and the following in GEMscript:
public setFontSize()
{

```
    if (document.cbFontSize.value() == 0) {
        InternalRAM.string(0) = "\f{Bitstream Vera Sans, 12pt (anti-aliased)}_";
     }
     else {
         InternalRAM.string(0) = "\f{Bitstream Vera Sans, 18pt (anti-aliased)}_";
     }
  }
```

This is the stringField with the checkBox unchecked:



This is the stringField with the checkBox checked



Note: In order to use the font change modifier you must have the Cache Fonts option enabled in the Project Properties dialog, Misc tab. This is on by default for new projects, but may be set to off for opened projects.

Numbers

Integers can be entered in hexadecimal or decimal. Decimal examples are used in this document. If hexadecimal is desired, precede the number with 0x. For example:

Amulet:uart1.invokeRPC(10)

is equivalent to

Amulet:uart1.invokeRPC(0x0A)

Image Formats

With the exception of the Animated Image, which only accepts the animated .GIF file format, all other images used by image objects or widgets can use any of these options:

GIF

This is the most compact format to store images in the Amulet system. Images can only contain 256 colors, and one of these (per image) may optionally be assigned as the transparency color and will not be drawn. Because of this color loss, this is typically not suitable for photographic images. Also, the transparency is 1-bit, i.e. either full-on or full-off, which does not give you the rounded edges look of high quality graphics.

PNG

This is the workhorse format of most Amulet projects due to the lossless compression, full color quality, and 256 levels of transparency for nice rounded edges. The files usually are larger than .GIF format, but they are much easier/quicker to decompress than .JPEG

JPEG

This is a compact but lossy file format which is best used for photographs. It does not support any kind of transparency.

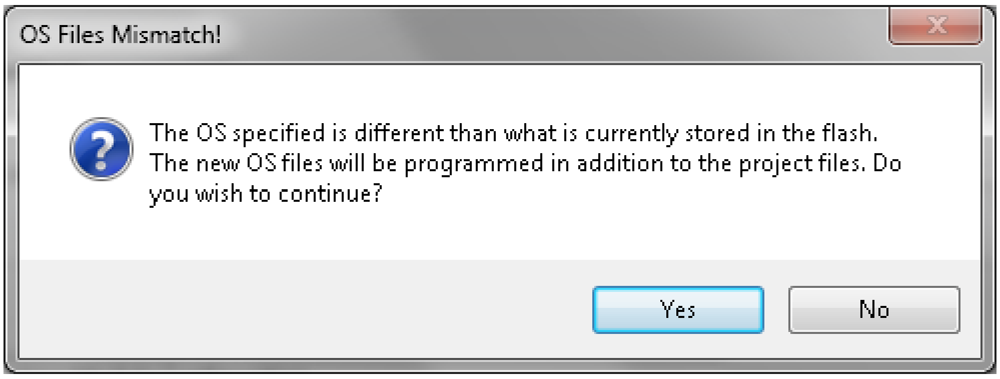
Alternatively, **.BMP .TIFF, and .PICT** formats can be used, but these are not native formats to the Amulet OS, so GEMstudio will automatically convert them to PNGs for you.

Programming the Amulet OS

The files stored in the onboard embedded flash are comprised of a mix of project related files and generic OS files. When programming a completely blank embedded flash, both the project and OS files will be programmed in. But, when programming a project into an embedded flash that already has the proper OS files, only the project files will be programmed in.

When the Program Project button is hit, the OS files in the embedded flash are tested to see if they match the settings specified in the project. If so, the project will be programmed in.

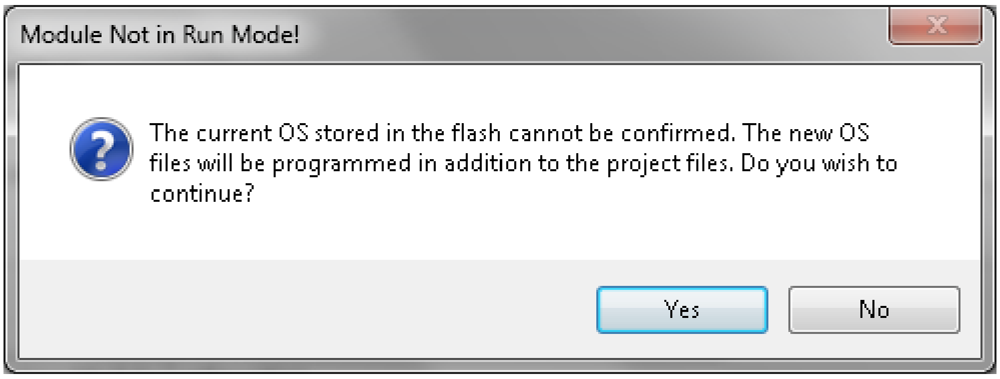
But, if the OS files do not match the settings specified in the project, the following pop-up window is displayed:


OS Files Mismatch!

The OS specified is different than what is currently stored in the flash. The new OS files will be programmed in addition to the project files. Do you wish to continue?

Pressing No will close the pop-up window and will close the Program Flash window. Pressing Yes will program in both the new OS files as well as the project files. This enforces that the OS files always match the project files.

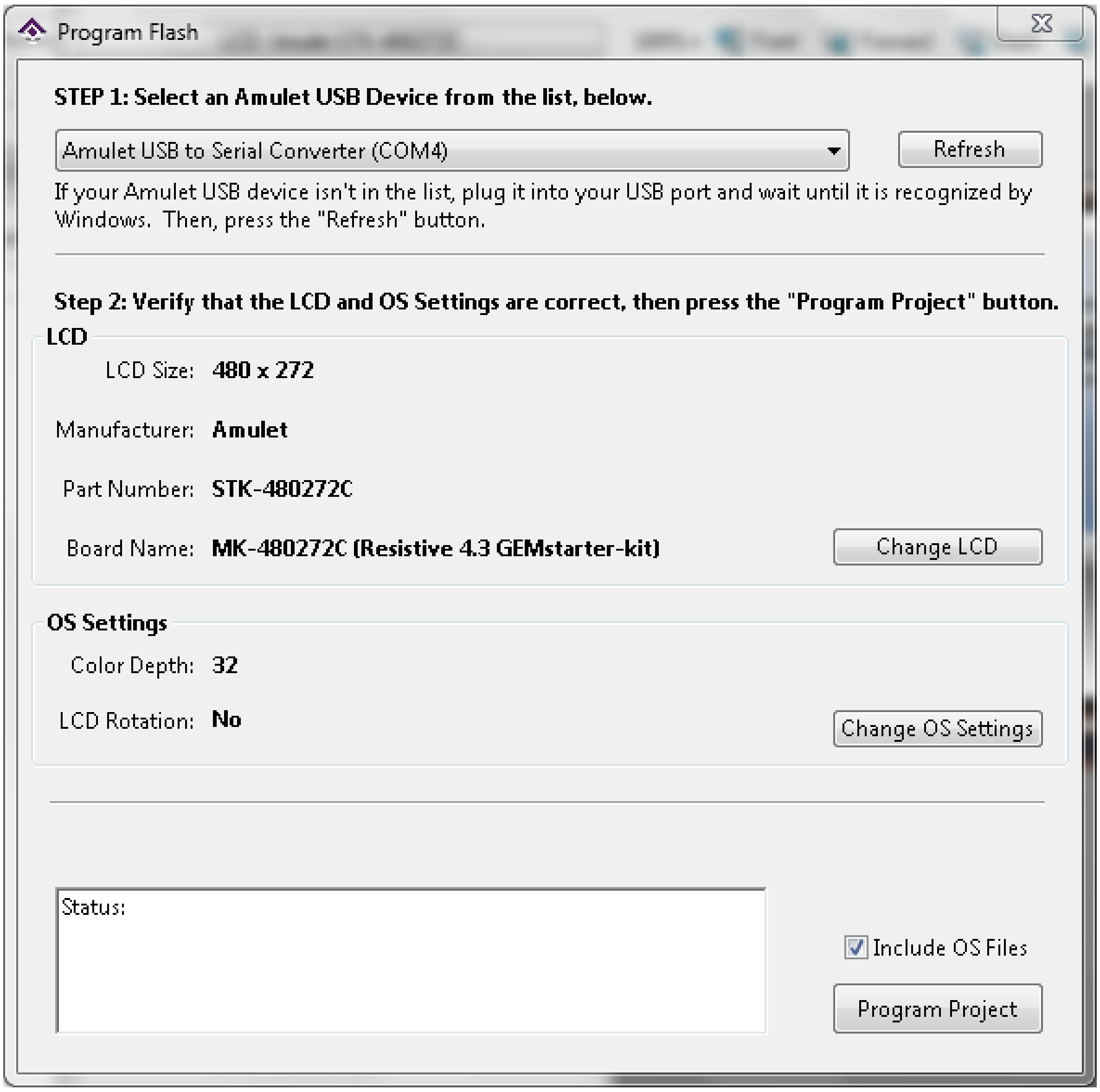
In the MK-07C-HP Module , the OS files can be tested whether the module is in Run Mode or Program Mode. But, in the standard module, the OS files can only be tested when the module is in Run Mode and currently running a project. If the Program Project button is hit while a standard module is in Program Mode, the following pop-up window is displayed:


Module Not in Run Mode!

The current OS stored in the flash cannot be confirmed. The new OS files will be programmed in addition to the project files. Do you wish to continue?

Pressing No will close the pop-up window and will close the Program Flash window. Pressing Yes will program in both the new OS files as well as the project files. This enforces that the OS files always match the project files.

If it is ever desired to manually force the OS to be programmed in, check the Include OS Files checkbox prior to hitting the Program Project button in the Program Flash window.


Program Flash

STEP 1: Select an Amulet USB Device from the list, below.

Amulet USB to Serial Converter (COM4)

If your Amulet USB device isn't in the list, plug it into your USB port and wait until it is recognized by Windows. Then, press the "Refresh" button.

Step 2: Verify that the LCD and OS Settings are correct, then press the "Program Project" button.

LCD

LCD Size:	**480 x 272**
Manufacturer:	**Amulet**
Part Number:	**STK-480272C**
Board Name:	**MK-480272C (Resistive 4.3 GEMstarter-kit)**

OS Settings

Color Depth:	**32**
LCD Rotation:	**No**

Status:

✓ Include OS Files

This will program both the OS files and project files together.

Localization

GEMstudio provides an easy interface to help your project to become translated into other languages. In general, the GNU gettext method is used. This is broken up into several steps and the examples rely on a third party translator's tool called Poedit.

Note: In order to use localization you must have the Cache Fonts option enabled in the Project Properties dialog, Misc tab. This is on by default for new projects, but may be set to off for opened projects.

Marking Strings for Translation

The first step in translation is marking which strings in the project need to be translated. The marked strings get extracted from the source code into a format that is friendly with translator's tools, such as Poedit. GEMstudio automatically marks labels in FunctionButtons, CustomButtons, RadioButtons and Checkboxes, the options strings in the List and StringField widgets, the static text in the printf of NumericField and StringField widgets, and of course the text of Static Text widgets. To mark strings in non-automatic locations, you can simply take the source string to be translated and wrapping it with a gettext function:

```
gettext("Source String")
```

There are several other areas which you may or may not want to mark strings:

1) Setting a string type using the IWCs InternalRAM.string(x).setValue("String") or document.setValue("String")

Dynamic strings such as those in InternalRAM string variables or Intrinsic values of other StringField objects can be marked for translation by inserting the method "gettext("String")" right into the setValue function:

Amulet:InternalRAM.String(x).setValue(gettext("String"))
Amulet:document.widgetName.setValue(gettext("String"))

This does double duty as it marks the string for translation so that the string extractor tool will detect it, and this will also tell the Amulet OS to translate the string at runtime into the currently selected language.

2) Setting a string type using GEMscript

There are two times when you may want to mark a string in GEMscript. There is the method similar to the one above, in which you want to set a String to a value directly such as:

InternalRAM.string(x) = "String"
document.widgetName.stringValue = "String"

The "String" can be wrapped with gettext:

InternalRAM.string(x) = gettext("String")
document.widgetName.stringValue = gettext("String")

This will call send the "String" to the OS for translation, pull it back into GEMscript, then send it out to the OS again to set the variable or widget value. This is not very efficient as several copies of the String are allocated. Another option is to just mark the string for translation, but don't actually do the translation in GEMscript, using the _noop version of gettext:

InternalRAM.string(x) = gettext_noop("String")
document.widgetName.stringValue = gettext_noop("String")

This will not translate the string at runtime (in GEMscript) as there are other ways to translate the dynamic string when displaying the string.

3) Dynamic Strings

You may want to translate strings retrieved from your host processor which are not part of the normal GEMstudio project, for example in a StringField HREF of Amulet:UART.string(0).getext(), but you don't know how to mark them for translation. To accomplish this you can wrap your string with gettext, and then comment it out:

//gettext_noop("String From Host")

The extractor tool will walk through all code including comments and find all of these strings.

Context

If multiple strings are found to be exactly the same, they will not generate a new entry to be translated. Sometimes one word in English can have two different meanings, which are different words in another language. When only using `gettext` both will have a single source string and a single translation. For example, "Open" might mean open a file, or open a door, which are different phrases in some languages. To get around this, we use a "context" identifier to distinguish the two strings. We have implemented a "context" parameter in many widgets and the "`pgettext`" version of `gettext` supports a second context parameter. The "p" stands for "particular" to indicate you are looking for a particular translation. The format is:

```
pgettext("context String", "Source String")
```

This can be used in the same places as `gettext` to mark strings.

1) Setting a string type using the IWCs InternalRAM.string(x).setValue("String") or document.setValue("String")

Amulet:InternalRAM.String(x).setValue(pgettext("context","String"))
Amulet:document.widgetName.setValue(pgettext("context","String"))

2) Setting a string type using GEMscript

InternalRAM.string(x) = pgettext("context","String")
document.widgetName.stringValue = pgettext("context","String")

Guidelines For Marking Strings

Before strings can be marked for translations, they sometimes need to be adjusted. Usually preparing a string for translation is done right before marking it. The GNU gettext specification recommends several steps in order to ensure that the translator understands the intent of the original message. The GNU spec can be found here but the pertinent sections are listed below. In some cases the code examples are not valid code in GEMstudio, but the example illustrates the point. What you have to keep in mind while doing the marking is the following.

Decent English style.

Translatable strings should be in good English style. If slang language with abbreviations and shortcuts is used, often translators will not understand the message and will produce very inappropriate translations.

```
"%s: is parameter\n"
```

This is nearly untranslatable: Is the displayed item *a* parameter or *the* parameter?

```
"No match"
```

The ambiguity in this message makes it unintelligible: Is the program attempting to set something on fire? Does it mean "The given object does not match the template"? Does it mean "The template does not fit for any of the objects"?

In both cases, adding more words to the message will help both the translator and the English speaking user.

Entire Sentences

Translatable strings should be entire sentences. It is often not possible to translate single verbs or adjectives in a substitutable way.

```
printf ("File is %s protected", rw ? "write" : "read");
```

Most translators will not look at the source and will thus only see the string "File is %s protected", which is unintelligible. Change this to

```
printf (rw ? "File is write protected" : "File is read protected");
```

This way the translator will not only understand the message, she will also be able to find the appropriate grammatical construction. A French translator for example translates "write protected" like "protected against writing".

Entire sentences are also important because in many languages, the declination of some word in a sentence depends on the gender or the number (singular/plural) of another part of the sentence. There are usually more interdependencies between words than in English. The consequence is that asking a translator to translate two half-sentences and then combining these two half-sentences through dumb string concatenation will not work, for many languages, even though it would work for English. That’s why translators need to handle entire sentences.

Split at Paragraphs

Often sentences don’t fit into a single line. If a sentence is output using two subsequent printf statements, like this

```
printf ("Locale charset \"%s\" is different from\n", lcharset);
printf ("input file charset \"%s\".\n", fcharset);
```

the translator would have to translate two half sentences, but nothing in the POT file would tell her that the two half sentences belong together. It is necessary to merge the two printf statements so that the translator can handle the entire sentence at once and decide at which place to insert a line break in the translation (if at all):

```
printf ("Locale charset \"%s\" is different from\n\
input file charset \"%s\".\n", lcharset, fcharset);
```

You may now ask: how about two or more adjacent sentences? Like in this case:

```
puts ("Apollo 13 scenario: Stack overflow handling failed.");
puts ("On the next stack overflow we will crash!!!");
```

Should these two statements merged into a single one? I would recommend to merge them if the two sentences are related to each other, because then it makes it easier for the translator to understand and translate both. On the other hand, if one of the two messages is a stereotypic one, occurring in other places as well, you will do a favor to the translator by not merging the two. (Identical messages occurring in several places are combined by the extracting tool, so the translator has to handle them once only.)

Translatable strings should be limited to one paragraph; don't let a single message be longer than ten lines. The reason is that when the translatable string changes, the translator is faced with the task of updating the entire translated string. Maybe only a single word will have changed in the English string, but the translator doesn't see that (with the current translation tools), therefore she has to proofread the entire message.

Avoid string concatenation.

Hardcoded string concatenation is sometimes used to construct English strings:

```
strcpy (s, "Replace ");
strcat (s, object1);
strcat (s, " with ");
strcat (s, object2);
strcat (s, "?");
```

In order to present to the translator only entire sentences, and also because in some languages the translator might want to swap the order of object1 and object2, it is necessary to change this to use a format string:

```
sprintf (s, "Replace %s with %s?", object1, object2);
```

Avoid unusual markup and unusual control characters.

Unusual markup or control characters should not be used in translatable strings. Translators will likely not understand the particular meaning of the markup or control characters.

For example, if you have a convention that '|' delimits the left-hand and right-hand part of some GUI elements, translators will often not understand it without specific comments. It might be better to have the translator translate the left-hand and right-hand part separately.

Translating Dynamic Strings

By default, the dynamic string area (%s) of the StringField is not automatically translated. The source of the dynamic strings could already be translated, so the programmer must decide based upon the source.

Pre-translated Strings:

When the string is already translated (either coming pre-translated from the Host processor or manually Translated in GEMscript) then the href of the StringField should use the .value() method. For example:

```
Amulet:InternalRAM.string(x).value()
Amulet:document.widgetName.value()
Amulet:UART.string(x).value()
```

Untranslated Strings:

When the string needs to be translated, then the href of the StringField should use the .gettext() method. For example:

```
Amulet:InternalRAM.string(x).gettext()
Amulet:document.widgetName.gettext()
Amulet:UART.string(x).gettext()
```

Context in Dynamic Strings:

Even if a Dynamic String has a context, the gettext method is still used in the StringField instead of pgettext. The dynamic string must have the context embedded into the source string in a specific format. That format is:

"Context" + 0x04 + "Source String" + 0x00

There is no null termination after the Context string, then Context and Source String are separated with a single \4, the EOT character, then the Null terminated Source String. In ASCII Hex, the above string would look like this:

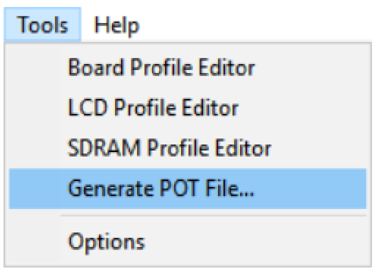
```
   C  o  n  t  e  x  t     S  o  u  r  c  e     S  t  r  i  n  g
0x43 6F 6E 74 65 78 74 04 53 6F 75 72 63 65 20 53 74 72 69 6E 67 00
```

When you set a string with gettext when you Mark Strings for translation, the runtime code will always set the string in the proper language. So the concatenation method described here would typically only be used for strings arriving from an outside source, such as the serial port.

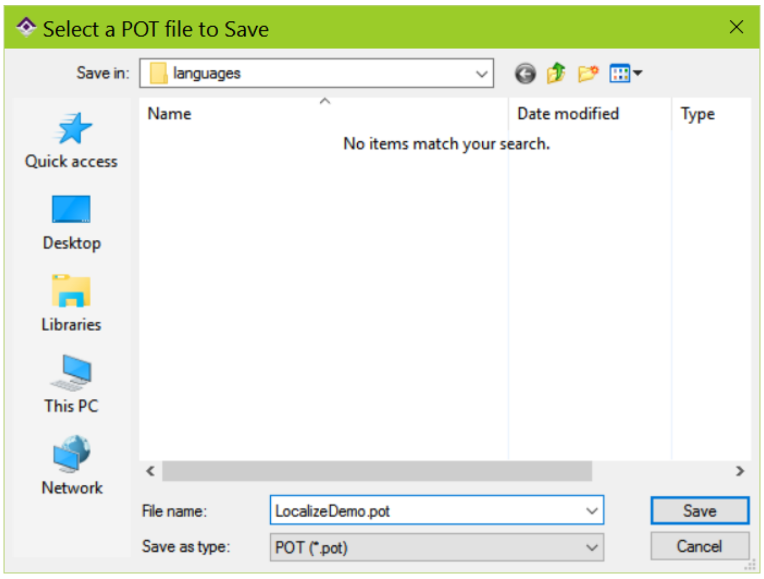
Extracting Strings to be Translated

Once all the strings that need to be translated have been marked, then they can be quickly extracted to generate a Portable Object Template, or .POT file. This is a human-readable format which translator tools such as Poedit can open in order to create translations for specific languages.

To generate the .POT file, Navigate to the Tools menu in GEMstudio and select "Generate POT File..."



This will bring up a Dialog box to choose the save location. By default the location is a "languages" sub-directory under your last opened project's directory, and it will create this directory if it doesn't already exist.



The resultant .pot file is what you would send to the translator of each language.

Format of the .POT

Upon examining the .POT file in a text editor, you will find that each string is composed of 1 or more comments starting with a '#' and exactly 2 or 3 lines of code. The output for a Widget named `MyFuncButton_1` on a page named `Page_1` with the label of "Open" would look like this:

```
#: label parameter for Widget MyFuncButton_1 in Page_1
msgid "Open"
msgstr ""
```

Where:

`#:` is an automatically generated comment that tells us where this string came from.
`msgid` is the string to be translated
`msgstr` is the translated string

Message Context

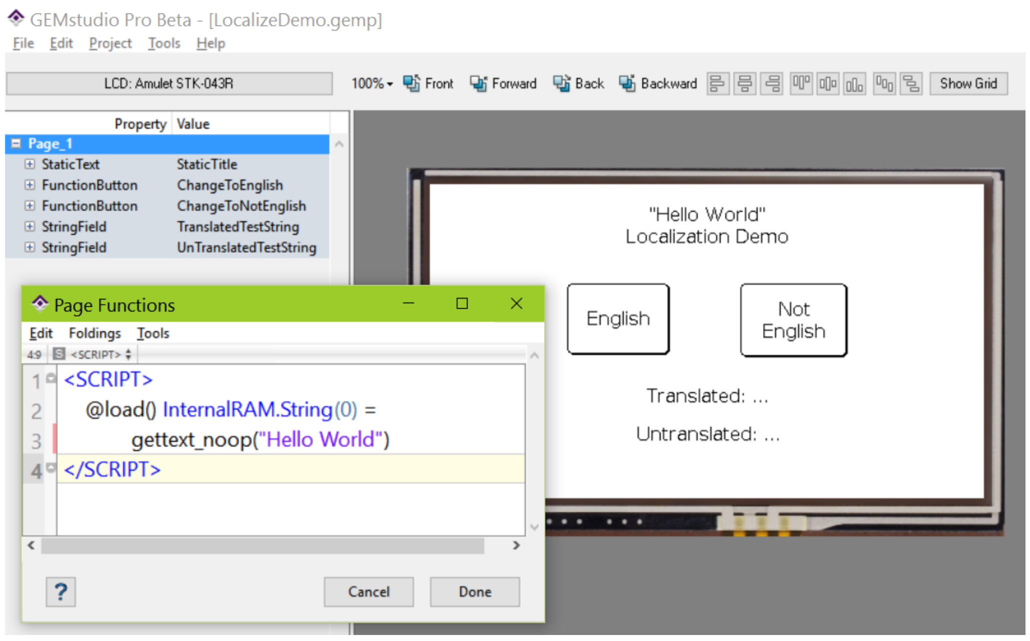
When the extracting tool sees that a context exists, it will generate an additional line of code. In the example below, two buttons with the same label in English can handle different strings in another language.

```
#: label parameter for Widget MyFuncButton_1 in Page_1
msgctxt "File"
msgid "Open"
msgstr ""

#: label parameter for Widget MyFuncButton_2 in Page_1
msgctxt "Door"
msgid "Open"
msgstr ""
```

As a side note, the `msgid` is actually appended to the `msgctxt` string when generating the language database, and these strings are searched with a binary search, so try to avoid excessively long context strings as this may slow performance in certain circumstances.

The Translator's Job

Unless you as the programmer are bilingual, it is usually not the programmers responsibility to translate the strings in a project to a new language. If you are that bilingual programmer, or your translator is not familiar with the Poedit tool, this will walk through the required steps. Assuming you have followed the previous steps, you should now have a .POT file.
As an example, we will look at the LocalizeDemo.gemp project that comes with GEMstudio in your folder: Documents/ GEMstudio Pro/Amulet Examples/Localization, and it looks like this in GEMstudio:


GEMstudio Pro Beta - [LocalizeDemo.gemp]
File Edit Project Tools Help
LCD: Amulet STK-043R
100% Front Forward Back Backward Show Grid
Property Value
Page_1
StaticText StaticTitle
FunctionButton ChangeToEnglish
FunctionButton ChangeToNotEnglish
StringField TranslatedTestString
StringField UnTranslatedTestString
Page Functions
Edit Foldings Tools
"Hello World"
Localization Demo
English
Not English
Translated: ...
Untranslated: ...
Cancel
Done

```
<SCRIPT>
   @load() InternalRAM.String(0) =
        gettext_noop("Hello World")
</SCRIPT>
```

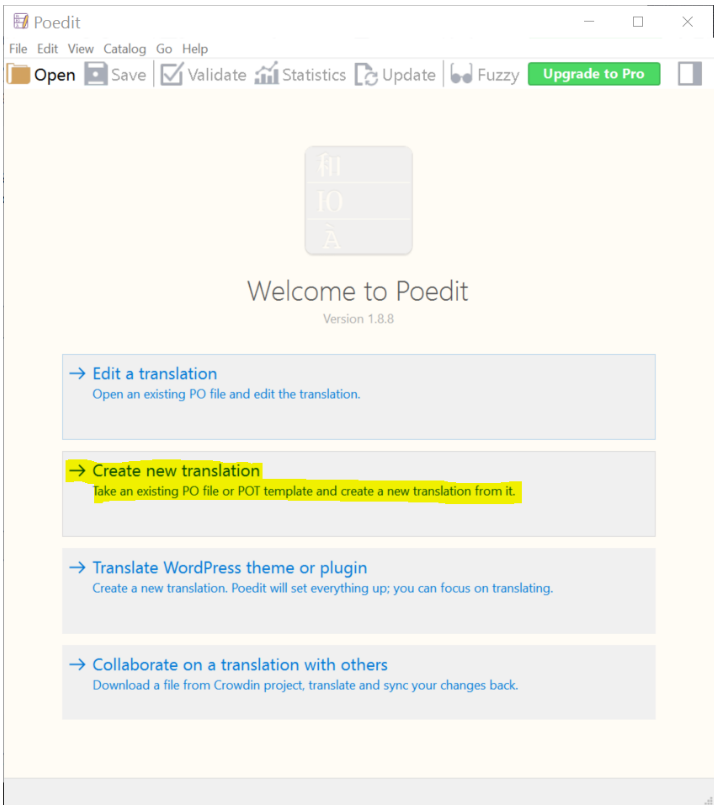
There are several strings to translate, including one marked in GEMscript which writes an **untranslated** string to a string variable, which is then read by both StringField widgets. For demonstration purposes, one of those StringField widgets will translate at runtime using the gettext() method, and the other will not translate.

Generating a POT file from this project gives the following, which is ready to import into Poedit:

```
#: label parameter for Widget ChangeToEnglish in Page_1
msgid "English"
msgstr ""

#: label parameter for Widget ChangeToNotEnglish in Page_1
msgid "Not English"
msgstr ""

#: printf parameter for Widget TranslatedTestString in Page_1
msgid "Translated: %s"
msgstr ""

#: printf parameter for Widget UnTranslatedTestString in Page_1
msgid "Untranslated: %s"
msgstr ""

#: text parameter for Widget StaticTitle in Page_1
msgid "\"Hello World\"\nLocalization Demo"
msgstr ""

#: Line 3 in the Page Functions for Page_1
msgid "Hello World"
msgstr ""
```

Poedit

When you first open Poedit you get a splash screen with several options. The first time here for each project, choose the "Create new translation". If you've already started the translation and are adding new strings to to it, Pick "Edit a translation" and then select the "Update from POT File..." option in the Catalog menu.


Poedit
File Edit View Catalog Go Help
Open Save Validate Statistics Update Fuzzy Upgrade to Pro

Welcome to Poedit

Version 1.8.8

→ Edit a translation

Open an existing PO file and edit the translation.

→ Create new translation

Take an existing PO file or POT template and create a new translation from it.

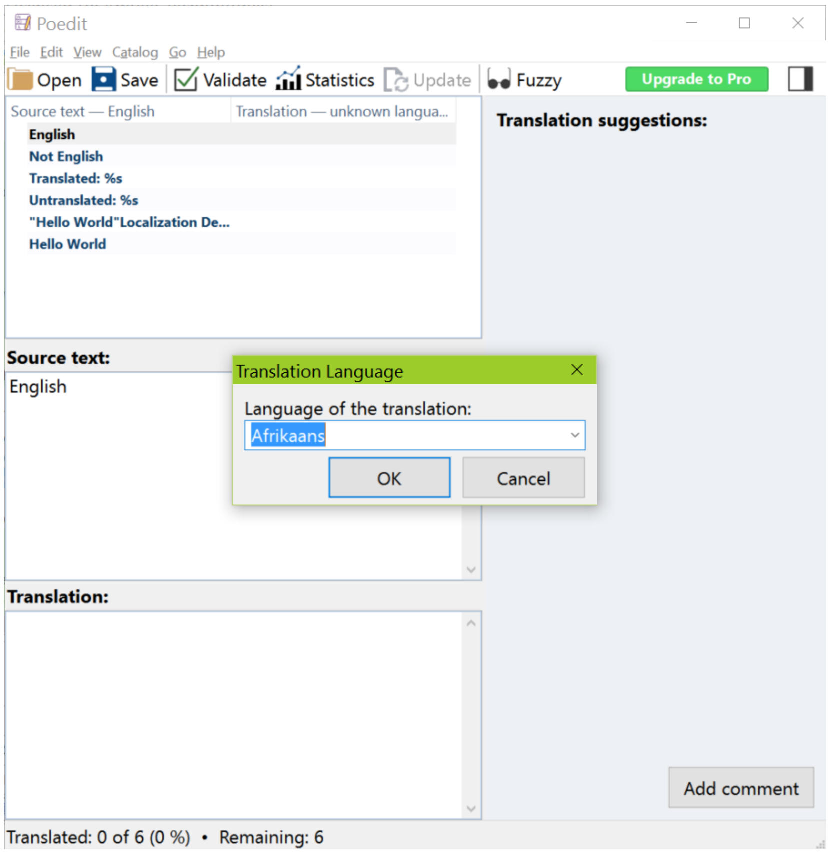
→ Translate WordPress theme or plugin

Create a new translation. Poedit will set everything up; you can focus on translating.

→ Collaborate on a translation with others

Download a file from Crowdin project, translate and sync your changes back.

After navigating to and opening the file from the LocalizeDemo project, if it is your first time here you will be prompted to pick a language for the translation. I'm just choosing the first item in the current list, Afrikaans. By no means do I know any Afrikaans, so this translation will be an utter sham, but through it I hope to show the process.


Poedit
File Edit View Catalog Go Help
Open Save Validate Statistics Update Fuzzy Upgrade to Pro
Translation suggestions:
Add comment

Source text — English	Translation — unknown langua...
English	
Not English	
Translated: %s	
Untranslated: %s	
"Hello World"Localization De...	
Hello World	

Source text:

English

Translation Language

Language of the translation:

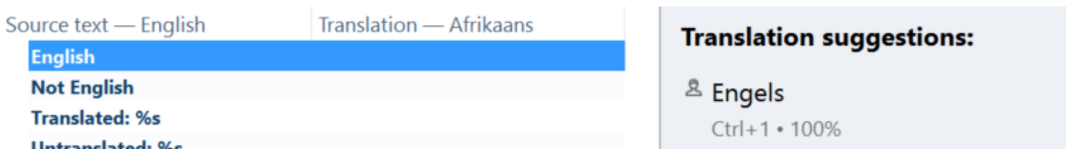
Afrikaans

OK Cancel

Translation:

Translated: 0 of 6 (0 %) • Remaining: 6

All of the Strings in the .POT file have been imported into a .PO file and now we can start translating. We can see all of the strings, and by clicking on any line under the source text we can even get a suggestion from Poedit (limited use in free version)



Source text — English | Translation — Afrikaans

English

Not English

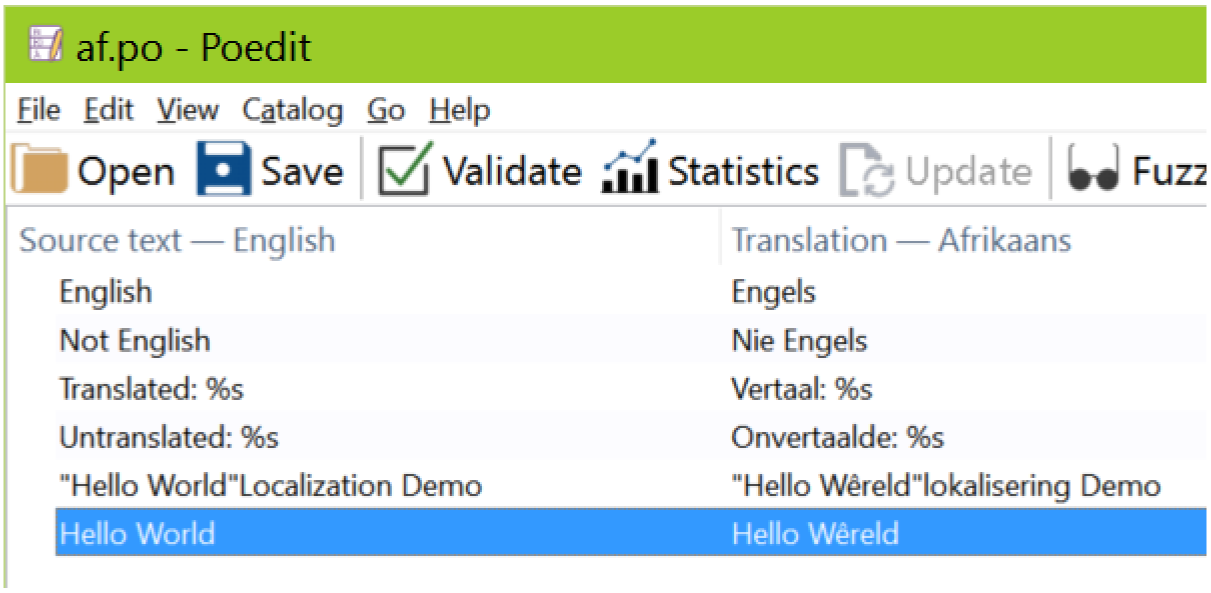
Translated: %s

Translation suggestions:

Engels

Ctrl+1 • 100%

Using Poedit's suggested translations or Google's translate feature I come up with the "quick and dirty" translations for each string. **Note**: If a string does not have a translation, or if the string searched for is not in the file at all, then it will just use the English version.


af.po - Poedit
File Edit View Catalog Go Help
Open Save Validate Statistics Update Fuzz

Source text — English	Translation — Afrikaans
English	Engels
Not English	Nie Engels
Translated: %s	Vertaal: %s
Untranslated: %s	Onvertaalde: %s
"Hello World"Localization Demo	"Hello Wêreld"lokalisering Demo
Hello World	Hello Wêreld

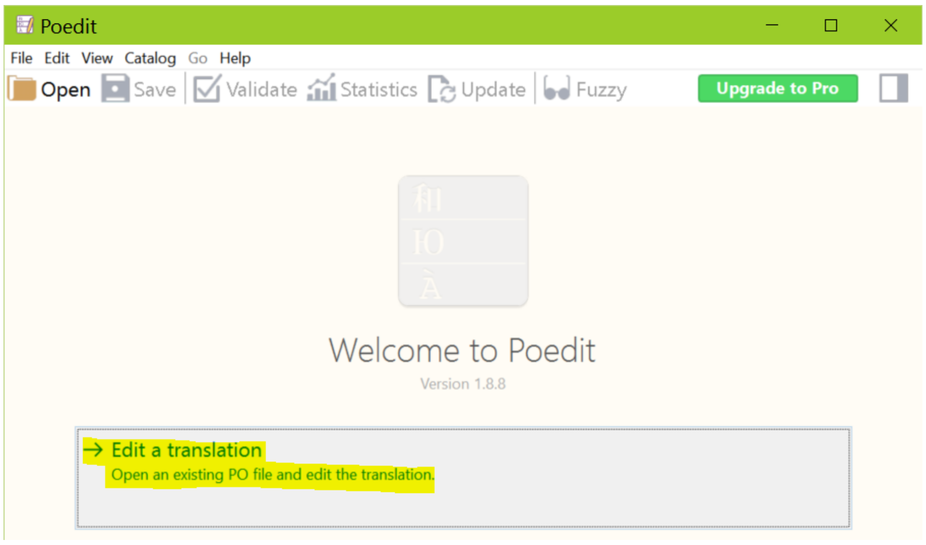
By Hitting the Save button, it defaults to the location of the imported POT file, and the Language code and optional Country Code of the selected language. In this case, just "af.po" for Afrikaans. You don't want to change the name of the file, as this is a standardized format, and we will be using the name of this file back in GEMstudio.

Saving the .po file from Poedit also, by default, compiled the "af.po" file into its compiled version, "af.mo". These .mo files are the files that GEMstudio uses. The .mo files should be saved into the same folder as the .pot file, under the languages folder of your project's root directory. Once there, they will become available as resources in your GEMstudio project.

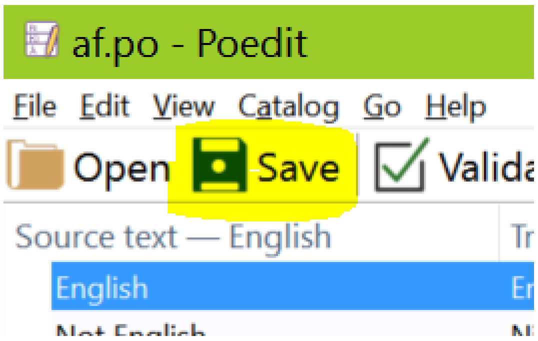
Compile .PO to .MO

A translator might be more comfortable giving you a PO file instead of a pre-compiled MO. One reason is that the PO can have translator comments, which can be questions for the programmer to clarify the meaning. These comments are all lost in a compiled MO. Another reason is they may not be familiar with Poedit and more comfortable just editing the PO file manually. In any case, the programmer might have to compile the MO themselves, but this is a very simple process with Poedit

The first step is to open the PO file in Poedit. Open Poedit and Select "Edit a translation"


Poedit
File Edit View Catalog Go Help
Open Save Validate Statistics Update Fuzzy
Upgrade to Pro
Welcome to Poedit
Version 1.8.8
Edit a translation
Open an existing PO file and edit the translation.

Next, hit the Save button to compile to the same source directory as the PO. Alternatively, Select the "Compile to MO..." option from the File menu to choose the save location manually.



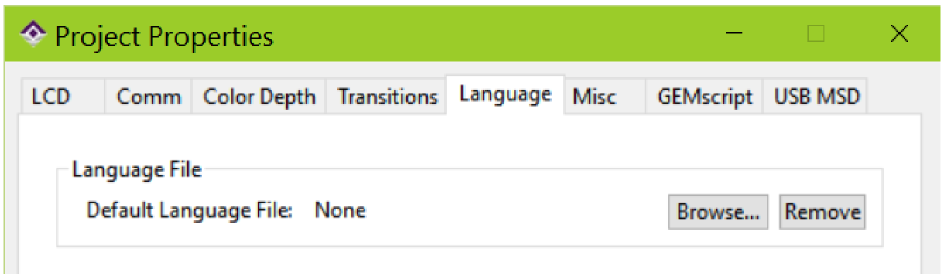
If the PO file was already in the right spot (languages folder of GEMstudio project root), then you're done. If not, just copy it to the right spot, though its probably a good idea to keep all of your translation files in one spot (like the languages sub-folder of your GEMstudio project!)

Using .MO files

Once you have one or more .mo files in the languages sub-folder of your project directory, you can set the default language, change the language at runtime, or save the changed language setting to flash so that the next boot will default to that language.

Change or Remove the Default Language

To set the default language of your project, select Project Properties from the Tools menu, then navigate to the Language Tab



Default Language File: None

Select Browse to navigate to and open one of your .mo files to use that as the default language. Select Remove to start with the language the GUI was designed in (not necessarily English)

Changing the language at Runtime

When you want to change the language at runtime, you can use a simple command:

```
Amulet:language.set(filename.mo)
```

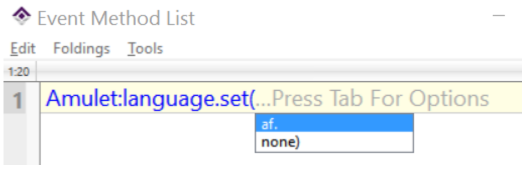
Where:
`filename.mo` is the name of the .mo file that contains the desired language translation.
The format of `filename` is standardized. It contains a 2 or 3 character language code in lowercase and a 2 character optional country code in uppercase. If the country code exists, they are separated by an underbar. For example:

Filipino - fil.mo
Canadian French - fr_CA.mo

Usage:
In the HREF Editor window of any control widget, or in GEMscript, you can use autocomplete feature by Tabbing to select, or manually type:

```
 Amulet:language.set(
```

Then hitting Tab one more time will give a list of all of the filenames of .mo files that exist in the proper directory. Using only the "Afrikaans" language file from the previous pages of this help, you would see this:



```
Amulet:language.set(...Press Tab For Options
```

The `af` represents the Afrikaans language file and `none` means use no .mo file at all. After selecting "af" you are prompted to tab one more time to complete the filename and the rest of the statement with `.mo)`

Set Current Language as Default

It is often convenient to have the user interface boot up to the local language without intervention. This is easily accomplished by saving a change to the current language selection to flash. This is done with the command:

Amulet:language.saveToFlash()

Poedit

This Help uses the Poedit tool written by Václav Slavík to demonstrate the role of the translator in the localization process.

Poedit is a free translation file editor that is available from https://poedit.net/

There is a Pro version, but the examples and screenshots provided here are all from the free version.

GEMstudio API

Overview

This section will cover all the tools you have access to when designing your Graphical User Interface in GEMstudio Pro.

Amulet Function Calls

Amulet has created a number of I/O objects, referred to as Amulet Widgets which can be added into your GEMstudio project. There are two types of Amulet Widgets: Control Widgets and View Widgets. Control Widgets are input objects, like function buttons, sliders, radio buttons, etc... Control Widgets have a function, or a set of functions, that can be applied to them. View Widgets are output objects, like bargraphs, numeric fields, string fields, etc... View Widgets call a function which returns the data used as the input for that particular widget. See Amulet Widgets for a detailed description of each widget.

Amulet widgets use the **href** parameter to specify a function call. A function call can be a jump to another page, a request for the value of an external variable, a command to invoke a Remote Procedure Call, and much more. A function call can also send a command to other widgets, known as Inter-Widget Communication (IWC). See Appendix B for a comprehensive list of all available functions.

Function Call Conventions

Amulet function calls borrow some of it syntax from Java Script, a scripting language used within HTML. Except for calling scripts and jumps to other pages, all href function calls start with **"Amulet:"**. If it is desired to jump to another page, then just use the name of the file to link to in the href parameter (i.e. NewPageName.open()).

Amulet function calls also borrow concepts from Java, an Object-Oriented Programming(OOP) language. When it is required to interface to an external server, use **"Amulet:uart0."** The **uart0**. can be thought of as a UART object. As in object oriented programming, each object has its own set of data and a set of well-defined interfaces to that data. As in Java, these interfaces are known as methods. Methods are just functions that are specific to a particular object. Each object has its own set of methods.

Examples:
Amulet:uart0.byte(0).value()
uart0 specifies that a serial message will go out uart0.
byte(0) specifies byte variable 0.
value() specifies the value of byte variable 0 is returned.

Amulet:internalRAM.word(5).setValue(0xF020)
internalRAM specifies the dual port RAM onboard the Amulet.
word(5) specifies the internal RAM word variable 5.
setValue(0xF020) specifies internal RAM word variable 5 is to be set to the value 0xF020.

Amulet:internalRAM.string(5).setValue("Your String")
internalRAM specifies the dual port RAM onboard the Amulet.
string(5) specifies the internal RAM string variable 5.
setValue("Your String") specifies that internal RAM string variable 5 is to contain the null terminated string "Your String". For more information regarding string variables, see the note regarding string variables.

The same nomenclature as Java is used, where a method is called by using the object's name followed by the dot operator, followed by the method. Amulet has added a new wrinkle with the concept of multiple byte, word and string variables. Since there are 256 different byte variables, 256 different word variables, 256 different color variable, and 256 different 25-character (plus Null) string variables available, there needs to be a way of specifing the type of variable as well as the variable number. Therefore, if the object is a byte, word, color, or string variable, the nomenclature is of the following type:

> **Amulet:object.variable(variable #).method(argument, if needed).**
> **Amulet:** specifies an Amulet specific command.
> **object** can be UART or InternalRAM. variable.
> **variable** can be byte, word, color or string.
> **variable #** can be a number from 0-255
> **method** can be any number of methods described in Appendix B.

See example href command below, where a function button, when pressed, causes a Remote Procedure Call # 5 to be sent out one of the UARTs to the external server:

> **Amulet:uart0.invokeRPC(5)**

The **href** line above invokes the **invokeRPC()** method on a UART object called **uart0**. That is, it calls **invokeRPC()** relative to the **uart0** UART object. Thus, the call to **uart0.invokeRPC()** causes **uart0** in the Amulet controller to send out an "invokeRPC" command.

The method **invokeRPC()** requires an argument (a parameter that is passed to the method that the method uses as its input). If the parenthesis are left blank, the argument passed to any method is the intrinsic value of the calling widget/object. Only Control Widgets/Objects have an intrinsic value. Function buttons can specify an intrinsic value by specifying the **intrinsicValue** parameter or by including a number between the ()'s. Widgets that can have multiple intrinsic values, like lists and sliders, must not include a number between the ()'s, since their intrinsic value is dependent upon the state of the widget.

For a list of all available functions, see Appendix B.

Control Widget Function Calls

Control Widgets are user input widgets such as list widgets, slider widgets, radio buttons, checkboxes and function buttons. META REFRESH objects can also call functions, like Control Widgets, but they can also act like a View Widget. Typically, Control Widget/Object function calls are initiated by a "hit" of the widget. A "hit" can occur one of two ways.

The typical way is the physical "hit", which occurs when the active region of the object/widget on the touchscreen is touched and then released while still within the bounds of the active region. The object/widget must be in focus when letting up on the touchscreen for the object/widget to initiate its function calls. If you touch an active region, but move off the region while still touching the touchscreen, the object/widget will lose focus, therefore, letting up on the touchscreen will do nothing.

The alternative way a "hit" can occur is to have one Control Object/Widget invoke the forceHit() method of another Control Object/Widget. See IWC documentation for more information.

There are three exceptions to the rule that a "hit" initiates all function calls. The first exception is scrolling, such as in a Slider Widget. In the default mode, as soon as a pen down event occurs within the boundaries of a slider, a function call is initiated. If, while still in the pen down state, new values of the slider are selected, new function calls will be initiated. The second exception is the auto-repeat feature of button type widgets. If an auto-repeatable button is touched, and stays in a pen down state longer than the time specified by the repeatDelay attribute, then a function call is initiated. As long as the button remains in a pen down state, the function calls will repeat at the frequency specified by the repeatRate attribute. The third exception is in the case of buttons which are set up to call their HREF upon pen-down AND pen-up, by setting the **ExecuteOn** attribute to "Both". Immediatly after this type of button is hit, it will execute its HREF. It will execute the HREF again (or the next HREF in the case of sequenced function calls) upon either a successful pen-up within the button, or when leaving the button. This allows you to create a button that performs different actions upon pen-down and pen-up without worrying about getting out of sequence if the user scrolls off of the button before lifting.

The Touch Area Widget has a number of "hit" options that no other widget has. The onRelease parameter sends out a "hit" like all other widgets, but the following options are unique to the Touch Area:
onAutoRepeat
onDoubleTap
onSlideOff
onSlideOn
onTap
onTouch

Control Objects/Widgets cannot call functions that return a value. See Appendix B for a comprehensive list of all valid control functions.

Multiple Function Calls

Control Objects/Widgets can call multiple functions at one time by separating the functions by a comma. There is a limit of 36 multiple functions that can be attached to a single Control Widget/Object **href**. To illustrate how to use multiple functions, see the following example. To create a function button that invokes an external RPC #5 and then links to **page2**, use the following href commands:

Amulet:uart0.invokeRPC(5), page2.open()

Multiple function calls are performed in the order they are entered on the line, from left to right*. In the previous example, **Amulet:uart0.invokeRPC(5)** would be called first, and then a jump to **page2** would follow. As soon as a page is linked to, any subsequent function calls are discarded. For this reason, if it is desired to link to another page, that must be the last function call within a multiple function call.

***Important notes regarding the order of multiple function calls:** If there is a mix of UART and InternalRAM or IWC function, the functions might not be performed from left to right. The reason for this is that UART functions are loaded into a UART transmit buffer whereas InternalRAM and IWC functions are loaded into a different function buffer. Due to the nature of UART transmissions, they will take a considerable amount of processor time to complete the function call in comparison to InternalRAM and IWC functions. Therefore, any InternalRAM or IWC function which is part of a multiple function call will most likely be finished prior to any UART function call which is part of the same multiple function call.

Sequenced Function Calls

Control Objects/Widgets can also call sequenced functions by separating the functions by semi-colons. Sequenced functions allow for different function calls at each successive "hit". Sequenced function calls are performed in the order they are entered on the line, from left to right. The sequences continually wrap, so the first sequence follows the last sequence. To illustrate how to use sequenced functions, see the following example. To create a toggle type custom button that invokes an external RPC #5 when toggled down and invokes an external RPC #6 when toggled up, use the following href commands:

Amulet:uart0.invokeRPC(5);Amulet:uart0.invokeRPC(6)

Sequenced function calls can also be made up of multiple function calls. There is a limit of 36 different sequences per Control Widget/Object **href** and each sequence can have a maximum of 36 multiple function calls. To help illustrate this, use the previous example, but instead of invoking RPC #5 for one sequence, and then RPC #6 for another sequence, let's assume we would like the first sequence to invoke RPC #4 and RPC #5, and the second sequence to invoke RPC #6 and RPC #7. To accomplish this, use:

Amulet:uart0.invokeRPC(4),Amulet:uart0.invokeRPC(5);Amulet:uart0.invokeRPC(6),Amulet:uart0.invokeRPC(7)

notice the sequences are separated by the semi-colon, and the multiple function calls are separated by the commas.

Using Commas and Semi-colons within Strings stored in Function Calls

Since commas and semi-colons are used to distinguish Multiple Function Calls and Sequenced Function Calls respectively, in order to use commas and semi-colons within strings that are used in function calls, you must use the escape character '\' prior to the comma or semi-colon. For example, to set InternalRAM string variable 5 to the string "To use a comma, use the escape character; same for the semi-colon.", the href would look like this:

Amulet:InternalRAM.string(5).setValue("To use a comma\, use the escape character\; same for the semi-colon.")

Using Backslashes, Commas and Semi-colons within Strings stored in Function Calls

Since commas and semi-colons are used to distinguish Multiple Function Calls and Sequenced Function Calls respectively, in order to use commas and semi-colons within strings that are used in function calls, you must use the escape character '\' prior to the comma or semi-colon. Conversely, since the backslash('\') is used as the escape character, in order to use a backslash within a string, you must use two backslashes in a row. For example, to set InternalRAM string variable 5 to the string "To use a comma, or backslash \ use the escape character; same for the semi-colon.", the href would look like this:

Amulet:InternalRAM.string(5).setvalue("To use a comma\, or back slash \\ use the escape character\; same for the semi-colon.")

If you fail to use the escape character in front of commas, semi-colons and back slashes, you will get an error when you try to Run or Program the project, although it won't explicitely say that the escape character is missing. Rather you will get an error complaining of a missing link.

View Widget Function Calls

View Widgets are used for displaying data. View Widgets include lineplots, bargraphs, and numeric fields. View Widget function calls are initiated by either a timer event or the IWC (inter-widget communication) method **forceUpdate**. The frequency of the timer event is specified by the **updateFreq** parameter of each view widget. They can only call a single function, and that function must return either a byte, word(2-bytes), or UTF-8 string. The returned value is used as the input to the View Widget.

Some Control Widgets can also have limited View widget functionality via the **initHref** parameter. This allows a widget to poll some type of data, internal or external, for use to initialize the widget's state. For instance, the initial position of a slider's handle can "remember" what it was from the last time it left the page by using the same InternalRAM variable in the Href and InitHref. The initHref is called anytime the widget is initialilzed, including the **forceUpdate** method. See each widget's documentation on how to use the initHref.

Amulet Page Functions

GEMstudio Pro uses Page Functions in order to enter logic that goes beyond the simple href function calls that control widgets can launch. Page functions are split up into two categories, but both are entered on the same dialog box

1) GEMscript is a simple C-like scripting language with added functionality of Amulet-style function calls. Script code is separated with an opening <script> and closing </script> tag to differentiate them from other objects in the Page Functions dialog. The GEMscript API is in its own document accessible through the GEMstudio Help menu.

2) META Refresh Objects are multi-purpose objects which can be launched either by a timer or by being invoked by another function or widget. Page functions can also perform an if-then-else logic, allowing for more complex timers. Click here for more in-depth documentation on META Refresh Objects and the different ways they can be used as Amulet page functions.

For further instructions and tips on the Page Functions Editor, please click here.

Below are a couple of templates that can be used to copy and paste into the Page Functions window:

To create a script that automatically launches once upon page load:

```
<script>
@load()
{
	internalRAM.string(0) = "Hello World!"
}
</script
```

To create an unconditionally launched function:

```
<meta http-equiv="Refresh" content="0,0.01;url=Amulet:InternalRAM.byte(0).setValue(0xFF);name=initMeta">
```

To create a conditionally launched function:

```
<meta http-equiv="Refresh" content="0.2;
	if=Amulet:InternalRAM.byte(0).value();
	eq=0xFF;
	then=Amulet:InternalRAM.byte(0).setValue(0), Page_2.open();
	else=Amulet:InternalRAM.byte(0).increment();
name=testerMeta">
```

For a list of all available functions, see Appendix B.

<EMBED>

For including a text file into the GEMstudio project, the **<EMBED>** function is used. All text within the included file will be treated as if it were part of the page and located within the page at the same location as the **<EMBED>** tag. This function can be used anywhere within the page. It is especially useful for replacing a number of META Refresh tags that exists in multiple pages, but cannot be used for GEMscript. If you wish to include the same GEMscript on multiple pages, please see Nested Pages

URL="filename", where "filename" is the name of the text file to include at that specific location within the page.
Example:

```
<EMBED URL="examplefilename">
```

Amulet Internal RAM

Amulet has over 8 KBytes of onboard RAM which we have turned into virtual dual port RAM. There are 256 different byte variables, 256 different word (16-bit) variables, 256 different Color (32-bit) variables, and 256 different 25-character null terminated string variables (25+1=26 bytes allocated per string variable). Amulet Widgets can read or write to these Internal RAM variables. The Internal RAM variables can also be saved in flash, thus giving the variables permanence. In addition, an external processor can read and write to these Internal RAM variables as well. The external processor can send an unsolicited serial message to the Amulet to read or write to the Internal RAM variables. This means that the external processor is not required to be the Amulet's slave. Follow this link to learn more about the communication protocol. You can setup your pages to a) have the Amulet always be the master, b) have your processor always be the master or c) have a dual master system.

Some of the main features of InternalRAM:

1) Internal RAM variables can survive from page to page of your GUI project.
2) Internal RAM can be saved back to the flash, so the variable can persist even after powering down.
3) Internal RAM variables can be used as arguments within Amulet methods. i.e. (Amulet:UART.byte(0).setValue(InternalRAM.byte(2)))
4) Internal RAM variables can be used as variable indices. i.e. (Amulet:UART.byte(InternalRAM.byte(0)).setValue(2))
5) Internal RAM variables can be set or requested from GEMscript. i.e. (new int a = InternalRAM.byte(0))

The following documentation refers to Internal RAM usage in function calls outside of GEMscript. For syntax and examples inside GEMscript, please see the GEMscript API documentation, available from GEMstudio's Help menu.

Internal RAM nomenclature

The href nomenclature for Internal RAM's is **Amulet:InternalRAM.variableType(variableNumber).method()**

Where:

Amulet: is the Amulet script escape telling the compiler that Amulet specific commands follow.

InternalRAM is the specifier for Amulet's dual port Internal RAM.

variableType is the type of variable, either byte, word or string.

variableNumber is the variable index within the variable type. 0-255 for bytes, 0-255 for words and 0-198 for strings.

method() is the name of the method to be performed by the InternalRAM variable.

As a point of reference, the nomenclature to specify the Internal RAM variables in the code is the same as specifying external variables. External variables expect the UART or USB object to be used whereas the Internal RAM variables use the InternalRAM object. For example, to have an Amulet Bargraph Widget tied to an external byte variable # 2, the **href** attribute would look like this:

"Amulet:uart0.byte(2).value()"

To be tied to an Internal RAM byte variable # 2, then the href attribute would look like this:

"Amulet:InternalRAM.byte(2).value()"

The uart0 object specifies the Amulet will send out a serial message to an external processor that is connected to the UART port uart0 requesting the value of the external byte variable number 2. The InternalRAM object specifies that the Amulet will read the value of the Internal RAM byte variable number 2. The InternalRAM object will not send out any serial requests since the Amulet is capable of reading the Internal RAM directly.

Using InternalRAM variables as method arguments

A powerful feature of Internal RAM variables is that they can be used as arguments within Amulet href methods. This means that they can be changed at run-time by your external processor. Internal RAM byte, word and string variables can all be used as method arguments. For example, to have a button send out an RPC that is defined by Internal RAM byte variable #1, you would use the following nomenclature:

"Amulet:uart0.invokeRPC(InternalRAM.byte(1))"

The value which is contained within Internal RAM byte variable #1 would be sent out as the RPC number over the UART port named uart0.
Another example which sends out an Internal RAM string variable:

"Amulet:uart0.string(0).setValue(InternalRAM.string(2))"

This sends the string contained within Internal RAM string variable #2 out the UART port named uart0 as a "set external string variable #0" command to an external processor.

When using Internal RAM variables as method arguments, use the following naming conventions:

InternalRAM.byte(x)
InternalRAM.string(x)
InternalRAM.word(x)

Do not precede with **Amulet:** or end with **.value()**.

Using InternalRAM byte variables as variable indices

Another powerful feature of Internal RAM variables is that they can be used as variable indices for the base variable type. Since variable indeces only range from 0x00-0xFF, only Internal RAM byte variables can be used as variable indices. For example, to have a button set an external byte variable that is defined by InternalRAM byte #1 to a value of 0x20, you would use the following:

"Amulet:UART.byte(InternalRAM.byte(1)).setValue(0x20)"

The value contained within Internal RAM byte variable #1 would be used as the external byte variable number to set to 0x20.
Another example which reads the Internal RAM word variable defined by InternalRAM byte variable #0 would use the following:

"Amulet:InternalRAM.word(InternalRAM.byte(0)).value()"

The value contained within Internal RAM byte variable #0 would be used as the Internal RAM word variable number to read from.
When using Internal RAM variables as variable indices, use the following naming convention:

InternalRAM.byte(x)

Do not precede with **Amulet:** or end with **.value()**.

Note: Internal RAM byte variables can only be used as a variable index of the base variable type of an href function. It cannot be used as an index to an Internal RAM variable being used as a method argument. That means the following CANNOT be used:

"Amulet:uart0.invokeRPC(InternalRAM.byte(InternalRAM.byte(5)))"

It is acceptable to have an href which uses Internal RAM variables as both a variable index as well as an argument. It can be a little confusing to look at, though. The following is a valid href:

"Amulet:InternalRAM.word(InternalRAM.byte(5)).setValue(InternalRAM.word(6))"

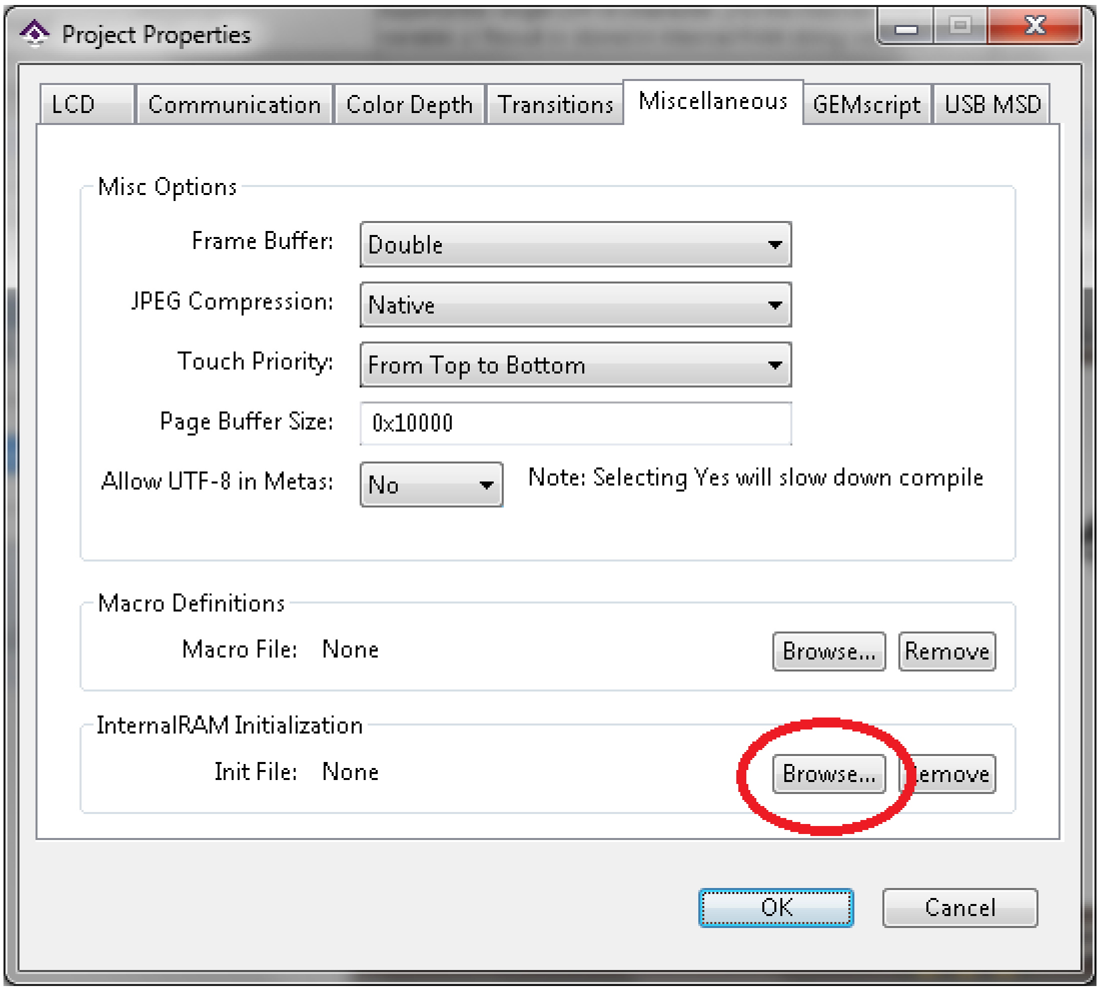
Which would result in the Internal RAM word variable number defined by the value of InternalRAM byte variable #5 set to the value of Internal RAM word variable #6.

Using InternalRAM string variable for a button label

When using an InternalRAM string as the label for a button which is using **FromInitHref** as its label, the initHref parameter of the button should look like: **Amulet:InternalRAM.label(x).value()**, where x is the index of the Internal RAM string variable. There is not a separate bank of Internal RAM label variables, this function will return the string associated with Internal RAM string variable # x. Due to the requirement of the button to have a label returned to it, it was necessary to create an InternalRAM label variable, but it shares the exact same memory space as the Internal RAM string variables. The Internal RAM label should only be used in this application.

Initializing Internal RAM variables at compile time

By default, all Internal RAM variables are initialized to zero. You can initialize Internal RAM variables at compile time by including an initialization file in your project. The initialization file must have an "ini" extension. You can include the initialization file in your project by selecting the file in the Project Options, under the Miscelaneous tab:



Misc Options

Frame Buffer: Double

JPEG Compression: Native

Touch Priority: From Top to Bottom

Page Buffer Size: 0x10000

Allow UTF-8 in Metas: No

Note: Selecting Yes will slow down compile

Macro Definitions

Macro File: None

InternalRAM Initialization

Init File: None

Inside the initialization file, any line preceded with // is treated as a comment. All initializations must be located in the far left column, so do not tab over. The GEMcompiler recognizes both decimal and hexadecimal numbers.

Initializing single Internal RAM variables

The syntax within the include file to initialize a single Internal RAM variable is as follows:

InternalRAM.variableType(variableNumber) = value

Where:

variableType is the type of variable, either byte, word, color or string.

variableNumber is the variable index within the variable type. 0-255 for bytes, 0-255 for words, and 0-255 for strings.

value is the initialization value of the Internal RAM variable (range if byte 0-0xFF, if word 0-0xFFFF, if color #rrggbb or #rrggbbaa, if string 1-250 character string)

Examples:

InternalRAM.byte(0xFE) = 0x7F
InternalRAM.word(32) = 4000
InternalRAM.color(0) = #800080
InternalRAM.color(1) = #408000FF
InternalRAM.color(2) = "green"
InternalRAM.color(3) = orange
InternalRAM.string(0) = "First String"

Internal RAM colors can be entered using the #rrggbb or #rrggbbaa convention. If the aa is left off, the alpha channel is saved as fully opaque (0xFF) internally. Colors can also be initialized by using the HTML color names with or without quotes. If the Internal RAM color variable is going to be used as a generic 32-bit variable as opposed to an actual color then the value should be entered using either decimal or hexadecimal numbers as opposed to the #rrggbb convention. The reason for this is because at compile time, the #rrggbbaa format is translated into a 0xaabbggrr hexadecimal format. If the Internal RAM color is initialized using a decimal or hexadecimal number the compiler will not perform the translation. For instance, entering InternalRAM.color(0) = #4080C0 would internally be saved as 0xFFC08040, but entering InternalRAM.color(0) = 0x4080C0 would internally be saved as 0x004080C0.

There is an important thing to note regarding the 256 25-character plus 1 byte Null strings. It is acceptable to have strings and initialize strings that are longer than 25 characters. You just need to be aware that the string will run on into the next string variable's RAM space. So, if you know your Internal RAM strings are going to be more than 25 characters, you might want to only use every other string variable index. i.e. InternalRAM.string(0), InternalRAM.string(2), InternalRAM.string(4)...etc. Keep in mind that this will effectively give you only 128 51-character string variables instead of the standard 256. In this example, the strings are 51 characters instead of 50 beacause the null between string(0) and string(1) becomes a usable character.

When initializing Internal RAM strings, user-defined wraps can be specified by entering "\n" within the string. Since we use double quotes to define a string, to have a literal double quote appear in the string, enter two double quotes in a row.

User-defined wrap example: **InternalRAM.string(0) = "top line\nbottom line"**
Double quote example: InternalRAM.string(0) = """this phrase"" is quoted"

Initializing multiple Internal RAM variables

It is also possible to initialize a block of contiguous Internal RAM variables. The syntax to initialize a block of Internal RAM variables is as follows:

InternalRAM.variableType(variableNumberStart-variableNumberEnd) = value

Where:

variableType is the type of variable, either byte, word, color or string.

variableNumberStart is the variable index within the variable type. 0-255 for bytes, 0-255 for words, 0-255 for color, and 0-255 for strings.

variableNumberEnd is the variable index within the variable type. 0-255 for bytes, 0-255 for words, 0-255 for color, and 0-255 for strings.

value is the initialization value of the Internal RAM variable (range if byte 0-0xFF, if word 0-0xFFFF, if color 0 - 0xFFFFFFFF, if string 1-25 character string)

Example:

InternalRAM.byte(0xFC-0xFF) = 0x7F
InternalRAM.word(0x00-0xFF) = 0xFFFF
InternalRAM.color(0x00-0xFF) = 0xFFFFFFFF
InternalRAM.string(0-10) = "undecided"

Note: When initializing a block of contiguous Internal RAM strings, the initialization string must be 25 characters or less. By default, each Internal RAM string variable holds a maximum of 25 characters and null character.

Initializing special attributes in Internal RAM string variables

When initializing string variables, it is possible to specify special attributes, such as font style, line feeds and upper ASCII characters.

The font styles available are plain, bold, italic, underline and strikethrough. If it is desired to set the font style within the initInternalRAM file, that can be done by using the font style escape sequence **"%s(xxx)"**, where xxx is bit representation of the desired font style. See the table below for the list of font styles and their corresponding bit location. Each font style is represented by a single bit within the font style byte. Multiple font styles can be specified at one time, except in the case of plain, which must stand alone.

Font Style	**Bit Location**
Italic	0x02
Strikethrough	0x04
Bold	0x20
Underline	0x40
Plain	0x80

For example, to initialize InternalRAM string #0 with a string that is formatted to look like this: **"this is bold"** you would use the following:

InternalRAM.string(0) = "this is %s(0x20)bold"

Line feeds are entered in the initInternalRAM file as "\n". For example, to initialize InternalRAM string #1 with a string that is formatted to look like this:

"this is
the break"

You would use the following:

InternalRAM.string(0) = "this is\nthe break"

Reloading the initialized InternalRAM variables on a per page basis

By default, the InternalRAM variables are loaded from the serial data flash into the internal SRAM of the Amulet GEM-compliant Chip only once upon power up. If it is desired to reload the internal SRAM with the InternalRAM variables that are stored in the serial data flash on a per page basis, you can use an Amulet META tag with the **ReloadInternalRAM** attribute to specify this. There are three dot modifiers that will specify the scope of the reloading from serial data flash: **.page**, **.project,** and **.notThisPage**. The dot modifier **.page** will reload the InternalRAM only on the page that the meta is included. By using the dot modifier **.project**, every page in the project will reload the InternalRAM from the serial data flash unless a page has the meta using **ReloadInternalRAM.notThisPage**.

If the InternalRAM is to be reloaded, it happens immediately upon the loading of the page. Any time that page is navigated to, the InternalRAM will be reloaded.
Example usage:

```
<META name="Amulet" content="ReloadInternalRAM.page">
```

Internal RAM specific methods

There are a number of methods that are specific to the Internal RAM variables. All four variable types have their own specific methods. See the tables below for a description of all available methods. If you need more complex control over InternalRAM variables, you can read and write to them from a scripting language called GEMscript. In GEMstudio, go to the Help menu and select GEMscript API for more information on GEMscript.

InternalRAM Byte Method Descriptions

InternalRAM Byte Methods	Descriptions
Amulet:InternalRAM.byte(z).add(x)	Add the byte value x to the Internal RAM byte variable z. Result is stored in Internal RAM byte variable z.
Amulet:InternalRAM.byte(z).and(x)	Logically AND the Internal RAM byte variable z with the byte value x. Result is stored in Internal RAM byte variable z.
Amulet:InternalRAM.byte(z).copyToRamByte(x)	Copy value of Internal RAM byte variable z into Internal RAM byte variable x.
Amulet:InternalRAM.byte(z).dec()	Decrement the value of Internal RAM byte variable z.
Amulet:InternalRAM.byte(z).div(x)	Divide the Internal RAM byte variable z by the byte value x. Result is stored in Internal RAM byte variable z.
Amulet:InternalRAM.byte(z).inc()	Increment the value of Internal RAM byte variable z.
Amulet:InternalRAM.byte(z).maskedValue(x)	Return the value of Internal RAM byte variable z ANDed with the mask x.
Amulet:InternalRAM.byte(z).mul(x)	Multiply the Internal RAM byte variable z by the byte value x. Result is stored in Internal RAM byte variable z.
Amulet:InternalRAM.byte(z).or(x)	Logically OR the Internal RAM byte variable z with the byte value x. Result is stored in Internal RAM byte variable z.

Amulet:InternalRAM.byte(z).setValue(x)	Set the value of Internal RAM byte variable z to the byte value x.
Amulet:InternalRAM.byte(z).sub(x)	Subtract the byte value x from the Internal RAM byte variable z. Result is stored in Internal RAM byte variable z.
Amulet:InternalRAM.byte(z).value()	Return the value of Internal RAM byte variable z.
Amulet:InternalRAM.byte(z).xor(x)	Logically EXCLUSIVE OR the Internal RAM byte variable z with the byte value x. Result is stored in Internal RAM byte variable z.

InternalRAM Word Method Descriptions

InternalRAM Word Methods	Descriptions
Amulet:InternalRAM.word(z).add(x)	Add the word value x to the Internal RAM word variable z. Result is stored in Internal RAM word variable z.
Amulet:InternalRAM.word(z).and(x)	Logically AND the Internal RAM word variable z with the word value x. Result is stored in Internal RAM word variable z.
Amulet:InternalRAM.word(z).calArraySet()	Sets the calibration constants into successive Internal RAM word variables, starting with z. InternalRAM.word(z)= mx128 InternalRAM.word(z+1) = bx InternalRAM.word(z+2) = my128 InternalRAM.word(z+3) = by Used to determine the calibration constants to enter into a touch panel configuration file. See Removing Touchpanel Calibration for more info.
Amulet:InternalRAM.word(z).copyToRamWord(y)	Copy value of Internal RAM word variable z into Internal RAM word variable y.
Amulet:InternalRAM.word(z).dec()	Decrement the value of Internal RAM word variable z.
Amulet:InternalRAM.word(z).div(x)	Divide the Internal RAM word variable z by the word value x. Result is stored in Internal RAM word variable z.
Amulet:InternalRAM.word(z).inc()	Increment the value of Internal RAM word variable z.

Amulet:InternalRAM.word(z).maskedValue(x)	Return the value of Internal RAM word variable z ANDed with the mask x.
Amulet:InternalRAM.word(z).mul(x)	Multiply the Internal RAM word variable z by the word value x. Result is stored in Internal RAM word variable z.
Amulet:InternalRAM.word(z).setValue(x)	Set the value of Internal RAM word variable z to the word value x.
Amulet:InternalRAM.word(z).sub(x)	Subtract the word value x from the Internal RAM word variable z. Result is stored in Internal RAM word variable z.
Amulet:InternalRAM.word(z).value()	Return the value of Internal RAM word variable z.
Amulet:InternalRAM.word(z).xor(x)	Logically EXCLUSIVE OR the Internal RAM word variable z with the word value x. Result is stored in Internal RAM word variable z.

InternalRAM Color Method Descriptions

InternalRAM Color Methods	Descriptions
Amulet:InternalRAM.color(z).set(x)	Sets the Internal RAM color variable z to x using color entry conventions. If using the color variable as a 32-bit integer use a decimal or hex number, but if using as a color variable, use the #rrggbb or #rrggbbaa format.
Amulet:InternalRAM.color(z).setBlue(x)	Sets the Blue Byte of Internal RAM color variable z to x
Amulet:InternalRAM.color(z).setGreen(x)	Sets the Green Byte of Internal RAM color variable z to x
Amulet:InternalRAM.color(z).setRed(x)	Sets the Red Byte of Internal RAM color variable z to x
Amulet:InternalRAM.color(z).value()	Returns the 32-bit value of Internal RAM variable color z

InternalRAM String Method Descriptions

InternalRAM String Methods	Descriptions
Amulet:InternalRAM.string(z).appendChar('y')	Append the single UTF-8 character y to the Internal RAM string variable z. Result is stored in Internal RAM string variable z.

Amulet:InternalRAM.string(z).appendString("y")	Append the string y to the Internal RAM string variable z. Result is stored in Internal RAM string variable z.
Amulet:InternalRAM.string(z).appendToRamString(x)	Append the string stored at Internal RAM string variable z to the Internal RAM string variable x. Result is stored in the Internal RAM string variable pointed to by the Internal RAM byte variable x. It might be easier to visualize it as: InternalRAM.string(InternalRAM.byte(x)) Note, the above is for visualization purposes only, do not attempt to use the above syntax.
Amulet:InternalRAM.string(z).appendViaByteVarPtr(x)	Append the string stored at Internal RAM string variable z into the Internal RAM string variable whose index is the byte value stored at Internal RAM byte x. Result is stored in the Internal RAM string variable pointed to by the Internal RAM byte variable x. It might be easier to visualize it as: InternalRAM.string(InternalRAM.byte(x)) Note, the above is for visualization purposes only, do not attempt to use the above syntax.
Amulet:InternalRAM.string(z).backspace()	Delete last character from the Internal RAM string variable z.
Amulet:InternalRAM.string(z).clear()	Clear out the Internal RAM string variable z.
Amulet:InternalRAM.string(z).copyToRamString(x)	Copy string stored at Internal RAM string variable z into Internal RAM string variable x. Result is stored in Internal RAM string variable x.
Amulet:InternalRAM.string(z).copyViaByteVarPtr(x)	Copy string stored at Internal RAM string variable z into the Internal RAM string variable whose index is the byte value stored at Internal RAM byte x. Result is stored in the Internal RAM string variable pointed to by the Internal RAM byte variable x.

Amulet:InternalRAM.string(z).setChar('y')	Stores the UTF-8 character y into Internal RAM string variable z. setChar() adds the null termination to the character y.
Amulet:InternalRAM.string(z).setValue("y")	Stores the UTF-8 string y into Internal RAM string variable z.
Amulet:InternalRAM.string(z).stringToByte(x)	Converts the ASCII string of a decimal number stored in Internal RAM string variable z to an actual number to be stored in Internal RAM byte variable x.
Amulet:InternalRAM.string(z).stringToWord(x)	Converts the ASCII string of a decimal number stored in Internal RAM string variable z to an actual number to be stored in Internal RAM word variable x.
Amulet:InternalRAM.string(z).value()	Return the null terminated string of Internal RAM string variable z.

Miscellaneous InternalRAM Method Descriptions

InternalRAM Methods	Descriptions
Amulet:InternalRAM.clearRPCBuf()	Clears the internalRAM RPC buffer.
Amulet:InternalRAM.invokeRPC(x)	Adds the RPC x, to the internalRAM RPC buffer.
Amulet:InternalRAM.saveToFlash()	Saves the current state of all the Internal RAM variables (byte, word and string) in the serial data flash. There is a 100,000 max write limit on the life of the serial data flash.

Amulet Widgets

Widgets are broken up into three categories:

1. Basic Objects do not have any touchpanel input nor do they provide any dynamically configurable output.
2. Control widgets take user inpup from the touchpanel and invoke functions to control other actions.
3. View widgets provide visualization of data

Object

The basic object consists of three types of objects. They have limited or no methods

Image

Although they cannot represent data like other View Widgets can, Images do have internal methods that can be invoked by other Control Widgets/Objects. See Inter-Widget Communications for the list of available methods for each object/widget.

Image Parameter Attributes:

Parameter="file" value="image" — Specifies the image to use. See Image Formats for supported image types.

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Image is to start out invisible. If the Image starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="cacheImage" value="CHECKED" or "UNCHECKED" — Specifies if the Image is loaded into project-specific SDRAM in an uncompressed format for immediate use from any page within the project. All cached images are loaded into SDRAM prior to the first page being displayed, so the more images that are cached, the longer it will take for the initial page to be displayed. Page-to-page transition times can be dramatically reduced by caching images.

Optional Image Parameter Attributes:

Parameter="jpegOverride" value="drive:/folder(s)/file" — Specifies the JPEG image that will take precedence over the image specified in the **file** attribute. **drive:** specifies the flash drive, where 0: is the onboard eMMC and 1: is the SD card slot. **drive:** is optional and defaults to 0:, the onboard eMMC, if not specified. **/folder(s)** specifies the location of the file within the given drive. If **/folder(s)** is not specified the default drive is 0:/Amulet_msd/images. If the initial / is not present then the folder will be relative to 0:/Amulet_msd/images, but if the initial / is present, the folder will be relative to the root of 0:/. **file** is the name of the JPEG image. The **file** need not have the .jpeg or .jpg extension, but the file must be a valid jpeg image that has the exact same dimensions as the image specified in the **file** attribute. For examples, "0:/Amulet_msd/images/new.jpg" is equal to "/Amulet_msd/images/new.jpg" which is equal to "new.jpg". Only valid in the High Performance Module.

Parameter="noSDRAM" value="CHECKED" or "UNCHECKED" — Specifies if the image(s) are NOT loaded into page-specific SDRAM prior to drawing the object. Has no effect if the images are already in the global cache. Having a copy of these images in the SDRAM speeds up the transition between the image and the invisible state. Default is UNCHECKED.

Animated Image

Animated Images have internal methods that can be invoked by other Control Widgets/Objects. See Inter-Widget Communications for the list of available methods for each object/widget.

Animated Image Parameter Attributes:

Parameter="file" value="image" — Specifies the animation to use. Image type can be animated .GIF.

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Animated Image is to start out invisible. If the Animated Image starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="cacheImage" value="CHECKED" or "UNCHECKED" — Specifies if the Animated Image is loaded into project-specific SDRAM in an uncompressed format for immediate use from any page within the project. All cached images are loaded into SDRAM prior to the first page being displayed, so the more images that are cached, the longer it will take for the initial page to be displayed. Page-to-page transition times can be dramatically reduced by caching images.

Parameter="noSDRAM" value="CHECKED" or "UNCHECKED" — Specifies if the image(s) are NOT loaded into page-specific SDRAM prior to drawing the object. Has no effect if the images are already in the global cache. Having a copy of these images in the SDRAM speeds up the transition between the images and the invisible state. Default is UNCHECKED.

Static Text

Static Text is an object that enables the user to write text on the display. Unlike the String Field widget, the Static Text object does not invoke any function(s).

Static Text Parameter Attributes:

Parameter="text" value="string" — Specifies the text to be displayed. See Using Amulet Formatted Text for more formatting options.

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the text is to start out invisible or not. If the attribute is not CHECKED, then by default the text is visible. If the text starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="fillColor" value= See color entry conventions — Specifies the desired background color. See section on colors for more information. If no fill color is specified, the default color is the current background color.

Parameter="font" value="font, font size" — Specifies the font and font size used for the text. The corresponding .auf/.aauf file must be included in the Amulet/Color/Configuration/Fonts folder, the project root folder, or the project root/Fonts folder. See Amulet GEM Font Converter for more information regarding the creation of .auf/.aauf files. Default is Bitstream Vera Sans. The default font size for the text is 12pt.

Parameter="fontColor" value= See color entry conventions — Specifies the text color. See section on colors for more information. If no font color is specified, the default color is black.

Parameter="fontStyle" value="BOLD" or "ITALIC" or "PLAIN" — Specifies the style associated with the text font. The available font styles are:

- BOLD — The option text is bold. (i.e. text)
- ITALIC — The option text is italicized. (i.e. text)
- PLAIN — The option text is standard font. (i.e. text)

This attribute defines the default font style of the static text defined in the text attribute.

Optional Static Text Parameter Attributes:

Parameter="border" value="number" — Specifies width, in pixels, of the border around the dimensions of the text. Default is 0, meaning no border.

Parameter="borderColor" value= See color entry conventions — Specifies the desired text border color. See section on colors for more information. If no border color is specified, the default color is black. Only applicable if border has a value of 1 or greater.

Parameter="context" value="String"— Specifies the context of the text string for translation. No context is not the same as an empty context. See Context under the Localization feature for more info.

Parameter="colorInvert" value="REGION" or "STRING" or "NONE" — Specifies if the string is shown with alternate colors. If REGION selected, the fillColorAlt and fontColorAlt will be used instead of fillColor and fontColor. If STRING selected, only fontColorAlt will be used instead of fontColor. If NONE selected, fillColor and fontColor will be used. Only one value is allowed; you cannot mix color inversion properties. Default is NONE.

Parameter="fillColorAlt" value= See color entry conventions — Specifies the desired background color if colorInvert set to REGION. See section on colors for more information. If no alternate fill color is specified, the default alternate fill color is the logical inverse of the current fillColor.

Parameter="fontColorAlt" value= See color entry conventions — Specifies the desired font color if colorInvert set to either REGION or STRING. See section on colors for more information. If fontColorAlt is not specified, the default alternate font color is the logical inverse of the current fontColor.

Control Widgets

Control Widgets enable the user to interact the GUI through the touchpanel

CheckBox

A Check Box is a labeled, usually square region used to invoke a function (or set of functions) whose argument is the value of the Check Box. To toggle (check or uncheck) a Check Box, click on the Check Box (or label). Check Boxes can also be grouped to invoke functions whose argument is the cumulative value of all checked Check Boxes (logical ORing). Check Boxes that have the same groupName are considered part of a Check Box group. Within a Check Box group, any combination of Check Boxes can be checked. If none of the Check Boxes are set within a group, then 0x00 is the href function call argument. Therefore, you must give each Check Box a checked value that is different on a binary bit level. If one Check Box has a checked value of 0x01, then another Check Box could have a value of 0x02, and the next a value of 0x04 (not 0x03 because that would be the ORed value of Check Boxes 1 and 2 when checked).

Checkbox Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Check Box is to start out invisible. If the Check Box starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="boxAlign" value="LEFT" or "RIGHT" — Specifies the location of the Check Box in relation to the label text.

Parameter="checkedValue" value="number" — Specifies the Check Box value if checked. If the Check Box is part of a group, single bits should be used, such as 0x01, 0x02, 0x04 etc. All boxes within a Check Box group must be assigned unique binary numbers. That is, if one Check Box has a value of 0x01(bit 0), no other Check Box within the group can use bit 0. If the box is part of a group, the range is 1 - 255 (0x01 - 0xFF). If a lone Check Box, the checkedValue can be a BYTE, WORD or STRING. See note regarding Control Widget intrinsic values.

Parameter="font" value="font, font size" — Specifies the font and font size used for the Check Box label. See Amulet GEM Font Converter for more information regarding the creation of .auf font files. Default font is Bitstream Vera Sans. Default font size is 12pt.

Parameter="fontColor" value= See color entry conventions — Specifies the desired font color. See section on colors for more information. If no font color is specified, the default color is black.

Parameter="fontStyle" value="BOLD" or "ITALIC" or "PLAIN" — Specifies the style associated with the Check Box label font. The available font styles are:

- BOLD — The option text is bold. (i.e. text)
- ITALIC — The option text is italicized. (i.e. text)
- PLAIN — The option text is standard font. (i.e. text)

Parameter="groupName" value="text" — For a grouped Check Box, specifies the Check Box group assignment. Attribute not needed if a lone Check Box.

Parameter="href" value="function(s)" — The function (or multiple/sequenced functions) invoked when a Check Box is toggled (checked or unchecked). If the Check Box is NOT part of a Check Box group, the argument of the function is determined by the state of the Check Box. If checked, the argument is determined by the checkedValue parameter;

if unchecked, the argument is determined by the unCheckedvalue parameter. If the Check Box is part of a group, the argument is the cumulative value of all "checked" boxes (logical ORing). All Check Boxes within a group must call the same href function(s). See Appendix B for all available href functions for the Check Box widget.

Parameter="initialCondition" value="ON" or "OFF" or "FromInitHref" — Specifies the initial condition of a Check Box when the page is loaded; ON specifies a checked Check Box; OFF an unchecked Check Box. If FromInitHref is selected, the function specified by the initHref attribute is called. The returned value determines the initial condition of all Check Boxes within the group. If part of a group, a Check Box is checked when its checkedValue corresponds to a set bit in the returned byte (logical ANDing). If a lone Check Box, the Check Box is checked when its internalNumber equals the returned initHref value.

Parameter="label" value="text" — Specifies the name that appears to the right or left of the Check Box. See Using Amulet Formatted Text for more formatting options.

Parameter="cacheImage" value="CHECKED" or "UNCHECKED" — Specifies if all three Check Box images are loaded into project-specific SDRAM in an uncompressed format for immediate use from any page within the project. All cached images are loaded into SDRAM prior to the first page being displayed, so the more images that are cached, the longer it will take for the initial page to be displayed. Page-to-page transition times can be dramatically reduced by caching images.

Optional Checkbox Parameter Attributes:

Parameter="noSDRAM" value="CHECKED" or "UNCHECKED" — Specifies if the image(s) are NOT loaded into page-specific SDRAM prior to drawing the object. Has no effect if the images are already in the global cache. Having a copy of these images in the SDRAM speeds up the transition between the images and the invisible state. Default is UNCHECKED.

Parameter="context" value="String"— Specifies the context of the label string for translation. No context is not the same as an empty context. See Context under the Localization feature for more info.

Parameter="unCheckedValue" value="number" — Specifies the unchecked Check Box value. This attribute is only valid when NOT part of a Check Box group. The checkedValue can be a BYTE, WORD or STRING. See note regarding Control Widget intrinsic values.

Parameter="trackingImage" value="image" — Specifies the image to use when the Check Box is in the pen down state. If this attribute is not present, then a default image, a black and white square with a check in it, is used. See Image Formats for supported image types.

Parameter="emptyImage" value="image" — Specifies the image to use when the Check Box is in the unchecked state. If this attribute is not present, then a default image, an empty black and white square, is used. See Image Formats for supported image types.

Parameter="fullImage" value="image" — Specifies the image to use when the Check Box is in the checked state. If this attribute is not present, then a default image, a grayed out block and white square, is used. See Image Formats for supported image types.

Parameter="trackingImage" value="image" — Specifies the image to use when the Check Box is in the pen down state. If this attribute is not present, then a default image, a black and white square with a check in it, is used. See Image Formats for supported image types.

Parameter="hrefOnHitOnly" value="CHECKED" or "UNCHECKED" — Specifies if the Check Box href function(s) is launched only upon a "hit". By default, hrefOnHitOnly is UNCHECKED, which means if the Check Box has an initHref function, the href of the check box (group or lone) is launched upon loading the page. The href is also launched after a forceUpdate() call. By setting hrefOnHitOnly to CHECKED, the href will only be launched upon the Check Box physically being selected or upon a forceHit() call.

Parameter="initHref" value="function" — Specifies the function called when the page is loaded. Use this attribute when FromInitHref is used as the initialCondition. All Check Boxes within a group must have the same function(s). If part of a group, the function must return a BYTE value, and the value returned is compared to the checkedValue. If the checkedValue bit is set in the returned value, the Check Box will initially start out in the "on" position. If a lone Check Box, then the returned value is compared to the internalNumber and if equal, the lone Check Box initially starts out in the "on" position. See Appendix B for all available functions.

Parameter="internalNumber" value="number" — Specifies the internal number of the lone Check Box, which is used to determine if the initial condition of the lone Check Box is on or off, if FromInitHref is used as the initialCondition. Should only be used if using initHref. If the value returned from the initHref function matches the internalNumber of the lone Check Box, then the Check Box starts out in the "on" position. If the internalNumber is not specified, the checkedValue is used instead.

Parameter="updateDelay" value ="number" — The number specifies the delay time from when the page is loaded until the first initHref function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to 0.01 (immediately upon page load).

CustomButton

The Custom Button Widget uses two images (a pressed and a not pressed) to create a Custom Button. Custom Buttons are Amulet anchors that allow a separate pressed image to appear instead of merely inverting the image. Custom Buttons can be set to be either a "spring-loaded" or a "toggle" button. By default, when hit, a Custom Button invokes a function (or set of functions). Optionally, Custom Buttons can be set to auto-repeat while pressed. Initial delay and repeat frequency can both be customized.
Each Custom Button can have a user-defined label (text or numeric) within the button image. If the label is specified as "fromInitHref", the label will be based upon a string variable that is passed from the initHref function at run-time. The label text will automatically wrap if the string exceeds the width of the Custom Button. User-defined wraps can be specified by entering "\n" at the point of the desired wrap.

Custom Buttons can also be set up to auto-repeat. When pressed, an auto-repeat Custom Button delays a user-defined amount of time then invokes a function (or set of functions) at a user-defined frequency while the Custom Button is still being pressed. As a side benefit of the auto-repeat functionality, a Custom Button can be created that will appear to invoke its function(s) immediately upon being pressed instead of waiting until the Custom Button is released. To create an instant hit Custom Button, set the delay very small and the frequency at 0. The Custom Button will invoke its function(s) after the very short delay time and not repeat again.
NOTE: To display a literal \ symbol in the label, use a double backslash in the string (e.g. 25 \\ 5 would display 25 \ 5 within the button).

Custom Button Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Custom Button is to start out invisible. If the Custom Button starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="buttonType" value="TOGGLE" or "SPRING-LOADED" — Specifies the action of the Custom Button when hit. TOGGLE causes the Custom Button to depress (or invert) on a pen down event and stay depressed on the ensuing pen up event. SPRING-LOADED causes the Custom Button to depress on a pen down event and return to its original state on the following pen up event. SPRING-LOADED is the default.

Parameter="downImage" value="image" — Image used when the Custom Button is pressed. See Image Formats for supported image types.

Parameter="font" value="font, font size" — Specifies the font and font size used for the Custom Button label. See Amulet GEM Font Converter for more information regarding the creation of .auf font files. Default font is Bitstream Vera Sans. The default font size is 12pt.

Parameter="fontColor" value= See color entry conventions — Specifies the desired font color. See section on colors for more information. If no font color is specified, the default color is black.

Parameter="fontStyle" value= "PLAIN" or "BOLD" or "ITALIC" — Specifies the style associated with the Custom Button label font. The available font styles are:

- PLAIN — The option text is standard font. (i.e. text)
- BOLD — The option text is bold. (i.e. text)
- ITALIC — The option text is italicized. (i.e. text)

Parameter="href" value="function(s)" — The function (or multiple/sequenced functions) invoked when the Custom Button is hit. See Appendix B for all available functions for the Custom Button widget.

Parameter="onButtonPress" value="ALPHA" or "CUSTOM" or "DEPRESS" — Specifies the look of the Custom Button during a pen down condition. CUSTOM causes the downImage to appear and label text, if any, to shift down and to the right to give the illusion of being pressed. ALPHA blends a transparent color specified by alphaColor with the upImage and prohibits the downImage from appearing. DEPRESS gives the illusion of the Custom Button being pressed using only the upImage and the downImage will not appear. CUSTOM is the default.

Parameter="upImage" value="image" — Image used when Custom Button is not pressed. See Image Formats for supported image types.

Parameter="cacheImage" value="CHECKED" or "UNCHECKED" — Specifies if both Custom Button images are loaded into project-specific SDRAM in an uncompressed format for immediate use from any page within the project. All cached images are loaded into SDRAM prior to the first page being displayed, so the more images that are cached, the longer it will take for the initial page to be displayed. Page-to-page transition times can be dramatically reduced by caching images.

Optional Custom Button Parameter Attributes:

Parameter="alphaColor" value= See color entry conventions — If onButtonPress equals ALPHA, this attribute specifies the alpha color mask used to notify the user the Custom Button has been depressed. See section on colors for more information. If no alpha color is specified, the default color is a transparent gray. If the color specified does not have an alpha component, then it will be completely opaque, which will result in a colored rectangle appearing in place of the Custom Button while it is touched.

Parameter="executeOn" value= "HIT" or "RELEASE" or "BOTH" — Specifies when the href function is launched, either when the Custom Button is hit, when the Custom Button is released, or when the Custom Button is both hit and released. If nothing is specified, the default is to "RELEASE".

Parameter="horizontalAlign" value="LEFT" or "CENTER" or "RIGHT" — Specifies the horizontal alignment of the string associated with the label attribute within the Custom Button dimensions. Only one value is allowed; you cannot mix horizontal alignments. Default is CENTER.

Parameter="initHref" value="function" — Specifies the function called when the page is loaded. Use this attribute only when FromInitHref is used as the label. Only available function is of type Amulet:UART.label(x).value(). See note regarding the use of InternalRAM label variables as Custom Button labels.

Parameter="label" value="text" or "FromInitHref" — Specifies the text that appears inside the Custom Button. See Using Amulet Formatted Text for more formatting options. The Custom Button will NOT automatically re-size to fit the text. If there is enough vertical room, text will automatically wrap. Any text that will not fit within the confines of the Custom Button will be truncated. User-defined wraps can be specified by entering "\n" within the text at the spot you would like the wrap to occur. There is a maximum of 5 lines for a Custom Button label. The name field can be left blank; blank is the default. To have the label be dynamically entered at runtime by the server, enter FromInitHref. By default, the dynamic Custom Button label can be a maximum of 25 bytes in length. To increase the maximum number of bytes, put the desired number in parentheses after FromInitHref. For example, to have a dynamic label up to 50 bytes long, use FromInitHref(50). The attribute initHref needs to be of the type Amulet:InternalRAM.label(x).value() or Amulet:

(UART or USB).label(x).value(). It will be called only once upon the loading of the page, with the string returned from the server becoming the Custom Button label. See note regarding the use of InternalRAM label variables as Custom Button labels.

Parameter="noSDRAM" value="CHECKED" or "UNCHECKED" — Specifies if the image(s) are NOT loaded into page-specific SDRAM prior to drawing the object. Has no effect if the images are already in the global cache. Having a copy of these images in the SDRAM speeds up the transition between the images and the invisible state. Default is UNCHECKED.

Parameter="context" value="String"— Specifies the context of the label string for translation. No context is not the same as an empty context. See Context under the Localization feature for more info.

Parameter="repeatDelay" value="number" — Time to delay from when the Custom Button is initially pressed until it starts to auto-repeat. Specified in seconds, with a single floating-point number. The range is 0.01 - 655.35.

Parameter="repeatRate" value="number" — The href function call frequency while the Custom Button is being pressed, after the initial delay determined by repeatDelay. Specified in seconds, with a single floating-point number. The range is 0.00 - 655.35. 0.00 means do not repeat.

Parameter="verticalAlign" value="TOP" or "MIDDLE" or "BOTTOM" — Specifies the vertical alignment of the string associated with the label attribute within the Custom Button dimensions. Only one value is allowed; you cannot mix vertical alignments. Default is MIDDLE.

Parameter="updateDelay" value ="number" — The number specifies the delay time from when the page is loaded until the first initHref function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to 0.01 (immediately upon page load).

CustomSlider

The Custom Slider Widget acts like a regular Slider Widget, except for the fact that you get to specify the channel image and two different handle images. The Custom Slider Widget invokes a function (or set of functions) either upon change or release, depending upon the hrefEvent parameter. If hrefEvent is setup as on Change, the Custom Slider invokes the function(s) whenever the handle moves. If hrefEvent is setup as on Release , the Custom Slider invokes the function(s) only upon releasing the handle. Touching anywhere on the channel causes the handle to snap to that location and invoke the function(s) with the corresponding argument. The argument is determined by the location of the handle and the limits setup with the min and max attributes. The location of the min value is determined by the minAt attribute(left, right, top or bottom). The height and width dimensions determines whether the Custom Slider is horizontal or vertical. The longer dimension establishes the path that the slider handle travels. An image, channelImage, is used as the slider's channel, which must be the same dimensions as the Custom Slider Widget. Two different images are used for the slider handle. One image, handleImage, is used when the handle is not touched, and another, handleTrackingImage, for when the handle is active. As an option, you can specify the offset of the handle from the center of the channel by using the handleOffset parameter.

It is possible to use a handleTrackingImage that contains transparent components. If the handleTrackingImage does not have transparency components, then a single copy of the handleTrackingImage will be moved along the channel and the transparency components will not be updated. If the image has transparency components, the handleTrackingImage will be forced to be redrawn at each new location. This will result in a slightly slower update of the tracking handle.

Custom Slider Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Custom Slider is to start out invisible or not. If the attribute is not CHECKED, then by default the Custom Slider is visible. If the Custom Slider starts out invisible, the only way to make it visible again is via the IWC method reappear(). If invisible not specified, the default is UNCHECKED.

Parameter="channelImage" value="image" — Image used as the background over which the Custom Slider handle travels. Image dimensions MUST be exactly the same as the dimensions of the Custom Slider Widget. See Image Formats for supported image types.

Parameter="handleImage" value="image" — Image used as the Custom Slider handle when not pressed. See Image Formats for supported image types.

Parameter="handleTrackingImage" value="image" — Image used as the Custom Slider handle when pressed. See Image Formats for supported image types.

Parameter="href" value="function(s)" — The function (or multiple/sequenced functions) invoked upon the event specified in hrefEvent. See Appendix B for all available functions for the Custom Slider widget.

Parameter="hrefEvent" value="On Change" or "On Release" — The event which triggers the launching of the href function. If set to On Change, the Custom Slider will launch the function call whenever the handle is moved. If set to On Release, the Custom Slider will only launch the function call upon the release of the Custom Slider handle. If hrefEvent is not specified, the default is On Change.

Parameter="initialCondition" value="number" or "FromInitHref" — Specifies handle position when the page is loaded. The position value becomes the argument given to the href function(s). If FromInitHref is selected, the function specified by the InitHref attribute is called. The returned value determines the handle position. [The range is 0 - 65535 (0x00 - 0xFFFF)]

Parameter="max" value="number" — The maximum value used as the argument given to the function(s) specified in href. By default, maximum value is achieved when handle is full-right on a horizontal slider, or full-top on a vertical slider. The range is 1 - 65535 (0x01 - 0xFFFF).

Parameter="min" value="number" — The minimum value used as the argument given to the function(s) specified in href. By default, minimum value is achieved when handle is full-left on a horizontal slider, or full-bottom on a vertical slider. The range is 0 - 65534 (0x00 - 0xFFFE).

Parameter="minAt" value=""LEFT" or "RIGHT" or "TOP" or "BOTTOM"" — Determines where the minimum value of the Custom Slider is located. As the handle sweeps from the minAt location, the value increases until the maximum value is reached at the opposite extreme of the minAt location. Default values are LEFT for horizontal sliders and BOTTOM for vertical sliders. The options are:

- LEFT — The slider value increases from left to right. (horizontal slider only)
- RIGHT — The slider value increases from right to left. (horizontal slider only)
- TOP — The slider value increases from top to bottom. (vertical slider only)
- BOTTOM — The slider value increases from bottom to top. (vertical slider only)

Optional Custom Slider Parameter Attributes:

Parameter="handleOffset" value="number" — Specifies the number of pixels from the center of the channel the handle is located. If a vertical slider, positive numbers shift the handle to the right and negative numbers shift it to the left. If a horizontal slider, positive numbers shift the handle to the bottom and negative numbers shift it to the top. [The range is -100 through 100]

Parameter="hrefOnHitOnly" value="CHECKED" or "UNCHECKED" — Specifies if the Custom Slider href function(s) is launched only upon a "hit" or not. By default, hrefOnHitOnly is UNCHECKED, which means the Custom Slider will launch its href upon loading the page. The href is also launched after a forceUpdate() call. By setting hrefOnHitOnly to CHECKED, the href will only be launched upon the Custom Slider physically being selected or upon a forceHit() call.

Parameter="initHref" value="function" — Specifies the function called when the page is loaded. Use this attribute whenever FromInitHref is used as the initialCondition. The value returned from this function call will be used as the initial condition of the Custom Slider handle. See Appendix B for all available functions.

Parameter="updateDelay" value ="number" — The number specifies the delay time from when the page is loaded until the first initHref function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to 0.01 (immediately upon page load).

Parameter="waitForInit" value="CHECKED" or "UNCHECKED" — Only valid if FromInitHref is used as the initialCondition. Specifies if the Custom Slider handle will wait for valid data before being displayed on the channel. If CHECKED, the Custom Slider handle will not display until the data from the initHref function is received. If UNCHECKED, or the attribute is not present, the Custom Slider handle momentarily starts at the minAt location until the initHref function receives its data. If waitForInit not specified, the default is UNCHECKED.

FunctionButton

Function Button Widget

Function buttons are Amulet anchors that provide either a "spring-loaded" or a "toggle" button image, depending upon the button type, to invoke a function (or set of functions) when hit. By default, Function Button images appear to depress when touched. You can also setup the button image to "shade" when touched with the alpha color. By default, when hit, a button invokes a function (or set of functions). Optionally, Function Buttons can be set to auto-repeat while pressed. Initial delay and repeat frequency can both be customized.

Each Function Button can have a user-defined label (text or numeric) within the button image. If the label is specified as "fromInitHref", the label will be based upon a string variable that is passed from the initHref function at run-time. The label text will automatically wrap if the string exceeds the width of the Function Button. User-defined wraps can be specified by entering "\n" at the point of the desired wrap.
Function Buttons can also be set up to auto-repeat. When pressed, an auto-repeat button delays a user-defined amount of time then invokes a function (or set of functions) at a user-defined frequency while the button is still being pressed. As a side benefit of the auto-repeat functionality, a Function Button can be created that will appear to invoke its function(s) immediately upon being pressed instead of waiting until the button is released. By setting the delay very small and the frequency at 0, the button will invoke its function(s) after the very short delay time and not repeat again. NOTE: To display a literal \ symbol in the label, use a double backslash in the string (e.g. 25 \\ 5 would display 25 \ 5 in the button).

Function Button Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Function Button is to start out invisible or not. If the attribute is not present, then by default the Function Button is visible. If the Function Button starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="buttonType" value="TOGGLE" or "SPRING-LOADED" — Specifies the action of the Function Button when hit. TOGGLE causes the button to depress (or invert) on a pen down event and stay depressed on the ensuing pen up event. SPRING-LOADED causes the button to depress on a pen down event and return to its original state on the following pen up event. SPRING-LOADED is the default.

Parameter="onButtonPress" value="DEPRESS" or "ALPHA" — Specifies the look of the Function Button during a pen down condition. DEPRESS gives the illusion of the button being pressed. ALPHA shades the image of the button with a transparent color defined by alphaColor. DEPRESS is the default.

Parameter="href" value="function(s)" —The function (or multiple/sequenced functions) invoked when the Function Button is hit. See Appendix B for all available functions for the Function Button widget.

Parameter="label" value="text" or "FromInitHref" — Specifies the text that appears inside the Function Button. See Using Amulet Formatted Text for more formatting options. The button will NOT automatically re-size to fit the text. If there is enough vertical room, text will automatically wrap. Any text that will not fit within the confines of the button will be truncated. User-defined wraps can be specified by entering "\n" within the text at the spot you would like the wrap to occur. There is a maximum of 5 lines for a button label. The name field can be left blank; blank is the default. To have the label be dynamically entered at runtime by the server, enter FromInitHref. By default, the dynamic button label can be a maximum of 25 characters in length. To increase the maximum number of characters, put the desired number in parentheses after FromInitHref. For example, to have a dynamic label up to 50 characters long, use FromInitHref(50). The attribute initHref needs to be of the type Amulet:UART.label(x).value(). It will be called only once upon the loading of the page, with the string returned from the server becoming the button label. See note regarding the use of InternalRAM label variables as button labels.

Optional Function Button Parameter Attributes:

Parameter="context" value="String"— Specifies the context of the label string for translation. No context is not the same as an empty context. See Context under the Localization feature for more info.

Parameter="borderColor" value= See color entry conventions — Specifies the desired Function Button border color. See section on colors for more information. If no border color is specified, the default color is black.

Parameter="fillColor" value= See color entry conventions — Specifies the desired Function Button fill color. See section on colors for more information. If no fill color is specified, the default color is the current background color.

Parameter="font" value="font, font size" — Specifies the font used for the Function Button label. See Amulet GEM Font Converter for more information regarding the creation of .auf font files. Default is Bitstream Vera Sans. The font size for the Function Button label defaults to 12pt.

Parameter="fontColor" value= See color entry conventions — Specifies the desired font color. See section on colors for more information. If no font color is specified, the default color is black.

Parameter="fontStyle" value="Plain" or "Bold" or "Italic"— Specifies the style associated with the button label font. The available font styles are:

- Bold — The option text is bold. (i.e. text)
- Italic — The option text is italicized. (i.e. text)
- Plain — The option text is standard font. (i.e. text)

Parameter="alphaColor" value= See color entry conventions — If onButtonPress equals ALPHA, this attribute specifies the alpha color mask used to notify the user the Function Button has been depressed. See section on colors for more information. If no alphaColor is specified, the default color is a transparent gray. If the color specified does not have an alpha component, then it will be completely opaque, which will result in a colored rectangle appearing in place of the button while it is touched.

Parameter="executeOn" value= "HIT" or "RELEASE" or "BOTH" — Specifies when the href function is launched, either when the button is hit, when the button is released, or when the button is both hit and released. If nothing is specified, the default is to "RELEASE".

Parameter="horizontalAlign" value="LEFT" or "CENTER" or "RIGHT" — Specifies the horizontal alignment of the string associated with the label attribute within the Function Button dimensions. Only one value is allowed; you cannot mix horizontal alignments. Default is CENTER.

Parameter="initHref" value="function" — Specifies the function called when the page is loaded. Use this attribute only when FromInitHref is used as the label. The string value returned from the function call will be used as the button label. Only available function is of type Amulet:UART.label(x).value(). See note regarding the use of InternalRAM label variables as button labels.

Parameter="intrinsicValue" value="text" or "number" — Specifies the intrinsic value of the button, which can be passed to functions in the button's own Href or queried by other objects who request this button's intrinsic value. The value can be a UTF-8 encoded string or a number from 0-65535.

Parameter="repeatDelay" value="number" — Time to delay from when teh Function Button is initially pressed until it starts to auto-repeat. Specified in seconds, with a single floating-point number. The range is 0.00 - 655.35.

Parameter="repeatRate" value="number" — The href function call frequency while the Function Button is being pressed, after the initial delay determined by repeatDelay. Specified in seconds, with a single floating-point number. The range is 0.00 - 655.35. 0.00 means do not repeat.

Parameter="updateDelay" value ="number" — The number specifies the delay time from when the page is loaded until the first initHref function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to 0.01 (immediately upon page load).

Parameter="verticalAlign" value="TOP" or "MIDDLE" or "BOTTOM" — Specifies the vertical alignment of the string associated with the label attribute within the Function Button dimensions. Only one value is allowed; you cannot mix vertical alignments. Default is MIDDLE.

ImageScroller

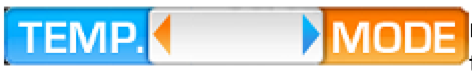


Image Scroller Widget

The Image Scroller Widget is a versatile Control Widget that can be used in many ways depending upon how the attributes are set. The Image Scroller takes either a single image or assembles an array of images into one long image that can be scrolled through either vertically or horizontally. The scrolling characteristics can be tuned so it appears to have inertia when released or it can stop scrolling once the finger has been removed from the touch panel. The number of stops within the image is determined by the numberOfSelections attribute. After scrolling and releasing the Image Scroller, it will invokes the href function(s) with the value of the "stop" being the argument. The value of the "stop" is determined by the min, max and the numberOfSelections attributes.

The Image Scroller can be used as a full-page image slideshow that allows you to scroll from one page to the other, either vertically or horizontally. The Image Scroller can also be used to create a list of selectable image objects that share a common function (or set of functions). By specifying an initial condition, that "stop" is centered within the viewport of the Image Scroller when the page is loaded, the href function(s) are invoked and the value associated with that "stop" is used as the argument. The Image Scroller can also be used to toggle between two sides of an image by setting the allowWrap attribute to Toggle.

If a vertical orientation (meaning the height of the image is longer than the width) is selected, GEMstudio will not allow the width of the Image Scroller to be adjusted less than the width of the scroller image. Changing the height of the Image Scroller is changing the height of the visible viewport. Only the portion of the Image Scroller image that is currently within the viewport will actually be visible. Conversely, if a horizontal orientation (meaning the width of the image is longer than the height) is selected, GEMstudio will not allow the height of the Image Scroller to be adjusted less than the height of the scroller image.

An optional overlayImage can be specified that sits on top of the visible viewport. Since the Image Scroller image is underneath the overlayImage, in order to see the ImageScroller image, the overlayImage must have some transparent regions, thus the overlayImage should be a .png or .gif with transparency. If using a vertical Image Scroller, the overlayImage height will be the same height as the viewport, but the overlayImage width can be smaller or larger than the viewport width. The overlayImage will be centered horizontally within the viewport of a vertical Image Scroller. Conversely, if using a horizontal Image Scroller ,the overlayImage width will be the same width as the viewport, but the overlayImage height can be smaller or larger than the viewport height. The overlayImage will be centered vertically within the viewport of a horizontal Image Scroller.

Note: For best performance, the number of pixels in the height of a vertical scrollerImage should be divisible by the number of stops as determined by numberOfSelections. For instance, if it is desired to have 8 stops along a vertical scrollerImage, the height of the scrollerImage should be divisible by 8. The same holds true for a horizontal scrollerImage. The number of pixels in the width of a horizontal scrollerImage should be divisible by the number of stops as determined by numberOfSelections.

Image Scroller Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Image Scroller is to start out invisible. If the Image Scroller starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="allowWrap" value= "No Wrap" or "Toggle" or "Wrap" — Specifies the wrapping capability of the Image Scroller. No Wrap specifies the Image Scroller will not scroll beyond the bounds of the image. Wrap specifies the Image Scroller will scroll beyond the bounds of the image, wrapping around to the opposite end of the image seamlessly. Toggle is used if numberOfSelections is 2 and it is desired to have a snap-to movement between the two stops. If toggle used, merely touching the ImageScroller will toggle between the 2 stops, no scrolling required.

Parameter="cacheImage" value="CHECKED" or "UNCHECKED" — Specifies if the Image Scroller image is loaded into project-specific SDRAM in an uncompressed format for immediate use from any page within the project. All cached images are loaded into SDRAM prior to the first page being displayed, so the more images that are cached, the longer it will take for the initial page to be displayed. Page-to-page transition times can be dramatically reduced by caching images.

Parameter="granularity" value="number" — Specifies the minimum number of pixels that must be scrolled before the image moves. The image will move by the same number of pixels as scrolled, so this number is the smallest increment that can be scrolled. The smaller the granularity, the smoother the scroll. 1 gives the smoothest scroll, but depending upon the size of the Image Scroller image, could be very slow. The default is 10. The range is 1 - 255 (0x01-0xFF).

Parameter="href" value="function(s)" — The function (or multiple/sequenced functions) invoked when the Image Scroller has stopped scrolling and the finger is released. See Appendix B for all available functions for the Image Scroller widget.

Parameter="incWhenScrolling" value= "Down" or "Left" or "Right" or "Up" — Specifies which scrolling direction increments the intrinsic value of the Image Scroller. Vertical Image Scrollers should use Down or Up while Horizontal Image Scrollers should use Left or Right.

Parameter="inertiaDuration" value="number" — Specifies the relative duration the Image Scroller will continue to scroll once the image is no longer being touched. The absolute duration of the inertia is based upon the inertiaDuration, the inertiaSpeed and the speed of the scroll at the time the finger was released from the touch panel. If doing full page image transitions, it is usually best to limit the inertiaDuration to 0. If scrolling through a list of image stops then 5 is a good place to start. If allowWrap is set to Toggle, inertiaDuration is ignored. The range is 0 - 255 (0x00-0xFF).

Parameter="inertiaSpeed" value="number" — Specifies the relative speed the Image Scroller will continue to scroll once the image is no longer being touched. The absolute speed of the inertia is based upon the inertiaSpeed and the speed of the scroll at the time the finger was released from the touch panel. The default is 7, which is a good place to start. If allowWrap is set to Toggle, inertiaSpeed is ignored. The range is 0 - 255 (0x00-0xFF).

Parameter="initialCondition" value="string" or "FromInitHref" — Specifies the starting stop of the Image Scroller when the page is loaded. The value associated with the relative location on the image selected is the argument given to the href function(s). If "FromInitHref" is selected, the function specified by the InitHref attribute is called. Whichever stop has the same value as the returned value from the initHref function will be centered within the Image Scroller viewport.

Parameter="max" value="number" — The maximum value used as the argument given to the function(s) specified in href. On horizontal Image Scrollers that have the incWhenScrolling attribute set to Right, the maximum value is achieved when scrolled fully to the right. On horizontal Image Scrollers that have the incWhenScrolling attribute set to Left, the maximum value is achieved when scrolled fully to the left. On vertical Image Scrollers that have the incWhenScrolling attribute set to Down, the maximum value is achieved when scrolled fully to the bottom. On vertical Image Scrollers that have the incWhenScrolling attribute set to Up, the maximum value is achieved when scrolled fully to the top. The range is 1 - 65535 (0x01 - 0xFFFF).

Parameter="min" value="number" — The minimum value used as the argument given to the function(s) specified in href. On horizontal Image Scrollers that have the incWhenScrolling attribute set to Right, the minimum value is achieved when scrolled fully to the left. On horizontal Image Scrollers that have the incWhenScrolling attribute set to Left, the minimum value is achieved when scrolled fully to the right. On vertical Image Scrollers that have the incWhenScrolling attribute set to Down, the minimum value is achieved when scrolled fully to the top. On vertical Image Scrollers that have the incWhenScrolling attribute set to Up, the minimum value is achieved when scrolled fully to the bottom. The range is 0 - 65534 (0x00 - 0xFFFE).

Parameter="numberOfSelections" value="number" — Specifies the number of "stops" along the image. The range is 2 - 65534 (0x02 - 0xFFFE).

Parameter="orientation" value="HORIZONTAL" or "VERTICAL" — Specifies if the Image Scroller travels horizontally or vertically. The orientation parameter will override the orientation determined by the height and width dimensions. *Not currently supported.*

Parameter="sequence" value="image1;image2;image3(etc.)" — Specifies the sequence of images to scroll through, i.e. the scroller image. See Image Formats for supported image types. Alternative to supplying one single ScrollerImage.

Optional Image Scroller Parameter Attributes:

Parameter="hrefOnHitOnly" value="CHECKED" or "UNCHECKED" — Specifies if the Image Scroller href function(s) is launched only upon a "hit" or not. By default, hrefOnHitOnly is UNCHECKED, which means the Image Scroller will launch its href upon loading the page. The href is also launched after a forceUpdate() call. By setting hrefOnHitOnly to CHECKED, the href will only be launched upon the Image Scroller physically being selected or upon a forceHit() call.

Parameter="fillColor" value= See color entry conventions — Specifies the desired background fill color. See section on colors for more information. If no fill color is specified, the default color is completely transparent.

Parameter="initHref" value="function" — Specifies the function called when the page is loaded. Use this attribute whenever FromInitHref is used as the initialCondition. The value returned from this function call will determine which stop will initially be displayed within the viewport. The value should be between the min and max values. See Appendix B for all available functions.

Parameter="overlayHeight" value="number" — Specifies the height in pixels of the overlayImage. In a vertical Image Scroller, this also specifies the height of the viewport.

Parameter="overlayImage" value="image" — Specifies the image to use as the overlay within the viewport of the Image Scroller. In order to be useful, the overlay image must have a transparent section to allow for the viewing of the Image Scroller image underneath it. Therefore, the overlayImage must be a.png or .gif with transparency.

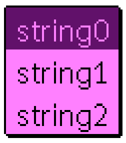
Parameter="overlayWidth" value="number" — Specifies the width in pixels of the overlayImage. In a horizontal Image Scroller, this also specifies the width of the viewport.

Parameter="scrollerImage" value="image" — Specifies the image to use as the Image Scroller. See Image Formats for supported image types.

Parameter="updateDelay" value ="number" — The number specifies the delay time from when the page is loaded until the first initHref function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to 0.01 (immediately upon page load).

Parameter="waitForInit" value="CHECKED" or "UNCHECKED" — Only valid if FromInitHref is used as the initialCondition. Specifies if the Image Scroller will wait for valid data before being displayed. If CHECKED, the Image Scroller will not display until the data from the initHref function is received. If waitForInit not specified, the default is UNCHECKED.

List



List Widget

Use the List Widget to create a list of selectable text objects that share a common function (or set of functions) and that are vertically aligned in a box. Selecting a list object highlights that object on the display and invokes the href function(s) with the value of the list item being the argument. By specifying an initial condition, that option is highlighted when the page is loaded, the href function(s) are invoked and the value associated with that list object is used as the argument.

Important note: The HEIGHT and WIDTH attributes for List.class DO NOT specify the actual size of the List box drawn on the screen. Instead, these attributes specify the size of the cell that contains the list box, and the relative position of other objects on the screen. The Amulet OS draws the box starting from the top-left corner of the widget. The code optimizes the width of the drawn box based on the width of the option titles and the specified font. The maximum number of visible list items that can fit in a box based on the HEIGHT attribute determines the actual height of the drawn box. Any remaining list items will be hidden from view, but are reachable via an arrow that allows for page scrolling through a list. You can also scroll through the list an item at a time by keeping the pen down and moving above or below the drawn list box.

List Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the List is to start out invisible. If the List starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="fillColor" value= See color entry conventions — Specifies the desired list fill color. See section on colors for more information. If no fill color is specified, the default color is the current background color. If the alpha is FF, then the fill is fully opaque.

Parameter="font" value="font, font size" — Specifies the font used for the option names within the list box. See Amulet GEM Font Converter for more information regarding the creation of .auf font files. Default is Bitstream Vera Sans. Default font size is 12pt.

Parameter="fontColor" value= See color entry conventions — Specifies the desired font color. See section on colors for more information. If no font color is specified, the default color is black.

Parameter="fontStyle" value="PLAIN" or "BOLD" or "ITALIC" — Specifies the style associated with the list box font. If BOLD or ITALIC is not selected, PLAIN is the default fontStyle. The available font styles are:

- PLAIN - The option text uses the standard font.
- BOLD — The option text is bold. (i.e. text)
- ITALIC — The option text is italicized. (i.e. text)

Parameter="href" value="function(s)" — The function (or multiple/sequenced functions) invoked when a list item is hit. See Appendix B for all available functions for the List widget.

Parameter="initialCondition" value="string" or "FromInitHref" — Specifies which option string is highlighted when the page is loaded. The value associated with the highlighted option string is the argument given to the href function(s). If "FromInitHref" is selected, the function specified by the InitHref attribute is called. Whichever option has the same intrinsic value as the returned value from the initHref function will be initially highlighted.

Parameter="options" value="see image below" — Specifies the list strings and their values. The value can be any number from 0 - 65535 (0x00 - 0xFFFF) or a STRING (which must be in single quotes (' '). (See note regarding Control Widget intrinsic values.) The first option displays at the top of the list and each subsequent option displays directly below the previous. To add more options, hit the plus button in the lower left. See Using Amulet Formatted Text for more formatting options.

Parameter="selectionColor" value= See color entry conventions — Specifies the color of the highlighted entry in the list box. See section on colors for more information. If no selection color is specified, the default color is black. If the alpha is FF, then the selectionColor is fully opaque.

Parameter="selectionFontColor" value= See color entry conventions — Specifies the color of the font within the highlighted entry in the list box. See section on colors for more information. If no selection color is specified, the default color is white.

Optional List Parameter Attributes:

Parameter="context" value="String"— Specifies the context of the options strings for translation. No context is not the same as an empty context. See Context under the Localization feature for more info.

Parameter="downArrow" value="image" — Specifies the "page down arrow" image to use when the list has more items than can be viewed. If this attribute is not present, then a default image, the Amulet logo, is used. See Image Formats for supported image types.

Parameter="downArrowAlt" value="image" — Specifies the "page down arrow" image to use when the down arrow is selected. If this attribute is not present, then a default image, the Amulet logo, is used. See Image Formats for supported image types.

Parameter="hrefOnHitOnly" value="CHECKED" or "UNCHECKED" — Specifies if the list href function(s) is launched only upon a "hit" or not. By default, hrefOnHitOnly is UNCHECKED, which means the list will launch its href upon loading the page. The href is also launched after a forceUpdate() call. By setting hrefOnHitOnly to CHECKED, the href will only be launched upon the list physically being selected or upon a forceHit() call.

Parameter="initHref" value="function" — Specifies the function called when the page is loaded. Use this attribute whenever FromInitHref is used as the initialCondition. The value returned from this function call will determine which option string starts out highlighted. The value should match the intrinsic value of one of the options strings. See Appendix B for all available functions.

Parameter="scrollRate" value="seconds" — Specifies the rate at which the list scrolls when selecting the area directly below or above the list box. The default scroll rate is .45 seconds. The range is 0.01 - 655.35.

Parameter="upArrow" value="image" — Specifies the "page up arrow" image to use when the list has more items than can be viewed. If this attribute is not present, then a default image, upArrow.gif, located in Amulet/Color/Configuration/Widgets/List/, is used. See Image Formats for supported image types.

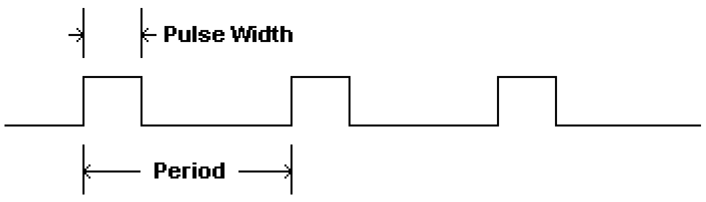
Parameter="upArrowAlt" value="image" — Specifies the "page up arrow" image to use when the up arrow is selected. If this attribute is not present, then a default image, upArrowAlt.gif, located in Amulet/Color/Configuration/Widgets/List/, is used. See Image Formats for supported image types.

Parameter="updateDelay" value ="number" — The number specifies the delay time from when the page is loaded until the first initHref function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to 0.01 (immediately upon page load).

Parameter="borderColor" value=See color entry conventions — Specifies the color of the list box border. See section on colors for more information. If no selection color is specified, the default color is black. If the alpha is FF, then the border is fully opaque.

PWM

A Pulse Width Modulation (PWM) widget is an invisible object that can send out a waveform out one of two or three PWM pins on the Amulet Color Chip. The waveform period and pulse width, in ms, can be specified as well as which PWM pin the waveform will come out on. The waveform peak is 3.3v and the trough is at 0v. The peak and trough levels cannot be modified.



Note: The PWM widget is not required to use the PWM ports. This widget provides an IWC interface for other widgets, but you can also use the Hardware Access methods inside of a script. To learn more, find GEMstudio Pro's Help menu and select GEMscript API. Then navigate to GEMscript API -> Hardware Access -> PWM.

PWM Parameter Attributes:

Parameter="channel" value="0" or "1" or "2" — Specifies which PWM channel on the Amulet Color Chip the waveform will be output. The default value is 0. Channel 0 is also the backlight dimming channel on many Amulet boards. (The MK-07C-HP Module only supports channels 0 and 1, but the backlight dimming is controlled by a separate dedicated pwm object.)

Parameter="initialCondition" value="ON" or "OFF" or "NO CHANGE" — Specifies if the waveform is to be active immediately upon loading the page. "No Change" means the PWM object will be created for the page, but the period and pulse width values will not be set when the page is loaded and the PWM will not be explicitly set to either on or off. This allows for re-instantiating a PWM object which has already been created earlier in the project without changing its setup values and current state. The default value is ON.

Parameter="periodIn_ms" value="time in ms" — Specifies the period of the waveform in ms. Range is 0.01-2040. (MK-07C-HP Module range is 0.003-129000)

Parameter="pulseWidthIn_ms" value="time in ms" — Specifies the pulse width of the waveform in ms. Range is 0-2040. (MK-07C-HP Module range is 0-129000)

RadioButton

Radio Button Widget

A Radio Button is a labeled, usually round button used to make a single selection from several options. To set a radio button, click on either the button or the adjacent label. All radio buttons that have the same groupName are considered part of a radio button group. Only one radio button within a group can be set at any one time. In contrast to a group of CheckBox widgets which must have identical HREF parameters to function as a group, each radio button in a group can invoke its own href function (or set of functions).

When using initHref to determine the initialCondition of the radio button group, the value of the data returned from the initHref function must match the internalNumber of one of the radio buttons in the group. If the internalNumber is not specified, the first radio button found in the project will be assigned internal number 1, with the internal numbers incrementing with each subsequent radio button found which is part of the same radio button groupName.

Radio Button Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Radio Button is to start out invisible. If the Radio Button starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="buttonAlign" value="LEFT" or "RIGHT" — Specifies the location of the Radio Button in relation to the label text.

Parameter="font" value="font" — Specifies the font and font size used for the Radio Button label. See Amulet GEM Font Converter for more information regarding the creation of .auf font files. Default is Bitstream Vera Sans. The default font size for the radio button label is 12pt.

Parameter="fontColor" value= See color entry conventions — Specifies the font color used for the Radio Button label. See section on colors for more information. If no font color is specified, the default color is black.

Parameter="fontStyle" value="BOLD" or "ITALIC" or "PLAIN" — Specifies the style associated with the Radio Button label font. The available font styles are:

- BOLD — The option text is bold. (i.e. text)
- ITALIC — The option text is italicized. (i.e. text)
- PLAIN — The option text is the standard font. (i.e. text)

Parameter="groupName" value="text" — Specifies the Radio Button group this radio button is a part of.

Parameter="href" value="function(s)" — The function (or multiple/sequenced functions) invoked when a Radio Button is set. Unlike checkboxes, radio buttons within a group can call different href function(s). See Appendix B for all available functions for the Radio Button widget.

Parameter="initialCondition" value="ON" or "OFF" or "FromInitHref" — Specifies the initial condition of the radio button when the page is loaded. If "FromInitHref" is selected, the function specified by the InitHref attribute is called. The returned byte value determines which single button (if any) within the group is selected; the returned value must exactly match one of the button's internalNumber.

Parameter="label" value="text" — Specifies the name that appears to the right or left of the radio button. See Using Amulet Formatted Text for more formatting options.

Parameter="cacheImage" value="CHECKED" or "UNCHECKED" — Specifies if all three Radio Button images are loaded into project-specific SDRAM in an uncompressed format for immediate use from any page within the project. All cached images are loaded into SDRAM prior to the first page being displayed, so the more images that are cached, the longer it will take for the initial page to be displayed. Page-to-page transition times can be dramatically reduced by caching images.

Optional Radio Button Parameter Attributes:

Parameter="context" value="String"— Specifies the context of the label string for translation. No context is not the same as an empty context. See Context under the Localization feature for more info.

Parameter="emptyImage" value="image" — Specifies the image to use when the Radio Button is in the not set state. If this attribute is not present, then a default image is used. See Image Formats for supported image types.

Parameter="fullImage" value="image" — Specifies the image to use when the Radio Button is in the set state. If this attribute is not present, then a default image is used. See Image Formats for supported image types.

Parameter="hrefOnHitOnly" value="CHECKED" or "UNCHECKED" — Specifies if the Radio Button href function(s) is launched only upon a "hit" or not. By default, hrefOnHitOnly is UNCHECKED, which means if the radio button has an initHref function, the href of the radio button which starts out "on" is launched upon loading the page. The href is also launched after a forceUpdate() call. By setting hrefOnHitOnly to CHECKED, the href will only be launched upon the radio button physically being selected or upon a forceHit() call.

Parameter="initHref" value="function" — Specifies the function called when the page is loaded. Use this attribute whenever FromInitHref is used as the initialCondition. All radio buttons within a group must have the same initHref function. The value of the data returned from the initHref function must match the internalNumber of one of the radio buttons in the group. If there is See Appendix B for all available functions.

Parameter="internalNumber" value="number" — Specifies the internal number of the Radio Button, used by the OS to determine which radio button is on. Should only be used if using initHref. Each button within a radio button group must have a unique internal number. If the value of the data returned from the initHref function matches the internalNumber of the radio button, then that radio button starts out in the "on" position. If the internalNumber is not specified, the first radio button found in the project will be assigned internal number 1, with the internal numbers incrementing with each subsequent radio button found which is part of the same radio button group.

Parameter="noSDRAM" value="CHECKED" or "UNCHECKED" — Specifies if the image(s) are NOT loaded into page-specific SDRAM prior to drawing the object. Has no effect if the images are already in the global cache. Having a copy of these images in the SDRAM speeds up the transition between the images and the invisible state. Default is UNCHECKED.

Parameter="trackingImage" value="image" — Specifies the image to use when the Radio Button is in the pen down state. If this attribute is not present, then a default image is used. See Image Formats for supported image types.

Parameter="updateDelay" value ="number" — The number specifies the delay time from when the page is loaded until the first initHref function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to 0.01 (immediately upon page load).

Scribble

The Scribble Widget allows for freehand drawing on a canvas. Assuming a stylus is being used, when the stylus sets down in the canvas of the Scribble Widget and starts moving within the canvas, a freehand line is drawn, following the directions of the stylus. The freehand line can be 1 to 15 pixels thick and can be any color. The Scribble canvas can have an optional background image and border. The background image, if specified, is the first image that is shown upon entering the page unless the initBackground attribute is set to FALSE. An InterWidget Communications method, uploadImage(), allows for another widget/object to make the Scribble Widget transfer the raw image data to an external processor via an xmodem protocol. The raw image is in the Amulet bitmap format.

If the Scribble canvas is saved using the IWC method saveCanvas(), the current state of the visible canvas is saved to the serial data flash, overwriting the current image specified by the canvas attribute. If it is desired to keep an initial image to be displayed upon page loading and you are using the saveCanvas(), you must use the background attribute to specify a separate initial image that will not get overwritten.

Scribble Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Scribble Widget is to start out invisible. If the Scribble Widget starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="href" value="function" — The function invoked upon receiving the uploadImage() IWC method. Only valid function is currently Amulet:UART.xmodemUploadImage().

Parameter="lineColor" value= See color entry conventions — Specifies the color of the active freehand drawing line. See section on colors for more information. If no line color is specified, the default color is black. If the alpha is FF, then the line color is fully opaque.

Parameter="lineWeight" value="pixels" — Defines the weight (thickness) of the active freehand drawing line in pixels. The range is 1- 15 (0x01 - 0x0F).

Optional Scribble Parameter Attributes:

Parameter="border" value="number" — Specifies width, in pixels, of the border around the dimensions of the Scribble Widget. Default is 0, meaning no border.

Parameter="borderColor" value=See color entry conventions — Specifies the color of the border around the dimensions of the Scribble Widget. See section on colors for more information. If no border color is specified, the default color is black. Only applicable if border has a value of 1 or greater. If the alpha is FF, then the border color is fully opaque.

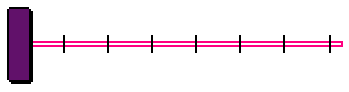
Parameter="background" value="image" — Specifies the optional background image used for the canvas of the Scribble Widget. This is an optional parameter. The image does not have to be the exact same dimensions as the Scribble Widget, but it will start drawing the background image from the topleft corner of the Scribble Widget. By default, the background image is initially displayed upon page load. If the canvas is saved, it will replace the canvas image and the background image will be left untouched. See Image Formats for supported image types.

Parameter="canvas" value="image" — Specifies the image used for the canvas of the Scribble Widget. The canvas can contain a background image or it can be a completely blank image, but the dimensions of the canvas MUST be exactly the same as the dimensions of the Scribble Widget. If the canvas is saved, the image overwrites the existing canvas image. If the initial image is desired to be saved, use the background attribute. See Image Formats for supported image types.

Parameter="initImage" value="Canvas" or "Background" or "Fillcolor" — Specifies if the background image is displayed initially upon page load. If set to Canvas, the canvas image is displayed initially upon page load. Defaults to Canvas if initImage is not specified.

Parameter="fillColor" value= See color entry conventions — Specifies the desired lineplot line color. See section on colors for more information. If no fill color is specified, the default color is white. If backgroundImage specified, fillColor is not used. If the alpha is FF, then the fill color is fully opaque.

Slider


Slider Widget

The Slider Widget invokes a function (or set of functions) either upon 1) any movement of the handle (onChange) or 2) only when the handle has been released (onPenUp). The argument is determined by the location of the handle relative to the limits setup with the min and max attributes. The location of the min value is determined by the minAt attribute(left, right, top or bottom). The height and width dimensions determines whether the slider is horizontal or vertical. The longer dimension establishes the path that the slider travels, while the shorter dimension determines the maximum handle size. In addition, the widget creates a 3-pixel wide "channel" along the slider path. To move the slider, touch the handle and drag it along the channel. If hrefEvent is setup as onChange, the slider invokes the function(s) whenever the handle moves. If hrefEvent is setup as onPenUp, the slider invokes the function(s) only upon releasing the handle. Touching anywhere on the channel causes the handle to snap to that location and invoke the function(s) with the corresponding argument. Tick marks are optional. As an option, you can specify the offset of the handle from the center of the channel by using the handleOffset parameter.

Slider Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Slider is to start out invisible. If the Slider starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="channelColor" value= See color entry conventions — Specifies the color of the channel the handle slides along. See section on colors for more information. If no channel color is specified, the default color is black.

Parameter="handleColor" value= See color entry conventions — Specifies the color inside of the handle. See section on colors for more information. If no handle color is specified, the default color is white.

Parameter="handleFrameColor" value= See color entry conventions — Specifies the color of the handle frame. See section on colors for more information. If no handle frame color is specified, the default color is black.

Parameter="handleTrackingColor" value= See color entry conventions — Specifies the color of the handle when being touched. See section on colors for more information. If no handle frame color is specified, the default color is black.

Parameter="href" value="function(s)" — The function (or multiple/sequenced functions) invoked upon the event specified in hrefEvent. See Appendix B for all available functions for the Slider widget.

Parameter="hrefEvent" value="On Change" or "On Release" — The event which triggers the launching of the href function. If set to On Change, the slider will launch the function call whenever the handle is moved. If set to On Release, the slider will only launch the function call upon the releasing of the slider handle. If hrefEvent is not specified, the default is On Change.

Parameter="initialCondition" value="number" or "FromInitHref" — Specifies handle position when the page is loaded. The position value becomes the argument given to the href function(s). If FromInitHref is selected, the function specified by the InitHref attribute is called. The returned value determines the handle position. [The range is 0 - 65535 (0x00 - 0xFFFF).]

Parameter="max" value="number" — The maximum value used as the argument given to the function(s) specified in href. Maximum value is achieved when handle is full-right on a horizontal slider, or full-top on a vertical slider. The range is 1 - 65535 (0x01 - 0xFFFF).

Parameter="min" value="number" — The minimum value used as the argument given to the function(s) specified in href. Minimum value is achieved when handle is full-left on a horizontal slider, or full-bottom on a vertical slider. The range is 0 - 65534 (0x00 - 0xFFFE).

Parameter="minAt" value=""LEFT" or "RIGHT" or "TOP" or "BOTTOM"" — Determines where the minimum value of the slider is located. As the handle sweeps from the minAt location, the value increases until the maximum value is reached at the opposite extreme of the minAt location. Default values are LEFT for horizontal sliders and BOTTOM for vertical sliders. The options are:

- LEFT — The slider value increases from left to right. (horizontal slider only)
- RIGHT — The slider value increases from right to left. (horizontal slider only)
- TOP — The slider value increases from top to bottom. (vertical slider only)
- BOTTOM — The slider value increases from bottom to top. (vertical slider only)

Parameter="tickColor" value= See color entry conventions — Specifies the color of the ticks drawn on the channel. See section on colors for more information. If no tick color is specified, the default color is black. If the alpha is FF, then the tick color is fully opaque.

Parameter="tickCount" value="number" — The total number of tick marks along the slider channel. If 0, no tick marks are visible. If tickCount is not given, the default is 0. The range is 0 - 255 (0x00 - 0xFF).

Optional Slider Parameter Attributes:

Parameter="handleOffset" value="number" — Specifies the number of pixels from the center of the channel the handle is located. If a vertical slider, positive numbers shift the handle to the right and negative numbers shift it to the left. If a horizontal slider, positive numbers shift the handle to the bottom and negative numbers shift it to the top. [The range is -100 through 100]

Parameter="handleThickness" value="number" — The width (in pixels) of the handle if a horizontal slider, or the height of the handle if a vertical slider. If handleThickness is not given, the default is 11 pixels. The range is 4 - 255 (0x04 - 0xFF).

Parameter="hrefOnHitOnly" value="CHECKED" or "UNCHECKED" — Specifies if the slider href function(s) is launched only upon a "hit" or not. By default, hrefOnHitOnly is UNCHECKED, which means the slider will launch its href upon loading the page. The href is also launched after a forceUpdate() call. By setting hrefOnHitOnly to CHECKED, the href will only be launched upon the slider physically being selected or upon a forceHit() call.

Parameter="initHref" value="function" — Only valid if FromInitHref is used as the initialCondition. Specifies the function called when the page is loaded. The value returned from this function call will be used as the initial condition of the Slider handle. See Appendix B for all available functions.

Parameter="orientation" value="HORIZONTAL" or "VERTICAL" — Specifies if the handle is to travel horizontally or vertically. The orientation parameter will override the orientation determined by the height and width dimensions.

Parameter="tickLength" value="number" — The length (size) of each tick mark, in pixels. The range is 1 - 255 (0x01 - 0xFF). If tickLength is not given, the default is 9 pixels.

Parameter="tickPosition" value="CENTER" or "TOP" or "BOTTOM" or "LEFT" or "RIGHT" — The position of the tick marks in relation to the channel. . The range for a horizontal slider is CENTER, TOP or BOTTOM. The range for a vertical slider is CENTER, LEFT or RIGHT. If tickPosition is not given, the default is CENTER (inside the channel).

Parameter="updateDelay" value ="number" — The number specifies the delay time from when the page is loaded until the first initHref function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to 0.01 (immediately upon page load).

Parameter="waitForInit" value="CHECKED" or "UNCHECKED" — Only valid if FromInitHref is used as the initialCondition. Specifies if the Slider handle will wait for valid data before being displayed on the channel. If CHECKED, the Slider handle will not display until the data from the initHref function is received. If UNCHECKED, or the attribute is not present, the Slider handle momentarily starts at the minAt location until the initHref function receives its data. This is useful when the initHref contains a UART or USB method and may not return immediately.

Touch Area

Touch Area Widget

Touch Areas are rectangular touchable regions that can be placed anywhere on the display. Touch Areas are not visible when not touched, and have no visual feedback by default when touched, although they can be set up to shade the area underneath the Touch Area with a transparent color, or invert the colors in the area underneath the Touch Area.
Instead of a single HREF that is launched upon release, like most other control widgets, the Touch Area has a number of events that can all have their own function call(s) associated with them. There are a number of events that are valid for all touch screens, like onRelease, onSlideOff, onSlideOn, onTap, onDoubleTap, and onTouch. There are also many gesture events that are only valid on certain capacitive touch panels like onPinch, onZoom, onSwipeEast, onMultiTouchSwipeEast, etc... Only the optional events that are applicable for the currently selected touch panel will show up in the Add/Remove Parameter dialog box.
Like Function Buttons, Touch Areas can also be set up to auto-repeat. When pressed, an auto-repeat Touch Area delays a user-defined amount of time, then invokes a function (or set of functions) at a user-defined frequency while the Touch Area is still being pressed. See onAutoRepeat, repeatDelay and repeatRate for more information.

Touch Area Parameter Attributes:

Parameter="onRelease" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area is touched and then released while still within the Touch Area boundaries. See Appendix B for all available functions for the onRelease event.

Parameter="visualFeedback" value="None" or "Alpha" or "Invert" — Specifies the visual action of the Touch Area while being touched. "None" causes no visual change on the display. "Alpha" shades the area underneath the Touch Area with a transparent color defined by alphaColor. "Invert" causes the colors in the area underneath the Touch Area to become inverted. Sliding off or releasing the Touch Area returns the area underneath the Touch Area back to its original state. "None" is the default.

Optional Touch Area Parameter Attributes:

Parameter="alphaColor" value= See color entry conventions — If visualFeedback equals "Alpha", this attribute specifies the alpha color mask used to notify the user the Touch Area has been depressed. See section on colors for more information. If no alphaColor is specified, the default color is a transparent gray. If the color specified does not have an alpha component, then it will be completely opaque, which will result in a colored rectangle appearing in place of the Touch Area while it is touched.

Parameter="gestureDelay" value="number" — Time to delay from when Touch Area initially senses a gesture until it starts to act upon the gesture (i.e. launch the function(s) tied to the gesture event). This delay helps filter out false gestures prior to sensing the true gesture being performed. Specified in seconds, with a single floating-point number. A gestureDelay of 0.00 results in the gesture never being seen. The range is 0.00 - 655.35. Default is 0.05 if the parameter is not specified.

Parameter="gestureRate" value="number" — The gesture function call frequency while Touch Area is sensing a gesture, after the initial delay determined by gestureDelay. This rate helps throttle the frequency of gesture functions that will be launched. If a gesture is not being sensed, this gestureRate has no effect. Specified in seconds, with a single floating-point number. A gestureRate of 0.00 results in the gesture function(s) being launched once after the given gestureDelay time and then not repeated. The range is 0.00 - 655.35. Default is 0.25 if the parameter is not specified.

Parameter="onAutoRepeat" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area is touched for at least the time specified in repeatDelay. The function(s) will be launched at the given update rate specified by repeatRate as long as the Touch Area remains touched. See Appendix B for all available functions for the onAutoRepeat event.

Parameter="onDoubleTap" value="function(s)" — the function (or multiple/sequenced functions) invoked after the Touch Area is tapped twice in a row. To be considered a Double Tap, the Touch Area needs to be touched for 150ms or less twice in a row with the time between the two taps at 75ms or less. It is possible to adjust the time required to be considered a Double Tap by contacting Amulet Technologies. See Appendix B for all available functions for the onDoubleTap event.

Parameter="onJogBackward" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Jog Backward gesture. To be considered a Jog Backward gesture, the Touch Area needs to be touched by a single finger and then rotated in a counter-clockwise manner. As long as the finger stays rotating counter-clockwise, the Jog Backward gesture will be sensed. Use gestureRate to specify the maximum rate the onJogBackward function(s) will be called. See Appendix B for all available functions for the onJogBackward event. (Jog Backward Gesture only available on select capacitive touch panels)

Parameter="onJogForward" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Jog Forward gesture. To be considered a Jog Forward gesture, the Touch Area needs to be touched by a single finger and then rotated in a clockwise manner. As long as the finger stays rotating clockwise, the Jog Forward gesture will be sensed. Use gestureRate to specify the maximum rate the onJogForward function(s) will be called. See Appendix B for all available functions for the onJogForward event. (Jog Forward Gesture only available on select capacitive touch panels)

Parameter="onMultiTouchDoubleTap" value="function(s)" — the function (or multiple/sequenced functions) invoked after the Touch Area is tapped twice in a row using two or more fingers. To be considered a Multi Touch Double Tap, the Touch Area needs to be touched by two or more fingers for 150ms or less twice in a row with the time between the two taps at 75ms or less. It is possible to adjust the time required to be considered a multi touch double tap by contacting Amulet Technologies. See Appendix B for all available functions for the onMultiTouchDoubleTap event. (Multi Touch Double Tap Gesture only available on select capacitive touch panels)

Parameter="onMultiTouchSwipeEast" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Multi Touch Swipe East gesture. To be considered a Multi Touch Swipe East gesture, the Touch Area needs to be touched by two or more fingers and then swiped to the right. As long as the fingers continue swiping to the right, the Multi Touch Swipe East gesture will be sensed. Use gestureRate to specify the maximum rate the onMultiTouchSwipeEast function(s) will be called. See Appendix B for all available functions for the onMultiTouchSwipeEast event. (Multi Touch Swipe East Gesture only available on select capacitive touch panels)

Parameter="onMultiTouchSwipeNorth" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Multi Touch Swipe North gesture. To be considered a Multi Touch Swipe

North gesture, the Touch Area needs to be touched by two or more fingers and then swiped to the top. As long as the fingers continue swiping to the top, the Multi Touch Swipe North gesture will be sensed. Use gestureRate to specify the maximum rate the onMultiTouchSwipeNorth function(s) will be called. See Appendix B for all available functions for the onMultiTouchSwipeNorth event. (Multi Touch Swipe North Gesture only available on select capacitive touch panels)

Parameter="onMultiTouchSwipeNorthEast" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Multi Touch Swipe North East gesture. To be considered a Multi Touch Swipe North East gesture, the Touch Area needs to be touched by two or more fingers and then swiped to the top and the right. As long as the fingers continue swiping to the top and the right, the Multi Touch Swipe North East gesture will be sensed. Use gestureRate to specify the maximum rate the onMultiTouchSwipeNorthEast function(s) will be called. See Appendix B for all available functions for the onMultiTouchSwipeNorthEast event. (Multi Touch Swipe North East Gesture only available on select capacitive touch panels)

Parameter="onMultiTouchSwipeNorthWest" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Multi Touch Swipe North West gesture. To be considered a Multi Touch Swipe North West gesture, the Touch Area needs to be touched by two or more fingers and then swiped to the top and the left. As long as the fingers continue swiping to the top and the left, the Multi Touch Swipe North West gesture will be sensed. Use gestureRate to specify the maximum rate the onMultiTouchSwipeNorthWest function(s) will be called. See Appendix B for all available functions for the onMultiTouchSwipeNorthWest event. (Multi Touch Swipe North West Gesture only available on select capacitive touch panels)

Parameter="onMultiTouchSwipeSouth" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Multi Touch Swipe South gesture. To be considered a Multi Touch Swipe South gesture, the Touch Area needs to be touched by two or more fingers and then swiped to the bottom. As long as the fingers continue swiping to the bottom, the Multi Touch Swipe South gesture will be sensed. Use gestureRate to specify the maximum rate the onMultiTouchSwipeSouth function(s) will be called. See Appendix B for all available functions for the onMultiTouchSwipeSouth event. (Multi Touch Swipe South Gesture only available on select capacitive touch panels)

Parameter="onMultiTouchSwipeSouthEast" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Multi Touch Swipe South East gesture. To be considered a Multi Touch Swipe South East gesture, the Touch Area needs to be touched by two or more fingers and then swiped to the bottom and the right. As long as the fingers continue swiping to the bottom and the right, the Multi Touch Swipe South East gesture will be sensed. Use gestureRate to specify the maximum rate the onMultiTouchSwipeSouthEast function(s) will be called. See Appendix B for all available functions for the onMultiTouchSwipeSouthEast event. (Multi Touch Swipe South East Gesture only available on select capacitive touch panels)

Parameter="onMultiTouchSwipeSouthWest" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Multi Touch Swipe South West gesture. To be considered a Multi Touch Swipe South West gesture, the Touch Area needs to be touched by two or more fingers and then swiped to the bottom and the left. As long as the fingers continue swiping to the bottom and the left, the Multi Touch Swipe South West gesture will be sensed. Use gestureRate to specify the maximum rate the onMultiTouchSwipeSouthWest function(s) will be called. See Appendix B for all available functions for the onMultiTouchSwipeSouthWest event. (Multi Touch Swipe South West Gesture only available on select capacitive touch panels)

Parameter="onMultiTouchSwipeWest" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Multi Touch Swipe West gesture. To be considered a Multi Touch Swipe West gesture, the Touch Area needs to be touched by two or more fingers and then swiped to the left. As long as the fingers continue swiping to the left, the Multi Touch Swipe West gesture will be sensed. Use gestureRate to specify the maximum rate the onMultiTouchSwipeWest function(s) will be called. See Appendix B for all available functions for the onMultiTouchSwipeWest event. (Multi Touch Swipe West Gesture only available on select capacitive touch panels)

Parameter="onMultiTouchTap" value="function(s)" — the function (or multiple/sequenced functions) invoked after the Touch Area is tapped using two or more fingers. To be considered a Multi Touch Tap, the Touch Area needs to be touched by two or more fingers for 150ms or less. It is possible to adjust the time required to be considered a multi touch tap by contacting Amulet Technologies. See Appendix B for all available functions for the onMultiTouchTap event. (Multi Touch Tap Gesture only available on select capacitive touch panels)

Parameter="onPinch" value="function(s)" — the function (or multiple/sequenced functions) invoked after the Touch Area senses the Pinch gesture. The Pinch gesture is sensed when the Touch Area is touched by two fingers that are apart and then pinched in together. As long as the two fingers are pinching in, the Pinch gesture will be sensed. See Appendix B for all available functions for the onPinch event. (Pinch Gesture only available on select capacitive touch panels)

Parameter="onSlideOff" value="function(s)" — the function (or multiple/sequenced functions) invoked after touching the Touch Area then sliding off the Touch Area while still making contact with the touch panel. The Slide Off event will only launch once for every Slide Off occurrence. It is possible to have multiple Slide Off occurrences if the finger slides back on the Touch Area and then off again. See Appendix B for all available functions for the onSlideOff event.

Parameter="onSlideOn" value="function(s)" — the function (or multiple/sequenced functions) invoked after touching the Touch Area then sliding off and then sliding back on the Touch Area while maintaining contact with the touch panel. The Slide On event will only launch once for every Slide On occurrence. It is possible to have multiple Slide On occurrences if the finger slides back off the Touch Area and then back on again. See Appendix B for all available functions for the onSlideOn event.

Parameter="onSwipeEast" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Swipe East gesture. To be considered a Swipe East gesture, the Touch Area needs to be touched by one finger and then swiped to the right. As long as the finger continues swiping to the right, the Swipe East gesture will be sensed. Use gestureRate to specify the maximum rate the onSwipeEast function(s) will be called. See Appendix B for all available functions for the onSwipeEast event. (Swipe East Gesture only available on select capacitive touch panels)

Parameter="onSwipeNorth" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Swipe North gesture. To be considered a Swipe North gesture, the Touch Area needs to be touched by one finger and then swiped to the top. As long as the finger continues swiping to the top, the Swipe North gesture will be sensed. Use gestureRate to specify the maximum rate the onSwipeNorth function(s) will be called. See Appendix B for all available functions for the onSwipeNorth event. (Swipe North Gesture only available on select capacitive touch panels)

Parameter="onSwipeNorthEast" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Swipe North East gesture. To be considered a Swipe North East gesture, the Touch Area needs to be touched by one finger and then swiped to the top and the right. As long as the finger continues swiping to the top and the right, the Swipe North East gesture will be sensed. Use gestureRate to specify the maximum rate the onSwipeNorthEast function(s) will be called. See Appendix B for all available functions for the onSwipeNorthEast event. (Swipe North East Gesture only available on select capacitive touch panels)

Parameter="onSwipeNorthWest" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Swipe North West gesture. To be considered a Swipe North West gesture, the Touch Area needs to be touched by one finger and then swiped to the top and the left. As long as the finger continues swiping to the top and the left, the Swipe North West gesture will be sensed. Use gestureRate to specify the maximum rate the onSwipeNorthWest function(s) will be called. See Appendix B for all available functions for the onSwipeNorthWest event. (Swipe North West Gesture only available on select capacitive touch panels)

Parameter="onSwipeSouth" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Swipe South gesture. To be considered a Swipe South gesture, the Touch Area needs to be touched by one finger and then swiped to the bottom. As long as the finger continues swiping to the bottom, the Swipe South gesture will be sensed. Use gestureRate to specify the maximum rate the onSwipeSouth function(s) will be called. See Appendix B for all available functions for the onSwipeSouth event. (Swipe South Gesture only available on select capacitive touch panels)

Parameter="onSwipeSouthEast" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Swipe South East gesture. To be considered a Swipe South East gesture, the Touch Area needs to be touched by one finger and then swiped to the bottom and the right. As long as the finger continues swiping to the bottom and the right, the Swipe South East gesture will be sensed. Use gestureRate to specify the maximum rate the onSwipeSouthEast function(s) will be called. See Appendix B for all available functions for the onSwipeSouthEast event. (Swipe South East Gesture only available on select capacitive touch panels)

Parameter="onSwipeSouthWest" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Swipe South West gesture. To be considered a Swipe South West gesture, the Touch Area needs to be touched by one finger and then swiped to the bottom and the left. As long as the finger continues swiping to the bottom and the left, the Swipe South West gesture will be sensed. Use gestureRate to specify the maximum rate the onSwipeSouthWest function(s) will be called. See Appendix B for all available functions for the onSwipeSouthWest event. (Swipe South West Gesture only available on select capacitive touch panels)

Parameter="onSwipeWest" value="function(s)" — The function (or multiple/sequenced functions) invoked after the Touch Area senses a Swipe West gesture. To be considered a Swipe West gesture, the Touch Area needs to be touched by one finger and then swiped to the left. As long as the finger continues swiping to the left, the Swipe West gesture will be sensed. Use gestureRate to specify the maximum rate the onSwipeWest function(s) will be called. See Appendix B for all available functions for the onSwipeWest event. (Swipe West Gesture only available on select capacitive touch panels)

Parameter="onTap" value="function(s)" — the function (or multiple/sequenced functions) invoked after the Touch Area is tapped using one finger. To be considered a Tap, the Touch Area needs to be touched by one finger for 150ms or less. It is possible to adjust the time required to be considered a tap by contacting Amulet Technologies. See Appendix B for all available functions for the onTap event.

Parameter="onTouch" value="function(s)" — the function (or multiple/sequenced functions) invoked after the Touch Area is initially touched. The onTouch function(s) are called once immediately upon touching the Touch Area. See Appendix B for all available functions for the onTouch event.

Parameter="onZoom" value="function(s)" — the function (or multiple/sequenced functions) invoked after the Touch Area senses the Zoom gesture. The Zoom gesture is sensed when the Touch Area is touched by two fingers that are slightly apart and then separated further apart from each other. As long as the two fingers are moving away from each other, the Zoom gesture will be sensed. See Appendix B for all available functions for the onZoom event. (Zoom Gesture only available on select capacitive touch panels)

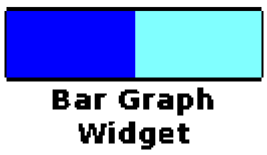
Parameter="repeatDelay" value="number" — Time to delay from when Touch Area is initially pressed until it starts to auto-repeat the onAutoRepeat function(s). Specified in seconds, with a single floating-point number. A repeatDelay of 0.00 results in the onAutoRepeat never firing. The range is 0.00 - 655.35.

Parameter="repeatRate" value="number" — The onAutoRepeat function(s) call frequency while Touch Area is being pressed, after the initial delay determined by repeatDelay. Specified in seconds, with a single floating-point number. A repeatRate of 0.00 results in the onAutoRepeat function(s) being launched only once after the given repeatDelay time and then not repeated. The range is 0.00 - 655.35.

View Widgets

View Widgets enable the user to visualize text, numerical values, or arrays of values.

BarGraph



A Bar Graph Widget is a live bargraph that represents the numerical value returned from a function call. The Bar Graph moves from left-to-right, right-to-left, bottom-to-top, or top-to-bottom.

Bargraph Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Bar Graph is to start out invisible. If the Bar Graph starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="fillColor" value= See color entry conventions — Specifies the desired bargraph fill color. See section on colors for more information. If no fill color is specified, the default color is black. If the alpha is FF, then the fill color is fully opaque.

Parameter="borderColor" value= See color entry conventions — Specifies the desired bargraph border color. See section on colors for more information. If no border color is specified, the default color is the fillColor color. If the alpha is FF, then the border color is fully opaque.

Parameter="backgroundColor" value= See color entry conventions — Specifies the desired bargraph background color. See section on colors for more information. If no background color is specified, the default color is white. if the alpha is FF, then the background color is fully opaque.

Parameter="href" value="function" — The function called to retrieve the widget input. See Appendix B for all available functions for the Bargraph Widget. The function is called at an update rate specified by the updateFreq attribute.

Parameter="min" value="number" — Minimum value returned from the href function; must be less than max. If the function returns a byte, the range is 0 - 254 (0x00 - 0xFE). If the function returns a word, the range is 0 - 65534 (0x00 - 0xFFFE).

Parameter="max" value="number" — Maximum value returned from the href function; must be greater than min. If the function returns a byte, the range is 1 - 255 (0x01 - 0xFF). If the function returns a word, the range is 1 - 65535 (0x01 - 0xFFFF).

Parameter="sweepFrom" value="LEFT" or "RIGHT" or "TOP" or "BOTTOM" — Determines where to begin drawing the bargraph. The options are:

- LEFT — The bargraph is drawn from left-to-right. (default)
- RIGHT — The bargraph is drawn from right-to-left.
- TOP — The bargraph is drawn from top-to-bottom.
- BOTTOM — The bargraph is drawn from bottom-to-top

Parameter="updateFreq" value ="number" — The number specifies the href function call frequency (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. A value of 0.00 means update never.

Parameter="updateDelay" value ="number" — The number specifies the delay time from when the page is loaded until the first href function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to updateFreq number.

Optional Bar Graph Parameter Attributes:

Parameter="waitForInit" value="CHECKED" or "UNCHECKED" — Specifies if the Bar Graph will wait for valid data before being displayed. If CHECKED, the Bar Graph will not display until the data from the href function is received. If UNCHECKED, or the attribute is not present, the Bar Graph momentarily starts at the min location until the href function receives its data. If waitForInit not specified, the default is UNCHECKED. This is useful when the Href contains a UART or USB method and may not return immediately.

Dynamic Image

Dynamic Image Widget

The Dynamic Image Widget is an uncompressed image which provides a space holder to display images which can be loaded serially at runtime. This is particularly useful for end-user uploaded images which do not change frequently. For images that change frequently or which are not required to persist between power cycles, a host processor can draw directly to the Frame Buffer.

The size of the image to be uploaded must be the exact same size as the canvas image. Images are uploaded via xmodem crc protocol. The image must be in either the Amulet bitmap or JPEG formats. The first 24-bytes of any image being uploaded, known as the header bytes, must be the same as the header bytes of the canvas image. The first six bytes are the flash header bytes and the next five bytes are the image header bytes. These first 24 bytes can be found in the .inc file, which will be generated in the MAP folder of your project after saving or programming your project.

To correctly use the Dynamic Image Widget, the IWC method, Amulet:loadFlash(return), needs to be used. Once that method is invoked, the Amulet will start sending 'C's, ready to receive the incoming xmodem data from an external source. Once the image is fully sent, and the xmodem protocol is complete, meaning the external source sent an EOT(0x04) and the Amulet answered back with an ACK(0x06), the external source needs to send an ETB(0x17), which will take the Amulet out of the xmodem mode and return to the active page. The Dynamic Image Widget must then be sent a reset IWC method. The easiest way to accomplish this all is via a META REFRESH using a trigger. The META would look something like:

```
<META HTTP-EQUIV="REFRESH"
CONTENT="0,0.25;ONVAR=Amulet:UART.byte(5).value();TRIGGER=0xFF;URL=Amulet:loadFlash(return),Amulet:docun
```

In the above case, when the external processor is ready to send over a new image, it would respond back to the Amulet:UART.byte(5).value() request with a value of 0xFF, which would cause the Amulet to enter the loadFlash routine, which means C's will start coming from the Amulet. After the image has been uploaded to the Amulet and the external processor sends the ETB(0x17), the Amulet returns to the active page and then performs a reset condition to the Dynamic Image Widget called Dyn1, which forces a repaint using the new image which was just uploaded.

Dynamic Image Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Dynamic Image is to start out invisible. If the Dynamic Image starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="canvas" value="image" — Specifies the image used for the canvas of the Dynamic Image Widget. This is a required parameter. The canvas can contain a default image or it can be a completely blank image, but the dimensions of the canvas MUST be exactly the same as the dimensions of the Dynamic Image Widget. See Image Formats for supported image types.

ImageBar

Image Bar Widget

The Image Bar Widget uses two images (empty bar and full bar) to create a custom "bargraph". A byte (or word) returned from a function call is the widget input. Based upon the value of the byte (or word) returned, a percentage of the empty bar and a percentage of the full bar are displayed. For further customization, the wipe between empty and full can be from left-to-right, right-to-left, bottom-to-top, or top-to-bottom. When the minimum value or less is returned, the entire "empty" image is displayed. When the maximum value or greater is returned, the entire "full" image is displayed.

Image Bar Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Image Bar is to start out invisible. If the Image Bar starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="empty" value="image" — Image used when minimum value is returned from the href function. See Image Formats for supported image types.

Parameter="full" value="image" — Image used when maximum value is returned from the href function. See Image Formats for supported image types.

Parameter="href" value="function" — The function called to retrieve the widget input. See Appendix B for all available functions for the Image Bar widget. The function is called at an update rate specified by the updateRate attribute.

Parameter="max" value="number" — Maximum value returned from the href function; must be greater than min. If the function returns a byte, the range is 1 - 255 (0x01 - 0xFF). If the function returns a word, the range is 1 - 65535 (0x01 - 0xFFFF).

Parameter="min" value="number" — Minimum value returned from the href function; must be less than max. If the function returns a byte, the range is 0 - 254 (0x00 - 0xFE). If the function returns a word, the range is 0 - 65534 (0x00 - 0xFFFE).

Parameter="sweepFrom" value="LEFT" or "RIGHT" or "TOP" or "BOTTOM" — Determines where (within the image bar cell) to begin the image transition. The options are:

- LEFT — The "full" and "empty" images transition from left-to-right.
- RIGHT — The "full" and "empty" images transition from right-to-left.
- TOP — The "full" and "empty" images transition from top-to-bottom.
- BOTTOM — The "full" and "empty" images transition from bottom-to-top.

Parameter="updateFreq" value ="number" — Specifies the href function call frequency (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. A value of 0.00 means update never.

Parameter="updateDelay" value ="number" — Specifies the delay time from when the page is loaded until the first href function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to the updateFreq number.

Optional Image Bar Parameter Attributes:

Parameter="waitForInit" value="CHECKED" or "UNCHECKED" — Only valid if href is used. Specifies if the Image Bar will wait for valid data before displaying an image on the channel. If CHECKED, the Image Bar will not display until the data from the href function is received. If UNCHECKED, or the attribute is not present, the Image Bar momentarily starts at the min location until the href function receives its data. If waitForInit not specified, the default is UNCHECKED. This is useful when the Href contains a UART or USB method and may not return immediately.

ImageSequence

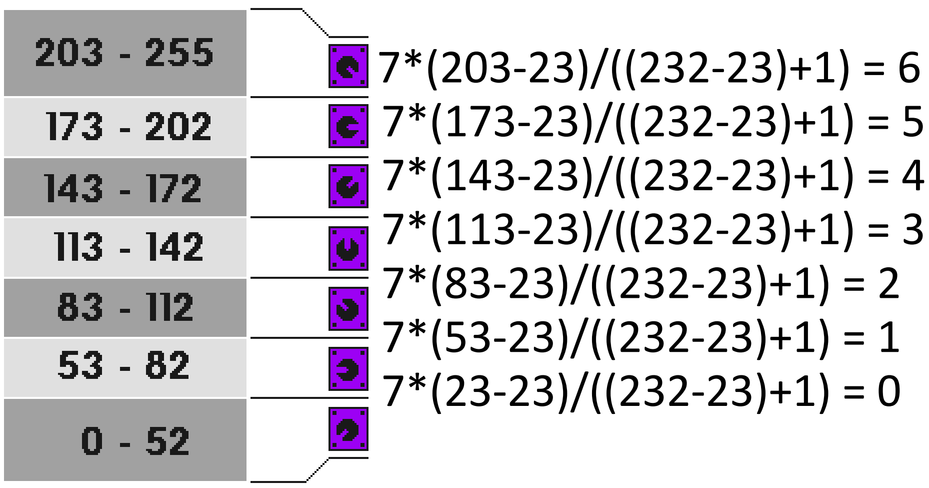
Image Sequence Widget

The Image Sequence Widget is similar to an animated image. However, instead of being linked to a timer event, the transition between images is linked to a byte (or word) returned from an href function call. The displayed image is determined by scaling the value returned from the href function call. The following algorithm is used:

Image # = {(total # of images) * (value of byte from function - min)}/ (max - min + 1)

The resultant number, truncated to an integer, is the number of the image to display, where the first image in the sequence is numbered 0. For all images to be displayable, (max-min+1) must be greater than or equal to the number of images. Also, if the href function returns a value less than the specified min, the value will be treated as equal to the specified min. Likewise, if the href function returns a value greater than the specified max, the value will be treated as equal to the specified max.
Consider the following example.

A sequence of 7 separate images: Image0 = , ... Image6 = .
An href function that returns a byte that ranges between a min of 23 and a max of 232. Although the value returned by the href function may vary with time, only one of the seven images will be displayed at any one time. Table 1, below, maps each value range to the image that will be displayed. Note that values below 23 are treated like a 23, and values above 232 are treated like a 232.



203 - 255	7*(203-23)/((232-23)+1) = 6
173 - 202	7*(173-23)/((232-23)+1) = 5
143 - 172	7*(143-23)/((232-23)+1) = 4
113 - 142	7*(113-23)/((232-23)+1) = 3
83 - 112	7*(83-23)/((232-23)+1) = 2
53 - 82	7*(53-23)/((232-23)+1) = 1
0 - 52	7*(23-23)/((232-23)+1) = 0

Table 1. Image Sequence Example

Image Sequence Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Image Sequence is to start out invisible. If the Image Sequence starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="cacheImage" value="CHECKED" or "UNCHECKED" — Specifies if the entire sequence of images are loaded into project-specific SDRAM in an uncompressed format for immediate use from any page within the project. All cached images are loaded into SDRAM prior to the first page being displayed, so the more images that are cached, the longer it will take for the initial page to be displayed. Page-to-page transition times can be dramatically reduced by caching images.

Parameter="href" value="function" — The function called to retrieve the widget input. See Appendix B for all available functions for the Image Sequence widget. The function is called at an update rate specified by the updateRate attribute.

Parameter="max" value="number" — Maximum value returned from the href function; must be greater than min. If the function returns a byte, the range is 1 - 255 (0x01 - 0xFF). If the function returns a word, the range is 1 - 65535 (0x01 - 0xFFFF).

Parameter="min" value="number" — Minimum value returned from the href function; must be less than max. If the function returns a byte, the range is 0 - 254 (0x00 - 0xFE). If the function returns a word, the range is 0 - 65534 (0x00 - 0xFFFE).

Parameter="sequence" value="image1;image2;image3(etc.)" — List of images used. See Image Formats for supported image types. The range is 1 - 65535 images.

Parameter="updateFreq" value ="number" — Specifies the href function call frequency (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. A value of 0.00 means update never.

Parameter="updateDelay" value ="number" — Specifies the delay time from when the page is loaded until the first href function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to the number in updateFreq.

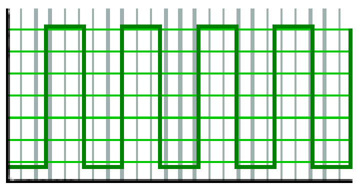
Parameter="waitForInit" value="CHECKED" or "UNCHECKED" — Specifies if the Image Sequence will wait for valid data before any image will be displayed on the LCD. If CHECKED, the Image Sequence will not display any image until the first packet of data is received. If UNCHECKED, or the attribute is not present, the Image Sequence starts out displaying the first image until the first packet of data is received. This is useful when the Href contains a UART or USB method and may not return immediately.

Optional Image Sequence Parameter Attributes:

Parameter="noSDRAM" value="CHECKED" or "UNCHECKED" — Specifies if the image(s) are NOT loaded into page-specific SDRAM prior to drawing the object. Has no effect if the images are already in the global cache. Having a copy of these images in the SDRAM speeds up the transition between the images and the invisible state. Default is UNCHECKED.

Parameter="loadImmediately" value="CHECKED" or "UNCHECKED" — Specifies if the images are loaded immediatly into the SDRAM on page load. This parameter is only applicable if the noSDRAM and the cacheImage parameters are UNCHECKED. If loadImmediately is CHECKED, then all the images in the Image Sequence will be loaded in the sdram for faster use after page load. However, this will delay the page load. If loadImmediately is UNCHECKED, then, the images won't be loaded in the sdram up front, but will do so as each image is requested, as long as noSDRAM is also UNCHECKED. Once each image has been displayed once, the image switching performance will be the same. The default for loadImmediately is UNCHECKED. Use this when the timing between images is critical, but you dont have the SDRAM to globally allocate cached images.

Line Graph


Line Graph Widget

The Line Graph Widget is a static 2-dimensional line graph that represents an array of bytes (or words) returned from an href function call. Each byte (or word) in the array represents a y-coordinate on the line graph. The x-coordinate is the index of that array distributed evenly throughout the graph. The line weight (thickness) is user-defined, as well as the number of samples in the x direction and the sampling rate. The line graph can be drawn over a background image. The graph is scaled in the y-direction based on yMin and yMax.

Multiple line graphs can be handled by a single Line Graph Widget. Each line graph shares the same yMin, yMax, xSamples and updateRate, but each graph needs its own href, lineWeight and linePattern values. The line graph that uses the first function specified in href has the line weight specified by the first number in lineWeight and the pattern specified by the first number in linePattern. The array specified in the second href function must be adjacent to the array specified in the first href function. For example, if displaying two line graphs, with an xSample of 10, the href functions would have to look something like this: Amulet:InternalRAM.bytes(0).array(10),Amulet:InternalRAM.bytes(10).array(10).

Line Graph Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Line Graph is to start out invisible or not. If the attribute is not present, then by default the Line Graph is visible. If the Line Graph starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="backgroundImage" value="image" — Image used as the Line Graph background image. Image dimensions should be the same as the dimensions of the Line Graph Widget. See Image Formats for supported image types. If no image specified, the default background will be the color specified in fillColor.

Parameter="fillColor" value= See color entry conventions — Specifies the desired background color. If no fillColor is specified, the default color is white. If backgroundImage specified, fillColor is not used.

Parameter="href" value="function" — The function called to retrieve the widget input. See Appendix B for all available functions for the Line Graph Widget. The function is called at an update rate specified by the updateRate attribute. If using multiple graphs, separate each function by a comma.

Parameter="lineColor" value= See color entry conventions — Specifies the desired line graph line color. If no line color is specified, the default color is black. If the alpha is FF, then the line color is fully opaque. If using multiple graphs, separate each color by a comma. Only the first color can be specified by the color picker. Subsequent colors must be entered manually, separated by a comma.

Parameter="lineWeight" value="pixels" — Defines the weight (thickness) of the active line graph in pixels. If using multiple graphs, separate each weight by a comma. The range is 1- 7 (0x01 - 0x07).

Parameter="updateFreq" value ="number" — Specifies the href function call frequency (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. A value of 0.00 means update never.

Parameter="updateDelay" value ="number" — Specifies the delay time from when the page is loaded until the first href function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to the updateRate value.

Parameter="xSamples" value="number" — Number of samples along the horizontal x-axis. The range is 2 - 125 (for a UART byte array), 2 - 62 (for a UART word array), 2 - 256 (for an InternalRAM byte array) and 2 - 256 (for an InternalRAM word array).
NOTE: The value must be less than the value specified in the WIDTH attribute of the tag.
TIP: In order for the graph to reach the far right of the dimensions of the Line Graph, refer to the following algorithm: # of pixels between plot points = (WIDTH of - 1) / (xSamples-1).

For example, if you want to completely fill the graph of a 101 pixel wide line graph, then you should have a number that is one greater than a perfect divisor of 100 as your xSamples. So, your xSamples could be 3, 5, 6, 11, 21, 26, 41, 51. If 6 is chosen, the # of pixels between plot points is (101-1)/(6-1) = 20 pixels.

Parameter="yMax" value="number" — Maximum value returned from the href function; must be greater than yMin. If the function returns a byte, the range is 1 - 255 (0x01 - 0xFF). If the function returns a word, the range is 1 - 65535 (0x01 - 0xFFFF).

Parameter="yMin" value="number" — Minimum value returned from the href function; must be less than yMax. If the function returns a byte, the range is 0 - 254 (0x00 - 0xFE). If the function returns a word, the range is 0 - 65534 (0x00 - 0xFFFE).

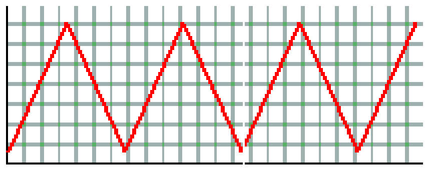
Optional Line Graph Parameter Attributes:

Parameter="axisColor" value= See color entry conventions — Specifies the desired linegraph axis color. See section on colors for more information. If no axis color is specified, the default color is black. This attribute is only applicable when showAxis is CHECKED.

Parameter="enableVertical" value="CHECKED" or "UNCHECKED" — Specifies if the Line Graph can have vertical lines or not. If CHECKED, then two data points are needed for every plot point. If a vertical line is not desired, then both points should be the same value. Basically, every x coordinate gets two y-coordinates. So, if xSamples is set to 10, you will need to furnish the Line Graph with 20 data points. If this attribute is not present, it defaults to false.

Parameter="showAxis" value="CHECKED" or "UNCHECKED" — Specifies if the Line Graph is to have an x and y axis frame. By default the Line Graph does not have a frame. Checking showAxis will cause the Line Graph to have a frame.

Line Plot


Line Plot Widget

The Line Plot Widget is a live 2-dimensional line plot that represents a byte (or word) returned from an href function call. The x and y-axis' are drawn based on the width and height. The line plots from left-to-right, continuously wrapping and does not get erased upon the wrap. The plot is updated at each new x-sample and the current location is kept via a vertical cursor that is the same height as the y-axis. The plot is scaled in the y-direction based on yMin and yMax.

Multiple line plots can be handled by a single Line Plot Widget. Each line plot shares the same yMin, yMax, xSamples and updateRate, but each plot needs its own href, lineColor, and lineWeight values. To add the multiple values to the href, lineColor, and lineWeight, use a comma to separate the values.

Line Plot Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Line Plot is to start out invisible or not. If the attribute is not present, then by default the Line Plot is visible. If the Line Plot starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="axisColor" value= See color entry conventions — Specifies the desired lineplot axis color. See section on colors for more information. If no axis color is specified, the default color is black. If the alpha is FF, then the axis color is fully opaque.

Parameter="backgroundImage" value="image" — Image used as the Line Plot background image. Image dimensions should be the same as the dimensions of the Line Plot Widget. See Image Formats for supported image types. If no image specified, the default background will be the color specified in fillColor.

Parameter="cursorColor" value= See color entry conventions — Specifies the desired lineplot cursor color. See section on colors for more information. If no cursor color is specified, the default color is black. If the alpha is FF, then the cursor color is fully opaque.

Parameter="fillColor" value= See color entry conventions — Specifies the desired background color. See section on colors for more information. If no fill color is specified, the default color is white. If backgroundImage specified, fillColor is not used. If the alpha is FF, then the fill color is fully opaque.

Parameter="href" value="function" — The function called to retrieve the widget input. See Appendix B for all available functions for the Line Plot Widget. The function is called at an update rate specified by the updateRate attribute. If using multiple plots, separate each function by a comma.

Parameter="lineColor" value=See color entry conventions — Specifies the desired lineplot line color. See section on colors for more information. If no line color is specified, the default color is black. If the alpha is FF, then the line color is fully opaque. If using multiple plots, separate each color by a comma. Only the first color can be specified by the color picker. Subsequent colors must be entered manually, separated by a comma.

Parameter="lineWeight" value="pixels" — Defines the weight (thickness) of the active line plot in pixels. If using multiple plots, separate each weight by a comma. The range is 1- 7 (0x01 - 0x07).

Parameter="updateFreq" value ="number" — Specifies the href function call frequency (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. A value of 0.00 means update never.

Parameter="updateDelay" value ="number" — Specifies the delay time from when the page is loaded until the first href function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the
delay time defaults to the value of updateFreq.

Parameter="xSamples" value="number" — Number of samples along the horizontal x-axis. The range is 2 - 638 (0x02 - 0x27E). NOTE: The value must be less than the value specified in the WIDTH attribute of the tag.

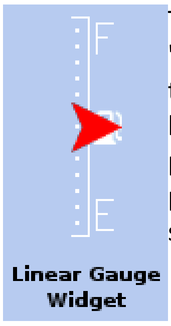
Parameter="yMax" value="number" — Maximum value returned from the href function; must be greater than yMin. If the function returns a byte, the range is 1 - 255 (0x01 - 0xFF). If the function returns a word, the range is 1 - 65535 (0x01 - 0xFFFF).

Parameter="yMin" value="number" — Minimum value returned from the href function; must be less than yMax. If the function returns a byte, the range is 0 - 254 (0x00 - 0xFE). If the function returns a word, the range is 0 - 65534 (0x00 - 0xFFFE).

Optional Line Plot Parameter Attributes:

Parameter="noAxis" value="CHECKED" or "UNCHECKED" — Specifies if the Line Plot is to have an x and y axis. If the attribute is not present, then by default the Line Plot has an x and y axis.

Linear Gauge



The Linear Gauge Widget uses two images (a background image and a pointer image) to create a custom "linear gauge". A byte (or word) returned from a function call is the input. The pointer travels linearly on the background image and is positioned based upon the value of the byte (or word) returned from an href function call. The height and width dimensions of the background image determine whether the pointer travels horizontally or vertically. By default, the longer dimension establishes the path that the pointer travels. Optionally, the orientation can be specified if it is desired to have the pointer travel the shorter dimension.

Linear Gauge Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Linear Gauge is to start out invisible or not. If the attribute is not present, then by default the Linear Gauge is visible. If the Linear Gauge starts out invisible, the only way to make it visiblare again is via the IWC method reappear().

Parameter="backgroundImage" value="image" — Image used as the linear background image. Image dimensions must be exactly the same as the dimensions of the Linear Gauge Widget. See Image Formats for supported image types.

Parameter="href" value="function" — The function called to retrieve the widget input. See Appendix B for all available functions for the Linear Gauge widget. The function is called at an update frequency specified by the updateFreq attribute.

Parameter="max" value="number" — Maximum value returned from the href function; must be greater than min. If the function returns a byte, the range is 1 - 255 (0x01 - 0xFF). If the function returns a word, the range is 1 - 65535 (0x01 - 0xFFFF).

Parameter="min" value="number" — Minimum value returned from the href function; must be less than max. If the function returns a byte, the range is 0 - 254 (0x00 - 0xFE). If the function returns a word, the range is 0 - 65534 (0x00 - 0xFFFE).

Parameter="minAt" value=""LEFT" or "RIGHT" or "TOP" or "BOTTOM"" — Determines where the minimum value of the linear gauge is located. When the minimum value is returned, the pointer is located at the minAt location. As the value increases, the pointer travels the length of the linear background image until the maximum value is reached and the pointer is then located at the opposite extreme of the minAt location. Default values are LEFT for horizontal linear gauges and BOTTOM for vertical linear gauges. The options are:

- LEFT — The pointer value increases from left to right. (horizontal linear gauge only)
- RIGHT — The pointer value increases from right to left. (horizontal linear gauge only)
- TOP — The pointer value increases from top to bottom. (vertical linear gauge only)
- BOTTOM — The pointer value increases from bottom to top. (vertical linear gauge only)

Parameter="pointerImage" value="image" — Image used as the pointer which travels across the linear background image. See Image Formats for supported image types.

Parameter="updateFreq" value ="number," — Specifies the href function call frequency (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. A value of 0.00 means update never.

Parameter="updateDelay" value ="number" — Specifies the delay time from when the page is loaded until the first href function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If the second number is not specified, then the delay time defaults to the value of updateFreq.

Parameter="waitForInit" value="CHECKED" or "UNCHECKED" — Specifies if the Linear Gauge pointer will wait for valid data before being displayed on the linearImage. If CHECKED, the Linear Gauge pointer will not display until valid data is received. If UNCHECKED, or the attribute is not present, the Linear Gauge pointer starts at the minAt location until the first packet of data is received. This is useful when the Href contains a UART or USB method and may not return immediately.

Optional Linear Gauge Parameter Attributes:

Parameter="orientation" value="HORIZONTAL" or "VERTICAL" — Specifies if the pointer icon is to travel horizontally or vertically. The orientation
parameter will override the orientation determined by the height and width dimensions.

Parameter="pointerOffset" value="number" — Specifies the number of pixels from the center of the channel the handle is located. If a vertical slider, positive numbers shift the handle to the right and negative numbers shift it to the left. If a horizontal slider, positive numbers shift the handle to the bottom and negative numbers shift it to the top. [The range is -100 through 100].

Numeric Field

The Numeric Field Widget uses a byte (or word) returned from an href function call to display a mixture of static text and a live number. The string is input using the standard C printf format. The Numeric Field Widget can display in integer, hexadecimal, and floating-point formats. Like printf, the variable is entered using the % character. The first digit following the % specifies the number of character spaces allocated to the live numeric field (including a decimal point, and plus (+) or minus (-) symbols). You can also have static text preceding and following the live numeric field. For example, to create a numeric field that displays "Output = 2.25 Volts", the printf field would read: "Output = %5.2f Volts". In this example, "5" specifies the number of character spaces, ".2" specifies the number of digits to the right of the decimal, and "f" specifies floating-point numbers.
If the width of the numeric field widget is less than required, the string will be truncated. Make sure your dimensions are large enough to hold all your text and numbers.

Numeric Field Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the Numeric Field is to start out invisible or not. If the attribute is not present, then by default the Numeric Field is visible. If the Numeric Field starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="border" value="number" — Specifies width, in pixels, of the border around the dimensions of the numeric field. Default is 0, meaning no border.

Parameter="borderColor" value= See color entry conventions — Specifies the desired Numeric Field border color. See section on colors for more information. If no border color is specified, the default color is black. Only applicable if border is set to something other than 0. If the alpha is FF, then the border is fully opaque.

Parameter="fillColor" value= See color entry conventions — Specifies the desired Numeric Field fill color. See section on colors for more information. If no fill color is specified, the default color is the current background color. If the alpha is FF, then the fillColor is fully opaque.

Parameter="font" value="font, font size" — Specifies the font and font size used for the static text defined in printf. See Amulet GEM Font Converter for more information regarding the creation of .auf font files. Default is Bitstream Vera Sans. The default font size for the static text is 12pt.

Parameter="fontColor" value= See color entry conventions — Specifies the desired static text font color. See section on colors for more information. If no font color is specified, the default color is black.

Parameter="fontStyle" value="PLAIN" or "BOLD" or "ITALIC" — Specifies the style associated with the font of the static text defined in printf. PLAIN overrides any other style. The available font styles are:

- PLAIN — The option text uses the standard font. (i.e. text)
- BOLD — The option text is bold. (i.e. text)
- ITALIC — The option text is italicized. (i.e. text)

Parameter="href" value="function" — The function called to retrieve the widget input. See Appendix B for all available functions for the Numeric Field Widget. The function is called at an update rate specified by the updateFreq attribute.

Parameter="horizontalAlign" value="LEFT" or "CENTER" or "RIGHT" — Specifies the horizontal alignment of the string associated with the printf attribute within the Numeric Field dimensions. Only one value is allowed; you cannot mix horizontal alignments. Default is LEFT.

Parameter="max" value="number" — Maximum value returned from the href function; must be greater than min. If the function returns a byte, the range is 1 - 255 (0x01 - 0xFF). If the function returns a word, the range is 1 - 65535 (0x01 - 0xFFFF).

Parameter="maxFld" value="number" — Specifies what the Numeric Field displays when the maximum value is returned from the href function call. When using hexadecimal, you must precede the minFld number with 0x. By default, the 0x will NOT be displayed unless the "#" flag is used in the printf field. The maxFld value does NOT have to be greater than the minFld value. Range is -65535 to 65535 when using integers and floating point numbers and 0 to 0xffff when using hex numbers. See note below regarding the span between minFld and maxFld.

Note regarding the span between minFld and maxFld: When using integers and floating point numbers, the value stored by the Amulet OS is a 16-bit number. When using a floating point number, the decimal point is removed and the digits to the right of the decimal point are concatenated with those to the left of the decimal point. So, 655.35 is stored as 65535 (the maximum 16-bit number). In addition, the span between minFld and maxFld is limited to a 16-bit number. For example if the min is -65535, then the largest max can be is 0 (which would result in a span of 65535). Therefore, even though -65535 is a valid min and 65535 is a valid max, the span is larger than a 16-bit number (causing an Amulet compiler error).

Parameter="min" value="number" — Minimum value returned from the href function; must be less than max. If the function returns a byte, the range is 0 - 254 (0x00 - 0xFE). If the function returns a word, the range is 0 - 65534 (0x00 - 0xFFFE).

Parameter="minFld" value="number" — Specifies what the numeric field displays when the minimum value is returned from the href function call. When using hexadecimal, you must precede the minFld number with 0x. By default, the 0x will NOT be displayed unless the "#" flag is used in the printf field. The minFld value does NOT have to be less than the maxFld value. (Negative slope is permissible.) Range is -65535 to 65535 when using integers and floating point numbers, and 0 to 0xffff when using hex numbers. See note regarding the span between minFld and maxFld.

Formatting Value	minFld Example	maxFld Example
%3i	-20	10
%3i	200	999
%3i	200	-40
%5.2f	-2.00	0.00
%5.2f	0.00	25.00
%6.2f	-50.00	50.00
%#4x	0x00	0xff
%2X	0xFF	0xAA

Table 1. Numeric Field formatting examples using the right-justified default. (The implied ranges are arbitrary.)

Parameter="printf" value="text %format text" — Specifies the text and the formatted numeric field to be displayed (similar to the standard C program printf command). The Numeric Field Widget can display integer, hexadecimal, and floating-point numbers. See Using Amulet Formatted Text for more formatting options.
To display integers, the format is %ai, where "a" is the number of character spaces, and "i" specifies integers. With floating-point numbers, the format is %a.bf, where "a" is the total number of character spaces, "b" is the number of digits to the right of the decimal point, and "f" specifies floating-point numbers. With hexadecimal numbers, the format is %aX or %ax, where "a" is the number of character spaces, and "X" specifies that the hexadecimal digits will be upper case (A-F), while "x" specifies lower case (a-f).

NOTE: To display a % symbol in the numeric field, use %% (e.g. Duty Cycle(%%)=%5.2f will display Duty Cycle(%)=99.99). There are also flags that change the numeric field format. Format flags are entered between the % and the character space specification. The flags are: "-", "+", "0", "#" and " ". The flags are defined, as follows:

- "-" — specifies that the numeric field is left-justified. (The default is right-justified.)
- "+" — specifies that positive numbers are preceded with a plus sign.
- "0" — specifies that a right-justified numeric field lead with zeroes. (The default is right-justified with leading spaces).
- "#"— specifies that displayed hexadecimal numbers are preceded with 0x.
- " " — a blank space specifies that a left-justified numeric field lead with a single space when displaying a positive number, and lead with a negative sign (-) when displaying a negative number.

Parameter="updateFreq" value ="number" — Specifies the href function call frequency (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. A value of 0.00 means update never.

Parameter="updateDelay" value ="number" — Specifies the delay time from when the page is loaded until the first href function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to the value in updateFreq.

Parameter="verticalAlign" value="TOP" or "MIDDLE" or "BOTTOM" — Specifies the vertical alignment of the string associated with the printf attribute within the Numeric Field dimensions. Only one value is allowed; you cannot mix vertical alignments. Default is TOP.

Parameter="waitForInit" value="CHECKED" or "UNCHECKED" — Specifies if the Numeric Field will wait for valid data before being displayed on the LCD. If CHECKED, the Numeric Field will not display any dynamic numbers until the first packet of data is received. If UNCHECKED, or the attribute is not present, the Numeric Field starts out displaying the minimum value until the first packet of data is received. This is useful when the Href contains a UART or USB method and may not return immediately.

Optional Numeric Field Parameter Attributes:

Parameter="context" value="String"— Specifies the context of the label string for translation. No context is not the same as an empty context. See Context under the Localization feature for more info.

Parameter="colorInvert" value="REGION" or "STRING" or "NONE" — Specifies if the string is shown with alternate colors. If REGION selected, the fillColorAlt and fontColorAlt will be used instead of fillColor and fontColor. If STRING selected, only fontColorAlt will be used instead of fontColor. If NONE selected, String Field will use fillColor and fontColor. Only one value is allowed; you cannot mix color inversion properties. Default is NONE.

Parameter="fillColorAlt" value= See color entry conventions — Specifies the desired Numeric Field fill color if colorInvert set to REGION. See section on colors for more information. If no alternate fill color is specified, the default alternate fill color is the logical inverse of the current fillColor.

Parameter="fontColorAlt" value= See color entry conventions — Specifies the desired font color if colorInvert set to either REGION or STRING. See section on colors for more information. If fontColorAlt is not specified, the default alternate font color is the logical inverse of the current fontColor.

String Field

The String Field Widget calls a function that returns either a null-terminated string of UTF-8 characters, or a one-byte index into a list of pre-built strings. The string field can display a mixture of static text and a dynamic string. The static text is input using the standard C printf format. Like printf, the conversion specification begins with a % and ends with the conversion character "s".

If a UTF-8 string is received, then the acquired string is copied to the dynamic string buffer, which has a length that defaults to 25 bytes. This dynamic string is inserted where the conversion specification (i.e. %s) resides in the printf string. To increase the maximum number of characters up to 1000, see precision below.
If a byte is received, the pre-built string that has the same value as the acquired byte is inserted where the conversion specification resides in the printf string. The values associated with each pre-built string are specified similar to C's "enum" specifier, where the first string will have a value of 00, the second item 01, etc, unless an explicit value is given.

The static part of the string is automatically passed through localization, if specified, and the dynamic parts of the string can be passed through the localization if desired. Once translated the string is drawn into the canvas, an off-screen buffer to store the pixel data that is usually the size of the stringField widget, but may be expanded for example to enable scrolling text and moved using one of the 6 canvas operations addCanvasX/Y, setCanvasX/Y, and subCanvasX/Y.

If the width of the String Field canvas is less than required, and the height of the string field canvas is tall enough, the string will wrap automatically. If there is not enough room to wrap, the string will be truncated. User-defined wraps can be specified by entering "\n" within the static text, or by sending a 0x0A in the dynamic text, at the spot you would like the wrap to occur. There is a maximum of 20 wrapped lines per String Field.

String Field Parameter Attributes:

Parameter="invisible" value="CHECKED" or "UNCHECKED" — Specifies if the String Field is to start out invisible or not. If the attribute is not CHECKED, then by default the String Field is visible. If the String Field starts out invisible, the only way to make it visible again is via the IWC method reappear().

Parameter="border" value="number" — Specifies width, in pixels, of the border around the dimensions of the String Field. Default is 0, meaning no border.

Parameter="borderColor" value= See color entry conventions — Specifies the desired String Field border color. See section on colors for more information. If no border color is specified, the default color is black. Only applicable if border has a value of 1 or greater.

Parameter="fillColor" value= See color entry conventions — Specifies the desired String Field fill color. See section on colors for more information. If no fill color is specified, the default color is the current background color.

Parameter="font" value="font, font size" — Specifies the font and font size used for the String Field text. The corresponding .auf file must be included in the Amulet/Color/Configuration/Fonts folder, the root folder, or the root/Fonts folder. See Amulet GEM Font Converter for more information regarding the creation of .auf files. Default is Bitstream Vera Sans. The default font size for the String Field text is 12pt.

Parameter="fontColor" value= See color entry conventions — Specifies the desired String Field text color. See section on colors for more information. If no font color is specified, the default color is black.

Parameter="fontStyle" value="BOLD" or "ITALIC" or "PLAIN" — Specifies the style associated with the string field font. The available font styles are:

- BOLD — The option text is bold. (i.e. text)
- ITALIC — The option text is italicized. (i.e. text)
- PLAIN — The option text is standard font. (i.e. text)

This attribute defines the default font style of both the static text defined in the printf attribute as well as the dynamic string returned from the href function. If it is desired to change the dynamic string's font style at run time, see the UART Protocol documentation regarding the font style escape byte.

Parameter="horizontalAlign" value="LEFT" or "CENTER" or "RIGHT" — Specifies the horizontal alignment of the string associated with the printf attribute within the String Field dimensions. Only one value is allowed; you cannot mix horizontal alignments. Default is LEFT.

Parameter="href" value="function" — The function called to retrieve the widget input. See Appendix B for all available functions for the String Field Widget. The function is called at an update rate specified by the updateFreq attribute.

Parameter="printf" value="text %format text" — Specifies the text and the formatted string field to be displayed (similar to the standard C program printf command). See Using Amulet Formatted Text for more formatting options. The string is input using the standard C printf format. Like printf, the conversion specification begins with a % and ends with the conversion character "s". By default, the dynamic string can be a maximum of 25 characters in length. To increase the maximum number of characters, see precision below. Between the % and the conversion character there may be, in order:

- A minus sign, which specifies left adjustment of the dynamic string.
- A number that specifies the minimum field width. The dynamic string will be printed in a field at least this wide. If necessary it will be padded on the left (or right, if left adjustment is called for) to make up the field width.
- A period, which separates the field width from the precision.
- A number, the precision, that specifies the maximum number of characters to be printed from a string. By default, the precision length is 25. If a dynamic string longer than 25 characters is desired, set the precision to the maximum length of the string. The maximum precision size is 500.

NOTE ON WRAPPING: If the width of the String Field widget is less than required, and the height of the string field widget is tall enough, the string will wrap automatically. If there is not enough room to wrap, the string will be truncated. User-defined wraps can be specified by entering "\n" within the static text, or by sending a 0x0A in the dynamic text, at the spot you would like the wrap to occur. There is a maximum of 20 wrapped lines per String Field. The following table shows the effect of a variety of specifications in printing "hello, world" (12 characters). We have put colons around each field so you can see its extent.

```
:%s:          :hello, world:
:%10s:        :hello, world:
:%.10s:       :hello, wor:
:%-10s:       :hello, world:
:%.15s:       :hello, world:
:%-15s:       :hello, world   :
:%15.10s:     :     hello, wor:
:%-15.10s:    :hello, wor     :
```

NOTE: To display a literal % symbol in the string field, use a double percent command in the string (e.g. %s at 100 %% displays your string at 100 %). To display a literal \ symbol in the string field, use a double backslash command in the string (e.g. %s \\ 100 displays your string \ 100).

Parameter="updateFreq" value ="number" — Specifies the href function call frequency (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. A value of 0.00 means update never.

Parameter="updateDelay" value ="number" — Specifies the delay time from when the page is loaded until the first href function call (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. If this number is not specified, then the delay time defaults to the value in updateFreq.

Parameter="verticalAlign" value="TOP" or "MIDDLE" or "BOTTOM" — Specifies the vertical alignment of the string associated with the printf attribute within the String Field dimensions. Only one value is allowed; you cannot mix vertical alignments. Default is TOP.

Parameter="waitForInit" value="CHECKED" or "UNCHECKED" — Specifies if the String Field will wait for valid data before being displayed on the LCD. If CHECKED, the String Field will not display any text, static or dynamic, until the first packet of data is received. If UNCHECKED, or the attribute is not present, the String Field starts out displaying only the static text, if any specified, until the first packet of data is received. This is useful when the Href contains a UART or USB method and may not return immediately.

Optional String Field Parameter Attributes:

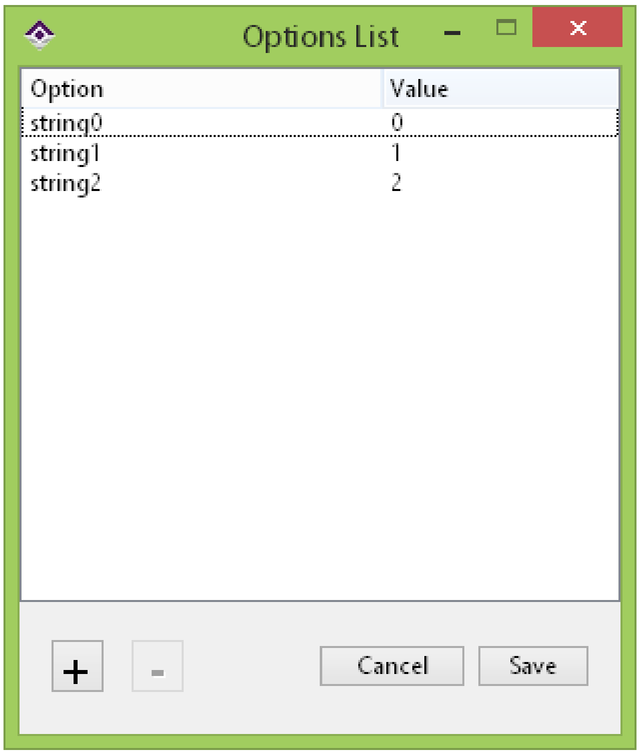
Parameter="canvasWidth" value="number"— Specifies the width of the canvas, an off screen buffer to which the text is drawn. You can move the canvas around using the canvas control IWCs, such as setCanvasX, which can be found here. If not specified, defaults to the width of the StringField.

Parameter="canvasHeight" value="number"— Specifies the height of the canvas, an off screen buffer to which the text is drawn. You can move the canvas around using the canvas control IWCs, such as setCanvasX, which can be found here. If not specified, defaults to the height of the StringField.

Parameter="context" value="String"— Specifies the context of the static portion of the printf string and any "options" parameter strings for translation. No context is not the same as an empty context. See Context under the Localization feature for more info.

Parameter="initialCondition" value="string" — Specifies which options string is initially used when the page is loaded. It is acceptable to use the options and initialCondition attributes even when requesting the Amulet:UART.string(x).value(). The default string will be used until a valid string is received.

Parameter="options" value="see image below"— Specifies the strings and its associated value that can be displayed when using an href that returns a numerical value. The string in the options list whose value equals the value returned from the href function is displayed.



Option	Value
string0	0
string1	1
string2	2

Parameter="colorInvert" value="REGION" or "STRING" or "NONE" — Specifies if the string is shown with alternate colors. If REGION selected, the fillColorAlt and fontColorAlt will be used instead of fillColor and fontColor. If STRING selected, only fontColorAlt will be used instead of fontColor. If NONE selected, String Field will use fillColor and fontColor. Only one value is allowed; you cannot mix color inversion properties. Default is NONE.

Parameter="fillColorAlt" value= See color entry conventions — Specifies the desired String Field fill color if colorInvert set to REGION. See section on colors for more information. If no alternate fill color is specified, the default alternate fill color is the logical inverse of the current fillColor.

Parameter="fontColorAlt" value= See color entry conventions — Specifies the desired font color if colorInvert set to either REGION or STRING. See section on colors for more information. If fontColorAlt is not specified, the default alternate font color is the logical inverse of the current fontColor.

Intrinsic Values

Control and View Widgets have intrinsic values which can be used to exchange information.

For a Control widget, the intrinsic value represents the data that the widget could pass in an HREF. For example, the intrinsic value of a slider is the relative position of the handle, scaled between the min and max values. In the slider example, the intrinsic value changes every time the user moves the slider's handle. Other widgets, like a fuction button, have a more static value set at compile time.

Most Control Widgets' intrinsic value can be either a BYTE, WORD or STRING. The exceptions to this are the Slider, CustomSlider, ImageScroller and the grouped CheckBox widgets, which can only have numbers (BYTES or WORDS) for their intrinsic value. If the href function call is BYTE specific, (i.e. Amulet:uart0.byte(5).setvalue()), then the range of the intrinsic value should be 0-255(0x00-0xFF). You can alternately specify a BYTE by putting an ASCII character between single quotes (i.e. 'A', which would be the BYTE equivalent of 0x41 or 65 decimal.) If the intrinsic value can become greater than 255, then you should use WORD specific function calls. If the href function call is WORD specific, (i.e. Amulet:UART.word(5).setvalue()), then the range of the intrinsic value is 0-65535(0x0000-0xFFFF).

A View widget's intrinsic value represents they data it is currently displaying. In other words, the value returned from the last HREF call is copied to the intrinsic value of the widget. If the Widget's HREF is pointing at an InternalRAM byte, but the widget has not been updated since the InternalRAM byte has been updated, the Widget's intrinsic value will reflect the old value. This is particularly useful in more complex control of layered objects where the update rate is 0 because the automatic refresh may cause unwanted drawing.

Examples:
A view widget's HREF that displays the intrinsic value of a slider:
Amulet:document.MySlilder_1.value()
A control wiidget's HREF that writes its intrinsic value to an InternalRAM byte:
Amulet:InternalRAM.byte(0).setValue(instrinsicValue)
By default, empty parenthesis on a method which requires a parameter will pass the instrinsic value. So, the above is equivalent to:
Amulet:InternalRAM.byte(0).setValue()

Entering Static Strings

If the href function call is STRING specific, (i.e. Amulet:uart0.string(5).setvalue()), then the intrinsic value should be specified between double quotes (i.e. "My string"). The maximum length of a string value is 254 bytes.

Backlight Control

If using the Amulet MK-07C-HP Module or one of the standard Amulet modules that use PWM0 to control the LCD backlight, the command **Amulet:lcdBacklight.setValue(*y*)** can be used to control the LCD backlight intensity, where *y* can be any value from 0 to 255. 0 is backlight off and 255 is full bright.

Once the backlight intensity has been set, the value can be saved to the embedded flash using the command **Amulet:lcdBacklight.saveToFlash()**, which will allow the backlight to start out at that same intensity upon a power cycle or reset of the module.

Inter-Widget Communication

All widgets use the href parameter to specify a function call or a group of function calls. One type of function call is Inter-Widget Communication (IWC). IWC's allow one Amulet widget to invoke the methods of another Amulet widget. (See Appendix B for a comprehensive listing of all available function calls. See Appendix C for a detailed listing of which IWC methods apply to which widget or object, as well as a more in-depth description of what the IWC method actually does.)

IWC borrows some of its syntax from Java Script, a scripting language used within HTML. All IWC function calls start with "Amulet:document.". The "Amulet:" signifies that what follows is an Amulet specific command. The "document." represents the Document Object Model. Every Page has an object called document, which is one large object that contains every object (widget, text, images etc...) that is on that Page. To put it simply, "document." signifies that we're dealing with the current Page.

IWC also borrows concepts from Java, an Object-Oriented Programming(OOP) language. Each widget can be thought of as an object. As in OOP, each object has its own set of data and a set of well-defined interfaces to that data. As in Java, IWC refers to these interfaces as methods. Methods are just functions that are specific to a particular object. Each object has its own set of methods. The same nomenclature as Java is used, where a method is called by using the object's name followed by the dot operator, followed by the method.

The href nomenclature for IWC's is **Amulet:document.widgetName.method()**

Where: **Amulet:** is the Amulet script escape telling the compiler that Amulet specific commands follow.

document is the name of the Document Object Model, the generic name for the current page.

widgetName is the user-defined name of the called widget.

method() is the name of the method the called widget is to perform.

Control Widgets/Objects, using IWC, can send data to other widgets/objects or invoke other widgets'/objects' methods. View Widgets, using IWC, can request data from other widgets.

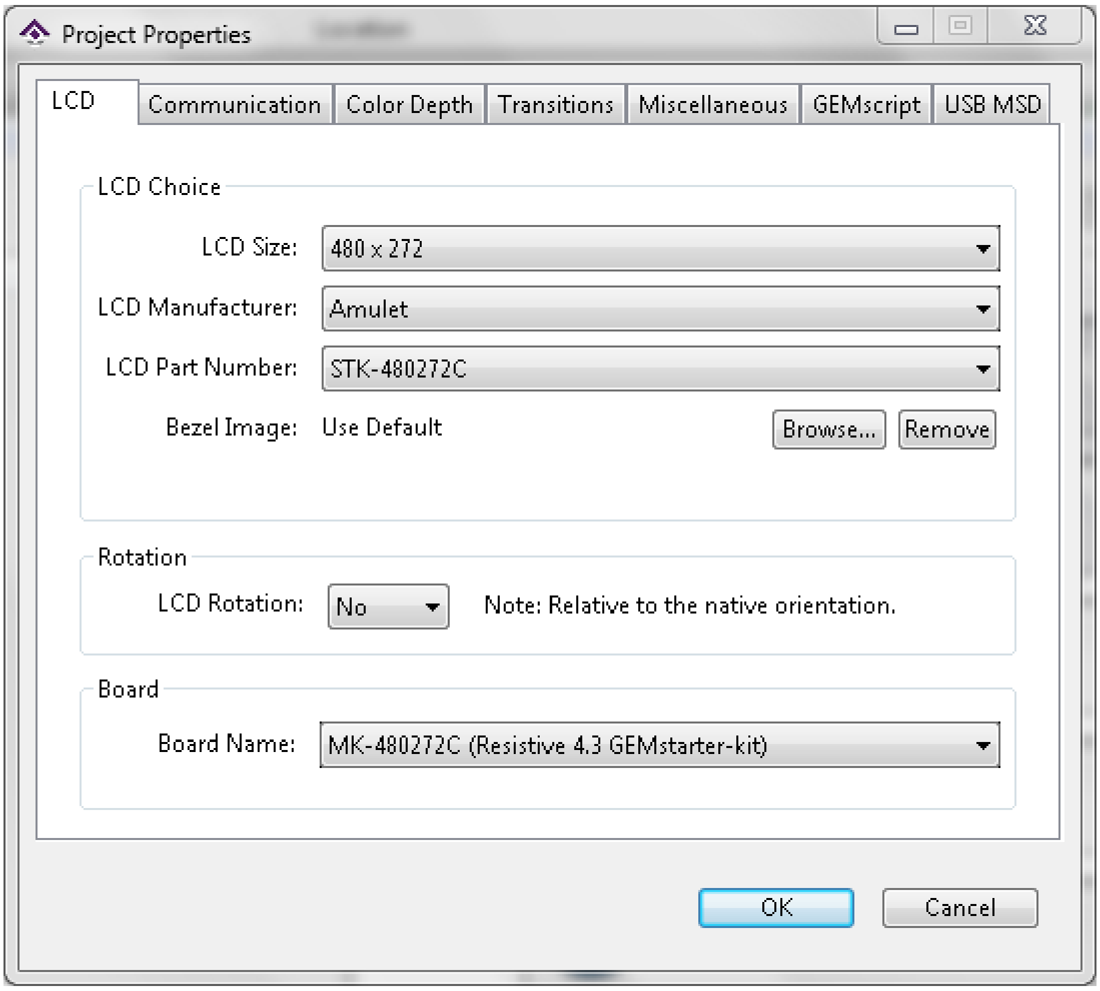
In addition to widgets, there are other objects that have methods that can be invoked using IWC. Images and animated images, referred to as View Objects, have methods that Control Widgets/Objects can call.

IWC Example

The simplest method to enter IWC calls in the href field is to use Amulet's auto-complete feature by hitting the Tab key. As an example of how quickly an IWC call can be added, follow the steps below:

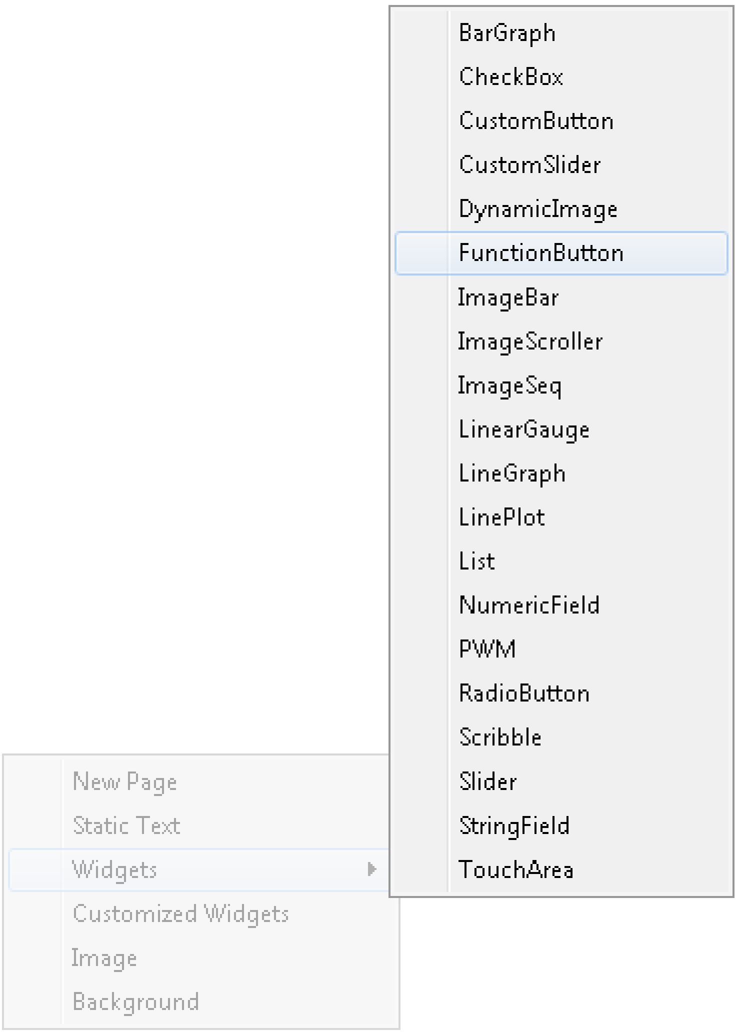
1) Open GEMstudio Pro and go to File > New Project

2) Press the OK button using the last used display settings (since that is not important for this example)



LCD Size: 480 x 272
LCD Manufacturer: Amulet
LCD Part Number: STK-480272C
Bezel Image: Use Default
LCD Rotation: No
Note: Relative to the native orientation.
Board Name: MK-480272C (Resistive 4.3 GEMstarter-kit)

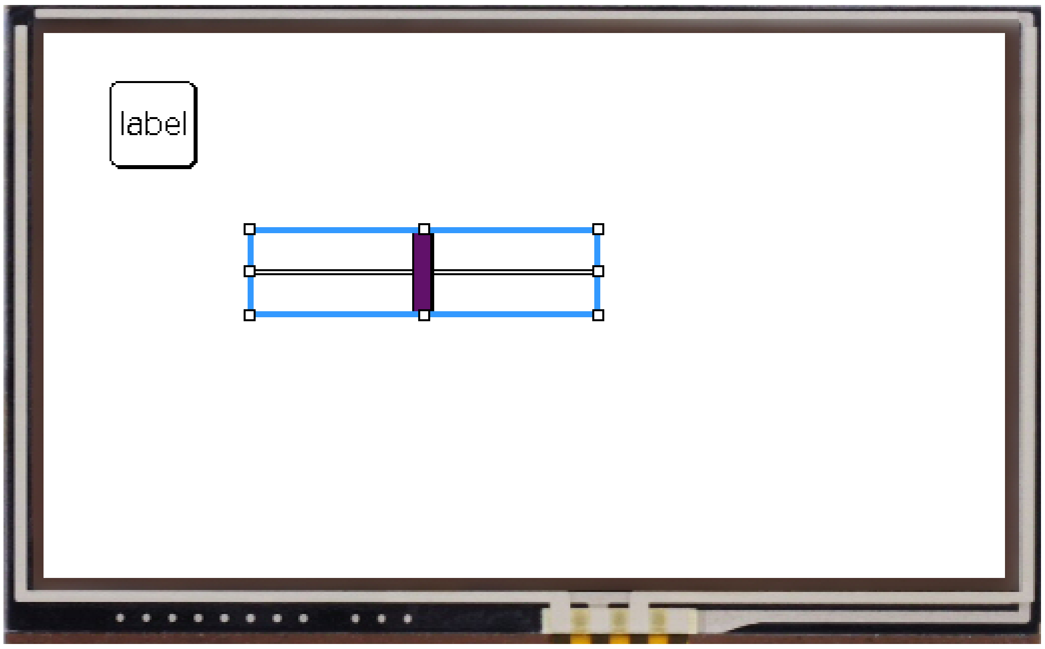
3) Press the ➕ button in the lower left to add an object.



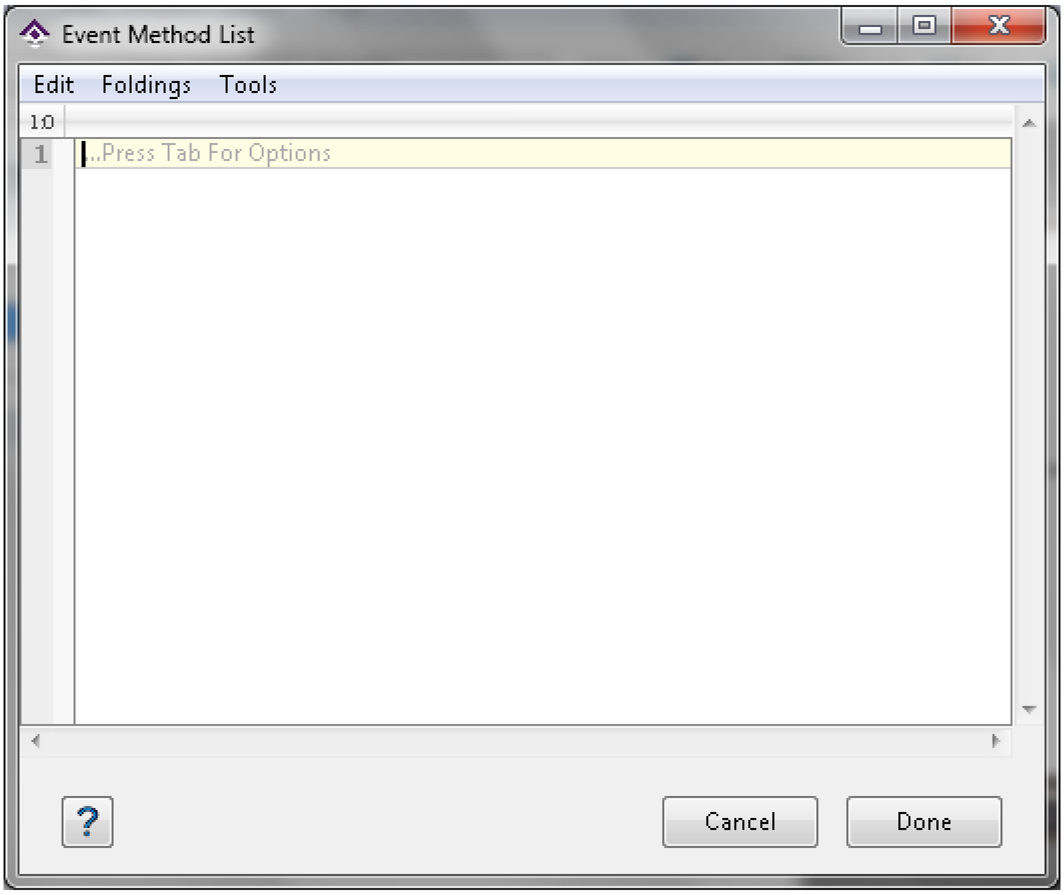
4) Navigate to Widgets and select FunctionButton

5) Follow the same steps to add a Slider.

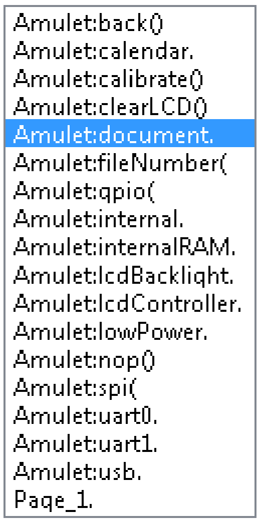
6) Go to the layout window and move the button to the upper left-hand corner so the two widgets are not on top of each other.



7) Select the button in the layout and the left-hand pane should now have the Function Button active with all of its attributes. Click on the empty href attribute and a new window will pop up. Notice the gray text that says ...Press Tab For Options. This is the key that auto-complete is ready to help.

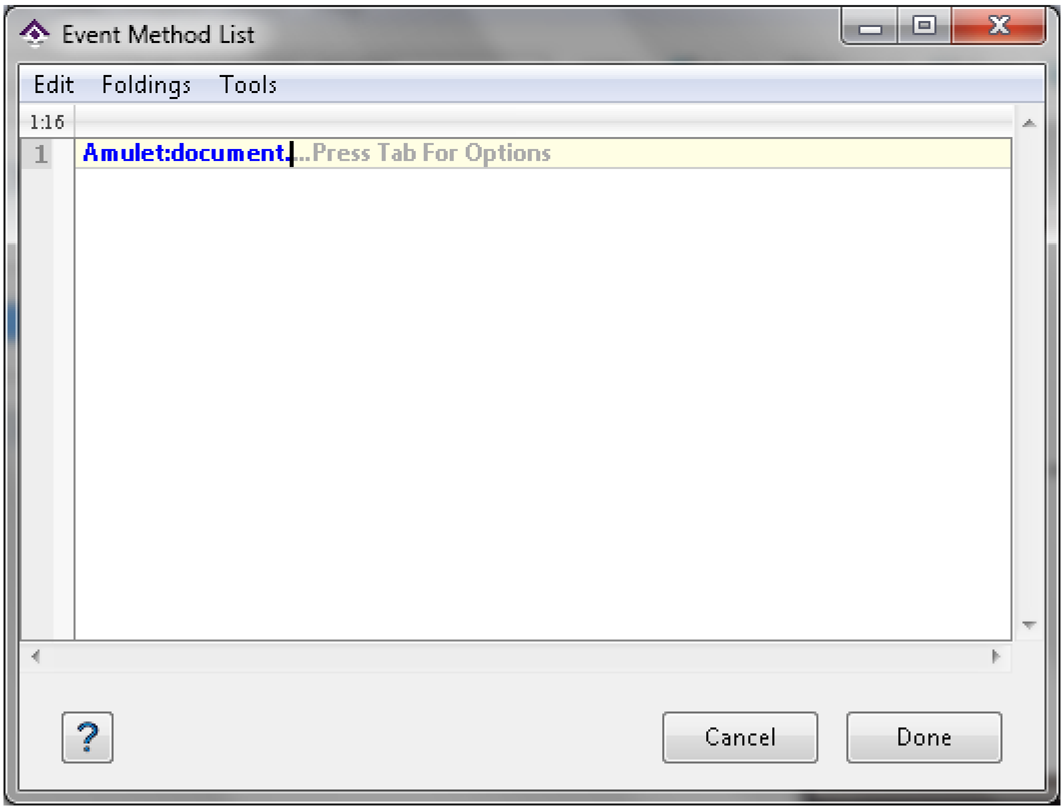


8) Press the Tab key and a new menu of all the available functions available for this Function Button to call will appear. Since this is an example on IWC function calls, select Amulet:document.



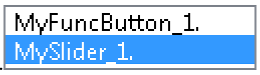
```
Amulet:back()
Amulet:calendar.
Amulet:calibrate()
Amulet:clearLCD()
Amulet:document.
Amulet:fileNumber(
Amulet:gpio(
Amulet:internal.
Amulet:internalRAM.
Amulet:lcdBacklight.
Amulet:lcdController.
Amulet:lowPower.
Amulet:nop()
Amulet:spi(
Amulet:uart0.
Amulet:uart1.
Amulet:usb.
Page_1.
```

9) The href editor will now have "Amulet:document." printed out and based on the gray "...Press Tab For Options" the auto-complete is ready to help some more.



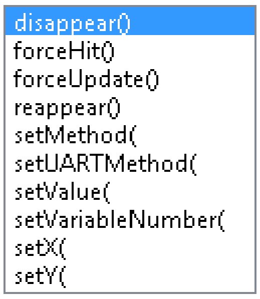
```
Amulet:document....Press Tab For Options
```

10) Press Tab again and the auto-complete will show the names of the objects on this page that the Function Button is capable of referencing. In this example, there are only two objects, and one of them is the Function Button itself.

Choose the Slider named MySlider_1 

```
MyFuncButton_1.
MySlider_1.
```

11) The href editor now has "Amulet:document.MySlider_1." printed out and we can see the auto-complete is ready again. Hit the Tab key and all the IWC methods that the Slider has available that the Function Button can call is in the next auto-complete menu. For this simple example, select disappear().



```
disappear()
forceHit()
forceUpdate()
reappear()
setMethod(
setUARTMethod(
setValue(
setVariableNumber(
setX(
setY(
```

12) The href editor finally has "Amulet:document.MySlider_1.disappear()" printed out and since the function call is complete, the gray text of "...Press Tab For Options" does not show up any longer. Using auto-complete, we entered an entire IWC function call and never had to type a single thing or even remember the exact syntax, for that matter.

13) To finish the example, hit the Run button and the GEMplayer will run the project you just created. GEMplayer should show the Function Button and the Slider in the exact same locations they were within the GEMstudio Pro layout view. Press the Function Button using your mouse and the Slider should disappear from the display. That is Inter-Widget Communication!

IWC Method Types

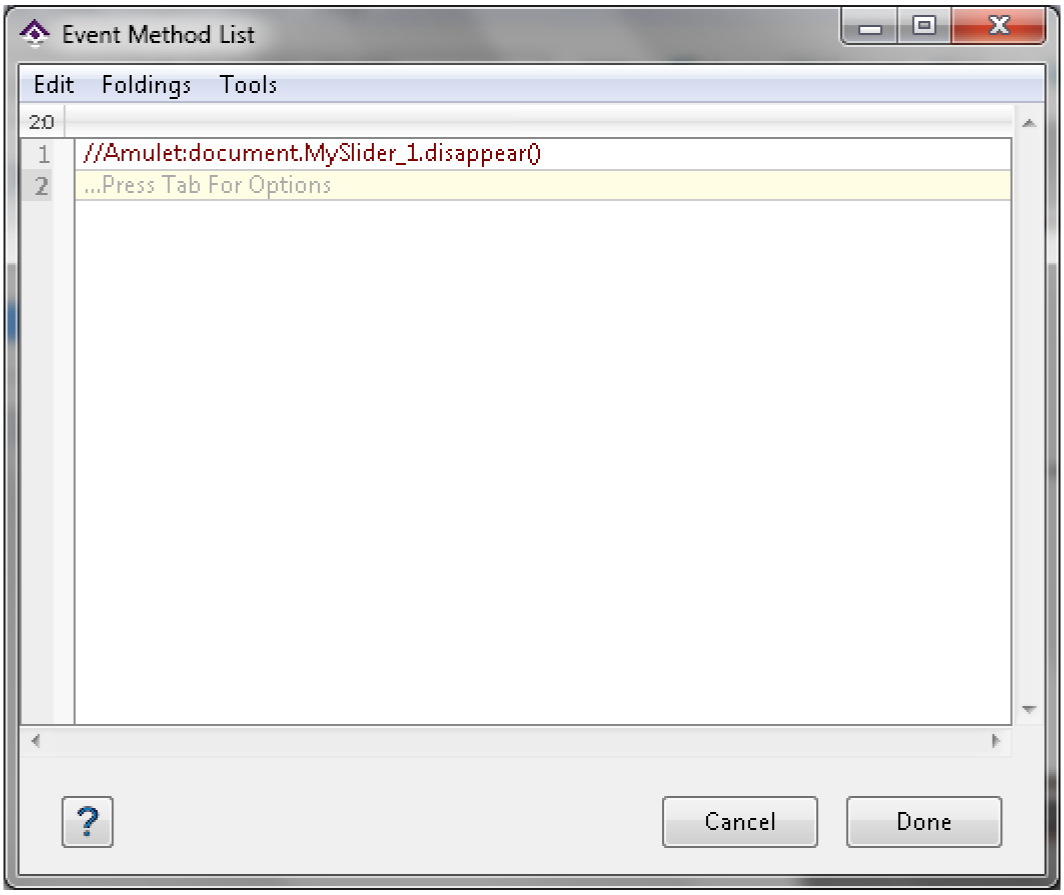
There are 3 different types of IWC methods. 1) methods that do not require parameters; 2) methods that require parameters; 3) methods that return a value.

1) Methods that do not require arguments (parameters):

There are a number of IWC methods that do not require any parameters passed to them, such as **disappear()**. When using auto-complete, if the function ends with () then that means there are no arguments needed.

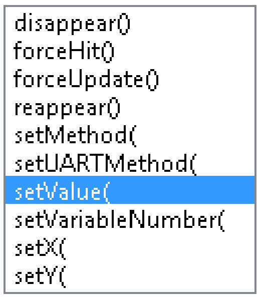
2) Methods that require arguments (parameters):

There are also a number of IWC methods that do require a parameter to be passed to them, such as **setValue(x)**. The parameter can be one of many different options. Again, looking at the auto-complete options will let you know what kind of parameters can be sent to each IWC method. For instance, using the same example as above, we can change the functionality of the Function Button's href by clicking on the href attribute again in the left-hand pane. For the sake of this example, use // to comment out the previous function call and then you should see "...Press Tab For Options" again.

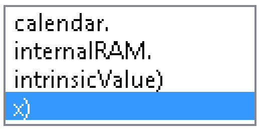


```
//Amulet:document.MySlider_1.disappear()
...Press Tab For Options
```

Go through the same steps as before to get "Amulet:document.MySlider_1." to print out. Hit the Tab key one more time and the same auto-complete menu will show up again where we picked "disappear()", but this time select "setValue(".

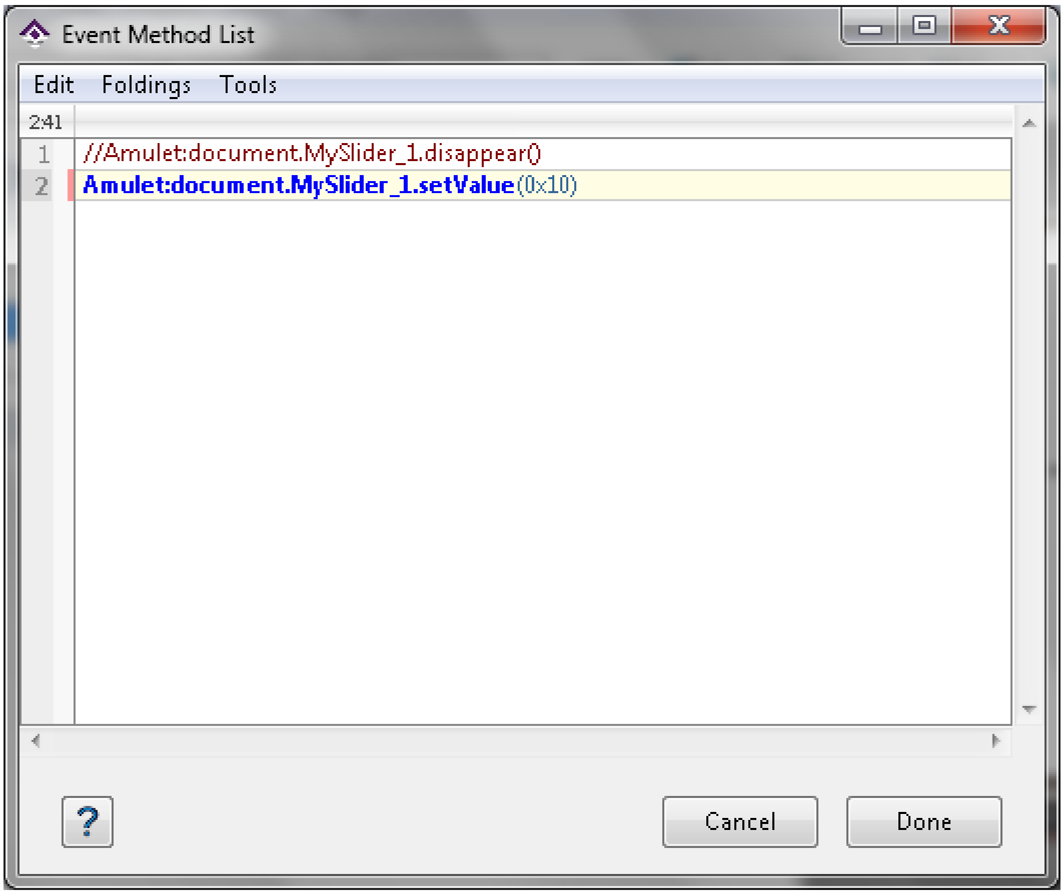


Hit the Tab key once more to get a menu of the parameters that can be used in the setValue method.



Based on this menu, the setValue method can take a calendar object, an internalRAM object, intrinsicValue, and x as its parameter. Notice that "calendar." and "internalRAM." both end with dots (.), meaning that they are objects, and if selected, another auto-complete menu will show up to allow you to drill down within the object. "intrinsicValue" is the current value of the calling object. In this case, a Function Button usually isn't given an intrinsic value, as "intrinsicValue" is most often used to determine the current value of an object that can change its value, such as a Slider or List widget. "x" is a place holder for an absolute integer value.

Leaving x in the href function call of "Amulet:document.MySlider_1.setValue(x)" will result in a compiler error because x is not a valid number. For this example, we will replace the x with a value of 0x10. Note: The absolute number can be written in either decimal or hexadecimal (precede with 0x) format. The href window should now look like this:



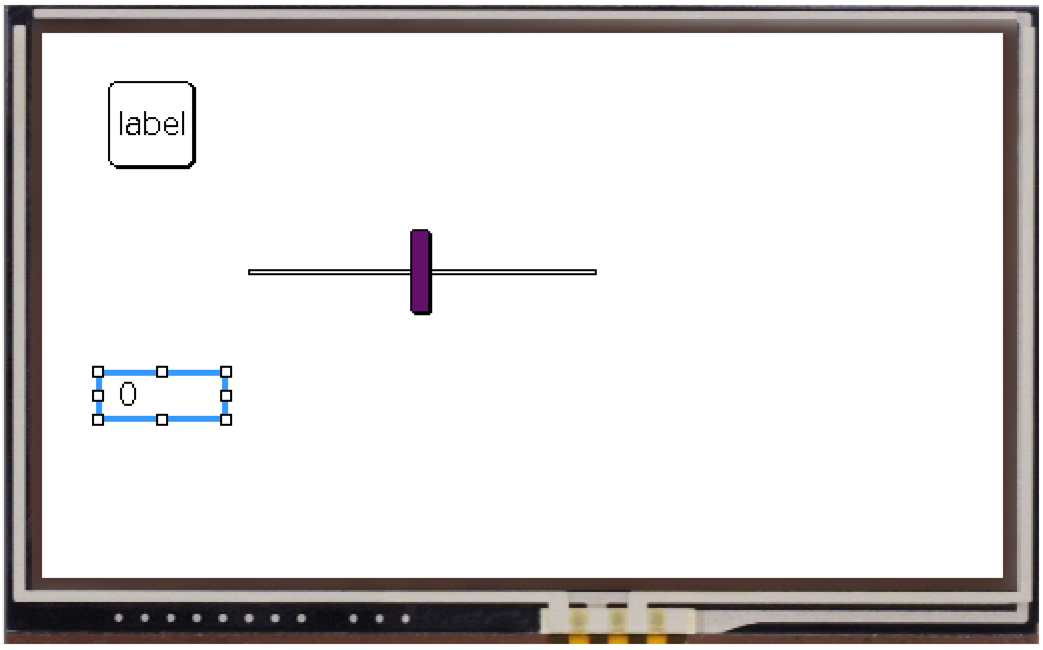
```
//Amulet:document.MySlider_1.disappear()
Amulet:document.MySlider_1.setValue(0x10)
```

Select Done and hit the Run button again. GEMplayer should run the project and if the Function Button is pressed, the handle of the Slider should snap to the left to where the value of 0x10 would be. That is an Inter-Widget Communication call with an argument.

3) Methods that return a value:

So far, the IWC calls in the examples have all been in the href of a Control Widget. IWC calls can also be used in the href of a View Widget, but instead of sending data, View Widgets request data.

To continue with the above example, go back to the ➕ button in the lower left hand corner of GEMstudio Pro and add a NumericField widget. Once added, move it so it doesn't overlap the Function Button or the Slider. The layout should look something like this:



Click on the href attribute of the NumericField and a similar href window should pop up. The gray "...Press Tab For Options" appears again letting us know auto-complete will work. Hit the Tab key and enter "Amulet:document.". Hitting Tab again will now show three objects to choose from, the Function Button, the Numeric Field, and the Slider. For this example, we will make the Numeric Field display the current value of the Slider, so choose the Slider from the list of three objects.

Hitting Tab one more time pops up a menu that lists "maskedValue(" and "value()". maskedValue is used to mask out bits in a hex value. For this example we just want the value of the Slider, so choose "value()".

Hit the Done button and then the Run button to start up GEMplayer. Use your mouse to move the Slider handle and the Numeric Field will display the value associated with the Slider. That is an Inter-Widget Communication call that returns a value.

Table 1. IWC method descriptions.

IWC Methods	**Description**
clearCanvas()	Clears the scribble/ dynamic image canvas completely, including any background images in the canvas.
buttonDown()	Sets the Custom/Function Button Widget to look like it is in the down state. This method does NOT invoke the href functions, it only affects the looks of the button, not the functionality.
buttonUp()	Sets the Custom/Function Button Widget to look like it is in the up state. This method does NOT invoke the href functions, it only affects the looks of the button, not the functionality.
disappear()	Object not visible on LCD. Counteracts the reappear() method.

forceHit()	Object performs its "hit" method without user input. The "hit" method will invoke all **href** functions of that object.
forceUpdate()	View Object performs its href method which is normally performed based upon the **updateFreq** parameter.
inverseRegionColor()	Object will display in reverse video. Counteracts the normalRegionColor() method.
inverseStringColor()	Object's text string will display in reverse video. Counteracts the normalStringColor() method.
nextEntry()	Highlighted box of a list box widget moves to next entry in the list. Effectively moves the highlighted box down one entry.
normalRegionColor()	Object will display in normal video. Counteracts the inverseRegionColor() method.
normalStringColor()	Object's text string will display in normal video. Counteracts the inverseStringColor() method.
previousEntry()	Highlighted box of a list box widget moves to previous entry in the list. Effectively moves the highlighted box up one entry.
reappear()	Object visible on LCD. Counteracts the disappear() method.
reset()	If called on a LinePlot widget, it clears the line plot screen. If called on a META Refresh Object, it will reset that META Refresh object's timer back to it's initial delayRate. If called on a Scribble or Dynamic Image Widget, it redraws the canvas image from flash.
saveCanvas()	Saves the current state of the canvas to the flash. Writes over the original canvas specified at compile time.
setLineWeight(x)	Sets the line weight for the scribble widget. (range 1-15)
setMethod(m)[2]	Changes the **href** method originally specified by the widget/object; only valid when the originally specified method is a single function. Cannot be used as part of a multiple href function.
setOnVarMethod(m)[2]	Changes the **ONVAR** method originally specified by the widget/object; only valid when the originally specified method is a single function. Cannot be used as part of a multiple href function.

setOnVarUARTMethod(m)[2]	Changes the **ONVAR** UART method originally specified by the widget/object; only valid when the originally specified method is a single function. Cannot be used as part of a multiple **href** function.
setOnVarVariableNumber(x)[1]	Changes the variable number originally specified by the object's onVar parameter. Can be used only if the object's onVar has a byte(y), word(y) or string(y). In all cases, the y gets changed to the argument specified in setOnVarVariableNumber(x).
setTrigger(x)[1]	Changes the equal, gt or lt value originally specified by the object to the byte value x.
setUARTMethod(m)[2]	Changes the **href** UART method originally specified by the widget/object; only valid when the originally specified method is a single function. Cannot be used as part of a multiple **href** function.
setUpdateRate(f)[3]	Changes the update rate originally specified by the widget/object. Argument is a floating point number, specifying time in seconds. Cannot be used as part of a multiple **href** function.
setValue(x)[1]	Object receives the value of the calling object. This allows a Control Widget to provide the input to a View Widget. This method is called from a Control Widget **href**. The type of data can be either a BYTE, WORD or STRING.
setVariableNumber(x)[1]	The variable number used in the href of the named widget will change to x, where x is the variable index used in the following variable types: byte(x), word(x) or string(x). The y gets changed to the argument specified in setVariableNumber(x).
setX(x)[10]	Changes the X coordinate of the top left corner of the named widget/object. Should be preceded by a disappear() method and followed by a reappear() method.
setY(x)[10]	Changes the Y coordinate of the top left corner of the named widget/object. Should be preceded by a disappear() method and followed by a reappear() method.
startUpdating()	View Widget starts updating the displayed data. Counteracts the stopUpdating() method.
stopUpdating()	View Widget stops updating the displayed data.

toggleRegionColor()	The named widget will either start or stop displaying in reverse video.
toggleStringColor()	The named widget's text string will either start or stop displaying in reverse video.
toggleUpdating()	Changes current state of View Widget; either starts or stops updating the displayed data.
uploadImage()	Scribble widget uploads its raw image data via the connection and protocol described in the href parameter.
value()	Returns the intrinsic value of the Control Object. The calling object then uses that value as its input. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget **href**. The value could be a BYTE, WORD or STRING, depending upon the type of data of the Control Object.

Table 2. IWC method descriptions unique to Animated Image Objects.

IWC Methods	Description of method for animated images
fastSpeed()	Increases animation speed.
oneFrame()	Advances animation one frame.
pause()	Stops animation.
play()	Starts animation in current direction.
playBackwards()	Starts animating backwards.
playForward()	Starts animating forward.
regularSpeed()	Normal animation speed.
slowSpeed()	Decreases animation speed.
superFastSpeed()	Fastest animation speed.
superSlowSpeed()	Slowest animation speed.

Table 3. Matrix of supported IWC methods for Widgets and Objects.

IWC methods	Widgets

	Bar graph	Check box	Custom button	Dyn. Image	Func. button	Image Bar	Image Scroller	Image Seq.	Line Plot	List	Num. Field	Radio Button	Sc
clearCanvas()				x									
disappear()	x	x	x	x	x	x	x	x	x	x	x	x	
forceHit()		x[4]	x		x		x			x		x[5]	
forceUpdate()	x	x[4a]	x		x	x	x	x	x	x	x	x	
inverseRegionColor()											x		
inverseStringColor()													
nextEntry()										x			
normalRegionColor()											x		
normalStringColor()													
previousEntry										x			
reappear()	x	x	x	x	x	x	x	x	x	x	x	x	
reset()				x					x				
saveCanvas()													
setLinePattern()													
setLineWeight()													
setMethod(m)[2, 8]	x	x[9]	x		x	x	x	x	x	x	x	x	
setTrigger(x)[1]													
setUpdateRate(f)[3, 8]	x					x		x	x		x		
setValue(x)[1]	x	x	x		x	x	x	x	x		x	x	
setVariableNumber(x)[1]	x					x		x	x		x		

setX(x)[10]	x	x	x	x	x	x	x	x	x	x	x	x	
setY(x)[10]	x	x	x	x	x	x	x	x	x	x	x	x	
startUpdating()	x					x		x	x		x		
stopUpdating()	x					x		x	x		x		
toggleUpdating()	x					x		x	x		x		
uploadImage()													
value()		x	x		x		x			x		x	
fastSpeed()													
oneFrame()													
pause()													
play()													
playBackwards()													
playForward()													
regularSpeed()													
slowSpeed()													
superFastSpeed()													
superSlowSpeed()													

1. Regarding x: For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. META REFRESH tags and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and 250-character strings in double quotes for STRINGs.
2. Regarding m: When setMethod(), setOnVarMethod(), setOnVarUARTMethod or setUARTMethod() is the IWC method, the argument should be the name of the method you want to set.
3. Regarding f: For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. META REFRESH tags and Function/Custom Buttons should use f. Like the regular updateFreq, use a floating point number to specify the update rate in seconds. Range for f is 0-655.35

4. forceHit() for checkBox widget: When imparting a forceHit on a single checkBox, that checkBox will toggle. You can also forceHit an entire checkBox group which will perform the href function(s), but will not toggle any checkboxes. To forceHit a checkbox group, use the **groupName** as the widgetName (rather than an individual checkBox name).
4a. forceUpdate() for checkBox widget: You can only impart a forceUpdate on an entire checkBox group (even if the checkBox is a lone box), which will perform the initHref function. To forceUpdate a checkbox group, use the **groupName** as the widgetName (rather than an individual checkBox name).
5. forceHit() for radioButton widget: You can only impart a forceHit to an individual radio button, not a radio button group.
6. setValue() for META REFRESH: setValue() cannot be invoked if the **ONVAR** attribute is present.
7. setMethod() and setUpdateRate(): When using the setMethod() or setUpdateRate() methods, multiple functions are not allowed within a single **href**. However, you can invoke a forceHit() on another object that will launch setMethod() or setUpdateRate(), which then allows you to use multiple functions since there is no such limitation on the forceHit() method.
8. setMethod() for checkBox widget: To change the method for a checkbox group, or an ungrouped checkbox, use the **groupName** as the *widgetName* (rather than an individual checkBox name). You cannot change the method for an individual checkBox within a group.
9. regarding x in setX() and setY(): When setX() or setY() is the IWC method, the argument needs to be a word value with the range being the viewable area of the given LCD.

Table 4. Matrix of supported IWC methods for Animated Image Object

IWC methods	Animated Image Object
disappear()	x
fastSpeed()	x
oneFrame()	x
pause()	x
play()	x
playBackwards()	x
playForward()	x
reappear()	x
regularSpeed()	x
slowSpeed()	x
superFastSpeed()	x
superSlowSpeed()	x

META Refresh Objects

META REFRESH <META HTTP-EQUIV="REFRESH">

The most powerful, and potentially the most confusing, object in the Amulet system is the META Refresh object. It can be thought of as an object that exists on a page, but isn't visible on the LCD.

There are four different ways to use the META Refresh control object:

1) Call a function(s) based upon a timer event.
2) Call a function(s) when a timer-based function returns a specific value. (if...then...else function) or (switch statement)
3) Initialize InternalRAM variables with a value returned from a timer-based function.
4) Be a container object that makes no function calls, essentially becoming a variable which other objects/widgets can reference.

1) Call a function(s) based upon a timer event.

<META HTTP-EQUIV="REFRESH" CONTENT="updateRate, delayRate;URL=function(s);VALUE=number;NAME=string">

This meta tag acts like an anchor that calls its function based upon a timer event instead of a user "hit". Notice the strange syntax with all of the semi-colon delimited fields enclosed within one set of quotes. Also, REFRESH must be all uppercase. CONTENT fields are described below:

updateRate,delayRate

The updateRate specifies the frequency that the IF function is called (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. 0.00 means update never. The delayRate is optional and specifies the delay time from when the page is loaded until the initial ONVAR function is called (specified in seconds, with a single floating-point number). The range is 0.01 - 655.35. If the delayRate is not specified, then the delay time defaults to the updateRate (frequency) value. If the delayRate is specified and the updateRate is 0.00, then the IF function(s) is called after the delay time specified by the delayRate and does not update again.

URL=function(s)

The allowable syntax for the "URL=function(s)"string are identical to that of the HREF attribute of the tag or the HREF attribute for a user-input widget. See Appendix B for available functions.

VALUE=number

Specifies the intrinsic value of this meta refresh object. This parameter is optional because the intrinsic value can be specified directly within the URL function call as the argument to the method. See note regarding Control Object intrinsic values.

NAME=string

Specifies the internal name of this meta refresh object. Used for Inter-Widget Communication only.

Examples:

To send out an "invoke RPC #5" message every 500ms, use the following META REFRESH object:

<META HTTP-EQUIV="REFRESH" CONTENT="0.5;URL=Amulet:UART.invokeRPC(5)">

To send out an "invoke RPC #5" message once immediately upon loading the page, but never again, use the following:

<META HTTP-EQUIV="REFRESH" CONTENT="0,0.01;URL=Amulet:UART.invokeRPC(5)">

Note: The 0 of "0,0.01" means that the URL function will not have an update rate. The 0.01 of "0,0.01" means that the URL function will be called 10ms after loading the page.

To launch to a page called parrot after 5 seconds, use the following:

```
<META HTTP-EQUIV="REFRESH" CONTENT="5;URL=parrot.open()">
```

2)Call a function(s) when a timer based function returns a specific value (if...then...else function) or (switch statement)

```
<META HTTP-EQUIV="REFRESH"
CONTENT="updateRate, delayRate;
IF=function;
{EQ | GT | LT | NEQ}=value;
THEN=function(s);
ELSE=function(s);
NAME=string">
```

This meta tag acts like an anchor that calls its THEN or ELSE function(s) when the timer-based IF function returns a specific value, instead of a user "hit". Notice the strange syntax with all of the semi-colon delimited fields enclosed within one set of quotes. There can only be a single IF statement per meta, but there can be multiple THEN statements with their corresponding trigger statement. By having multiple THEN statements, the meta can be used like a switch statement. The IF attribute is like a View Widget's HREF parameter. The THEN and ELSE attributes are like a Control Widget's HREF parameter. CONTENT fields are described below:

updateRate,delayRate

The updateRate specifies the frequency that the IF function is called (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35. 0.00 means update never. The delayRate is optional and specifies the delay time from when the page is loaded until the initial ONVAR function is called (specified in seconds, with a single floating-point number). The range is 0.01 - 655.35. If the delayRate is not specified, then the delay time defaults to the updateRate (frequency) value. If the delayRate is specified and the updateRate is 0.00, then the IF function(s) is called after the delay time specified by the delayRate and does not update again.

IF=function

The value returned by this function call is used to trigger the function(s) in THEN=. The behavior and syntax of this META attribute is identical to that of the HREF parameter for a View Widget. See Appendix B for available functions.

{EQ | GT | LT | NEQ}=number

This attribute specifies the value and condition that triggers the THEN= function(s). If the value returned from the IF= function meets the condition, the immediately following THEN= function(s) is called. The value of number can be a byte, word, InternalRAM byte variable or InternalRAM word variable.

THEN=function(s)

The allowable syntax for the "THEN=function(s)"string are identical to that of the HREF attribute of the tag or the HREF parameter for a Control Widget. See Appendix B for available functions.

ELSE=function(s) (Optional)

The allowable syntax for the "ELSE=function(s)"string are identical to that of the HREF attribute of the tag or the HREF parameter for a Control Widget. See Appendix B for available functions.

NAME=string

Specifies the internal name of this meta refresh object. Used for Inter-Widget Communication only.

if...then...else example:

For example, to create a META Refresh object that checks the value of InternalRAM.byte(0) every 500ms and if it is less than 5 it sends out an RPC(0), else it will send out an RPC(0xFF), use the following:

```
<META http-equiv="Refresh" content="0.5;
IF=Amulet:InternalRAM.byte(0).value();
LT=5;
THEN=Amulet:UART.invokeRPC(0);
ELSE=Amulet:UART.invokeRPC(0xFF);
NAME=metaRPCLauncher">
```

switch statement example:

For example, to create a META Refresh object that is similar to a switch statement that checks the value of InternalRAM.byte(0) every 500ms and if it equals 0 it sends out an RPC(0), if it equals 1 it sends out an RPC(1), if it equals 2 it sends out an RPC(2), if it equals 3 it sends out an RPC(3), else it will send out an RPC(0xFF), use the following:

```
<META http-equiv="Refresh" content="0.5;
IF=Amulet:InternalRAM.byte(0).value();
EQ=0;
THEN=Amulet:UART.invokeRPC(0);
EQ=1;
THEN=Amulet:UART.invokeRPC(1);
EQ=2;
THEN=Amulet:UART.invokeRPC(2);
EQ=3;
THEN=Amulet:UART.invokeRPC(3);
ELSE=Amulet:UART.invokeRPC(0xFF);
NAME=metaRPCLauncher">
```

3) Initialize Internal RAM variables with a value returned from a timer-based function.

```
META HTTP-EQUIV="REFRESH" CONTENT="updateRate,
delayRate;ONVAR=function;VALUE=InternalRAM.type(x) ;NAME=string">
```

This meta tag does not require a URL function since the META object only exists to initialize an Internal RAM variable value, either a BYTE, WORD or STRING.
The most obvious use for this type of META is for initializing an Internal RAM variable to an internal Amulet value. For example, to set Internal RAM word variable 0 to the current page number, you could use the following:

```
<META HTTP-EQUIV="Refresh"
CONTENT="0,0.01;ONVAR=Amulet:internal.fileNumber.value();value=InternalRAM.word(0)">
```

Since the value=InternalRAM.word(0), instead of the META Refresh saving the value in its own memory space, it actually saves it directly to Internal RAM word variable #0.

Another case where this could be useful is if someone wanted to initialize an Internal RAM variable, but wanted to maintain the slave relationship with the Amulet. Therefore, you could have the META request a variable and store it directly into an Internal RAM variable without ever sending a master message.

```
<META HTTP-EQUIV="Refresh"
CONTENT="0,0.01;ONVAR=Amulet:UART.byte(5).value();value=InternalRAM.byte(5)">
```

The above example will request the value of external byte variable #5 once 10ms after loading the page and save that value into InternalRAM byte variable #5.

```
<META HTTP-EQUIV="Refresh"
CONTENT="0,0.01;ONVAR=Amulet:UART.words(0).array(4);value=InternalRAM.words(0)">
```

The above example will request the value of external word variables #0 - #3 once 10ms after loading the page and save that value into InternalRAM word variables #0 - #3.

4) Be a container object (byte, word or string variable)

```
META HTTP-EQUIV="REFRESH" CONTENT="updateRate,
delayRate;ONVAR=function;URL=Amulet:nop();VALUE=number ;NAME=string">
```

This meta tag does not need to call any functions. It exists to hold a variable value, either a BYTE, WORD or STRING. Other control objects/widgets can set the value of this "variable" by using Amulet:document.name.setValue(), and the value of the "variable" can be read by using Amulet:document.name.value(), where name is the internal name given in NAME=string. See note regarding Control Object intrinsic values.

With the addition of InternalRAM variables, using the META as a container object is not needed. InternalRAM uses less uHTML space as well as the additional benefit of existing outside of a specific page. META objects are only valid in the page that they are defined in and will be reinitialized every time the page is re-entered. InternalRAM can survive from page to page, will not be reinitialized every time the page is re-entered, unless you specify it, and InternalRAM can actually be saved back to the flash, so the variable can persist even after powering down.

Examples:

To send out an"invoke RPC #5" message when the value of a slider (Slider1) equals 0xFF, which is polled every 500ms, use the following META REFRESH object:

```
<META HTTP-EQUIV="REFRESH"
CONTENT="0.5;ONVAR=Amulet:document.Slider1.value();TRIGGER=0xFF;URL=Amulet:UART.invokeRPC(5);NAME=Me
```

This META will continue to send the RPC every 500ms until the slider value no longer equals 0xFF. To have the META send it out only once, you could have the META URL include the following function after the invokeRPC function:
Amulet:document.MetaOne.setUpdateRate(0)
This will make the META turn itself off after the first invokeRPC is sent.
To send out a "set byte variable #2 to 0x78" message when the value of external byte variable #4 equals 0xFF, which is polled every 500ms, use the following META REFRESH object:

```
<META HTTP-EQUIV="REFRESH"
CONTENT="0.5;ONVAR=Amulet:UART.byte(4).value();TRIGGER=0xFF;URL=Amulet:UART.byte(2).setValue(0x78)">
```

To send out a 'set string variable #5 to "My String"' message when the value of external byte variable #4 equals 0xF0, which is polled every 500ms, use the following META REFRESH object:

```
<META HTTP-EQUIV="REFRESH"
CONTENT='0.5;ONVAR=Amulet:UART.byte(4).value();TRIGGER=0xF0;URL=Amulet:UART.string(5).setValue("My
String")'>
```

To launch to "Page1"when the value of external byte variable #1 is greater than 0xC0, which is polled every 500ms, and to "Page2" if the value is less than 0x40, use the following META REFRESH objects:

```
<META HTTP-EQUIV="REFRESH"
CONTENT="0.5;ONVAR=Amulet:UART.byte(1).value();TRIGGER.GT=0xC0;URL=../setValue()/Page1.open()">
<META HTTP-EQUIV="REFRESH" CONTENT="0.5;ONVAR=Amulet:UART.byte(1).value();TRIGGER.LT=0x40;URL=../
setValue()/Page2.open()">
```

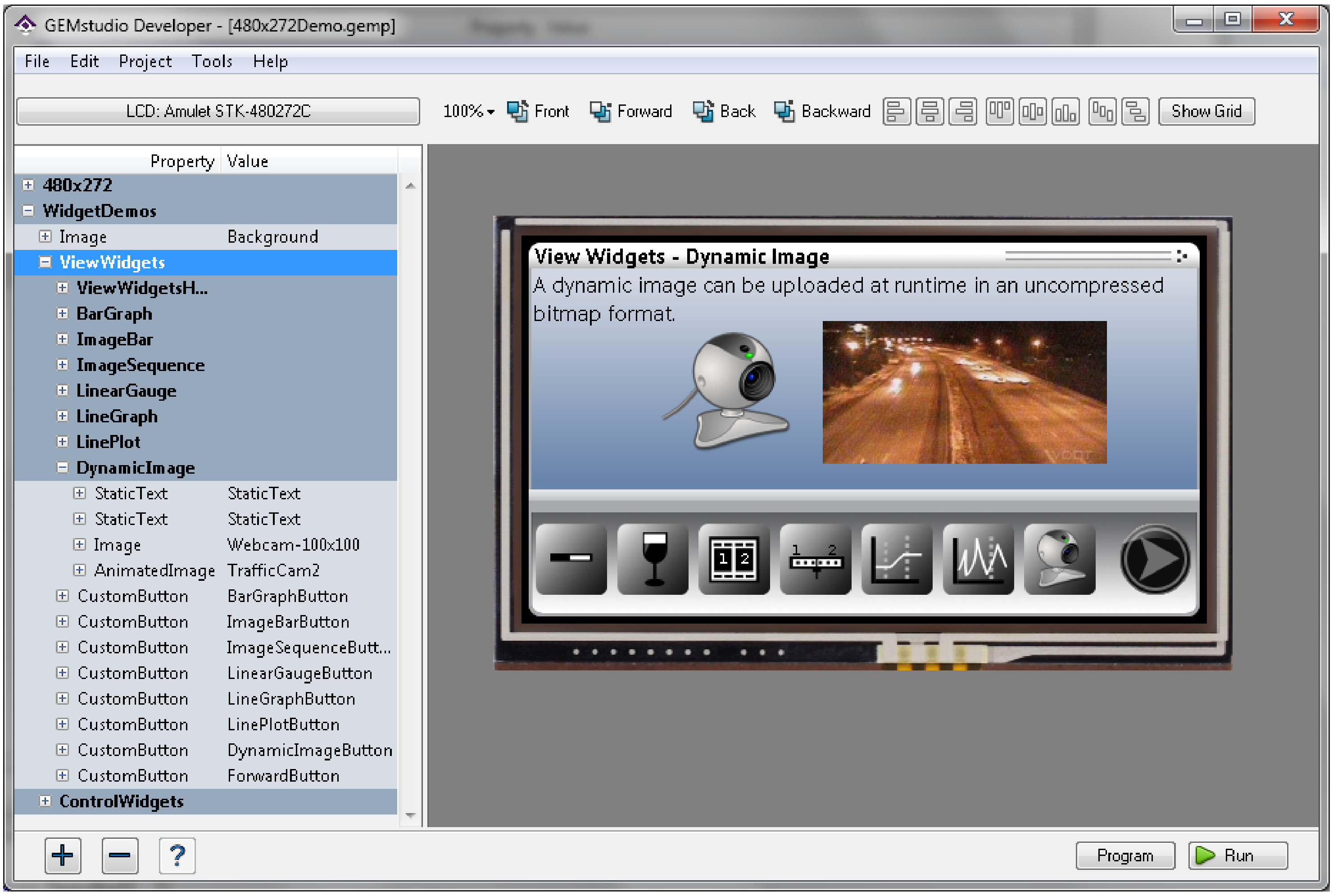
Nested Pages

GEMstudio allows pages to be nested. This can be accomplished by simply dragging one page into another. Once compiled, the nested page contains all of the objects on the parent page, including all of the code in the Page Functions. This is especially useful for groups of common objects, such as a row of navigation buttons, clocks and power indicators, or common code used on multiple pages.

An example of how to use nested pages can be found in the 480x272 Demo project. In the screenshot below, notice the page called **ViewWidgets**. **ViewWidgets** is a parent page to the 8 nested pages below it, namely ViewWidgetsH..., BarGraph, ImageBar, ImageSequence, LinearGauge, LineGraph, LinePlot, and DynamicImage. The CustomButtons named BarGraphButton, ImageBarButton, ImageSequenceButton, LinearGaugeButton, LineGraphButton, LinePlotButton, DynamicImageButton, and ForwardButton are all part of the ViewWidgets page.

In this screenshot, notice that the nested page named **DynamicImage** has been expanded and there are only four objects (two StaticTexts, and Image and an AnimatedImage) placed within the actual page. But notice in the layout window that all the CustomButtons show up as part of that page. This is because **ViewWidgets** is a parent page to **DynamicImage**, so all objects that are in **ViewWidgets** are also in **DynamicImage**. That means all the CustomButtons in **ViewWidgets** are also in all the other nested pages, even though they were only manually added to a single page. That is the power of parent and nested pages.

Also note that **ViewWidgets** is a nested page of **WidgetDemos**, which contains the common background that is used in all of the nested pages of both **ViewWidgets** and **ControlWidgets**.


GEMstudio Developer - [480x272Demo.gemp]
File Edit Project Tools Help
LCD: Amulet STK-480272C
100%
Front
Forward
Back
Backward
Show Grid
View Widgets - Dynamic Image
Program
Run

A dynamic image can be uploaded at runtime in an uncompressed bitmap format.

Property	Value
⊞ 480x272	
⊟ WidgetDemos	
⊞ Image	Background
⊟ ViewWidgets	
⊞ ViewWidgetsH...	
⊞ BarGraph	
⊞ ImageBar	
⊞ ImageSequence	
⊞ LinearGauge	
⊞ LineGraph	
⊞ LinePlot	
⊟ DynamicImage	
⊞ StaticText	StaticText
⊞ StaticText	StaticText
⊞ Image	Webcam-100x100
⊞ AnimatedImage	TrafficCam2
⊞ CustomButton	BarGraphButton
⊞ CustomButton	ImageBarButton
⊞ CustomButton	ImageSequenceButt...
⊞ CustomButton	LinearGaugeButton
⊞ CustomButton	LineGraphButton
⊞ CustomButton	LinePlotButton
⊞ CustomButton	DynamicImageButton
⊞ CustomButton	ForwardButton
⊞ ControlWidgets	

Amulet Communication Protocols

Amulet can use either an ASCII-based or a CRC-based serial communication protocol between the Amulet LCD module and your embedded device (external processor). The ASCII protocol has weak error checking, but can be easier for beginners to read in a debug terminal window. All new GEMstudio projects default to using the CRC protocol. This is very similar in structure to Modbus RTU, which includes a 16-bit CRC for robust error checking. These protocols can work at the same time on all UART and USB ports, which all have their own transmit and receive buffers as well as baud, parity and other configurations.

The Amulet system has two different ways of interfacing this protocol with an external processor. One method has the Amulet LCD module as the master and the external processor as the slave. The other method has the external processor as the master and the Amulet as the slave. Both methods can be run concurrently on the same GEMstudio page. Amulet master messages are initiated either by timer events or by user input from the touch panel. Amulet master messages are derived from compiled GEMstudio code stored in the serial flash, eMMC, or SD Card, depending on your Amulet module.

To set the Amulet as the master, an object needs to have **href** commands that start with **Amulet:uartx or Amulet:USB where x can be 0, 1, or 2**. If no **x** is specified, then it defaults to **0**. This command will queue a message in the port-specific transmit buffers of the Amulet OS whenever the control widget is activated, or in a view widget at an interval based upon the **updateRate** specified within that particular object. The Amulet expects a response from the external processor within 200ms, by default. This timeout can be altered in the project settings under the communications tab. If no valid response is received within the timeout, the Amulet OS will automatically resend up to 10 times before giving up. Each timeout increments the **.**

The Amulet does not need to be configured to be a slave. If the external processor chooses to be the master, it can send a valid Amulet message to the Amulet at any time or on any page and the Amulet will become the slave for that message. If the Amulet OS has any further queued master messages, it will once again become the master until the external processor chooses to be the master.

When the Amulet is the slave, the external processor can read and write to "virtual dual-port" RAM which resides on the Amulet side. The Amulet has 256 byte variables, 256 word variables, 256 26-character string variables and a 6 byte deep RPC buffer. Amulet Widgets can have **href** commands that start with **Amulet:InternalRAM** to access these "virtual dual-port" RAM variables.

The command opcodes are the same, regardless of who is the master or who is the slave. This means that a "Get byte variable" command sent to the Amulet looks exactly like a "Get byte variable" command sent to the external processor. The one difference is the slave ID, which is the first byte of most messages in the CRC protocol. If the host is sending a master message to the Amulet processor, the message will start with the Amulet ID. If the Amulet is sending a master message to the host processor, the message will start with the Host ID. Similarly, CRC responses start with the same ID as the original message. So if the Amulet sends a master message to the host, the message from the Amulet processor will start with the Host ID and the response from the host will also start with the Host ID. Conversely, if the external processor sends a master message to the Amulet processor, the message will start with the Amulet ID and the response from the Amulet processor will also start with the Amulet ID.

Communication Format

Communications between the Amulet LCD module and an external processor are asynchronous serial transmissions, with the following formatting options:

Baud Rate:

Standard baud rates: 9600, 14400, 19200, 28800, 38400, 56000, 57600, or 115200bps
Custom baud rate generation is also possible.

Parity:

None, Even, Odd, Space, Mark

Data Bits:

5, 6, 7, 8, 9

Stop Bits:

1, 1.5, 2

Mode Options:

Normal, RS485, HW_Handshaking, ISO7816_T_0, ISO7816_T_1, IRDA

Channel Mode Options:

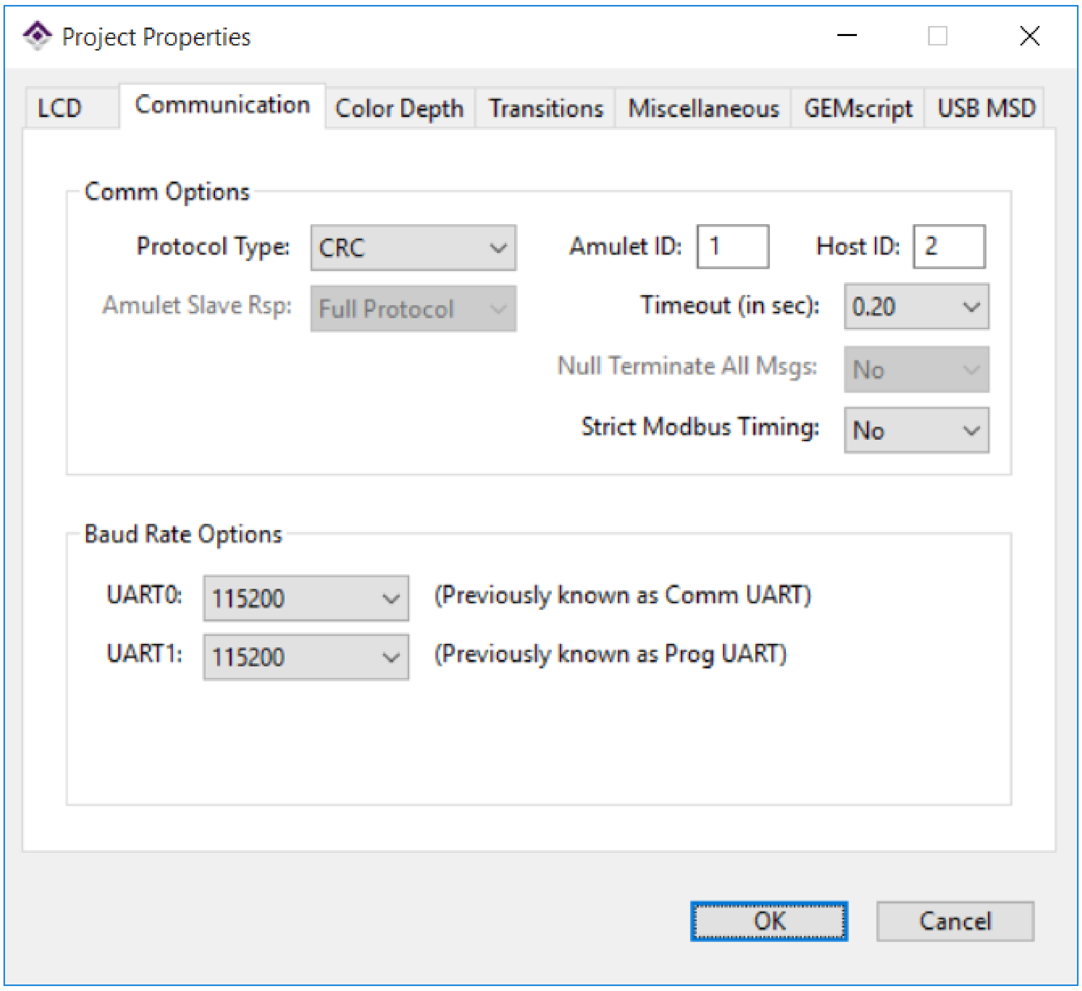
Normal, Auto, Local_Loopback, Remote_Loopback

The default baud rate is 115,200 bps. Other baud rates are set in the Project Options > Communication tab.

All settings other than baud rate are defined in a configuration file which can be found here, if GEMstudio is installed:
C:\ProgramData\AmuletTech\Global\Configuration\Board\UART
The Amulet MK-07C-HP Module uses "AGB53D default.uart" file, while all other Amulet color products will use the "AGB75L default.uart" file.

Communication Modifications

The Communications tab under the Project Properties dialog allows customization of several properties.



Protocol Type: CRC
Amulet ID: 1
Host ID: 2
Amulet Slave Rsp: Full Protocol
Timeout (in sec): 0.20
Null Terminate All Msgs: No
Strict Modbus Timing: No
UART0: 115200 (Previously known as Comm UART)
UART1: 115200 (Previously known as Prog UART)

Protocol Type

Select either ASCII or CRC protocol. The default commonications protocol is CRC

Amulet ID & Host ID (CRC only)

The default slave ID for the Amulet is 1 and for the Host it is 2. This is the first byte the master uses to address the slave, as per Modbus RTU specifications. If used on a peer to peer network, the slave ID numbers are not that important, as long as they are unique. If used in a multi-drop network where slave IDs are already defined, it might be necessary to give the Amulet and the Host new IDs to suit the existing network.

Amulet Slave Response (ASCII only)

When the Amulet receives a "Set" or "Draw" command, by default, it responds back with the corresponding response byte followed by an echo of all the bytes sent to the Amulet. To cut back on unnecessary bytes, the SlaveAckRsp META tag attribute can be used. Instead of sending an echo of the entire message, it will only respond with an ACK (0xF0). If no response is desired, the SlaveNoRsp attribute can be used.

Time-Out

This is the maximum time Amulet gives the host to respond to a message before resending. The default is 200ms.

Null-Terminate All Messages (ASCII only)

By default, the ASCII protocol does not provide for a termination character, except when sending strings. The Amulet can be forced to null terminate (0x00) every single response by setting this to Yes. This can be handy in the receive section of your serial protocol code if you don't have time to analyze each byte received as it is coming in.

Strict Modbus Timing (CRC only)

Modbus protocol specifies a 3.5 character delay between messages. If your host requires this delay, then Amulet might be sending messages too quickly after receiving a response, in the case that multiple messages are queued into the transmit buffer. Turning this setting on will enable a non-blocking delay to satisfy Modbus hardware that depends on this delay.

CRC Protocol

The Amulet CRC communication protocol piggybacks on the Modbus RTU standard protocol. That means if your software already support Modbus RTU communication, you are almost done, you just need to add support for the Amulet specific function opcodes. For those not familiar with Modbus RTU, there is one oddity when it comes to numbers greater than 8-bit. When dealing with data within the payload, data is transmitted big-endian, meaning Most Significant Byte first. But, when dealing with the 16-bit CRC, the CRC is transmitted little-endian, meaning Least Significant Byte first.

Amulet as Master

When using a UART or USB port, the Amulet CRC protocol is full-duplex, meaning both the Amulet and the host processor can transmit at the same time. All master messages do require a response message, though. The Amulet LCD module(master) can send 11 different types of messages to the external processor (slave):

- A "Get byte variable" request. **(Amulet:port.byte(x).value())**
- A "Get word variable" request. **(word = 2 bytes) (Amulet:port.word(x).value())**
- A "Get string variable" request. **(Amulet:port.string(x).value())**
- A "Get byte variable array" request. **(Amulet:port.bytes(x).array(y))**
- A "Get word variable array" request. **(Amulet:port.words(x).array(y))**
- A "Get label variable" request. **(Amulet:port.label(x).value())**
- A "Set byte variable" command. **(Amulet:port.byte(x).setValue(y))**
- A "Set word variable" command. **(Amulet:port.word(x).setValue(y))**
- A "Set string variable" command. **(Amulet:port.string(x).setValue(y))**
- A "Set color variable" command. **(Amulet:port.color(x).setValue(y))**
- An "Invoke Remote Procedure Call (RPC)" command. **(Amulet:port.invokeRPC(x))**

Where **port** can be uart0, uart1, uart2 (if applicable) or USB.

If the message is valid, the slave should either return the requested data (if a "Get" request) or confirm the message was received (if an "Invoke" or "Set" command). If the message is not valid, the slave should respond with an acknowledge (ACK) to have the Amulet move to the next request, or the slave can respond with a negative acknowledgement (NAK) to have the Amulet resend the last message. See Error Responses for how to send these error replys. If there is a CRC error, following the Modbus standard, the message should be ignored. If the message is not responded to, the Amulet will resend the message after a timeout period.

Amulet as Slave

In the Amulet communications CRC protocol, the slave should not respond until the master is done sending its message. If the Amulet is going to be the slave in the protocol, the external processor (master) can send twenty-four different types of messages to the Amulet LCD module (slave). The external processor can read from and write to all the Amulet Internal RAM variables. The external processor can also send graphic primitives (pixel, line, rectangle, and filled rectangle) to the Amulet. The external processor can also force the Amulet to jump to a specific page.

- A "Get Internal RAM byte variable" request.
- A "Get Internal RAM word variable" request. (word = 2 bytes)
- A "Get Internal RAM string variable" request.
- A "Get Internal RAM color variable" request. (color = 4 bytes)
- A "Get Internal RAM byte array" request.
- A "Get Internal RAM word array" request. (word = 2 bytes)
- A "Get Internal RAM color array" request. (color = 4 bytes)
- A "Get Internal RAM RPC buffer" request.
- A "Set Internal RAM byte variable" command.
- A "Set Internal RAM word variable" command. (word = 2 bytes)
- A "Set Internal RAM string variable" command.
- A "Set Internal RAM color variable" command. (color = 4 bytes)
- A "Set Internal RAM byte variable array" command.
- A "Set Internal RAM word variable array" command. (word = 2 bytes)
- A "Set Internal RAM color variable array" command. (color = 4 bytes)
- A "Set Graphic Primitive color (8-bit)" command.
- A "Set Graphic Primitive color (32-bit)" command.
- A "Set Graphic Primitive color (InternalRAM color index)" command.
- A "Set Graphic Primitive pen weight" command.
- A "Draw pixel" command.
- A "Draw line" command.
- A "Draw rectangle" command.
- A "Draw filled rectangle" command.
- A "Jump to specific page" command.

If the message is valid, the Amulet LCD module slave will either return the requested data (if a "Get" request) or confirm the message (if a "Draw" or "Set" command). If there is a CRC error, the Amulet will not respond at all. If the opcode is not a valid Amulet opcode, the Amulet will respond with a NAK and the "Illegal Function" error code #1.

CRC Commands

The protocol is the same regardless of who is the master. So if an external processor is requesting the value of a byte variable on the Amulet (which would be an Internal RAM byte variable), the command opcode would be the same as if the Amulet was requesting the value of a byte variable that resides on the external processor.

The table below defines seventeen messages that can be sent between the master and the slave, not counting the graphic primitive messages. The valid range of variables and Remote Procedure Calls is 0-0xFF. The valid range for byte variable values returned from the slave (in response to the "Get Byte variable" request) is also 0-0xFF. The valid range for word variable values returned from the slave (in response to the "Get Word variable" request) is 0-0xFFFF.

String and label variable values returned from the slave (in response to the "Get String variable" request) can have a maximum of 250 characters plus a null termination character (0x00).

The first byte of all master messages is the ID of the slave processor. If the master is the Amulet processor, the first byte is the Host ID. If the slave is responding to a master message, it will also start with the ID of the slave processor. The second byte is always the opcode of the function. See the summary for the full list of available opcodes. The remaining payload bytes are opcode dependent. The final two bytes are the LSByte and MSByte of the CRC, in that order. See the CRC documentation on how to derive the 16-bit CRC. Essentially, all bytes prior to the two CRC bytes are run through the CRC algorithm. The result is the last two bytes of the message. If the CRC received is not the same as the CRC derived, then the message should be disregarded as the integrity of the message cannot be assured.

For example, if the GEMstudio file being compiled has a view widget with an href of **Amulet:uart0.byte(0x1A).value()**, which will send out the "Get Byte Variable #0x1A" request on uart0, the message to be transmitted would consist of five bytes. The first byte is the Host ID (by default, 0x02), the second byte is the "Get Byte Variable" opcode (0x20), third byte is the byte variable number (0x1A), fourth byte is the LSByte of the 16-bit CRC of the first three bytes (0x48), and the seventh and final byte is the MSbyte of the 16-bit CRC (0x0B). So the five byte message looks like: **0x02 0x20 0xA1 0x48 0x0B**

NOTE: The slave must respond to every valid Amulet command. When commands are not responded to, a time-out will occur after 200ms, by default, and that message will be repeated until either a response is received or after a total of eleven attempts. After eleven attempts, all UART variables are reset in an attempt to resync with the slave processor.

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7...	Byte n-2	Byte n-1	Byte n
Get Byte Variable	Slave ID	0x20	Variable Index	CRC LSByte	CRC MSByte					
Slave Response	Slave ID	0x20	Variable Index	Value	CRC LSByte	CRC MSByte				
Get Word Variable	Slave ID	0x21	Variable Index	CRC LSByte	CRC MSByte					
Slave Response	Slave ID	0x21	Variable Index	Value MSByte	Value LSByte	CRC LSByte	CRC MSByte			
Get String Variable	Slave ID	0x22	Variable Index	CRC LSByte	CRC MSByte					
Slave Response	Slave ID	0x22	Variable Index	UTF-8 Char	UTF-8 Char	UTF-8 Char	UTF-8 Char...	0x00 (Null Terminator)	CRC LSByte	CRC MSByte
Get Color Variable	Slave ID	0x23	Variable Index	CRC LSByte	CRC MSByte					

Slave Response	Slave ID	0x23	Variable Index	Value Bits 31-24 (Alpha)	Value Bits 23-16 (Blue)	Value Bits 15-8 (Green)	Value Bits 7-0 (Red)	CRC LSByte	CRC MSByte	

Get Byte Variable Array	Slave ID	0x24	Variable Start Index	Count of Bytes in Array	CRC LSByte	CRC MSByte				
Slave Response	Slave ID	0x24	Variable Start Index	Count of Bytes in Array	Value	Value	Value...	Last Value in Array	CRC LSByte	CRC MSByte

---

Get Word Variable Array	Slave ID	0x25	Variable Start Index	Count of Words in Array	CRC LSByte	CRC MSByte				
Slave Response	Slave ID	0x25	Variable Start Index	Count of Words in Array	Value MSByte	Value LSByte	Value MSByte...	Last Value LSByte in Array	CRC LSByte	CRC MSByte

---

Get Color Variable Array	Slave ID	0x26	Variable Start Index	Count of Colors in Array	CRC LSByte	CRC MSByte				
Slave Response	Slave ID	0x26	Variable Start Index	Count of Colors in Array	Value Bits 31-24 (Alpha)	Value Bits 23-16 (Blue)	Value Bits 15-8... (Green)	Last Value Bits 7-0 (Red) in Array	CRC LSByte	CRC MSByte

---

Get RAM Remote Procedure Calls (RPC)[1]	Slave ID	0x27	CRC LSByte	CRC MSByte						
Slave Response	Slave ID	0x27	Count of RPCs in Buffer	RPC #1	RPC #2	RPC #3	RPC #4...	RPC #n	CRC LSByte	CRC MSByte

---

Get Label Variable	Slave ID	0x28	Variable Index	CRC LSByte	CRC MSByte					

Slave Response	Slave ID	0x28	Variable Index	UTF-8 Char	UTF-8 Char	UTF-8 Char	UTF-8 Char...	0x00 (Null Terminator)	CRC LSByte	CRC MSByte
Set Byte Variable	Slave ID	0x30	Variable Index	Value	CRC LSByte	CRC MSByte				
Slave Response	Slave ID	0x30 (ACK)	CRC LSByte	CRC MSByte						
Set Word Variable	Slave ID	0x31	Variable Index	Value MSByte	Value LSByte	CRC LSByte	CRC MSByte			
Slave Response	Slave ID	0x31 (ACK)	CRC LSByte	CRC MSByte						
Set String Variable	Slave ID	0x32	Variable Index	UTF-8 char	UTF-8 char	UTF-8 char	UTF-8 char...	0x00 (Null Terminator)	CRC LSByte	CRC MSByte
Slave Response	Slave ID	0x32 (ACK)	CRC LSByte	CRC MSByte						
Set Color Variable	Slave ID	0x33	Variable Index	Value Bits 31-24 (Alpha)	Value Bits 23-16 (Blue)	Value Bits 15-8 (Green)	Value Bits 7-0 (Red)	CRC LSByte	CRC MSByte	
Slave Response	Slave ID	0x33 (ACK)	CRC LSByte	CRC MSByte						
Set Byte Variable Array[1]	Slave ID	0x34	Variable Start Index	Count of Bytes in Array	Value	Value	Value...	Last Value in Array	CRC LSByte	CRC MSByte
Slave Response	Slave ID	0x34 (ACK)	CRC LSByte	CRC MSByte						
Set Word Variable Array[1]	Slave ID	0x35	Variable Start Index	Count of Words in Array	Value MSByte	Value LSByte	Value MSByte...	Last Value LSByte in Array	CRC LSByte	CRC MSByte
Slave Response	Slave ID	0x35	CRC	CRC						

		(ACK)	LSByte	MSByte						
Set Color Variable Array[1]	Slave ID	0x36	Variable Start Index	Count of Colors in Array	Value Bits 31-24 (Alpha)	Value Bits 23-16 (Blue)	Value Bits 15-8 (Green)...	Last Value Bits 7-0 (Red) in Array	CRC LSByte	CRC MSByte
Slave Response	Slave ID	0x36 (ACK)	CRC LSByte	CRC MSByte						
Invoke Remote Procedure Call (RPC)	Slave ID	0x37	RPC Index	CRC LSByte	CRC MSByte					
Slave Response	Slave ID	0x37 (ACK)	CRC LSByte	CRC MSByte						
Jump To Page Command[1]	Slave ID	0x50	Page Index MSByte	Page Index LSByte	CRC LSByte	CRC MSByte				
Slave Response	Slave ID	0x50 (ACK)	CRC LSByte	CRC MSByte						
Get Current Page Index[1]	Slave ID	0x51	CRC LSByte	CRC MSByte						
Slave Response	Slave ID	0x50 (ACK)	Page Index MSByte	Page Index LSByte	CRC LSByte	CRC MSByte				

[1] Denotes command is only applicable when Amulet is the Slave.

Table 1. Seventeen types of messages can be sent between the master and the slave, not counting the graphic primitives.

Synchronization--The master initiates all communications by sending a message to the slave. All valid messages from the master to the slave start with the slave's ID (By default, the Amulet ID is 01 and the host processor's ID is 02). The payload is comprised of the Amulet message opcode (Byte 2) up to, but not including, the CRC bytes. The number of bytes following the opcode is dependent upon the opcode itself as each opcode has its own message structure.
When the Amulet is the master, upon receiving the last byte of a valid message from the master, the slave then has, by default, 200ms to respond to the message before the master times out. After 200ms, if there is no response, the master will continue to repeat the message until a response is received. After 10 unsuccessful attempts, the Amulet will flush its transmit buffer and reset all UART variables.

Other time out durations are set in the Project Properties menu from within GEMstudio.
By Modbus RTU standard, if a CRC error occurs, the slave should not respond at all. The master will timeout and resend the previous message.

NOTE: The external processor slave must respond to every valid Amulet master command. When commands are not responded to, a time-out will occur, and that message will be repeated 10 more times before flushing the transmit buffer and resetting all UART variables.

RPC Buffer

If a setup where the Amulet is always the slave is needed or desired, then up to six RPCs can be buffered in the Amulet's Internal RAM. The external processor can request the contents of the RPC buffer by sending a "Get Internal RAM RPC buffer" request (0x27).

When the Amulet is the master and an RPC is invoked, the Amulet will immediately send out the RPC command. If the Amulet is setup to use the Internal RAM RPC buffer instead, then the Amulet will send the RPC to an RPC buffer. The RPC buffer can only be read by an external processor by sending a "Get Internal RAM RPC buffer" request (0x27). The Amulet will respond with a count byte and all the RPCs (up to six) stored in the RPC buffer. After sending out the contents of the RPC buffer, the Amulet will then flush the buffer.

For example, to request the contents of the InternalRAM RPC buffer, the following would be sent to the Amulet:

0x01 0x27 0x40 0x3A

Where:
0x01 is the default Amulet Slave ID
0x27 is the Amulet message to send
0x3A40 is the two-byte CRC, sent LSByte first

If the InternalRAM RPC buffer has the following RPCs in it, 0x01 0x05 0x05 0x06, the following would be returned by the Amulet:

0x01 0x27 0x04 0x01 0x05 0x05 0x06 0x6E 0x78

If the same request is made before the InternalRAM RPC buffer is repopulated, the following would be returned by the Amulet:

0x01 0x27 0x00 0x3B 0xF0

CRC-Based Graphic Primitives

Using GEMstudio to create your projects at compile time allows you to make rich user interfaces quickly and easily. Sometimes, though, the ability to draw graphic primitives like lines, rectangles and filled rectangles at runtime, is needed. GEMstudio does not inherently give you the ability to do this.

The CRC-based protocol allows the external processor to send unsolicited graphic primitives to the Amulet. The drawing of these graphic primitives is independent of the project that is currently being run on the Amulet. You do need to keep in mind that the project will still be running, so any widgets or objects that write to the LCD might write over the graphic primitives you send to the LCD.

There are four graphic primitive commands used to set the color and weight of the lines used. The color and line weight should be sent prior to sending the actual graphic primitive drawing commands. Once the color and line weight is set, it does not need to be resent unless a change in either color or line weight is desired.

1. Set the graphic primitive 8-bit color (only applicable if the color density for the project is set for 8-bit color)
2. Set the graphic primitive 32-bit color (only applicable if the color density for the project is set for 32-bit color)
3. Set the graphic primitive InternalRAM color index (if wanting the graphic primitive color to be based on the color stored in an InternalRAM.color variable.)
4. Set the graphic primitive line weight

All graphic primitive commands follow the Amulet CRC-based protocol convention, so the first byte of every command will be the Amulet ID, followed by the graphic primitive opcode, a payload of data dependent upon the opcode, and finally a 16-bit CRC.

When setting the graphic primitive 8-bit color, only a single byte is used as the index into the 8-bit color palette currently defined in the project properties. The graphic primitive 8-bit color opcode is 0x40. The payload is a single 8-bit index into the palette, making for a total of 5 bytes in the command.
When setting the graphic primitive 32-bit color, the order of each 8-bit color component is Alpha, Blue, Green, Red. The graphic primitive 32-bit color opcode is 0x41. The payload is made up of four 8-bit color components, making for a total of 8 bytes in the command.

When setting the graphic primitive InternalRAM.color index, only a single byte is used as the index into the InternalRAM.color array. The graphic primitive InternalRAM color opcode is 0x42. The payload is a single 8-bit index, making for a total of 5 bytes in the command.

When setting the graphic primitive line weight, only a single byte is used to set the width of the graphic primitive line. The graphic primitive line weight opcode is 0x43. The payload is a single byte, making for a total of 5 bytes in the command.

There are four graphic primitive drawing commands. All four commands assume that the color and line weight have already been set prior to sending the drawing commands.

1. Draw pixel
2. Draw line
3. Draw rectangle
4. Draw filled rectangle

The pixel primitive (opcode 0x44) draws a pixel at point1(x and y coordinates). All coordinates are specified by a 16-bit number, which are in the order of MSByte then LSByte. The payload is made up of the x and y coordinates, so there are a total of 4 bytes in the payload, making for a total of 8 bytes in the command. The color is determined by the most recently sent "set graphic primitive color "command, and is the size specified by the most recently sent "set graphic primitive line weight" command.

The line primitive (opcode 0x45) draws a line from point 1(x and y coordinates) to point 2(x and y coordinates). The payload is made up of two sets of x and y coordinates, so there are a total of 8 bytes in the payload, making for a total of 12 bytes in the command. The color is determined by the most recently sent "set graphic primitive color "command, and the line weight is specified by the most recently sent "set graphic primitive line weight" command.

The rectangle primitive (opcode 0x46) draws a rectangle with a given starting top left point(x and y coordinates) and a delta x and delta y. The payload is made up of an x and y coordinate plus a 16-bit delta x and a 16-bit delta y, so there are a total of 8 bytes in the payload, making for a total of 12 bytes in the command. The color is determined by the most recently sent "set graphic primitive color "command, and the line weight is specified by the most recently sent "set graphic primitive line weight" command.

The fill rectangle primitive draws a solid rectangle with a given starting top left point(x and y coordinates) and a delta x and delta y. The payload is made up of an x and y coordinate plus a 16-bit delta x and a 16-bit delta y, so there are a total of 8 bytes in the payload, making for a total of 12 bytes in the command. The color is determined by the most recently sent "set graphic primitive color "command. The line weight is not used for the file rectangle graphic primitive.

If a graphics primitive is sent that does not fit within the bounds of the given LCD (i.e. a delta x of 380 pixels on a 320 x 240 LCD) the Amulet will ignore the request. It will respond back serially, but the graphic primitive will not be drawn.

Note on 32-bit color and the Alpha Component

32-bit colors that are entered through GEMstudio are entered using the #rrggbbaa notation, where rr is the 8-bit red value, gg is the 8-bit green value, bb is the 8-bit blue value, and aa is the 8-bit alpha (transparency) value. Each value can be a number from 00 to ff (in hex). The level of transparency is set by the alpha channel. The alpha channel is fully transparent with a value of 00 and completely opaque (no transparency) with a value of FF.

Examples:

Draw a Line:

To draw a line from (0x05,0x07) to (0x65,0x67), using the color fully opaque alpha:0x00, blue:0x00, green:0x00, red:0x22, and a line weight of 4 the following would be sent to the Amulet:

First send the "set graphic primitive 32-bit color" command:

```
0x01, 0x40, 0x00,0x00,0x00,0x22, 0x80,0x1C
 |      |     {a:    b:    g:    r: } {   CRC   }
 ID    set            color
       color
       opcode
```

Then send the "set graphic primitive line weight" command:

```
0x01, 0x43, 0x04, 0x10,0x43
 |      |      |     {    CRC    }
 ID    set    line
       line   weight
       weight
```

Then send the "draw graphic primitive line" command:

```
0x01, 0x45, 0x00,0x05,0x00,0x07, 0x00,0x65,0x00,0x67, 0x9E,0x64
 |      |     { x & y of point1 }  { x & y of point2 }  {   CRC   }
 ID    draw
       line
```

Draw a Rectangle:

To draw a rectangle that is 0x10C pixels wide, 0x82 pixels tall, has a topleft point at (0x0A,0x05), a line weight of 2 and using line color blue:0xFF, green:0x00, red:0x00 and fully opaque, the following would be sent to the Amulet:

First send the "set graphic primitive 32-bit color" command:

```
0x01, 0x40, 0x00,0xFF,0x00,0x00, 0x30,0x35
 |      |     {a:    b:    g:    r:}  {   CRC   }
 ID    set            color
       color
       opcode
```

Then send the "set graphic primitive line weight" command:

```
0x01, 0x43, 0x02, 0x90,0xF1
 |      |      |     {   CRC   }
 ID    set    line
       line   weight
       weight
```

Then send the "draw graphic primitive rectangle" command:

```
0x01,0x46,0x00,0x0A,0x00,0x05,0x01,0x0C,0x00,0x82,0x1C,0x3F
   |        |    {     x & y of topleft     }{      delta x & y          }{    CRC
  }
  ID    draw
        rectangle
```

Draw a Filled Rectangle:

To draw a filled rectangle that is 0x140 pixels wide, 0xF0 pixels tall, has a topleft point at (0x00,0x00) and using the same color as the previously drawn graphic primitive, the following would be sent to the Amulet:

Since the color has already been set, no need to send that string. Since line weight is not used by the fill rectangle command, no need to send that either.
Send the "draw graphic primitive fill rectangle" command:

```
0x01, 0x47, 0x00,0x00,0x00,0x00, 0x01,0x40,0x00,0xF0, 0x36,0x5D
 |     |     { x & y of topleft } {   delta x & y   }  {  CRC   }
 ID   draw
      fill
      rectangle
```

Graphic Primitives

This table defines the eight types of messages regarding graphic primitives that can be sent between an external processor and the Amulet. If a graphics primitive is sent that does not fit within the bounds of the given LCD (i.e. a delta x of 380 pixels on a 320 x 240 LCD) the Amulet will not draw the graphic primitive.

Message	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Byte 9**	**Byte 10**	**Byte 11**
Set 8-bit Color for Primitive	0x01	0x40	8-bit color index	CRC LSByte	CRC MSByte						
Amulet Response	0x01	0x40 (ACK)	CRC LSByte	CRC MSByte							

Message	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Byte 9**	**Byte 10**	**Byte 11**
Set 32-bit Color for Primitive	0x01	0x41	Alpha	Blue	Green	Red	CRC LSByte	CRC MSByte			
Amulet Response	0x01	0x41 (ACK)	CRC LSByte	CRC MSByte							

Message	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Byte 9**	**Byte 10**	**Byte 11**
Set InternalRAM Color Index for Primitive	0x01	0x42	IR color index	CRC LSByte	CRC MSByte						

Amulet Response	0x01	0x42 (ACK)	CRC LSByte	CRC MSByte							

Message	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Byte 9**	**Byte 10**	**Byte 11**
Set Line Weight for Primitive	0x01	0x43	Line Weight	CRC LSByte	CRC MSByte						
Amulet Response	0x01	0x43 (ACK)	CRC LSByte	CRC MSByte							

Message	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Byte 9**	**Byte 10**	**Byte 11**
"Draw" Pixel Primitive	0x01	0x44	Pnt X MSByte	Pnt X LSByte	Pnt Y MSByte	Pnt Y LSByte	CRC LSByte	CRC MSByte			
Amulet Response	0x01	0x44 (ACK)	CRC LSByte	CRC MSByte							

Message	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Byte 9**	**Byte 10**	**Byte 11**
"Draw" Line Primitive	0x01	0x45	Pnt 1 X MSByte	Pnt 1 X LSByte	Pnt 1 Y MSByte	Pnt 1 Y LSByte	Pnt 2 X MSByte	Pnt 2 X LSByte	Pnt 2 Y MSByte	Pnt 2 Y LSByte	CRC LSByte
Amulet Response	0x01	0x45 (ACK)	CRC LSByte	CRC MSByte							

Message	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Byte 9**	**Byte 10**	**Byte 11**
"Draw" Rectangle Primitive	0x01	0x46	Pnt 1 X MSByte	Pnt 1 X LSByte	Pnt 1 Y MSByte	Pnt 1 Y LSByte	Delta X MSByte	Delta X LSByte	Delta Y MSByte	Delta Y LSByte	CRC LSByte
Amulet Response	0x01	0x46 (ACK)	CRC LSByte	CRC MSByte							

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
"Draw" Fill Rectangle Primitive	0x01	0x47	Pnt 1 X MSByte	Pnt 1 X LSByte	Pnt 1 Y MSByte	Pnt 1 Y LSByte	Delta X MSByte	Delta X LSByte	Delta Y MSByte	Delta Y LSByte	CRC LSByte
Amulet Response	0x01	0x47 (ACK)	CRC LSByte	CRC MSByte							

Error Responses

There are two types of communications errors that need to be handled by both the host and Amulet processors, CRC errors due to noise on the line and invalid requests. If the CRC check does not match, the slave should not respond at all. The master will timeout and resend its last message.
If an invalid request is received (i.e. requesting or setting a variable that does not exist), then the slave should respond with a NAK and an error code. The NAK is defined as the opcode with the Most Significant Bit set. Normally, the slave would respond back with the opcode sent to them, but in the case of an error, instead of the opcode, the slave would respond with the MSBit set plus the opcode sent to them. For instance, if the master was requesting a byte variable that did not exist, the opcode used to request a byte is 0x20. If an error existed, the slave would respond with 0xA0 (0x80 + 0x20) with the following byte being the error code.
List of error codes:

Error Code	Name	Explanation	Amulet Action if Error Sent By Host	Will Amulet Send?
01	Illegal Function	Opcode used was invalid	ignore	yes
02	Illegal Data Address	Variable number used was invalid	ignore	no
03	Illegal Data Value	Implied length of array is incorrect (array overflow)	ignore	yes
04	Slave Device Fail	Slave Device in failure mode	ignore	no
05	Ack	Positive Acknowledgement	ignore	no
06	Slave Busy	Slave currently too busy to process command	resend last message	no
07	Nak	Negative Acknowledgement	resend last message	no

08	Memory Parity Error	Not currently used	ignore	no

CRC Used In The Amulet CRC Communication Protocol

The Amulet CRC Communication Protocol uses a 16-bit CRC (Cyclic Redundancy Check) to ensure data integrity. Each message sent is comprised of a slave ID, an Amulet opcode, a variable number of bytes within the message payload, and finally the two bytes of the CRC. The slave ID, Amulet opcode, and all the bytes of the message payload are run through the CRC algorithm. The resulting 16-bit CRC is appended to the end of every message. The Least Significant Byte of the CRC is sent first, followed by the Most Significant Byte of the CRC.
When receiving a packet of information from the Amulet module, the calculated CRC should match the last two bytes of the packet. If not, then the packet should be disregarded as the integrity of the data cannot be assured.

For example, a master message from the Amulet chip requesting the value of byte variable # 5 would look like this:

0x02 0x20 0x05 0x09 0xC3

Where:

0x02 - Host Processor slave ID
0x20 - Request byte variable (Amulet opcode)
0x05 - Byte variable index #5
0x09 - CRC LSByte
0xC3 - CRC MSByte
Running 0x02 0x20 0x05 through the CRC algorithm results in the 16-bit number 0xC309
Code for the CRC algorithm:

```
#define CRC_SEED  0xFFFF
#define CRC_POLY  0xA001

int calcCRC(char *ptr, int count)
{
  unsigned short crc = CRC_SEED;    // initialize CRC
  int i;

  while (count-- > 0)
  {
     crc = crc ^ *ptr++;
     for (i=8; i>0; i--)
     {
        if (crc & 0x0001)
           crc = (crc >> 1) ^ CRC_POLY;
        else
           crc >>= 1;
     }
  }
  return crc;
}
```

Jump to specific page

It is possible to send a Master Message to the Amulet which will force it to jump to a specific page within the project. The page number is an internal 16-bit number that the Amulet Compiler generates. All pages and images are assigned an internal number which can be determined by looking at the Amulet link map. When this message is received by the Amulet, it will react as if the Amulet:fileNumber(x) was launched, meaning the Amulet will jump directly to the page specified by the 16-bit internal number. The Amulet will respond with a standard ACK message, complete with the 16-bit CRC.

You must NOT jump directly to an image file number, it must be a valid page. If you do errantly jump to a non-valid page, the Amulet OS will respond with a soft reset. This will act exactly as if the reset button was pressed.

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Jump To Page Command	Slave ID	0x50	Page Index MSByte	Page Index LSByte	CRC LSByte	CRC MSByte
Amulet Response	Slave ID	0x50 (ACK)	CRC LSByte	CRC MSByte		

Examples (assuming Amulet Slave ID is 0x01):
To jump to page 0x25, send the following sequence:
0x01 0x50 0x00 0x25 0xC0 0x12

To jump to page 0x103, send the following sequence:
0x01 0x50 0x01 0x03 0x40 0x58

Soft Reset

It is possible to send a Master Message to the Amulet which will force it to perform a soft reset. It will react exactly as if the reset button was pressed.
The message structure is essentially the Jump To Page command, but the page index is 0xFFFF.

Example:
To cause a soft reset, assuming the Amulet Slave ID is 0x01, send the following sequence:
0x01 0x50 0xFF 0xFF 0x00 0x79

Get Current Page Index

To make the Amulet respond with the flash index of the current page, send the "Get Current Page Index" command. It will respond with a 16-bit value. See the Amulet Link Map for more information

Get Current Page Index[1]	Slave ID	0x51	CRC LSByte	CRC MSByte		
Slave Response	Slave ID	0x50 (ACK)	Page Index MSByte	Page Index LSByte	CRC LSByte	CRC MSByte

Example:
To read the flash index of the current page, assuming an Amulet slave ID of 0x01, send the following sequence:
0x01 0x51 0xC1 0xDC
The Amulet will respond with the following if the current page is index 0x20.
0x01 0x51 0x00 0x20 0x51 0xD1

Flow Diagram Example

The flow diagram in the table below depicts a sample CRC communications session between the Amulet LCD module and an external processor. This sample is setup as a dual master system, with both the Amulet and the external processor sharing the responsibility of being the master. It is possible to have a system where only the Amulet is the master, only the external processor is the master, or as in this case, a dual master setup.
The variables used in this example have the following values:

External processor's byte variable 01 = 0x38
External processor's string variable 01 = "Abc"
External processor's word variable 03 = 0x10E8
Amulet Internal RAM byte variable 0xF4 = 0xA8
Amulet Internal RAM word variable 0x76 = 0x0000
Amulet Internal RAM RPC buffer = 0x52 only
Following the communication session, the following has occurred:
External processor's byte variable 01 = 0xFE
External processor performs user-defined RPC 02. (There are no reserved RPC #'s, so all 256 RPC's can perform any desired function on your processor.)
Amulet Internal RAM word variable 0x76 = 0x02c9
Amulet Internal RAM RPC buffer = empty

Amulet LCD Module	**Dir.**	**External Processor**	**Description**
0x02 0x20 0x01 0x08 0x00	>>		Get byte variable 1
	<<	0x02 0x20 0x01 0x38 0x00 0x14	Return byte of byte var 1
0x02 0x22 0x01 0x09 0x60	>>		Get string variable 1
	<<	0x02 0x22 0x01 0x41 0x62 0x63 0x00 0x9E 0x9 'A' 'b' 'c'	Return string of string var 1
0x02 0x30 0x01 0xFE 0x81 0x83	>>		Set byte variable 1 to 0xFE
	<<	0x02 0x30 0x00 0xC4	Ack set byte variable 1 to 0xFE
0x02 0x37 0x02 0x47 0xF1	>>		Invoke RPC2 (User-defined)
	<<	0x02 0x37 0x41 0x06	Ack invoke RPC2
0x02 0x21 0x03 0x88 0x51	>>		Get word variable 3
	<<	0x02 0x21 0x03 0x10 0xE8 0xAA 0x72	Return word of word var 3
0x02 0x20 0x04 0xC8 0x03	>>		Get byte variable 4

			(no such variable on server)*
	<<	0x02 0xA0 0x05 0x68 0x03	Error (ACK) Move on.
	<<	0x01 0x20 0xF4 0x38 0x47	Get Internal RAM byte var 0xF4
0x01 0x20 0xF4 0xA8 0x47 0xC6	>>		Return value of IR byte var 0xF4
	<<	0x01 0x31 0x76 0x02 0xC9 0x37 0xD0	Set IR word var 0x76 =>0x02C9
0x01 0x31 0xC1 0xF4	>>		Ack set IR word var 0x76
	<<	0x01 0x27 0x40 0x3A	Get Internal RAM RPCs
0x01 0x27 0x01 0x52 0x31 0xBE	>>		Return IR RPC buffer

*NOTE: If master requests an invalid variable, the slave should at least respond with an ACK error code. It will allow the communications to continue on without displaying bogus data. This really should never happen since you have control of both the GEMstudio page and your processor's variables.

Summary of Amulet CRC protocol

Summary of Amulet CRC protocol commands with default opcodes:

Command Opcode	Description	Command Can Be Sent by Amulet	Command Can Be Sent by External Processor
0x20	Get byte variable	X	X
0x21	Get word variable	X	X
0x22	Get string variable	X	X
0x23	Get color variable		X
0x24	Get byte variable array	X	X
0x25	Get word variable array	X	X
0x26	Get color variable array		X
0x27	Get RPC buffer		X
0x28	Get label variable	X	
0x30	Set byte variable	X	X
0x31	Set word variable	X	X

0x32	Set string variable	X	X
0x33	Set color variable	X	X
0x34	Set byte variable array		X
0x35	Set word variable array		X
0x36	Set color variable array		X
0x37	Invoke RPC	X	
0x40	Set Graphic Primitive 8-bit color		X
0x41	Set Graphic Primitive 32-bit color		X
0x42	Set Graphic Primitive InternalRAM color index		X
0x43	Set Graphic Primitive Line Weight		X
0x44	Draw Pixel		X
0x45	Draw Line		X
0x46	Draw Rectangle		X
0x47	Draw Filled Rectangle		X
0x50	Jump To Page		X
0x51	Get Current Page		X

ASCII Protocol

An ASCII communication protocol can be used between the Amulet LCD module and your embedded device (external processor). This type of protocol encodes all data into ASCII, but leaves control bytes like opcodes outside of the ASCII range, which is 0x00 to 0x7F. To encode data into ASCII, each byte is first broken into two 4-bit pieces, called nibbles. These nibbles can be represented by a single hexacedimal character 0-9, A-F which is conveniently indexed 0x30-0x46 in ASCII. So the simple formula for "ASCII-izing" any byte is:

Hi Nibble = ((byte >> 4) & 0x0F) + 0x30
Low Nibble = (byte & 0x0F) + 30

Each command consists of an opcode and then ASCII data. There are three exceptions for data that is not encoded into ASCII.
1) When setting or requesting strings, UTF-8 ecoding represents characters beyond the limited ASCII range typically for language support beyond English..
2) Also within strings, the Font Style Escape is an inline control byte, and the following data byte is not ASCII encoded either.
3) The Jump to Specific Page command is not encoded into ASCII

Amulet as Master

Amulet master messages are initiated either by timer events or by user input from the touch panel. The Amulet LCD module(master) can send eleven different types of messages to the external processor (slave):

- A "Get byte variable" request. **(Amulet:port.byte(x).value())**
- A "Get word variable" request. **(word = 2 bytes) (Amulet:port.word(x).value())**
- A "Get string variable" request. **(Amulet:port.string(x).value())**
- A "Get label variable" request. **(Amulet:port.label(x).value())**
- A "Get byte variable array" request. **(Amulet:port.bytes(x).value())**
- A "Get word variable array" request. **(Amulet:port.words(x).value())**

- An "Invoke Remote Procedure Call (RPC)" command. **(Amulet:port.invokeRPC())**
- A "Set byte variable" command. **(Amulet:port.byte(x).setValue(y))**
- A "Set word variable" command. **(Amulet:port.word(x).setValue(y))**
- A "Set string variable" command. **(Amulet:port.string(x).setValue(y))**
- A "Set color variable" command. **(Amulet:port.color(x).setValue(y))**

Where port can be uart0, uart1, uart2 (if applicable) or USB

If the message is valid, the slave should either return the requested data (if a "Get" request) or confirm the message (if an "Invoke" or "Set" command). If the message is not valid, the slave should respond with an acknowledge (0xF0) to have the Amulet move to the next request, or the slave can respond with a negative acknowledgement (NAK) (0xF1) to have the Amulet resend the last message.

Amulet as Slave

The Amulet communication protocol is half-duplex, meaning the slave should not respond until the master is done sending its message. The external processor (master) can send sixteen different types of ASCII messages to the Amulet LCD module (slave). The external processor can read from and write to all the Amulet Internal RAM variables or Draw directly into the display. The external processor can also force the Amulet to jump to a specific page.

- A "Get Internal RAM byte variable" request.
- A "Get Internal RAM word variable" request. (word = 2 bytes)
- A "Get Internal RAM color variable" request. (color = 4 bytes)
- A "Get Internal RAM string variable" request.
- A "Get Internal RAM RPC buffer" request.
- A "Set Internal RAM byte variable" command.
- A "Set Internal RAM word variable" command.
- A "Set Internal RAM color variable" command.
- A "Set Internal RAM string variable" command.
- A "Set Internal RAM byte variable array" command.
- A "Set Internal RAM word variable array" command.
- A "Draw line" command.
- A "Draw rectangle" command.
- A "Draw filled rectangle" command.
- A "Draw pixel" command.
- A "Jump to specific page" command.

If the message is valid, the Amulet LCD module slave will either return the requested data (if a "Get" request) or confirm the message (if a "Draw" or "Set" command). If the message is not a valid Amulet message, the Amulet will not respond.

ASCII Commands

The protocol is the same regardless of who is the master. So if an external processor is requesting the value of a byte variable on the Amulet (which would be an Internal RAM byte variable), the command opcode would be the same as if the Amulet was requesting the value of a byte variable that resides on the external processor.

The table below defines fifteen messages that can be sent between the master and the slave, not counting the graphic primitive messages. The valid range of variables and Remote Procedure Calls is 0-0xFF. The valid range for byte variable values returned from the slave (in response to the "Get Byte variable" request) is also 0-0xFF. The valid range for word variable values returned from the slave (in response to the "Get Word variable" request) is 0-0xFFFF. String and label variable values returned from the slave (in response to the "Get String variable" request) can have a maximum of 0x1000 bytes plus a null termination character (0x00).
Since this is an ASCII protocol, two bytes must be sent out for every byte of data to be transmitted. All messages will start with a command character followed by the ASCII representation of all data. For example, if the page being compiled has a view widget with an href of **Amulet:uart0.byte(0x1A).value()**, which will send the "Get Byte Variable

#0x1A" request out uart0, the message to be transmitted would consist of three bytes. The first byte is the command byte for "Get Byte Variable" 0xD0. The second byte is 0x31, which is the ASCII representation of the high nibble"1". The third and final byte is 0x41, which is the ASCII representation of the low nibble "A".

NOTE: The slave must respond to every valid Amulet command, even if it's only an acknowledge (0xF0). When commands are not responded to, a time-out will occur after 200ms, by default, and that message will be repeated until either a response is received or after a total of eleven attempts. After eleven attempts, all UART or USB variables are reset in an attempt to resync with the slave processor.

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	By 1
Amulet Get Byte Variable	0xD0	Variable Hi Nibble	Variable Lo Nibble								
Server Response	0xE0	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value Lo Nibble						
Amulet Get Word Variable	0xD1	Variable Hi Nibble	Variable Lo Nibble								
Server Response	0xE1	Variable Hi Nibble	Variable Lo Nibble	|-----MS Byte-----| Value Hi | Value Hi Hi Nibble|Lo Nibble		|-------LS Byte-------| Value Lo | Value Lo Hi Nibble|Lo Nibble					
Amulet Get String Variable	0xD2	Variable Hi Nibble	Variable Lo Nibble								
Server Response	0xE2	Variable Hi Nibble	Variable Lo Nibble	ASCII char	ASCII char	ASCII char	ASCII .... 0x00 char				
Amulet Get Color Variable	0xF4	Variable Hi Nibble	Variable Lo Nibble								
Server Response	0xF5	Variable Hi Nibble	Variable Lo Nibble	|--Red Byte--| Hi Nibble | Lo Nibble		|--Green Byte--| Hi Nibble|Lo Nibble		|--Blue Byte--| Hi Nibble|Lo Nibble		|--Alpha Byte--| Hi Nibble|Lo Nibb	

Amulet Get Label Variable	0xD3	Variable Hi Nibble	Variable Lo Nibble				
Server Response	0xE3	Variable Hi Nibble	Variable Lo Nibble	ASCII char	ASCII char	ASCII char	ASCII .... 0x00 char

Amulet Get Remote Procedure Calls (RPC)	0xD4**	RPC flag Hi Nibble	RPC flag Lo Nibble				
Server Response	0xE4	RPC flag Hi Nibble	RPC flag Lo Nibble	RPC #1 Hi Nibble	RPC #1 Lo Nibble	RPC #n Hi Nibble	RPC #n .... 0x00 Lo Nibble

Amulet Set Byte Variable	0xD5	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value Lo Nibble		
Server Response	0xE5	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value Lo Nibble		

Amulet Set Word Variable	0xD6	Variable Hi Nibble	Variable Lo Nibble	|-----MS Byte-----| Value Hi | Value Hi Hi Nibble|Lo Nibble	|------LS Byte------| Value Lo | Value Lo Hi Nibble|Lo Nibble	
Server Response	0xE6	Variable Hi Nibble	Variable Lo Nibble	|-----MS Byte-----| Value Hi | Value Hi Hi Nibble|Lo Nibble	|-----LS Byte-----| Value Lo | Value Lo Hi Nibble|Lo Nibble	

Amulet Set String Variable	0xD7	Variable Hi Nibble	Variable Lo Nibble	ASCII char	ASCII char	ASCII char	ASCII .... 0x00 char
Server Response	0xE7	Variable Hi Nibble	Variable Lo Nibble	ASCII char	ASCII char	ASCII char	ASCII .... 0x00 char

Amulet Set Color Variable	0xF6	Variable Hi Nibble	Variable Lo Nibble	|--Red Byte--| Hi Nibble|Lo Nibble	|--Green Byte--| Hi Nibble|Lo Nibble	|--Blue Byte--| Hi Nibble|Lo Nibble	|--Alpha Byte--| Hi Nibble|Lo Nibb
Server Response	0xF7	Variable Hi Nibble	Variable Lo Nibble	|--Red Byte--| Hi Nibble|Lo Nibble	|--Green Byte--| Hi Nibble|Lo Nibble	|--Blue Byte--| Hi Nibble|Lo Nibble	|--Alpha Byte--| Hi Nibble|Lo Nibb

Amulet Invoke Remote Procedure Call (RPC)	0xD8	RPC Hi Nibble	RPC Lo Nibble
Server Response	0xE8	RPC Hi Nibble	RPC Lo Nibble

Amulet Get Byte Variable Array	0xDD	Variable Hi Nibble	Variable Lo Nibble		
Server Response	0xED	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value .... 0x00 Lo Nibble

Amulet Get Word Variable Array	0xDE	Variable Hi Nibble	Variable Lo Nibble		
Server Response	0xEE	Variable Hi Nibble	Variable Lo Nibble	|-----MS Byte-----| Value Hi|Value Hi Hi Nibble|Lo Nibble	|-----LS Byte--------| Value Lo | Value Lo .... 0x00 Hi Nibble|Lo Nibble

Amulet Set Byte Variable Array	0xDF**	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value .... 0x00 Lo Nibble
Server Response	0xEF	Variable Hi Nibble	Variable Lo Nibble	Array Cnt Hi Nibble	Array Cnt Lo Nibble

Amulet Set Word Variable Array	0xF2**	Variable Hi Nibble	Variable Lo Nibble	|-----MS Byte-----| Value Hi | Value Hi Hi Nibble|Lo Nibble		|-----LS Byte-----| Value Lo | Value Lo Hi Nibble|Lo Nibble	....	0x00		
Server Response	0xF3	Variable Hi Nibble	Variable Lo Nibble	Array Cnt Hi Nibble	Array Cnt Lo Nibble					

** Denotes command is only applicable when Amulet is set as
Slave.

Table 1. Thirteen types of messages can be sent between the master and the slave, not counting the graphic primitives.

Synchronization--The master initiates all communications by sending a message to the slave. All valid messages from the Amulet to the external processor start with one of the opcode bytes. These opcodes are considered the Master Start Of Message (MSOM) bytes.

NOTE: These MSOM bytes signify the start of a message and they are not allowed in the body of any message except for when part of a UTF-8 string. For non-string messages, the only valid characters in the body of a message are: ASCII 0-9 (0x31-0x39), and A-F (0x41-0x46). If the slave receives any character other than those specified, the message should be considered errant, and the slave should start over hunting for a new MSOM character.

All slave responses must start with the counterpart of the MSOM character that began the message that is being responded to. The body of the response message starts with a byte for byte echo of the command message. The echo is then followed by any optional response data (in ASCII format).

Upon receiving the last byte of a valid message from the master, the slave then has, by default, 200ms to respond to the message before the master times out. After 200ms, if there is no response, the master will continue to repeat the message until a response is received. After 10 unsuccessful attempts, the Amulet will flush its transmit buffer and reset all UART variables.Other time out durations are set in the Communications Tab of the Project Properties dialog.

NOTE: The external processor slave must respond to every valid Amulet master command. It is okay to respond with a single byte of acknowledgement (0xF0) without transmitting data, but all Amulet commands should be responded to. When commands are not responded to, a time-out will occur, and that message will be repeated 10 more times before flushing the transmit buffer and resetting all UART variables.

ASCII-Based Graphic Primitives

Using GEMstudio to create your projects at compile time allows you to make rich user interfaces quickly and easily. Sometimes, though, the ability to draw graphic primitives like lines, rectangles and filled rectangles at runtime, is needed. GEMstudio does inherently give you the ability to do this via GEMscript commands, but sometimes the.
With the addition of a new communications protocol that allows the external processor to be the master and the Amulet to be the slave comes a way to send unsolicited graphic primitives to the Amulet. The drawing of these graphic primitives is independent of the uHTML that is currently being run on the Amulet. You do need to keep in mind that the uHTML will still be running, so any widgets or objects that write to the LCD might write over the graphics primitive you send to the LCD.

If using the ASCII-based protocol, the Amulet will respond to all graphic primitive commands with a response opcode (0xE9-0xEB) and an echo of all other bytes. If it is desired to have the Amulet respond with an ACK(0xF0) or no response at all, a SlaveAckRsp or Slave NoRsp can be used in the communications options. One thing to keep in mind is that the Amulet will use that attribute for all "Set" or "Draw" commands coming from your external processor.

The line primitive draws a line from point 1(x and y coordinates) to point 2(x and y coordinates), with a 32-bit color and that has a line weight of 1-15. The line primitive is drawn by sending out the "Draw line primitive" opcode(0xD9), followed by the x-coordinate of point 1, the y-coordinate of point 1, the x-coordinate of point 2, the y-coordinate of point 2, the line color and finally the line weight. The x and y coordinates are 16-bit numbers. The line color is made up of three 8-bit numbers, one blue, one green, and one red. The line weight is a 4-bit number. Because Amulet uses an ASCII protocol, each coordinate that is sent to the Amulet is comprised of four different bytes. For example, if the x-coordinate of point 1 is 0x0020, the 4 bytes that would be sent to the Amulet for that particular coordinate would be 0x30,0x30,0x32,0x30. The line color is comprised of six different bytes, two bytes for the blue component, two bytes for the green component, and two bytes for the red component. For example, if the desired line color is purple (blue=0x80, green=0x00, red=0x7E), the 6 bytes sent to the Amulet would be 0x38,0x30,0x30,0x30,0x37,0x45. Line weight, when sent to the Amulet, is only one byte. The line weight specifies the thickness of the line, range of 1-15. The entire line primitive message sent to the Amulet is a total of 24 bytes.

The rectangle primitive draws a rectangle with a given starting top left point(x and y coordinates) and a delta x and delta y, the line color and finally the line weight.. The rectangle primitive is drawn by sending out the "Draw rectangle primitive" opcode(0xDA), followed by four bytes specifying the x-coordinate of the topleft point, four bytes specifying the y-coordinate of the topleft point, four bytes specifying the delta x, four bytes specifying the delta y, six bytes specifying the line color and finally one byte specifying the line weight. The x and y coordinates and the delta x and delta y are all 16-bit numbers. The line color is comprised of three 8-bit color components, a blue component, a green component, and a red component. The line weight is a 4-bit number. The delta x is a 16-bit number specifying the length of the rectangle in the x direction and the delta y is a 16-bit number specifying the height of the rectangle in the y direction. The line weight specifies the thickness of the line, range of 1-15. The entire rectangle primitive message sent to the Amulet is a total of 24 bytes.

The fill rectangle primitive draws a solid rectangle with a given starting top left point(x and y coordinates) and a delta x and delta y, plus a line color. The fill rectangle primitive is drawn by sending out the "Draw fill rectangle primitive" opcode(0xDB), followed by four bytes specifying the the x-coordinate of the topleft point, four bytes specifying the y-coordinate of the topleft point, four bytes specifying the delta x, four bytes specifying the delta y, six bytes specifying the fill color and finally one filler byte. The x and y coordinates and the delta x and delta y are all 16-bit numbers. The delta x is a 16-bit number specifying the length of the rectangle in the x direction and the delta y is a 16-bit number specifying the height of the rectangle in the y direction. The fill color is comprised of six different bytes, two bytes for the blue component, two bytes for the green component, and two bytes for the red component. The line weight is a 4-bit number, but the line weight is not used by this primitive, but the byte still must be sent, range of 1-15. The entire fill rectangle primitive message sent to the Amulet is a total of 24 bytes.

If a graphics primitive is sent that does not fit within the bounds of the given LCD (i.e. a delta x of 380 pixels on a 320 x 240 LCD) the Amulet will just ignore the request. It will respond back serially, but the graphic primitive will not be drawn.

Notice that the order of the color bytes that are sent to the Amulet is blue, green, red. In the HTML, when specifying colors, the order is red, green, blue.

Examples:

Drawing a Line:
To draw a line from (0x05,0x07) to (0x65,0x67), using color blue:0x00, green:0x00, red:0x22, and a line weight of 4 the following would be sent to the Amulet:

```
0xD9,0x30,0x30,0x30,0x35,0x30,0x30,0x30,0x37,0x30,0x30,0x36,0x35,0x30,0x30,0x36,0x37
 |   {-----------------} {-----------------} {-----------------}
{-----------------}
 |        pnt 1, x            pnt 1, y             pnt 2, x            pnt 2, y

draw
    (0x05)              (0x07)               (0x65)             (0x67)
line
```

```
opcode

0x30,0x30,0x30,0x30,0x32,0x32,0x34
{---------------------------}  |
            color           line weight
 b:0x00    g:0x00    r:0x22  (0x04)
```

Drawing a Rectangle:
To draw a rectangle that is 0x10C pixels wide, 0x82 pixels tall, has a topleft point at (0x0A,0x05), a line weight of 2 and using line color blue:0x00, green:0x00, red:0x00, the following would be sent to the Amulet:

```
0xDA,0x30,0x30,0x30,0x41,0x30,0x30,0x30,0x35,0x30,0x31,0x30,0x43,0x30,0x30,0x38,0x32
 |   {-----------------} {-----------------} {-----------------}
{-----------------}
 |        pnt 1, x           pnt 1, y             delta x            delta y

draw
      (0x0A)              (0x05)              (0x10C)              (0x82)
rectangle
opcode

0x30,0x30,0x30,0x30,0x30,0x30,0x32
{---------------------------}  |
            color           line weight
 b:0x00    g:0x00    r:0x00  (0x04)
```

Drawing a Filled Rectangle:
To draw a filled rectangle that is 0x140 pixels wide, 0xF0 pixels tall, has a topleft point at (0x00,0x00) and using fill color blue:0x25, green:0x46, red:0x73, the following would be sent to the Amulet:

```
0xDB,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x31,0x34,0x30,0x30,0x30,0x46,0x30
 |   {-----------------} {-----------------} {-----------------}
{-----------------}
 |        pnt 1, x           pnt 1, y             delta x            delta y

draw
      (0x00)              (0x00)              (0x140)              (0xF0)
fill

rectangle
opcode

0x32,0x35,0x34,0x36,0x37,0x33,0x32
{---------------------------}  |
            color           line weight
b:0x25    g:0x46    r:0x73  (N/A)
```

Graphic Primitives

See Graphic Primitives for more information regarding the use of graphic primitives.

The table below defines the three types of messages regarding graphic primitives that can be sent between an external processor and the Amulet. If a graphics primitive is sent that does not fit within the bounds of the given LCD (i.e. a delta x of 380 pixels on a 320 x 240 LCD) the Amulet will not draw the graphic primitive.

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9

"Draw" Line Primitive	0xD9	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	**Byte 11**	**Byte 12**	**Byte 13**	**Byte 14**	**Byte 15**	**Byte 16**	**Byte 17**	**Byte 18**	**Byte 19**
Pnt 2 X Hi Byte Hi Nibble	Pnt 2 X Hi Byte Lo Nibble	Pnt 2 X Lo Byte Hi Nibble	Pnt 2 X Lo Byte Lo Nibble	Pnt 2 Y Hi Byte Hi Nibble	Pnt 2 Y Hi Byte Lo Nibble	Pnt 2 Y Lo Byte Hi Nibble	Pnt 2 Y Lo Byte Lo Nibble	Blue Hi Nibble	Blue Lo Nibble
Byte 20	**Byte 21**	**Byte 22**	**Byte 23**	**Byte 24**					
Green Hi Nibble	Green Lo Nibble	Red Hi Nibble	Red Lo Nibble	Line Weight					

Message	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Byte 9**
Amulet Response	0xE9	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	**Byte 11**	**Byte 12**	**Byte 13**	**Byte 14**	**Byte 15**	**Byte 16**	**Byte 17**	**Byte 18**	**Byte 19**
Pnt 2 X Hi Byte Hi Nibble	Pnt 2 X Hi Byte Lo Nibble	Pnt 2 X Lo Byte Hi Nibble	Pnt 2 X Lo Byte Lo Nibble	Pnt 2 Y Hi Byte Hi Nibble	Pnt 2 Y Hi Byte Lo Nibble	Pnt 2 Y Lo Byte Hi Nibble	Pnt 2 Y Lo Byte Lo Nibble	Blue Hi Nibble	Blue Lo Nibble
Byte 20	**Byte 21**	**Byte 22**	**Byte 23**	**Byte 24**					
Green Hi Nibble	Green Lo Nibble	Red Hi Nibble	Red Lo Nibble	Line Weight					

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
"Draw" Rectangle Primitive	0xDA	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	**Byte 11**	**Byte 12**	**Byte 13**	**Byte 14**	**Byte 15**	**Byte 16**	**Byte 17**	**Byte 18**	**Byte 19**
Delta X Hi Byte Hi Nibble	Delta X Hi Byte Lo Nibble	Delta X Lo Byte Hi Nibble	Delta X Lo Byte Lo Nibble	Delta Y Hi Byte Hi Nibble	Delta Y Hi Byte Lo Nibble	Delta Y Lo Byte Hi Nibble	Delta Y Lo Byte Lo Nibble	Blue Hi Nibble	Blue Lo Nibble
Byte 20	**Byte 21**	**Byte 22**	**Byte 23**	**Byte 24**					
Green Hi Nibble	Green Lo Nibble	Red Hi Nibble	Red Lo Nibble	Line Weight					

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
Amulet Response	0xEA	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	**Byte 11**	**Byte 12**	**Byte 13**	**Byte 14**	**Byte 15**	**Byte 16**	**Byte 17**	**Byte 18**	**Byte 19**
Delta X Hi Byte Hi Nibble	Delta X Hi Byte Lo Nibble	Delta X Lo Byte Hi Nibble	Delta X Lo Byte Lo Nibble	Delta Y Hi Byte Hi Nibble	Delta Y Hi Byte Lo Nibble	Delta Y Lo Byte Hi Nibble	Delta Y Lo Byte Lo Nibble	Blue Hi Nibble	Blue Lo Nibble
Byte 20	**Byte 21**	**Byte 22**	**Byte 23**	**Byte 24**					

Green Hi Nibble	Green Lo Nibble	Red Hi Nibble	Red Lo Nibble	Line Weight					

Message	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Byte 9**
"Draw"Fill Rectangle Primitive	0xDB	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	**Byte 11**	**Byte 12**	**Byte 13**	**Byte 14**	**Byte 15**	**Byte 16**	**Byte 17**	**Byte 18**	**Byte 19**
Delta X Hi Byte Hi Nibble	Delta X Hi Byte Lo Nibble	Delta X Lo Byte Hi Nibble	Delta X Lo Byte Lo Nibble	Delta Y Hi Byte Hi Nibble	Delta Y Hi Byte Lo Nibble	Delta Y Lo Byte Hi Nibble	Delta Y Lo Byte Lo Nibble	Blue Hi Nibble	Blue Lo Nibble
Byte 20	**Byte 21**	**Byte 22**	**Byte 23**	**Byte 24**					
Green Hi Nibble	Green Lo Nibble	Red Hi Nibble	Red Lo Nibble	Line Weight					

Message	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Byte 9**
Amulet Response	0xEB	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	**Byte 11**	**Byte 12**	**Byte 13**	**Byte 14**	**Byte 15**	**Byte 16**	**Byte 17**	**Byte 18**	**Byte 19**
Delta X Hi Byte Hi Nibble	Delta X Hi Byte Lo Nibble	Delta X Lo Byte Hi Nibble	Delta X Lo Byte Lo Nibble	Delta Y Hi Byte Hi Nibble	Delta Y Hi Byte Lo Nibble	Delta Y Lo Byte Hi Nibble	Delta Y Lo Byte Lo Nibble	Blue Hi Nibble	Blue Lo Nibble

Byte 20	Byte 21	Byte 22	Byte 23	Byte 24	
Green Hi Nibble	Green Lo Nibble	Red Hi Nibble	Red Lo Nibble	Line Weight	

RPC buffer

If a setup where the Amulet is always the slave is needed or desired, then up to six RPCs can be buffered in the Amulet's Internal RAM. The external processor can request the contents of the RPC buffer by sending a "Get Internal RAM RPC buffer" request (0xD4), followed by the RPC buffer flag byte (which is ASCII-ized like all data within the Amulet protocol). The RPC buffer flag byte options are **0x00:**send all RPCs from buffer, then flush. 0xFF:flush RPC buffer. When the Amulet is the master and an RPC is invoked, the Amulet will immediately send out the RPC command. If the Amulet is setup to use the Internal RAM RPC buffer instead, then the Amulet will send the RPC to an RPC buffer. The RPC buffer can only be read by an external processor by sending a "Get Internal RAM RPC buffer" request (0xD4). The Amulet will respond with all the RPCs (up to six) stored in the RPC buffer, and then a null termination character, to signify the end of the buffer. After sending out the contents of the RPC buffer, the Amulet will flush the buffer.

Software handshaking using a modified XON/XOFF protocol

In a system where the Amulet is the slave, it is possible to send a number of master messages in a row without waiting for a response from the Amulet. This becomes an important point when there are a large number of messages that need to be sent to the Amulet in a short period of time. If the Amulet is set to respond to every master message, like it will by default, or if you have setup the Amulet to respond with an ACK by using the SlaveAckRsp, only the last sent message will be responded to. So, if you send over 30 master messages packed together as a single stream of bytes, only one response message or ACK will be returned from the Amulet.

One of the problems of this technique is that the Amulet has a 0x1000-byte receive buffer which can be filled to capacity if too many bytes are sent to the Amulet. To counter this problem, we've created a modified form of the XON/XOFF software handshaking protocol. In order to use the Amulet XON/XOFF protocol, you should probably use the Communications Setting SlaveNoRsp so that the Amulet will not respond to any master messages that do not require a response. Whenever the Amulet receives an XOFF command (0x13) it will respond with an XON command (0x11). So, to safely use the software handshaking, you should send a number of messages which will be less than 0x1000 bytes in length, then terminate the stream of bytes with an XOFF command. Do not send any further commands until the Amulet has responded with the XON command. At that point, you can be assured that all the previous commands have been acted upon and the receive buffer is completely empty, thus allowing for another large string of bytes, up to 0x1000.

This raises the question, why not just use SlaveAckRsp and wait for the ACK. In a perfect world, that would work perfectly fine. Unfortunately, when talking about serial communications, it is folly to believe that everything will be perfect. If there is a slight delay in between messages, it is possible for the Amulet to sneak an ACK out before the entire master stream of bytes is completely done. This could result in the master seeing the ACK, and depending on how/when the code decides to look at the incoming messages, incorrectly assuming that the ACK was in response to the last individual message of the stream. This could result in the master starting to send another large stream of bytes. That might or might not be a problem. If the Amulet was able to finish parsing the final message of the stream, then the receive buffer counters will be reset and the next stream will fit in the receive buffer, assuming the stream is less than 0x1000 bytes. But, there could be a race condition where the Amulet doesn't quite finish parsing the final message of the stream before the master starts sending the new stream. This could result in entire streams being lost. Once again, that is not acceptable in most applications.

Therefore, if you are sending large streams to the Amulet, it is highly suggested that you use the SlaveNoRsp Meta attribute and also send an XOFF command at the conclusion of your stream. Do not send your next stream until the Amulet returns an XON command. This will close any race condition windows.

Jump to specific page

It is possible to send a Master Message to the Amulet which will force it to jump to a specific page within the project. The page number is an internal 16-bit number that the GEMstudio compiler generates. All pages and images are assigned an internal number which can be determined by looking at the Amulet link map. When this message is received by the Amulet, it will react as if the Amulet:fileNumber(x) was launched, meaning the Amulet will jump directly to the page specified by the 16-bit internal number. You must NOT jump directly to an image file number, it must be a valid page. If you do errantly jump to a non-valid page, the Amulet OS will respond with a soft reset. This will act exactly as if the reset button was pressed.

The message structure is different than all other ASCII commands. This message is NOT in ASCII. The first two bytes for jumping to a specific page are always 0xA0, 0x02. The next byte is the MSByte of the internal number and the following byte is the LSByte of the internal number. The final byte is the checksum byte, which when added to the first four bytes, should result with the LSByte of the sum being equal to 0x00.

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Jump To Page Command	0xA0	0x02	Page Index MSByte	Page Index LSByte	0x100 - Checksum LS Byte

Examples:

To jump to page 0x25:
0xA0,0x02,0x00,0x25,0x39
Notice that 0xA0+0x02+0x00+0x25+0x39 = 0x0100. LSByte of sum is 0x00.
To jump to page 0xF0:
0xA0,0x02,0x00,0xF0,0x6E
Notice that 0xA0+0x02+0x00+0xF0+0x6E = 0x200. LSByte of sum is 0x00.

Soft Reset

It is possible to send a Master Message to the Amulet which will force it to perform a soft reset. It will react exactly as if the reset button was pressed.
The message structure is similar to the Jump To Page command. This message is NOT in ASCII. The first four bytes are 0xA0, 0x02, 0xFF, 0xFF. The final byte is the checksum byte, which when added to the first four bytes, should result with the LSByte of the sum being equal to 0x00. Given that, the entire five byte string is 0xA0, 0x02, 0xFF, 0xFF, 0x60.
Example:

To cause a soft reset:
0xA0,0x02,0xFF,0xFF,0x60
Notice that 0xA0+0x02+0xFF+0xFF+0x60 = 0x0300. LSByte of sum is 0x00.

Flow Diagram Example

The flow diagram in the table below depicts a sample ASCII communications session between the Amulet LCD module and an external processor. This sample is setup as a dual master system, with both the Amulet and the external processor sharing the responsibility of being the master. It is possible to have a system where only the Amulet is the master, only the external processor is the master, or as in this case, a dual master setup.

The variables used in this example have the following values:

External processor's byte variable 01 = 0x38
External processor's string variable 01 = "Abc"
External processor's word variable 03 = 0x10E8
Amulet Internal RAM byte variable 0xF4 = 0xA8
Amulet Internal RAM word variable 0x76 = 0x0000
Amulet Internal RAM RPC buffer = 0x52 only
Following the communication session, the following has occurred:
External processor's byte variable 01 = 0xFE
External processor performs user-defined RPC 02. (There are no reserved RPC #'s, so all 256 RPC's can perform any desired function on your processor.)
Amulet Internal RAM word variable 0x76 = 0x02c9
Amulet Internal RAM RPC buffer = empty

Amulet LCD Module	**Dir.**	**External Processor**	**Descr**
0xD0 0x30 0x31 '0' '1'	>>		Get varia
	<<	0xE0 0x30 0x31 0x33 0x38 '0' '1' '3' '8'	Retur of byt
0xD2 0x30 0x31 '0' '1'	>>		Get varia
	<<	0xE2 0x30 0x31 0x41 0x62 0x63 0x00 '0' '1' 'A' 'b' 'c'	Retur of strir
0xD5 0x30 0x31 0x46 0x45 '0' '1' 'F' 'E'	>>		Set variable
	<<	0xE5 0x30 0x31 0x46 0x45 '0' '1' 'F' 'E'	Confi byte va
0xD8 0x30 0x32 '0' '2'	>>		Invok (User-c
	<<	0xE8 0x30 0x32 '0' '2'	Cor invoke
0xD1 0x30 0x33 '0' '3'	>>		Get varia
	<<	0xE1 0x30 0x33 0x31 0x30 0x45 0x38 '0' '3' '1' '0' 'E' '8	Retur of wor

0xD0 0x30 0x34 '0' '4'	>>		Get varia (not on
	<<	0xF0	Ackno (no
	<<	0xD0 0x46 0x34 'F' '4'	Get I RAM var
0xE0 0x46 0x34 0x41 0x38 'F' '4' 'A' '8'	>>		Return IR byte
	<<	0xD6 0x37 0x36 0x30 0x32 0x43 0x39 '7' '6' '0' '2' 'C' '9'	Set I var
0xE6 0x37 0x36 0x30 0x32 0x43 0x39 '7' '6' '0' '2' 'C' '9'	>>		Confrir of IR var
	<<	0xD4 0x30 0x30 '0' '0'	Get I RAM
0xE4 0x30 0x30 0x35 0x32 0x00 '0' '0' '5' '2'	>>		Re IR RPC (r termi

***NOTE:** If master requests an invalid variable or RPC, the slave should respond with an acknowledgment (0xF0).

Summary of Amulet ASCII protocol

Command Opcode	Response Opcode	Description	Command Can Be Sent by Amulet	Command Can Be Sent by External Processor
0xD0	0xE0	Get byte variable	X	X
0xD1	0xE1	Get word variable	X	X
0xD2	0xE2	Get string variable	X	X
0xD3	0xE3	Get label variable	X	
0xD4	0xE4	Get RPC buffer		X
0xD5	0xE5	Set byte variable	X	X

0xD6	0xE6	Set word variable	X	X
0xD7	0xE7	Set string variable	X	X
0xD8	0xE8	Invoke RPC	X	
0xD9	0xE9	Draw Line		X
0xDA	0xEA	Draw Rectangle		X
0xDB	0xEB	Draw Filled Rectangle		X
0xDC	0xEC	Draw Pixel		X
0xDD	0xED	Get byte variable array	X	
0xDE	0xEE	Get word variable array	X	
0xDF	0xEF	Set byte variable array		X
	0xF0	Acknowledgment (ACK)	X	X
	0xF1	Negative Acknowledgment (NAK)		X
0xF2	0xF3	Set word variable array		X
0xF4	0xF5	Get Color variable		X
0xF6	0xF7	Set Color variable	X	X

UTF-8

Sending strings to the Amulet which contain characters between 0x80-0xFFFF

When sending strings to the Amulet, all characters from 0x20-0x7F are valid. The Amulet protocol expects any characters above 0x7F to be in the UTF-8 format. See the documentation on UTF-8 below. In order for the Amulet to display characters over 0x7F, the font used to display these characters must have those characters explicitely defined in the .auf font file. By default, only the characters 0x20-0x7F are saved in the .auf file, but in the Amulet GEM Font Converter, you can optionally save all the characters up to 0xFFFF.

UTF-8

UTF-8 (8-bit Unicode Transformation Format) is a variable-length character encoding for Unicode. Like UTF-16 and UTF-32, UTF-8 can represent every character in the Unicode character set, but unlike them it has the special property of being backwards-compatible with ASCII.

UTF-8 encodes each character (code point) in 1 to 4 octets (8-bit bytes). The first 128 characters of the Unicode character set (which correspond directly to the ASCII) use a single octet with the same binary value as in ASCII.
The UTF-8 encoding is variable-width, with each character represented by 1 to 4 bytes. Each byte has 0–4 leading consecutive '1' bits followed by a '0' bit to indicate its type. 2 or more '1' bits indicates the first byte in a sequence of that many bytes. The scalar value of the Unicode code point is the concatenation of the non-control bits. In this table, zeros and ones in black represent control bits, each x represents one of the lowest 8 bits of the Unicode value, y represents the next higher 8 bits, and z represents the bits higher than that.

Unicode range		Encoded Bytes	Example
Hex	Binary		
U+0000 to U+007F	00000000 to 01111111	0xxxxxxx	'$' U+0024 = 00100100 → 00100100 → 0x24

U+0080 to U+07FF	00000000 10000000 to 00000111 11111111	110yyyxx 10xxxxxx	'¢' U+00A2 = 00000000 10100010 → 11000010 10100010 → 0xC2 0xA2
U+0800 to U+FFFF	00001000 00000000 to 11111111 11111111	1110yyyy 10yyyyxx 10xxxxxx	'€' U+20AC = 00100000 10101100 → 11100010 10000010 10101100 → 0xE2 0x82 0xAC

So the first 128 characters (US-ASCII) need one byte. The next 1,920 characters need two bytes to encode. This includes Latin letters with diacritics and characters from Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac and Tana alphabets. Three bytes are needed for the rest of the Basic Multilingual Plane (which contains virtually all characters in common use). Four bytes are needed for characters in the other planes of Unicode, which include less common CJK characters and various historic scripts.

With these restrictions, bytes in a UTF-8 sequence have the following meanings. The ones marked in red can never appear in a legal UTF-8 sequence. The ones in green are represented in a single byte. The ones in blue must only appear as the first byte in a multi-byte sequence, and the ones in orange can only appear as the second or later byte in a multi-byte sequence:

UTF-8 byte range			**Interpretation**
Binary	**Hex**	**Decim**	
00000000-0111	00-7F	0-127	Single-byte encoding (compatible with US-ASCII)
10000000-1011	80-BF	128-191	Second, third, or fourth byte of a multi-byte sequence
11000000-1100	C0-C1	192-193	Overlong encoding: start of 2-byte sequence, but would encode a code point ≤ 127
11000010-1101	C2-DF	194-223	Start of 2-byte sequence
11100000-1110	E0-EF	224-239	Start of 3-byte sequence
11110000-1111	F0-F4	240-244	Start of 4-byte sequence (including invalid codepoints between 110000 and 13FFFF)
11110101-1111	F5-F7	245-247	Restricted by RFC 3629: start of 4-byte sequence for codepoint above 10FFFF (actually, starts at 140000)
11111000-1111	F8-FB	248-251	Restricted by RFC 3629: start of 5-byte sequence
11111100-1111	FC-FD	252-253	Restricted by RFC 3629: start of 6-byte sequence
11111110-1111	FE-FF	254-255	Invalid: not defined by original UTF-8 specification

Dual Master Collisions

In a dual-master system, it is possible that both masters will choose to send a message out at the same time. If the Amulet sees an incoming master message coming from the external processor while it is in the process of sending a master message of its own, it will finish sending its master message and then immediately respond to the incoming message, assuming it is a valid message. After completely responding to the master message, the Amulet will then wait for a response to its own master message. If the host does not respond to the Amulet message, the Amulet will timeout and resend its own master message again.

Sending Master Messages Between Amulet Page Changes

When the Amulet changes from one page to another, all UART buffers are flushed, so if you are in the middle of sending the Amulet a Master Message while it is changing pages, it is possible that the Amulet will not fully receive your message. Another thing to keep in mind is that when first loading a page, the transmission of messages is halted until the page is fully rendered. The Amulet is capable of buffering incoming messages, but it will not process or respond to any incoming messages until the page is fully rendered. It can take over 500ms for some complex pages to be fully rendered, so if you were to send a Master Message at the beginning of a page change, the Amulet might not respond back for a while. If the Amulet does respond, it will have performed the request, albeit maybe not exactly when you thought it should.

With the above in mind, if your processor pounds Master Messages out at a rapid rate, you might want to have all your pages start out by sending an RPC, set byte or byteOut command that lets your processor know when it is okay to start transmitting again. You could also have an RPC, set byte or byteOut command go out prior to leaving any page, so your processor will know when to halt transmissions as well.

Font Style Escape

Sending strings to the Amulet which contain the font style escape byte

To change the font style, size, color, or the font itself in a UART string, see Using Amulet Formatted Text for formatting options. The same formatting used within GEMstudio works exactly the same on UART strings.

Sending A Single Byte Without A Response

When it is desired to send a single byte out the UART or USB without regard to the Amulet protocol, then the href command to use is **Amulet:port.byteOut(x)**, where x is a raw byte to send out and port is uart0, uart1, oart2(if applicable) or USB. By default, x is a decimal number, but if a hexadecimal number is desired, precede the byte with 0x to specify a hexadecimal number. For instance, to send out a single 0x3D out uart0, use the following: **Amulet:uart0.byteOut(0x3D)**.

To send out a single ASCII character, use single quotes around the ASCII character. For instance, to send out an equals sign, use the following: **Amulet:uart0.byteOut('=')**.

Note that both **Amulet:uart0.byteOut(0x3D)** and **Amulet:uart0.byteOut('=')** will send out the same 0x3D since the ASCII representation of the equals sign is 0x3D.

When the byteOut command is used, it is not part of the Amulet protocol, which means there are no header bytes and the data is not ASCII-ized for you. The byte you want to be sent out is what will be transmitted, nothing more and nothing less. This is a unidirectional message, meaning that the byte will be sent out, but it will not be expecting, nor accepting, any responses.

Sending A Stream Of Bytes Without A Response

To send a stream of raw bytes out the UART or USB can be accomplished by calling multiple **Amulet:port.byteOut()** functions separated by commas, but that is not the most efficient way. The href command **Amulet:port.streamOut(x1+x2+...xn)** will send out the stream of bytes without any formatting. Notice that each byte is separated by a plus sign (+). Like the byteOut command, it is not part of the Amulet protocol, so the message will be sent out, but it will not be expecting, nor accepting, any responses. The bytes to be sent out are by default decimal. If hexadecimal numbers are desired, precede the numbers with 0x. For instance, to send out a stream on uart0 consisting of 0x01, 0x11, 0x22, 0x33, use the following:

Amulet:uart0.streamOut(0x01+0x11+0x22+0x33)

As is the case with the byteOut() function call, the Amulet OS is not expecting, nor accepting, any responses to the streamOut() function call.

Sending A String Of Bytes Without A Response

To send a stream of raw bytes out one of the communication ports can be accomplished by calling multiple **Amulet:port.byteOut()** functions separated by commas, but that is not the most efficient way. The href command **Amulet:UART.stringOut('string') or Amulet:USB.stringOut('string')** will send out the string of bytes without any formatting and it will not send out the null termination character. You can also use InternalRAM string variables as strings to be sent out. For instance, to send out uart0 the value of InternalRAM.string(0), use the following:

Amulet:uart0.stringOut(InternalRAM.string(0))

If it is desired to send out an ASCII string, that can be accomplished by entering a string that is enclosed by either single or double quotes. (Please see the section on entering strings for more information) For example, to send a string out uart0 that says "123 ABC", use the following:

Amulet:uart0.stringOut('123 ABC')

One thing to note is that the ASCII string that is sent out is not null terminated. If a null termination character is required, then you can enter an Amulet:port.byteOut(0x00) immediately following the streamOut() function call, separated by a comma, such as:

Amulet:uart0.stringOut('123 ABC'),Amulet:uart0.byteOut(0x00)

As is the case with the byteOut() function call, the Amulet OS is not expecting, nor accepting, any responses to the stringOut() function call.

Customizing the Amulet protocol command opcodes

By default, the Amulet protocol uses the Command and Response Opcodes specified in this document. You can customize the Command and Response Opcodes at compile time by including a customization file in your project. The customization file must have an "ini" extension and must reside in the same directory as your GUI project. You must include the customization file in a META tag in the home page of your project. The syntax to include the file is:

<META NAME="initCommands" SRC="filename.ini">

The customization file, **filename**, must have a .ini extension and it must be located in the same directory as your .gemp file. Inside the customization file, any line preceded with // is treated as a comment. All customizations must be located in the far left column, so do not tab over. The GEMcompiler recognizes both decimal and hexadecimal numbers.

The default file, called **defaultCommands.ini** is located in the %AllUsersProfile%\AmuletTech\Global\Configuration \Protocol folder. If the above META tag is not specified, **defaultCommands.ini** is used as the opcode definition file. If you are going to make a customization file, you should copy the **defaultCommands.ini** and place it in the same directory as your .gemp file. The file can be renamed as long as it retains the .ini extension. The values of the opcodes can be changed to practically any single byte value between (1-254)[*]. All the names of the opcodes can be cross-referenced with the list below. The opcode names are self documenting.

[*]There are three opcodes that are off limits. They are 0x00, 0xA0 and 0xFF. These must not be used to customize any of the Command or Response Opcodes.

Note: For users who don't want to change the opcodes from an earlier version of Amulet software, we have included **origCommands.ini** in the %AllUsersProfile%\AmuletTech\Global\Configuration\Protocol folder. Copy origCommands.ini into your projects main directory and use the initCommands META with **origCommands.ini** as your SRC file.

Serial Terminal

Docklight is a testing, analysis and simulation tools for serial communication protocols. You can easily set it up to reply to Amulet protocol commands.

With the permission of Docklight, Amulet has provided a link to Docklight here which you can install separate from GEMstudio. The Scripting version is useful for calculating CRCs on the fly for dynamic messages. The base version is useful for ASCII protocol or static CRC messages where you calculate the CRC yourself.

By installing Docklight, you adhere to the End User License Agreement set forth by Docklight during the install.
If you would like to purchase the full version of Docklight, you can do so by going to Docklight's web site, http://www.docklight.de

Amulet Bitmap Format

All images stored within the Amulet embedded flash are saved in an Amulet proprietary format. Most images are compressed when saved, but when dealing with Dynamic Image Widgets, the GEMcompiler leaves the canvas image uncompressed. When an external processor sends a new image serially to the Amulet, the Amulet replaces the existing image stored in the embedded flash with the incoming image. To keep from overwriting other files within the embedded flash, the image to be sent to the Amulet must be the exact same dimensions as the canvas image.

The Amulet Bitmap Format is slightly different between the standard Amulet module and the Amulet MK-07C-HP Module.

Standard Module

The Standard module's first 24 bytes of the new image to be sent to the Amulet must be identical to the first 24 bytes of the original canvas image of the Dynamic Image Widget. The first 24 bytes can be found in the .inc file generated by the GEMstudio, located in a generated MAP folder in the same directory as your specific project.

The first 12 bytes of an Amulet file are the flash header bytes. The next 12 bytes are all image file specific.

byte 1 - 12: Amulet flash header bytes.
byte 13: # of rows, in pixels. (LSByte)
byte 14: # of rows, in pixels. (MSByte)
byte 15: # of columns, in pixels. (LSByte)
byte 16: # of columns, in pixels. (MSByte)
byte 17: image format.
byte 18: valid image key (0xA5)
byte 19-20: transparency flag and bytes per row
byte 21-24: transparency bytes

CRC

The final two bytes of the image file must be a CRC of the entire file, using a seed of 0x00 and a polynomial of 0x1021.
Sample code for the CRC algorithm used on Amulet image files:

```
#define CRC_SEED  0x0000
#define CRC_POLY  0x1021
int calcCRC(char *ptr, int count)
{
  unsigned short crc = CRC_SEED;    // initialize CRC
  int i;
  while (count-- > 0)
  {
     crc = crc ^ *ptr++;
     for (i=8; i>0; i--)
     {
        if (crc & 0x0001)
           crc = (crc >> 1) ^ CRC_POLY;
        else
           crc >>= 1;
     }
  }
  return crc;
 }
```

MK-07C-HP Module

The MK-07C-HP Module's first 20 bytes of the new image to be sent to the Amulet must be identical to the first 20 bytes of the original canvas image of the Dynamic Image Widget. The first 20 bytes can be found in the .inc file generated by the GEMstudio, located in a generated MAP folder in the same directory as your specific project.

The first 8 bytes of an Amulet file are the flash header bytes. The next 12 bytes are all image file specific.

byte 1 - 8: Amulet flash header bytes.
byte 9: # of rows, in pixels. (LSByte)
byte 10: # of rows, in pixels. (MSByte)
byte 11: # of columns, in pixels. (LSByte)
byte 12: # of columns, in pixels. (MSByte)
byte 13: image format.
byte 14: valid image key (0xA5)
byte 15-16: transparency flag and bytes per row
byte 17-20: transparency bytes

Amulet Link Map

After any project is compiled either via Run or Program, GEMstudio generates a link map for the specific project. The name of the link map has the same name as the project name, except it has a .inc extension. The .inc file is created in the the "MAP" folder, which is created in the same directory as the compiled project.
The .inc file is an unformatted text file which can be read by any standard text editor. The .inc file includes the name of each file in the given project. It is setup so a standard C compiler can use the .inc file as an include file, because all comments are bracketed by /* and */ and the definitions are all preceded by #define. This was chosen so the .inc file could be directly included in a C or Java project. If you are using a different programming language, you might have to modify or extract the information that you need to appease your compiler.

Sample .inc file

The following is from an actual .inc file.

```
/*      File Name                File Index                  */
/*      ---------                -----------                  */
#define Page_1                     0x20   /* File Size = 0x19E  */
/*      downarrow.gif              0x21   /* File Size = 0x34E  */
/*      uparrow.gif                0x22   /* File Size = 0x34E  */
/*      downarrowAlt.gif           0x23   /* File Size = 0x34E  */
/*      uparrowAlt.gif             0x24   /* File Size = 0x34E  */
#define Page_2                     0x25   /* File Size = 0x19E  */
```

```
/*      fonthdr.auf                 0xF   /* File Size = 0x28   */
/*      internalram.bin             0x6   /* File Size = 0x214E */
/*      lcdconfigproject.bin        0x3   /* File Size = 0xB6   */
```

Amulet:fileNumber(x)

Following the File Name is the File Index in hexadecimal format. This is the internal file number that the GEMcompiler assigns to each file. This number can be used to hyperlink directly to a given page within the project by using the **Amulet:fileNumber(x)** function, where x is the File Index number. Any Control Object/Widget could use **Amulet:fileNumber(x)** as its href function.

The **Amulet:fileNumber(x)** function works exactly the same as if you entered just a File Name. Using the above .map file, the HREF:

    Page_1.open()

is equal to:

    Amulet:fileNumber(0x20)

Amulet:internal.fileNumber.value()

The **Amulet:internal.fileNumber.value()** function returns the value of the file index number of the given page. Any View Widget that expects a word value can use the **Amulet:internal.fileNumber.value()** function. Using the above .map file again, a View Widget in Page_1 could have an href function that looks like:

Amulet:internal.fileNumber.value()

The above function would return a word value of 0x20

One way to use the **Amulet:internal.fileNumber.value()** function is to have each page within a project have a couple of METAs that looks like:

```
<META HTTP-EQUIV="Refresh"
CONTENT="0,0.01;URL=Amulet:nop();ONVAR=Amulet:internal.fileNumber.value();value=InternalRAM.word(0xFF)">

<META HTTP-EQUIV="Refresh" CONTENT="0,0.02
URL=Amulet:uart0.word(0xFF).setValue(InternalRAM.word(0xFF))">
```

The first META loads InternalRAM word variable #0xFF with the file index number of the given page. Just a note on why this saves the file index number to InternalRAM word variable #0xFF: META's by their nature, will hold data in their own memory space, not in InternalRAM, but this can be changed by using the **value** attribute in the META and specifying a specific Internal RAM variable. In this case, we set the value to InternalRAM.word(0xFF), so instead of saving its value in its own memory space, it now saves its value in Internal RAM word variable # 0xFF.

The second META then sends out a setWord command out uart0, setting external word variable #0xFF to the value of Internal RAM word variable #0xFF, which was just loaded with the file index number of the given page. Notice that the second META is launched 10ms after the first one. This allows for the updating of Internal RAM word variable #0xFF. Using the above .inc file again, if these two METAs were in Page_1.htm, they would be the equivalent to the following META:

<META HTTP-EQUIV="Refresh" CONTENT="0,0.02;URL=Amulet:uart0.word(0xFF).setValue(0x20)">

Image Files

All image files are each saved in thier own file on the Amulet system, and as thus, are included in the .inc file. Do NOT try to use Amulet:fileNumber(x) where x is the File Index to an image file. The Amulet could crash and for serial dataflash based products it could potentially trash the OS in the serial data flash, requiring a reloading of the Amulet OS files.

Any image which is not compressed will also have the first 12-bytes of the File Header stored in the .inc file. The File Header bytes are needed when using the Dynamic Image Widget. Any image used as a Dynamic Image **canvas** will not be compressed. Images can be sent to the Amulet via xmodem crc protocol which will overwrite the **canvas** image of a Dynamic Image Widget. Only images which are the exact same dimensions as the **canvas** image can be sent to the Dynamic Image Widget. The 12-byte File Header for any image sent to the Dynamic Image Widget must be the same as the original image used for the Dynamic Image **canvas**. The first 12 bytes are created by the GEMcompiler and cannot be derived, thus the need for inclusion in the link map. The last 12 bytes are part of the Amulet Bitmap Format and can be derived, but are included in the link map for cross checking purposes.

The File Header bytes are all hexadecimal numbers. The "0x" was left off for the sake of brevity.

Macro Preprocessor

The Amulet macro preprocessor allows you to create macros which are used to make textual substitution throughout the project. The macros are defined in a text file with a .macro extention, which is included in your project by choosing a macro file within the Misc. tab in the Project Settings dialog. Almost all text within the project will be scanned by the Amulet macro preprocessor and text substitutions will be made. The only exception is the Amulet defined parameters, such as the names of widget parameters like Font, Href, etc. All other text, including static text strings, will be scanned by the macro preprocessor and substitutions will occur when an exact match to the macro definition is found. Macro expansion occurs at compile time within the compiler.

The Amulet macro preprocessor is case sensitive. To be considered an exact match, the macro name must be found in the scanned string and must be surrounded by word separators. The following are considered word separators:

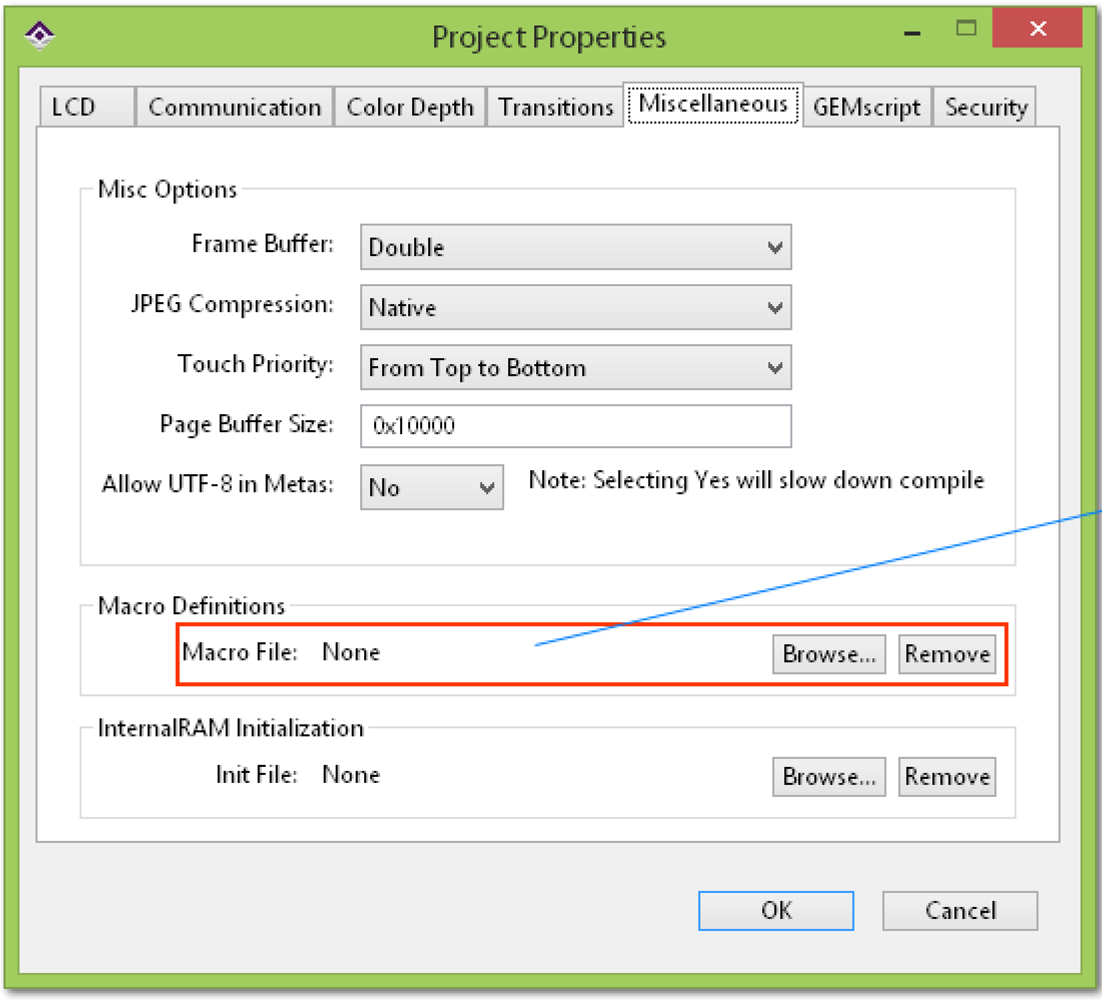
.?!,;:()="
and
start of string, end of string, and spaces.

Note: &nbsp; is NOT considered a space, so if it is desired to use the literal macro name within the project, it can be either preceded or followed by a &nbsp; which ends up looking on the Amulet LCD as a regular space. Another option is to precede or follow the literal variable name with a &#03; which puts in a space holder that the Amulet doesn't display.

Because the preprocessor scans all the text in your project, it is advised to use macro names that will not show up normally. We suggest starting all of your macro names with an uncommon character, like % or #. Another common practice is to bracket the macro name with %, such as %macro%. This will serve two purposes. One, it makes it easy to see your macros when looking at your code. Two, you won't have to worry about having the preprocessor do a text substitution when you didn't really mean it. The Amulet macro preprocessor is very flexible, though, so you are not required to use any special characters, but care should be taken in the naming of macros.

Enabling the Preprocessor:

By default, the preprocessor is not enabled. You can enable the preprocessor by specifying an include file. To do so, open the project settings, choose on the Miscellaneous tab, and select "Browse...". The open file dialog will appear. Navigate to your .macro file and select Open.



Misc Options

Frame Buffer: Double

JPEG Compression: Native

Touch Priority: From Top to Bottom

Page Buffer Size: 0x10000

Allow UTF-8 in Metas: No

Note: Selecting Yes will slow down compile

Macro Definitions

Macro File: None

InternalRAM Initialization

Init File: None

1 Macro Definition file

1 Macro Definition file

Macro File: None

Click Browse... to add or change a file. Click Remove to not use any file.

Defining macros:

All macros must start with **#define**, followed by white space, then the macro name, more white space, then the textual substitution. Neither the macro name nor the textual substitution are allowed to have any white space within it. White space is considered either spaces or tabs. Inside the include file, any text to the right of the comment characters // is treated as a comment. It is okay to have a comment on the same line as the macro definition.
As an example, the syntax is as follows:

#define macroName textSubstitution

Where:

#define specifies the creation of a macro.

macroName is the name of the macro.

textSubstitution is the text which will replace the macro name.

Examples:

```
#define %hour%    Amulet:uart0.byte(0)// Comment goes here.
#define %getHour  %hour%.value()     // expands to Amulet:uart0.byte(0).value()
#define *year     word(3)
#define +counter  7                  // Comment goes here.
```

As you can see from the examples above, it is legal to use macros to define other macros. Forward references are allowed. For example, both of the following are legal:

```
#define  %getCnt  %cnt.value()            // forward reference to %cnt is okay!
#define  %cnt     Amulet:InternalRAM.byte()
```

as well as

```
#define  %cnt     Amulet:InternalRAM.byte()
#define  %getCnt  %cnt.value()
```

Using macros to initialize InternalRAM:

A powerful feature of the preprocessor is that it enables you to use defined macros anywhere in the project, including in the initialization of Internal RAM variables. For example, to initialize an Internal RAM byte variable referenced by the macro **%counter** to a value of 16, you would use the following nomenclature in the Internal RAM initialization file:

InternalRAM.byte(%counter) = 16

You can also use a macro as the value of an InternalRAM variable as well. For example:

InternalRAM.byte(%counter) = %time

Utilization of SD Card

The MK-07C-HP Module includes 4GB of onboard embedded flash memory, but it also includes a Micro SD card slot that can be utilized. The Micro SD card can be used as a data logger by using GEMscript's file I/O functions. The Micro SD card can also be used to program the 4GB of onboard embedded flash.

Data Logging

The SD Card slot can be used to retrieve and store text files. This is particularly useful for data logging to capture the long term performance of a system. The building blocks to design a data logging system are in the GEMscript File I/O API. You can find them under the separate help file, accessed in GEMstudio through Help->GEMscript API

Software Programming

(Available on MK-07C-HP Module only)

The Micro SD card slot can be used to program the onboard embedded flash. This can be used at development time or out in the field for upgrades.

In order to use the Micro SD card as a programmer, the GEMstudio project must first be saved as either a .pdb Production File or .pdb Update File. Production Files include the project files as well as all the OS files required to run the project. Update Files can be saved with or without the OS files, but they do not require a touchpanel calibration. Only .pdb Production/Update Files can be used for Micro SD card programming, .gem files cannot be used for this purpose.

The .pdb Production/Update File needs to be placed in the root directory of the Micro SD card. In order to start the programming sequence, the Micro SD card needs to be placed into the Micro SD card slot and then either the power needs to be cycled or the reset button needs to be pressed. Upon restart, the .pdb file will be read and the onboard embedded flash will be programmed with the new files. Once the programming is finished, the .pdb file will be deleted from the Micro SD card unless the .pdb file's attributes are set to Read Only.

If the .pdb file's attributes are set to Read Only, it will not be deleted after programming. This is handy if it is desired to program multiple modules with a single Micro SD card since it won't be necessary to reload the .pdb file between programming sessions, but is probably not desirable if the Micro SD card will be kept in the MK-07C-HP Module. If the Micro SD card is left in the slot, the Amulet module will go through the programming cycle upon every reset or power cycle.

While the Amulet module is programming via .pdb file, the Graphics Display Card's red LED will toggle after each file is programmed and the boot logo will travel from the upper left hand corner down to the center of the display. When the programming is finished, the Graphics Display Card's red LED will stay off, the green LED will stay on, and the LCD will start displaying the newly programmed project.

Appendix A - UART Code examples

Below you will find implementations of the Amulet ASCII protocol written in C, BASIC, and assembly. Please note that these code snippets are presented here to illustrate the workings of the protocol, not serve as a model implementation.

C Source Code

```
/********************************************************************
*MAIN ROUTINE - initializes RingBuffer and sets the baud to 9600, then
*stays in an infinite loop polling the serial line to see if anything
*has been received, and then handling the byte received.
*NOTE: the serIn() function simply checks the serial line to see if
*anything is there, if not it returns
*********************************************************************/
int main()
{
     rbInit(buffer);
     setbaud(BAUD9600);
     while(1)
     {
          serIn(&buffer);
     }
     return(0);
}

/**************************************************************************
*Checks to see if anything is waiting on the serial line to be handled,
*if so, it puts it at the end of the Ringbuffer, otherwise it does nothing
*and returns
***************************************************************************/
void serIn(RingBuf *buf)
{
     if ((SCSR & RDRF) != 0)
     {
          tail = buf->tail;
          buf->serData[tail++] = SCDR;
          buf->tail = (tail & RB_SIZE_MASK);
          parseSerial();
     }
}

/*************************************************************************
*This function acts as the byte handler to the bytes that serIn() puts
*in the Ringbuffer. Checks to see if valid request type has been
*received and then sets server response value. Using a standard state
*machine, the program proceeds to set variables to hold the values of
*the next byte received on the serial lines and when the correct number
```

```
*of bytes have been received (3 bytes for all request types except
*setByte which needs 5 bytes) later calls functions that put the
*variables back out on the serial line for output
*************************************************************************/
void parseSerial(void)
{
     static char caseType;
     newByte = byteFromBuf(&buffer);
     if((newByte >= 0xD0) && (newByte      {
          serverResp = respMake(newByte);
          caseType = serverResp;
     }
     else if(state == 0)
          caseType = 0x00;
     if((state==0) && ((caseType >= 0xD0) && (caseType      {
          state++;
     }
     else if(state == 1)
     {
          hiNib = newByte;
          state++;
     }
     else if(state == 2)
     {
          loNib = newByte;
          if(caseType == 0xD5)
          {
               state++;
          }
          else
          {
               state = 0;
          }
          switch(caseType)
          {
               case 0xD0:
                    getByte();
                    break;
               case 0xD1:
                    getString();
                    break;
               case 0xD2:
                    getWord();
                    break;
               case 0xD8:
                    invokeRPC();
                    break;
          }
     }
     else if((state == 3) && (caseType == 0xD5))
     {
          setValHi = newByte;
          state++;
      }
     else if(state == 4)
     {
          setValLo = newByte;
          state = 0;
          setByte();
     }
}


/*********************************
*Hex to ascii conversion routine
*********************************/
char hex2ascii(char hex)
{
     return ((hex < 0x0A) ? (hex + '0') : (hex + ('A' - 0x0A)));
}

/*********************************
*Ascii to hex conversion routine
*********************************/
char ascii2hex(char ascii)
{
     return((ascii }

/************************************************************************
*Handler for a getByte function request
* Format of request string is three bytes => 0xD0 vH vL, where v = variable
being requested,
* vH is ASCII version of high nibble of v and vL is ASCII of low nibble of v.
* Returns five bytes => 0xE0 vH vL NH NL,
* where N = value of variable v (low byte),
* NH is ASCII version of high nibble of N and NL is ASCII of low nibble of N.
***************************************************************************/
void getByte(void)
{
     char index, byteValue, valueHiNib, valueLoNib;

     index = ascii2hex(hiNib) << 4;
     index |= ascii2hex(loNib);

     byteValue = byteData[index];

     valueHiNib = hex2ascii(byteValue >> 4);
     valueLoNib = hex2ascii(byteValue & 0x0f);

     putchar(serverResp);
     putchar(hiNib);
     putchar(loNib);
     putchar(valueHiNib);
     putchar(valueLoNib);
}

/**************************************************************************
*Handler for a getString function request
* Format of request string is three bytes => 0xD2 vH vL, where v = index of
string variable,
* vH is ASCII version of high nibble of v and vL is ASCII of low nibble of v.
* Returns variable number of byte: 0xE2 vH vL String+Null to client
*
* This routine is only looking to respond with one of two strings. Therefore, it
* is only looking at the least significant nibble of string variable index.
*
* Uses putchar from ICC C library to put individual characters onto serial line
******************************************************************************/
void getString(void)
{
     putchar(serverResp);
     putchar(hiNib);
     putchar(loNib);

     if(loNib == 0x30)
     {
          putstring(string1);
      }
     if(loNib == 0x31)
     {
          putstring(string2);
     }
}

/**********************************************************************
*Handler for a setByte function request
* Format of request string is five bytes => 0xD5 vH vL NH NL, where v = index of
byte variable,
* vH is ASCII version of high nibble of v and vL is ASCII of low nibble of v,
* N = value of variable v,
* NH is ASCII version of high nibble of N and NL is ASCII of low nibble of N.
* Returns five bytes => 0xE5 vH vL NH NL,
* where P vH vL NH NL are all duplicates of the bytes that were in the request
string.
**********************************************************************/
void setByte(void)
{
     char index, hexVal;

     index = ascii2hex(hiNib) << 4;
     index |= ascii2hex(loNib);

     hexVal = ascii2hex(setValHi) << 4;
     hexVal |= ascii2hex(setValLo);

     byteData[index] = hexVal;

     putchar(serverResp);
     putchar(hiNib);
     putchar(loNib);
     putchar(setValHi);
     putchar(setValLo);
}

/******************************************************************************
*Handler for a getWord function request
* Format of request string is three bytes => 0xD1 vH vL, where v = index of word
variable,
* vH is ASCII version of high nibble of v and vL is ASCII of low nibble of v.
* Returns seven bytes => 0xE1 vH vL PH PL NH NL,
* where P = value of variable v (high byte), N = value of variable v (low byte),
* PH is ASCII version of high nibble of P and PL is ASCII of low nibble of P,
* NH is ASCII version of high nibble of N and NL is ASCII of low nibble of N.
*******************************************************************************/
void getWord(void)
{
     char index, valMSBhinib, valMSBlonib, valLSBhinib, valLSBlonib;
     unsigned int wordValue;

     index = ascii2hex(hiNib) << 4;
     index |= ascii2hex(loNib);

     wordValue = wordData[index];

     valMSBhinib = hex2ascii((char)((wordValue >> 12) & 0x0f));
     valMSBlonib = hex2ascii((char)((wordValue >> 8) & 0x0f));
     valLSBhinib = hex2ascii((char)((wordValue >> 4) & 0x0f));
     valLSBlonib = hex2ascii((char)(wordValue & 0x0f));

     putchar(serverResp);
     putchar(hiNib);
     putchar(loNib);
     putchar(valMSBhinib);
     putchar(valMSBlonib);
     putchar(valLSBhinib);
     putchar(valLSBlonib);
}

/******************************************************************************
*Handler for an invokeRPC function
* Format of request string is three bytes => 0xD8 rH rL, where r = index of RPC,
* rH is ASCII version of high nibble of r and rL is ASCII of low nibble of r.
* Returns three bytes => 0xD8 rH rL,
* where rH rL are duplicates of the bytes that were in the request string.
*******************************************************************************/
void invokeRPC(void)
{
     char rpc, valMSBhinib, valMSBlonib, valLSBhinib, valLSBlonib;
     unsigned int wordValue;

     rpc = ascii2hex(hiNib) << 4;
     rpc |= ascii2hex(loNib);

     performRPC[rpc];     // this code could do any number of things based upon
                          // which Remote Procedure Call is requested.

     putchar(serverResp);
     putchar(hiNib);
     putchar(loNib);
}

/*********************************************
*Function to take a byte out of the buffer
*********************************************/
char byteFromBuf(RingBuf *buf)
{
     char byte;
     head = buf->head;
     byte = buf->serData[head];
     buf->head = (head + 1) & RB_SIZE_MASK;

     return byte;
}

/
*********************************************************************************
*Function to assign a serverResp value based on the byte taken out of the buffer
*********************************************************************************/
char respMake(char byte)
{
     // Response is always 0x10 greater than the command opcode
     return (byte + 0x10);
}

/**********************************************
*Function to put a string onto the serial line
**********************************************/
void putstring(char *str)
{
     char iloop;
     char index = 0;
     char value;

     for(iloop=0; iloop      {
          value = str[index];
          putchar(value);
          index++;
     }
}
```

Basic Source Code

The following BASIC source code was taken from actual server implementation based on a Basic Stamp interfaced to an Analog Devices ADXL202EB accelerometer.

```
'*********************************************************************************
'* The main loop of this code waits for valid Client Start of Message (CSOM)
characters then jumps to the appropriate
'* service routine
'*********************************************************************************
incoming VAR byte(6)  ' Incoming buffer
'*********************************************************************************
' serin Rpin, Baudmode, [STR incoming\L\E]
' serin = receive asynchronous serial data
' Rpin = Rx (instruct BASIC Stamp to use dedicated serial-input pin)
' Baudmode = N9600 (9600 baud, 8-bit data, no-parity, true polarity)
' [STR amuletMsg\L\E] = receive string of length L or until end character E is
received into amuletMsg array
'
' The command serin Rx, N9600, [STR incoming\L\E]
' will poll the serial line looking for serial data up to length L or until end
character E is encountered.
' Incoming data will be stored in the amuletMsg array.
'*********************************************************************************
serial_in:SERIN 16, 84, [RxType, VarType1, VarType2, SetVar1, SetVar2]
```

```
SERIN 16, 84, [STR incoming\6\0]      '* Read a max of 6 bytes or stop when
received a null termination character
RxType = incoming(0)
VarType1 = incoming(1)
VarType2 = incoming(2)

IF RxType = $D5 THEN setByte     'asking for a setByte
IF RxType = $D0 THEN getByte     'asking for a getByte
IF RxType = $D2 THEN getString   'asking for a getString
IF RxType = $D1 THEN getWord     'asking for a getWord
IF RxType > $D5 THEN serial_in   'value on line is not a function, go back to
wait for
                                 'another command
'*******************************************************************************
'* Handler for a getByte function request
'* Format of request string = 0xD0xx, where xx = variable being requested
'* Returns 0xE0xxNN, where NN equals the HEX data of the variable xx
'*******************************************************************************
getByte:
     ServerResp1 = $E0
     IF VarType2 = "5" THEN RateReturn
     IF VarType2 = "6" THEN TimeReturn
     IF VarType2 = "7" THEN VariableReturn
     'Variable 5 is being requested so return value stored for rate (in register
B20)
     RateReturn:
          SEROUT 16,84,[ServerResp1, VarType1, VarType2, HEX2 B20]
          GOTO serial_in

     'Variable 6 is being requested so return value stored for time (in register
B22)
     TimeReturn:
          SEROUT 16,84,[ServerResp1, VarType1, VarType2, HEX2 B22]
          GOTO serial_in
     'Variable 7 is being requested so return value stored for variable (in
register B21)
     VariableReturn:
          SEROUT 16,84,[ServerResp1, VarType1, VarType2, HEX2 B21]
          GOTO serial_in

'******************************************************************************
* Handler for a getString function request
'* Format of request string = 0xD2xx, where xx = index of string variable
'* Returns 0xE2xxString+Null to client
'* Returns requested data back to the screen
'******************************************************************************
getString:
     ServerResp1 = $E2
     IF VarType2 = "2" THEN Author_string
     IF VarType2 = "3" THEN Company_string
     Author_string:
          SEROUT 16,84,[ServerResp1, VarType1, VarType2, "Jacob Horn", NULL]
          GOTO serial_in
     Company_string:
          SEROUT 16,84,[ServerResp1, VarType1, VarType2, "Amulet Technologies",
               NULL]
          GOTO serial_in
```

```
'*****************************************************************
'* Handler for a setByte function request
'* Format of request string = 0xD5xxNN, where xx = variable to be set
'* and NN = HEX data
'* Returns 0xE5xxNN to client
'*****************************************************************
setByte:
     ServerResp1 = $E5

     IF VarType2 = "5" THEN Rate
     IF VarType2 = "6" THEN Time
     'Variable 5 has been called to be set
     Rate:
          'Choose which parameter associated with rate is to be set, using NN
values
          IF incoming(4) = "0" THEN one
          IF incoming(4) = "1" THEN two
          IF incoming(4) = "2" THEN five
          IF incoming(4) = "3" THEN ten
          one:
               B20 = 1
               GOTO setByteResp
          two:
               B20 = 2
               GOTO setByteResp
          five:
               B20 = 5
               GOTO setByteResp
          ten:
               B20 = 10
               GOTO setByteResp
     'Variable 6 has been called to be set
     Time:
          'Once again, choose which parameter is to be set using NN values
          IF incoming(4) = "0" THEN tenSeconds
          IF incoming(4) = "1" THEN thirtySeconds
          IF incoming(4) = "2" THEN fortyfiveSeconds
          IF incoming(4) = "3" THEN sixtySeconds
          tenSeconds:
               B22 = 10
               GOTO setByteResp
          thirtySeconds:
               B22 = 30
               GOTO setByteResp
          fortyfiveSeconds:
               B22 = 45
               GOTO setByteResp
          sixtySeconds:
               B22 = 60
               GOTO setByteResp
     setByteResp:
          SEROUT 16,84,[ServerResp1, VarType1, VarType2, incoming(3),
incoming(4)]
          GOTO serial_in

'********************************************************************************
```

```
'* Handler for a getWord function request
'* Format of request string = 0xD1xx, where xx = variable being requested
'* Returns 0xE1xxPPNN, where PP = high byte of variable xx, and NN = low byte of
variable xx
'****************************************************************************************
getWord:
     ServerResp1 = $E1
     IF VarType2 = "1" THEN YWORDvariable

     'Pulse X port pin, read, and save out x acceleration values
     PULSIN 4,1,xWord
     ServerResp2 = xWord.HIGHBYTE
     ServerResp3 = xWord.LOWBYTE

     'server data sent out over RS232 to client
     SEROUT 16,84,[ServerResp1, VarType1, VarType2, HEX2 ServerResp2,
          HEX2 ServerResp3]
     GOTO serial_in

     YWORDvariable:
          PULSIN 2,1,yWord

          ServerResp2 = yWord.HIGHBYTE
          ServerResp3 = yWord.LOWBYTE
          SEROUT 16,84,[ServerResp1, VarType1, VarType2, HEX2 ServerResp2,
               HEX2 ServerResp3]
          GOTO serial_in
END
```

Assembly Source Code

The following assembly code snippets were taken from an actual server implementation based on Atmel's AVR 8-bit RISC microcontroller (Part# AT90LS4433).

```
;*********************************************************************
;Main loop of program. Waits for valid Client Start Of Message (CSOM)
;characters, then vectors to the appropriate service routine.
;*********************************************************************

.equ GetCommand     = 0xD0
.equ GetResponse   = 0xE0
.equ StringCommand = 0xD2
.equ StringResponse = 0xE2
.equ SetCommand     = 0xD5
.equ SetResponse   = 0xE5
.equ InvokeCommand = 0xD8
.equ InvokeResponse = 0xE8
try_again:
   rcall   getch               ;Go and wait until a character is present in the
buffer
   cpi     buffer,GetCommand   ;Is it a get command?
   brne    is_it_S             ;Not get command, so check others
   rjmp    handleG
is_it_String:
   cpi     buffer,StringCommand;Is it a string command?
   brne   is_it_S             ;Not string command, so check for set
   rjmp   handleString
is_it_S:
   cpi     buffer,SetCommand   ;Is it a set command?
   brne   is_it_I             ;Not set command, so check for I
   rjmp   handleS
is_it_I:
   cpi buffer,InvokeCommand   ;Is it an invoke command?
   brne   try_again           ;Not a valid request so start over and wait for
next character
   rjmp   handleI

;*******************************************************************
;Handle Get Byte command to get variable data
; Format of request string = 0xD0xx
; where xx = variable requested
; Returns 0xE0xxNN to client where NN = HEX data of variable (xx)
;*******************************************************************
handleG:
    rcall   getByte             ;Read both nibbles of xx and assemble into a
byte. Return in buffer.
   brcc   try_again           ;If carry cleared, then invalid value, so start
over
   mov     which,buffer
   ldi     buffer,GetResponse
   rcall   putch               ;Acknowledge valid command
   mov     buffer,which
   swap   buffer             ;Rotate MSNibble into LSNibble position
   rcall   nib2ascii           ;Convert LSNibble of buffer to an ASCII character
   rcall   putch               ;Echo back MSNibble of variable (xx)
   mov     buffer,which
   rcall   nib2ascii           ;Convert LSNibble of buffer to an ASCII character
   rcall   putch               ;Echo back LSNibble of variable (xx)
   rcall   get_varH           ;Go get data (msn) for variable (xx)
   rcall   putch               ;Transfer data to the client
   rcall   get_varL           ;Go get data (lsn) for variable (xx)
   rcall   putch               ;Transfer data to the client
   rjmp   try_again           ;Done with Get command so start over

;Handle String command to tx a string back.
; Format of request string = 0xD2xx
; where xx = index of string variable
; Returns 0xE2xxString+Null to host.
;*******************************************************
handleString:
   rcall   getByte             ;Read both nibbles of xx and assemble into a
byte. Return in buffer.
   brcc   try_again           ;If carry cleared, then invalid value and start
over
   mov     which,buffer
   ldi     buffer,StringResponse
   rcall   putch               ;Acknowledge valid command
   mov     buffer,which
   swap   buffer             ;Rotate MSNibble into LSNibble position
   rcall   nib2ascii           ;Convert LSNibble of buffer to an ASCII character
   rcall   putch               ;Echo back MSNibble of variable (xx)
   mov     buffer,which
   rcall   nib2ascii           ;Convert LSNibble of buffer to an ASCII character
   rcall   putch               ;Echo back LSNibble of variable (xx)

   ...                         ;To simplify this snippet, all the code which
looks up the string(xx)
   ...                         ;in a look-up table was left out

   icall                       ;Time to go pound the null terminated string out
   rjmp   try_again           ;Done with string command so start over

;****************************************
;Handle S command to Set variable data
; Format of request string = 0xD5xxNN
; where xx = variable to set
; NN = HEX data for variable (xx)
; Returns 0xE5xxNN to client
;****************************************
handleS:
   rcall   getByte             ;Read both nibbles of xx and assemble into a
byte. Return in buffer.
   brcc   try_again           ;If carry cleared, then invalid value, so start
over
   mov     which,buffer
   rcall   getByte             ;Read both nibbles of NN and assemble into a
byte. Return in buffer.
   brcc   try_again           ;If carry cleared, then invalid value, so start
over
   mov     what,buffer

   ;This is where we would use the value of xx, now stored in 'which', to
determine
   ;where to store the value of NN, now stored in 'what'. For the sake of
brevity,
   ;this example only handles xx=00, while all other xx are ignored.

   cpi     which,0             ;If offset is zero, then VAR0 is the variable to
set
   brne   try_again
   sts     VAR0,what           ;Set VAR0
   ldi     buffer,SetResponse
   rcall   putch               ;Acknowledge valid command
   mov     buffer,which
   swap   buffer             ;Rotate MSNibble into LSNibble position
   rcall   nib2ascii           ;Convert LSNibble of buffer to an ASCII character
   rcall   putch               ;Echo back MSNibble of variable (xx)
   mov     buffer,which
   rcall   nib2ascii           ;Convert LSNibble of buffer to an ASCII character
   rcall   putch               ;Echo back LSNibble of variable (xx)
   mov     buffer,what
   swap   buffer             ;Rotate MSNibble into LSNibble position
   rcall   nib2ascii           ;Convert LSNibble of buffer to an ASCII character
   rcall   putch               ;Echo back MSNibble of variable (NN)
   mov     buffer,what
   rcall   nib2ascii           ;Convert LSNibble of buffer to an ASCII character
   rcall   putch               ;Echo back LSNibble of variable (NN)
   rjmp   try_again           ;Done with Set command so start over
```

```
;*************************************
;Handle I command to Invoke a function
; Format of request string = 0xD8xx
; where xx = function to invoke
; Returns 0xE8xx to client
;*************************************
handleI:
   rcall   getByte              ;Read both nibbles of xx and assemble into a
byte. Return in buffer.
   brcc   try_again            ;If carry cleared, then invalid value, so start
over
   mov     which,buffer

   ...                          ;To simplify this snippet, all the code which
looks up the function(xx)
   ...                          ;in a look-up table was left out

   icall                        ;Time to go execute the function (xx)
   ldi     buffer,InvokeResponse
   rcall   putch                ;Acknowledge valid command
   mov     buffer,which
   swap   buffer               ;Rotate MSNibble into LSNibble position
   rcall   nib2ascii            ;Convert LSNibble of buffer to an ASCII character
   rcall   putch                ;Echo back MSNibble of variable (xx)
   mov     buffer,which
   rcall   nib2ascii            ;Convert LSNibble of buffer to an ASCII character
   rcall   putch                ;Echo back LSNibble of variable (xx)
   rjmp   try_again            ;Done with Invoke command so start over
```

Appendix B - All Commands

The commands that can be used in a given widget's Href depend on what kind of widget it is.

Control Widget Href Functions

Amulet:back()	Returns to calling page.
Amulet:calendar.timePeriod.setValue(x)	Returns the year, month, day of the month, day of the week, hour, military hour, am or pm, minute, second. Where timePeriod is either year, month, day of the month, day of the week, hour, militaryHour, am_pm, minute, second and X is an integer with the following limits: • year: min=0000 max=65535 • month: min=1 (January) max=12 (December) • dayOfMonth: min=1 max=31 • dayOfWeek: min=1 (Sunday) max=7 (Saturday) • hour: min=0 max=23 • militaryHour: min=0 max=23 • am_pm: min=0 (am) max=1 (pm) • minute: min=0 max=59 • second: min=0 max=59

Amulet:calendar.timePeriod.value()	All time periods return an integer, with the same limits as timePeriod.setValue(x) EXCEPT for the hour, the hour will return min=1, max=12 and you must read am_pm to determine the actual time OR you can just read military hour and the min=0, max=23.
Amulet:calibrate()	Runs touchscreen calibration, returns to calling page when calibration completed. Calibration constants are then saved to serial data flash. There is a 100,000 max write limit on the life of the serial data flash. After 10,000 writes, the Amulet OS and the user's project should be reprogrammed to maintain data integrity.
Amulet:clearLCD()	Clears the entire LCD.
Amulet:disableDraw()	Prevents the display buffer from being updated with new data until enableDraw() is called.
Amulet:enableDraw()	Resumes updates to the display buffer and forces an immediate refresh if there is anything to draw.
Amulet:document.widgetName.method(x)	Performs the named method of the widget specified by widgetName. See Control Widget IWC Methods for a full list of available methods.
Amulet:fileNumber(x)	Jump to the file whose internal fle index is x. See Amulet link map file documentation for more information.
Amulet:gpio(x).clear()	Clears (voltage = 0) the Amulet-specific General Purpose I/O pin, specified by x.
Amulet:gpio(x).set()	Sets (voltage = Vdd) the Amulet-specific General Purpose I/O pin, specified by x.

Amulet:internal.disableMSD()	Disables the USB Mass Storage Device. The Amulet embedded flash will still work, it is no longer accessible via USB MSD, though. (MK-07C-HP Module only)
Amulet:internal.enableMSD()	Enables the USB Mass Storage Device. Counters the Amulet:internal.disableMSD() function.(MK-07C-HP Module only)
Amulet:internal.setCalPOC()	Sets the software calibration bit in the flash memory. Causes Amulet to go in calibration mode upon a soft or hard reset. Bit is cleared after succesful calibration.
Amulet:internal.setMSDDrive(x)	If the USB Mass Storage Device is enabled, then this specifies which flash appears as a MSD. 0 for the embedded flash, 1 for the Micro SD card. By default, the embedded flash(0) starts out enabled and is the active MSD Drive.(MK-07C-HP Module only)
Amulet:internal.softReset()	Forces an immediate software controlled reset of the Amulet processor. The equivalent of pressing the module's reset button.
Amulet:internalRAM.byte(z).method(x)[1]	Performs the named Internal RAM byte method. See Internal RAM documentation for list of available methods.
Amulet:internalRAM.color(z).method(x)[1]	Performs the named Internal RAM color method. See Internal RAM documentation for list of available methods.
Amulet:internalRAM.clearRPCBuf()	Clears the internalRAM RPC buffer.
Amulet:internalRAM.invokeRPC(x)[1]	Adds the RPC number, x, to the internalRAM RPC buffer.
Amulet:internalRAM.saveToFlash()	Saves the current state of all the Internal RAM variables (byte, word and string) in the serial data flash. There is a 100,000 max write limit on the life of the serial data flash. After 10,000 writes, the Amulet OS and the user's project should be reprogrammed to maintain data integrity.
Amulet:internalRAM.string(z).method(x)	Performs the named Internal RAM string method. See Internal RAM documentation for list of available methods.
Amulet:internalRAM.word(z).method(x)	Performs the named Internal RAM word method. See Internal RAM documentation for list of available methods.
Amulet:lcdBacklight.saveToFlash()	Saves the current value of the lcd backlight intensity in the embedded flash. See Backlight Control for more information.
Amulet:lcdBacklight.setValue(x)	Sets the value of the lcd backlight intensity to the value x. See Backlight Control for more information.
Amulet:lcdController.off()	Turns the LCD controller off. The display will go black, but everything on the page is still active as far as the touch panel is concerned.

Amulet:lcdController.on()	Turns the LCD controller back on if the LCD controller is off.
Amulet:lcdController.rotate(x)	Instantly rotates the LCD relative to the native LCD orientation. x can be 0, 90, 180, or 270. For best results, rotations should be on the same plane as the original project rotation setting. (i.e. If project was created with a rotation of 0, Amulet:lcdController.rotate(180) can be used, but not 90 or 270)
Amulet:loadFlash(back)	Used internally to halt all current activity and await flash programming commands from the Amulet GEM Compiler. Performs Amulet:back() upon exit.
Amulet:loadFlash(reset)	Used internally to halt all current activity and await flash programming commands from the Amulet GEM Compiler. Performs software reset upon exit.
Amulet:loadFlash(return)	Used to halt all current activity and await flash programming commands. Stays in same gem page upon exit.
Amulet:lowPower.sdram(mode)	The sdram can be put into one of the three modes below: • HIGH_PERFORMANCE (highest performance, full power usage) • LOW_POWER (slightly slower performace, about 15 mA savings) • LOWEST_POWER (slower performance, about 20 mA savings) (savings based MK-043R module)
Amulet:lowPower.sleep(mode)	The low power sleep mode puts the Amulet into a mode where page objects are no longer being updated and the OS stays in a tight loop updating the calendar object and looking for a touch wake up or a wake up on the negative edge on uart0 Rx line. Where mode can be one of the three modes below (savings based on 4.3" MK-043R module): • LCD_OFF/SLOW_CLK (savings of about 148mA) • LCD_OFF/FAST_CLK (savings of about 108mA) • LCD_ON/FAST_CLK (savings of about 20mA) NOTE: For MK-070C-HP, FAST_CLK is 536MHz and it does not have a SLOW_CLK mode. For all other products, FAST_CLK is the 80 MHz clock and SLOW_CLK is the 32KHz clock.
Amulet:nop()	Non-operational function; does nothing. May be useful in creating states for HREF state machines.

Amulet:screenDump()	Causes Amulet to jump to the screenDump page embedded in the OS files. ScreenDump page uploads the image on the LCD in the Amulet Bitmap Format using the xmodem protocol.
Amulet:SPI(z).byteOut(x)[1]	Sends a raw byte x out the SPI bus while taking Amulet SPI slave select line z low. Put x in single quotes to be treated as an ASCII character. (i.e. '9' is equal to 0x39)
Amulet:SPI(z).streamOut(x1+x2+...xn)	Sends a stream of raw bytes x1, x2 ... xn out the SPI bus while taking Amulet SPI slave select line z low. If the spixCSAAT=1 in the SPI Configuration settings, then the slave select will stay low between each byte, if spixCSAAT=0, then the slave select will go high between each byte.
Amulet:SPI(z).stringOut(x)[1]	Sends an ASCII string x, not null terminated, out the SPI bus while taking Amulet SPI slave select line z low. If the spixCSAAT=1 in the SPI Configuration settings, then the slave select will stay low between each byte, if spixCSAAT=0, then the slave select will go high between each byte. Where x is a single-quoted string or x can also be an InternalRAM.string variable.
Amulet:*port*.byteOut(x)[1]	Sends out a raw byte x out the communication port. Put x in single quotes to be treated as an ASCII character (i.e. '9' is equal to 0x39). The *port* selection can be USB, uart0, uart1, or uart2
Amulet:*port*.invokeRPC(x)[1]	Sends out the invokeRPC command over the communication port, with x being the RPC number and the *port* selection can be USB, uart0, uart1, or uart2.
Amulet:*port*.byte(z).setValue(x)[1]	Sends out the setByte command over the communication port, where z is the byte variable number, x is the value to set it to and the *port* selection can be USB, uart0, uart1, or uart2.
Amulet:*port*.byte(z).setValue(x)[1]	Sends out the setColor command over the communication port, where z is the color variable number, x is the color to set it to and the *port* selection can be USB, uart0, uart1, or uart2.
Amulet:*port*.setBaudRate(x)	Sets the baud rate of the port to x, where x can be any baud rate from 9600 to 115200. The *port* selection can be uart0, uart1, or uart2.
Amulet:*port*.streamOut(x1+x2+...xn)	Sends a stream of raw bytes x1, x2 ... xn out the communication port. No response is required or desired. The *port* selection can be USB, uart0, uart1, or uart2.
Amulet:*port*.stringOut(x)[1]	Sends an ASCII string x, not null terminated, out the communication port. No response is required or desired. Where x is a single-quoted string or x can also be an InternalRAM.string variable and the *port* selection can be USB, uart0, uart1, or uart2.

Amulet:*port*.string(z).setValue(x)[1]	Sends out the setString command over the communication port, where z is the string variable number, x is the string to set it to and the *port* selection can be USB, uart0, uart1, or uart2.
Amulet:*port*.word(z).setValue(x)[1]	Sends out the setWord command over the communication port, where z is the word variable number, and x is the value to set it to and the *port* selection can be USB, uart0, uart1, or uart2.
filename.open()	Jumps to the named page.

1. Regarding x: For Control Widgets that have intrinsic values, such as lists and sliders, use (intrinsicValue), since the intrinsic value of the selection will be sent out. META REFRESH tags and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and 250-character strings in double quotes for STRINGs.

Control Widget IWC Methods

Control Widget href functions for Inter-Widget Communications (IWC).

Amulet:document.widgetName.addCanvasX(x)	The named StringField widget will move its canvas by x pixels toward the right and redraw itself.
Amulet:document.widgetName.addCanvasY(x)	The named StringField widget will move its canvas by x pixels toward the bottom and redraw itself.
Amulet:document.widgetName.buttonDown()	The named Function/Custom Button Widget will appear as if it is currently being touched.
Amulet:document.widgetName.buttonUp()	The named Function/Custom Button Widget will appear as if it is currently NOT being touched.
Amulet:document.widgetName.clearCanvas()	The named Scribble/Dynamic Image Widget clears its canvas, including any background images.
Amulet:document.widgetName.disappear()	The named widget will clear itself from the LCD; if it is a View Widget it will also stop updating.
Amulet:document.widgetName.forceHit()	The named Control Widget will act as if it was "hit".
Amulet:document.widgetName.forceRefresh()	The named View Image Sequence Widget will paint the image at the next update, even if the incoming value is the same as the current state. Useful if an anchor is used around an Image Sequence Widget.
Amulet:document.widgetName.forceUpdate()	The named View Widget will act as if it's update rate time was activated. Allows for asynchronous updating.
Amulet:document.widgetName.inverseRegionColor()	The named widget will display in reverse video.

Amulet:document.widgetName.inverseStringColor()	The named widget's text string will display in reverse video.
Amulet:document.widgetName.nextEntry()	The named List widget will move its highlighted bar down to the next entry. Does not perform a "hit" on the new entry.
Amulet:document.widgetName.normalRegionColor()	The named widget will display in normal video.
Amulet:document.widgetName.normalStringColor()	The named widget's text string will display in normal video.
Amulet:document.widgetName.previousEntry()	The named List widget will move its highlighted bar up to the previous entry. Does not perform a "hit" on the new entry.
Amulet:document.widgetName.reappear()	The named widget will reappear on the LCD in its last location; if it is a View Widget it will also start updating.
Amulet:document.widgetName.refresh()	The named widget, if not currently in a disappeared state, will redraw on the LCD in its last location.
Amulet:document.widgetName.reset()	The named widget will initialize internal variables and redraw.
Amulet:document.widgetName.setCanvasX(x)	The named StringField widget will move its canvas to the X coordinate specified by x and redraw itself.
Amulet:document.widgetName.setCanvasY(x)	he named StringField widget will move its canvas to the Y coordinate specified by x and redraw itself.
Amulet:document.widgetName.setMethod(m)^2	The href method for the named widget will change to m, where m is the method name. (such as value() or disappear())
Amulet:document.widgetName.setOnVarMethod(m)^2	The IF= method for the named widget will change to m, where m is the method name. (such as value() or disappear())
Amulet:document.widgetName.setOnVarUARTMethod(m)^2	The ONVAR UART method for the named widget will change to m, where m is the method name. (such as Value())

Amulet:document.widgetName.setOnVarVariableNumber(x)[1]	The variable number used in the ONVAR of the named widget will change to x, where x is the variable index used in the following variable types: byte(x), word(x) or string(x).
Amulet:document.widgetName.setTrigger(x)[1]	The named Widget will change its equal, gt or lt value to the byte value x.
Amulet:document.widgetName.setUARTMethod(m)[2]	The href UART method for the named widget will change to m, where m is the UART:method name. (such as Value())
Amulet:document.widgetName.setUpdateRate(f)[3]	The update rate for the named widget will change to f, where f is a floating point number that represents the update rate in seconds.
Amulet:document.widgetName.setValue(x)[1]	The named widget will receive the intrinsic value of the calling widget, where x is the intrinsic value.
Amulet:document.widgetName.setValue(gettext("String"))	The named widget will receive the static string specified between the double quotes, translated into the [currently selected language](). If no exact translation is found, it will use the original string.
Amulet:document.widgetName.setValue(pgettext("context","String"))	The named widget will receive the static string specified in the second argument, translated into the [currently selected language]() using the context of the first argument. If no exact translation is found with the specified context, it will use the original string from the second arguent.
Amulet:document.widgetName.setVariableNumber(x)[1]	The variable number used in the href of the named widget will change to x, where x is the variable index used in the following variable types: byte(x), word(x) or string(x).
Amulet:document.widgetName.setX[4](x)	The named Widget will change its topleft x coordinate to the word value x.

Amulet:document.widgetName.setY^4(x)	The named Widget will change its topleft y coordinate to the word value x.
Amulet:document.widgetName.startUpdating()	The named View Widget will start updating the displayed data.
Amulet:document.widgetName.stopUpdating()	The named View Widget will stop updating the displayed data.
Amulet:document.widgetName.subCanvasX(x)	The named StringField widget will move its canvas by x pixels toward the left and redraw itself.
Amulet:document.widgetName.subCanvasY(x)	The named StringField widget will move its canvas by x pixels toward the top and redraw itself.
Amulet:document.widgetName.toggleRegionColor()	The named widget will either start or stop displaying in reverse video.
Amulet:document.widgetName.toggleStringColor()	The named widget's text string will either start or stop displaying in reverse video.
Amulet:document.widgetName.toggleUpdating()	The named View Widget will either start or stop updating the displayed data.

1. Regarding x: For Control Widgets that have intrinsic values, such as lists and sliders, the user can either leave the argument field empty or use "intrinsicValue" as the argument, the intrinsic value of the selection will be sent out. META REFRESH tags and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and 250-character strings in double quotes for STRINGs.

2. Regarding m - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.

3. Regarding f: For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. META REFRESH tags and Function/Custom Buttons should use f. Like the regular updateRate, use a floating point number to specify the update rate in seconds. Range for f is 0-655.35

4. Regarding setX and setY: These methods should most always be preceded by a disappear() method and followed by a reappear() method. The setting of the x and y coordinates are independent of the removal of the widget in the old coordinates and the displaying in the new coordinates.

View Widget Href Functions

Some functions are only valid for very specific widgets or use cases. A function that returns an array, for example, could not be used in a widget that only displays a single value at a time. See Widget-specific Href Functions for more

Amulet:document.widgetName.value()	Returns the intrinsic value (byte or word) of the named Control Widget.

Amulet:document.widgetName.maskedValue(y)	Returns the intrinsic value (byte or word) of the named Control Widget ANDed with the mask y.
Amulet:document.StringFieldName.maxTextWidth()	Returns the maximum width in pixels of the displayed text, factoring in line wraps
Amulet:document.StringFieldName.maxTextHeight()	Returns the the font height in pixels multiplied by the count of rows of text drawn.
Amulet:internal.fileNumber.value()	Returns the internal flash index number of the current page. See Amulet link map file documentation for more information.
Amulet:internal.timeouts(*port*).value()	Returns the count of consecutive communication time-outs for the port specified by *port* (uart0, uart1, uart2 or usb).
Amulet:internalRAM.byte(x).value()	Returns the value of internalRAM byte variable x.
Amulet:internalRAM.byte(x).maskedValue(y)	Returns the value of internalRAM byte variable x ANDed with the mask y.
Amulet:internalRAM.word(x).value()	Returns the value of internalRAM word variable x.
Amulet:internalRAM.word(x).maskedValue(y)	Returns the value of internalRAM word variable x ANDed with the mask y.
Amulet:math.randomByte.value()	Returns a pseudo-random byte.
Amulet:math.randomByte.maskedValue(y)	Returns a pseudo-random byte ANDed with the mask y.
Amulet:math.randomFilteredByte.value()	Returns a pseudo-random byte that is filtered.
Amulet:math.randomFilteredByte.maskedValue(y)	Returns a pseudo-random byte that is filtered ANDed with the mask y.
Amulet:NOP()	Returns nothing.
Amulet:*port*.byte(x).value()	Sends a getByte request over the communications port, where x is the byte variable number and the *port* selection can be USB, uart0, uart1, or uart2; returns the value of the byte variable x.
Amulet:*port*.byte(x).maskedValue(y)	Sends a getByte request over the communications port, where x is the byte variable number and the *port* selection can be USB, uart0, uart1, or uart2; returns the value of the byte variable x ANDed with the mask y.
Amulet:*port*.bytes(x).array(y)	Sends a getByteArray request over the communications port, where x is the starting byte variable number, y is the number of bytes in the arrayand the *port* selection can be USB, uart0, uart1, or uart2; returns the array of byte variables from x to (x+y). (Only valid when used in a META

	Refresh tag to load InternalRAM variables or in LineGraph Href.)
Amulet:*port*.word(x).value()	Sends a getWord request over the communications port, where x is the word variable number and the *port* selection can be USB, uart0, uart1, or uart2; returns the value of the word variable x.
Amulet:*port*.word(x).maskedValue(y)	Sends a getWord request over the communications port, where x is the word variable number and the *port* selection can be USB, uart0, uart1, or uart2; returns the value of the word variable x ANDed with the mask y.
Amulet:*port*.words(x).array(y)	Sends a getWordArray request over the communications port, where x is the starting word variable number, y is the number of words in the array and the *port* selection can be USB, uart0, uart1, or uart2; returns the array of word variables from x to (x+y). (Only valid when used in a META Refresh tag to load InternalRAM variables or in LineGraph Href.)

Widget-specific Href Functions

Some widgets have a very limited set of unique Href functions they are allowed to use. This section describes those special cases.

Line Graph Widget Href Functions

Amulet:internalRAM.bytes(x).array(y)	Returns the array of internalRAM bytes starting at variable x through variable (x+y).
Amulet:internalRAM.words(x).array(y)	Returns the array of internalRAM words starting at variable x through variable (x+y).
Amulet:*port*.bytes(x).array(y)	Sends a getByteArray request over the communications port, where x is the starting byte variable number, y is the number of bytes in the array and the *port* selection can be USB, uart0, uart1, or uart2. returns the array of byte variables from x to (x +y).
Amulet:*port*.words(x).array(y)	Sends a getWordArray request over the communications port, where x is the starting word variable number, y is the number of words in the array and the *port* selection can be USB, uart0, uart1, or uart2. returns the array of word variables from x to (x +y).

Scribble Widget Href Functions

This is the only href a scribble widget is allowed to have

Amulet:*port*.xmodemUploadImage()	The Scribble Widget will upload raw image data out the specified *port* using an xmodem protocol upon receiving the uploadImage() IWC method. *port* selection can be USB, uart0, uart1, or uart2.

StringField Widget Href Functions

Amulet:document.widgetName.value()	Returns the intrinsic value (byte, word or string) of the named Control Widget.
Amulet:internal.timeouts(*port*).value()	Receives the count of consecutive communication time-outs for the port specified by *port* (uart0, uart1, uart2 or usb). (Only valid if option field populated.)
Amulet:internal.OSVersionString.value()	Returns the string of the current Amulet OS version number.
Amulet:internalRAM.byte(x).value()	Returns the value of internalRAM byte variable x.
Amulet:internalRAM.byte(x).maskedValue(y)	Returns the value of internalRAM byte variable x ANDed with the mask y.
Amulet:internalRAM.string(x).value()	Returns the string of internalRAM string variable x.
Amulet:math.randomByte.value()	Receives a pseudo-random byte.
Amulet:math.randomByte.maskedValue(y)	Receives a pseudo-random byte ANDed with the mask y.
Amulet:uartn.byte(x).value()	Sends a getByte request over the UART, where x is the byte variable number and *n* is the uart port number; returns the value of the byte variable x. (Only valid if option field populated.)
Amulet:uartn.byte(x).maskedValue(y)	Sends a getByte request over the UART, where x is the byte variable number and *n* is the uart port number; returns the value of the byte variable x ANDed with the mask y. (Only valid if option field populated.)
Amulet:uartn.string(x).value()	Sends a getString request over the UART, where x is the string variable number and *n* is the uart port number; returns a null terminated ASCII string.

Appendix C - Inter-Widget Commands

Inter Widget Communication allows one Amulet widget to invoke the methods of another Amulet widget. (See Appendix B for a comprehensive listing of all available function calls.) In the topics below you will find a description of the valid IWC methods for each widget, in addition to a brief description of how those methods specifically act.

Animated Image

Methods

- **disappear()** - Makes the image not visible on the LCD.
- **fastSpeed()** - Increases animation speed.
- **oneFrame()** - Advances animation one frame.
- **pause()** - Stops animation.
- **play()** - Starts animation in current direction.
- **playBackwards()** - Starts animating backwards.
- **playForward()** - Starts animating forward.

- **reappear()** - Makes the image visible on the LCD. Counteracts the disappear() method.
- **refresh()** - Redraws the image on the LCD. Will not redraw if currently in a disappeared state.
- **regularSpeed()** - Normal animation speed.
- **setX(x)** - Sets the x-coordinate of the topleft corner of the animated image to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the animated image to the coordinate specified by the word x.
- **slowSpeed() -** Decreases animation speed.
- **superFastSpeed()** - Fastest animation speed.
- **superSlowSpeed()** - Slowest animation speed.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
- **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

BarGraph

Methods

- **disappear()** - Makes the BarGraph not visible on the LCD.
- **forceUpdate()** - Forces the BarGraph to call its href function immediately, regardless of the updateRate.
- **reappear()** - Makes the BarGraph visible on the LCD. Counteracts the disappear() method.
- **refresh()** - Redraws the BarGraph on the LCD. Will not redraw if currently in a disappeared state.
- **setValue(x)** - BarGraph uses x as its input. This allows a Control Widget to provide the input to a BarGraph.
- **setMethod(m)** - Changes the method originally specified by the BarGraph's href parameter.
- **setUARTMethod(m)** - Changes the UART method originally specified by the BarGraph's href parameter.
- **setUpdateRate(f)** - Changes the update rate originally specified by the BarGraph, the argument being a floating point number, specifying the time in seconds.
- **setVariableNumber(x)** - Changes the variable number originally specified by a BarGraph. Can only be used if the BarGraph href is byte(y).value() or word(y).value(). In either case, the y gets changed to the argument specified in setVariableNumber(x).
- **setX(x)** - Sets the x-coordinate of the topleft corner of the BarGraph to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the BarGraph to the coordinate specified by the word x.
- **startUpdating()** - BarGraph starts updating based upon its input data. Counteracts the stopUpdating() method.
- **stopUpdating()** - BarGraph stops updating.
- **toggleUpdating()** - Changes current state of BarGraph; either starts or stops updating.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and

Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
• **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

CheckBox

Methods

• **disappear()** - Makes the CheckBox not visible or touchable on the LCD.
• **forceHit()** - CheckBox performs its "hit" method without user input. The "hit" method will invoke all href functions of the CheckBox.. When imparting a forceHit on a single CheckBox, that CheckBox will toggle. You can also forceHit an entire CheckBox group which will perform the href function(s), but will not toggle any checkboxes. To forceHit a CheckBox group, use the groupName as the widgetName (rather than the individual CheckBox name).
• **forceUpdate()** - Forces the CheckBox group to call its initHref function immediately. Only valid if initialCondition is FromInitHref. Updates the CheckBox group and performs a "hit". To forceUpdate a CheckBox group, use the groupName as the widgetName (rather than the individual CheckBox name).
• **maskedValue(y)** - Sends the intrinsic value of the CheckBox to the calling widget.. The calling object then uses that value, ANDed with the mask y, as its input. This method is only valid if the intrinsic value is a byte or word. The mask y can be either a byte or word. This allows a CheckBox to provide the input to a View Widget. This method is called from a View Widget href.
• **reappear()** - Makes the CheckBox visible and touchable on the LCD. Counteracts the disappear() method.
• **refresh()** - Redraws the CheckBox on the LCD. Will not redraw if currently in a disappeared state.
• **setValue(x)** - The CheckBox intrinsic value is changed to x.
• **setMethod(m)** - Changes the method originally specified by the CheckBox's href parameter; only valid when the originally specified method is a single function. To change the method for a CheckBox group, or an ungrouped CheckBox, use the groupName as the widgetName (rather than in individual CheckBox name). You cannot change the method for an individual CheckBox within a group.
• **setUARTMethod(m)** - Changes the UART method originally specified by the CheckBox's href parameter; only valid when the originally specified method is a single function. To change the method for a CheckBox group, or an ungrouped CheckBox, use the groupName as the widgetName (rather than in individual CheckBox name). You cannot change the method for an individual CheckBox within a group.
• **setVariableNumber(x)** - Changes the variable number originally specified in the href function to x. Can only be used only if the Check Box href uses byte(y), word(y) or string(y). In all three of the cases, the value y is replaced with the value x. Only valid when the originally specified method is a single function. Can only be used on a Check Box Group, not an individual CheckBox.
• **setX(x)** - Sets the x-coordinate of the topleft corner of the individual CheckBox to the coordinate specified by the word x.
• **setY(x)** - Sets the y-coordinate of the topleft corner of the individual CheckBox to the coordinate specified by the word x.
• **value()** - Sends the intrinsic value of the CheckBox to the calling widget.. The calling object then uses that value as its input. This allows a CheckBox to provide the input to a View Widget. This method is called from a View Widget href.

Regarding Method Parameters

• **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.

• **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Custom Button

Methods

• **buttonDown()** - Sets the Custom Button Widget to look like its down state. This method does NOT invoke the href functions. It only affects the looks of the button, not the functionality.
• **buttonUp()** - Sets the Custom Button Widget to look like its up state. This method does NOT invoke the href functions. It only affects the looks of the button, not the functionality.
• **disappear()** - Makes the Custom Button not visible or touchable on the LCD.
• **forceHit()** - Custom Button performs its "hit" method without user input. The "hit" method will invoke all href functions of the Custom Button.
• **forceUpdate()** - Forces the Custom Button to call its initHref function immediately. Only valid if label is fromInitHref. Updates the Custom Button label without performing a "hit".
• **maskedValue(y)** - Sends the intrinsic value of the Custom Button to the calling widget. The calling object then uses that value, ANDed with the mask y, as its input. This method is only valid if the intrinsic value is a byte or word. The mask y can be either a byte or word. This allows a Custom Button to provide the input to a View Widget. This method is called from a View Widget href.
• **reappear()** - Makes the Custom Button visible and touchable on the LCD. Counteracts the disappear() method.
• **refresh()** - Redraws the Custom Button on the LCD. Will not redraw if currently in a disappeared state.
• **setMethod(m)** - Changes the method originally specified by the Custom Button's href parameter; only valid when the originally specified method is a single function.
• **setUARTMethod(m)** - Changes the UART method originally specified by the Custom Button's href parameter; only valid when the originally specified method is a single function.
• **setValue(x)** - The Custom Button intrinsic value is changed to x.
• **setVariableNumber(x)** - Changes the variable number originally specified in the href function to x. Can only be used only if the Custom Button href uses byte(y), word(y) or string(y). In all three of the cases, the value y is replaced with the value x. Only valid when the originally specified method is a single function.
• **setX(x)** - Sets the x-coordinate of the topleft corner of the individual Custom Button to the coordinate specified by the word x.
• **setY(x)** - Sets the y-coordinate of the topleft corner of the individual Custom Button to the coordinate specified by the word x.
• **value()** - Sends the intrinsic value of the Custom Button to the calling widget. The calling object then uses that value as its input. This allows a Custom Button to provide the input to a View Widget. This method is called from a View Widget href.

Regarding Method Parameters

• **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
• **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and

Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Custom Slider

Methods

• **disappear()** - Makes the Custom Slider not visible or touchable on the LCD.
• **forceHit()** - Custom Slider performs its "hit" method without user input. The "hit" method will invoke all href functions of that object.
• **forceUpdate()** - Forces the Custom Slider to call its initHref function immediately. Only valid if initialCondition is FromInitHref. Updates the Custom Slider and performs a "hit".
• **maskedValue(y)** - Sends the intrinsic value of the Custom Slider to the calling widget. The calling object then uses that value, ANDed with the mask y, as its input. This method is only valid if the intrinsic value is a byte or word. The mask y can be either a byte or word. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.
• **reappear()** - Makes the Custom Slider visible and touchable on the LCD. Counteracts the disappear() method.
• **refresh()** - Redraws the Custom Slider on the LCD. Will not redraw if currently in a disappeared state.
• **setValue(x)** - The Custom Slider intrinsic value is changed to x.
• **setMethod(m)** - Changes the method originally specified by the Custom Slider's href parameter; only valid when the originally specified method is a single function.
• **setUARTMethod(m)** - Changes the UART method originally specified by the Custom Slider's href parameter; only valid when the originally specified method is a single function.
• **setVariableNumber(x)** - Changes the variable number originally specified in the href function to x. Can only be used only if the Custom Slider href uses byte(y), word(y) or string(y). In all three of the cases, the value y is replaced with the value x. Only valid when the originally specified method is a single function.
• **setX(x)** - Sets the x-coordinate of the topleft corner of the Custom Slider to the coordinate specified by the word x.
• **setY(x)** - Sets the y-coordinate of the topleft corner of the Custom Slider to the coordinate specified by the word x.
• **value()** - Sends the intrinsic value of the Custom Slider to the calling widget. The calling object then uses that value as its input. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.

Regarding Method Parameters

• **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
• **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Dynamic Image

Methods

• **clearCanvas()** - Clears the display where the canvas image resided. Essentially does an erase of the Dynamic Image Widget. This only affects what is displayed on the LCD, it does not affect the actual canvas image stored in flash.
• **disappear()** - Makes the Dynamic Image not visible on the LCD.
• **reappear()** - Makes the Dynamic Image visible on the LCD. Counteracts the disappear() method.
• **reset()** - Makes the Dynamic Image redraw its image. Should be called after loading a new image using the Amulet:loadFlash(reset) function.

- **setX(x)** - Sets the x-coordinate of the topleft corner of the Dynamic Image to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the Dynamic Image to the coordinate specified by the word x.

FunctionButton

Methods

- **buttonDown()** - Sets the Function Button Widget to look like it is its down state. This method does NOT invoke the href functions. It only affects the looks of the button, not the functionality.
- **buttonUp()** - Sets the Function Button Widget to look like it is its up state. This method does NOT invoke the href functions. It only affects the looks of the button, not the functionality.
- **disappear()** - Makes the Function Button not visible or touchable on the LCD.
- **forceHit()** - Function Button performs its "hit" method without user input. The "hit" method will invoke all href functions of that object.
- **forceUpdate()** - Forces the Function Button to call its initHref function immediately. Only valid if label is fromInitHref. Updates the Function Button label without performing a "hit".
- **maskedValue(y)** - Sends the intrinsic value of the Function Button to the calling widget. The calling object then uses that value, ANDed with the mask y, as its input. This method is only valid if the intrinsic value is a byte or word. The mask y can be either a byte or word. This allows a Function Button to provide the input to a View Widget. This method is called from a View Widget href.
- **reappear()** - Makes the Function Button visible and touchable on the LCD. Counteracts the disappear() method.
- **refresh()** - Redraws the Function Button on the LCD. Will not redraw if currently in a disappeared state.
- **setMethod(m)** - Changes the method originally specified by the Function Button's href parameter; only valid when the originally specified method is a single function.
- **setUARTMethod(m)** - Changes the UART method originally specified by the Function Button's href parameter; only valid when the originally specified method is a single function.
- **setValue(x)** - The Function Button intrinsic value is changed to x.
- **setVariableNumber(x)** - Changes the variable number originally specified in the href function to x. Can only be used only if the Function Button href uses byte(y), word(y) or string(y). In all three of the cases, the value y is replaced with the value x. Only valid when the originally specified method is a single function.
- **setX(x)** - Sets the x-coordinate of the topleft corner of the Function Button to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the Function Button to the coordinate specified by the word x.
- **value()** - Sends the intrinsic value of the Function Button to the calling widget. The calling object then uses that value as its input. This allows a Function Button to provide the input to a View Widget. This method is called from a View Widget href.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
- **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Image

Methods

- **disappear()** - Makes the image not visible on the LCD.
- **reappear()** - Makes the image visible on the LCD. Counteracts the disappear() method.
- **refresh()** - Redraws the image on the LCD. Will not redraw if currently in a disappeared state.
- **setX(x)** - Sets the x-coordinate of the topleft corner of the image to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the image to the coordinate specified by the word x.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
- **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

ImageBar

Methods

- **disappear()** - Makes the Image Bar not visible on the LCD
- **forceUpdate()** - Forces the Image Bar to call its href function immediately, regardless of the updateRate.
- **reappear()** - Makes the Image Bar visible on the LCD. Counteracts the disappear() method.
- **refresh()** - Redraws the Image Bar on the LCD. Will not redraw if currently in a disappeared state.
- **setMethod(m)** - Changes the method originally specified by the Image Bar's href parameter.
- **setUARTMethod(m)** - Changes the UART method originally specified by the Image Bar's href parameter.
- **setUpdateRate(f)** - Changes the update rate originally specified by the Image Bar, the argument being a floating point number, specifying time in seconds.
- **setVariableNumber(x)** - Changes the variable number originally specified by an Image Bar. Can be used only if the Image Bar href is byte(y).value() or word(y).value(). In either case, the y gets changed to the argument specified in setVariableNumber(x).
- **setValue(x)** - Image Bar uses x as its input. This allows a Control Widget to provide the input to an Image Bar.
- **setX(x)** - Sets the x-coordinate of the topleft corner of the Image Bar to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the Image Bar to the coordinate specified by the word x.
- **startUpdating()** - Image Bar starts updating based upon its input data. Counteracts the stopUpdating() method.
- **stopUpdating()** - Image Bar stops updating.
- **toggleUpdating()** - Changes current state of Image Bar; either starts or stops updating.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice

when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

ImageScroller

Methods

• **disappear()** - Makes the Image Scroller not visible or touchable on the LCD.
• **forceHit()** - Image Scroller performs its "hit" method without user input. The "hit" method will invoke all href functions of that object.
• **forceUpdate()** - Forces the Image Scroller to call its initHref function immediately. Only valid if initialCondition is FromInitHref. Updates the Image Scroller and performs a "hit".
• **maskedValue(y)** - Sends the intrinsic value of the Image Scroller to the calling widget. The calling object then uses that value, ANDed with the mask y, as its input. This method is only valid if the intrinsic value is a byte or word. The mask y can be either a byte or word. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.
• **reappear()** - Makes the Image Scroller visible and touchable on the LCD. Counteracts the disappear() method.
• **refresh()** - Redraws the Image Scroller on the LCD. Will not redraw if currently in a disappeared state.
• **setMethod(m)** - Changes the method originally specified by the Image Scroller's href parameter; only valid when the originally specified method is a single function.
• **setUARTMethod(m)** - Changes the UART method originally specified by the Image Scroller's href parameter; only valid when the originally specified method is a single function.
• **setVariableNumber(x)** - Changes the variable number originally specified in the href function to x. Can only be used if the Image Scroller href uses byte(y), word(y) or string(y). In all three of the cases, the value y is replaced with the value x. Only valid when the originally specified method is a single function.
• **setX(x)** - Sets the x-coordinate of the topleft corner of the Image Scroller to the coordinate specified by the word x.
• **setY(x)** - Sets the y-coordinate of the topleft corner of the Image Scroller to the coordinate specified by the word x.
• **value()** - Sends the intrinsic value of the Image Scroller to the calling widget. The calling object then uses that value as its input. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.

Regarding Method Parameters

• **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
• **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

ImageSequence

Methods

• **disappear()** - Makes the Image Sequence not visible on the LCD
• **forceRefresh()** - Forces the Image Sequence to paint the image at the next update, even if the incoming value is the same as the current state. Useful if an anchor is used around an Image Sequence, to force the image to be the

correct polarity. If the image changes while the anchor is selected, it is possible to have the polarity get swapped. Having the surrounding anchor href perform this on the Image Sequence will allow the image to stay correctly synched.
• **forceUpdate()** - Forces the Image Sequence to call its href function immediately, regardless of the updateRate.
• **reappear()** - Makes the Image Sequence visible on the LCD. Counteracts the disappear() method.
• **refresh()** - Redraws the Image Sequence on the LCD. Will not redraw if currently in a disappeared state.
• **setMethod(m)** - Changes the method originally specified by the Image Sequence's href parameter.
• **setUARTMethod(m)** - Changes the UART method originally specified by the Image Sequence's href parameter.
• **setUpdateRate(f)** - Changes the update rate originally specified by the Image Sequence, the argument being a floating point number, specifying time in seconds.
• **setValue(x)** - Image Sequence uses x as its input. This allows a Control Widget to provide the input to an Image Sequence.
• **setVariableNumber(x)** - Changes the variable number originally specified by an Image Sequence. Can be used only if the Image Sequence href is byte(y).value() or word(y).value(). In either case, the y gets changed to the argument specified in setVariableNumber(x).
• **setX(x)** - Sets the x-coordinate of the topleft corner of the Image Sequence to the coordinate specified by the word x.
• **setY(x)** - Sets the y-coordinate of the topleft corner of the Image Sequence to the coordinate specified by the word x.
• **startUpdating()** - Image Sequence starts using its input data to determine which image to display. Counteracts the stopUpdating() method.
• **stopUpdating()** - Image Sequence stops using its input data to determine which image to display.
• **toggleUpdating()** - Changes current state of Image Sequence; either starts or stops using its input data to determine which image to display.

Regarding Method Parameters

• **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
• **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Line Graph

Methods

• **disappear()** - Makes the Line Graph not visible on the LCD.
• **forceUpdate()** - Forces the Line Graph to call its href function immediately, regardless of the updateRate.
• **reappear()** - Makes the Line Graph visible on the LCD. Counteracts the disappear() method.
• **refresh()** - Redraws the Line Graph on the LCD. Will not redraw if currently in a disappeared state.
• **reset()** - Clears the Line Graph
• **setMethod(m)** - Changes the method originally specified by the Line Graph's href parameter.
• **setUARTMethod(m)** - Changes the UART method originally specified by the Line Graph's href parameter.
• **setUpdateRate(f)** - Changes the update rate originally specified by the Line Graph, the argument being a floating point number, specifying time in seconds.
• **setValue(x)** - Line Graph uses x as its input. This allows a Control Widget to provide the input to a Line Graph.

- **setVariableNumber(x)** - Changes the variable number originally specified by an Line Graph. Can be used only if the Line Graph href is byte(y).value() or word(y).value(). In either case, the y gets changed to the argument specified in setVariableNumber(x).
- **setX(x)** - Sets the x-coordinate of the topleft corner of the Line Graph to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the Line Graph to the coordinate specified by the word x.
- **startUpdating()** - Line Graph starts plotting based upon its input data. Counteracts the stopUpdating() method.
- **stopUpdating()** - Line Graph stops plotting.
- **toggleUpdating()** - Changes current state of Line Graph; either starts or stops plotting.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
- **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Line Plot

Methods

- **disappear()** - Makes the Line Plot not visible on the LCD.
- **forceUpdate()** - Forces the Line Plot to call its href function immediately, regardless of the updateRate.
- **reappear()** - Makes the Line Plot visible on the LCD. Counteracts the disappear() method.
- **refresh()** - Redraws the Line Plot on the LCD. Will not redraw if currently in a disappeared state.
- **reset()** - Clears the Line Plot.
- **setMethod(m)** - Changes the method originally specified by the Line Plot's href parameter.
- **setUARTMethod(m)** - Changes the UART method originally specified by the Line Plot's href parameter.
- **setUpdateRate(f)** - Changes the update rate originally specified by the Line Plot, the argument being a floating point number, specifying time in seconds.
- **setValue(x)** - Line Plot uses x as its input. This allows a Control Widget to provide the input to a Line Plot.
- **setVariableNumber(x)** - Changes the variable number originally specified by an Line Plot. Can be used only if the Line Plot href is byte(y).value() or word(y).value(). In either case, the y gets changed to the argument specified in setVariableNumber(x).
- **setX(x)** - Sets the x-coordinate of the topleft corner of the Line Plot to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the Line Plot to the coordinate specified by the word x.
- **startUpdating()** - Line Plot starts plotting based upon its input data. Counteracts the stopUpdating() method.
- **stopUpdating()** - Line Plot stops plotting.
- **toggleUpdating()** - Changes current state of Line Plot; either starts or stops plotting.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.

- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
- **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Linear Gauge

Methods

- **disappear()** - Makes the Linear Gauge not visible on the LCD.
- **forceUpdate()** - Forces the Linear Gauge to call its href function immediately, regardless of the updateRate.
- **reappear()** - Makes the Linear Gauge visible on the LCD. Counteracts the disappear() method.
- **refresh()** - Redraws the Linear Gauge on the LCD. Will not redraw if currently in a disappeared state.
- **reset()** - Clears the Linear Gauge.
- **setMethod(m)** - Changes the method originally specified by the Linear Gauge's href parameter.
- **setUARTMethod(m)** - Changes the UART method originally specified by the Linear Gauge's href parameter.
- **setUpdateRate(f)** - Changes the update rate originally specified by the Linear Gauge, the argument being a floating point number, specifying time in seconds.
- **setValue(x)** - Linear Gauge uses x as its input. This allows a Control Widget to provide the input to a Linear Gauge.
- **setVariableNumber(x)** - Changes the variable number originally specified by a Linear Gauge. Can be used only if the Linear Gauge href is byte(y).value() or word(y).value(). In either case, the y gets changed to the argument specified in setVariableNumber(x).
- **setX(x)** - Sets the x-coordinate of the topleft corner of the Linear Gauge to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the Linear Gauge to the coordinate specified by the word x.
- **startUpdating()** - Linear Gauge starts updating based upon its input data. Counteracts the stopUpdating() method.
- **stopUpdating()** - Linear Gauge stops updating.
- **toggleUpdating()** - Changes current state of Linear Gauge; either starts or stops updating.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
- **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

List

Methods

- **disappear()** - Makes the List not visible or touchable on the LCD.
- **forceHit()** - List performs its "hit" method without user input. The "hit" method will invoke all href functions of that object.
- **forceUpdate()** - Forces the List to call its initHref function immediately. Only valid if initialCondition is FromInitHref. Updates the List and performs a "hit".

• **maskedValue(y)** - Sends the intrinsic value of the List to the calling widget. The calling object then uses that value, ANDed with the mask y, as its input. This method is only valid if the intrinsic value is a byte or word. The mask y can be either a byte or word. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.
• **nextEntry()** - Highlighted box of a list box widget moves to next entry in the list. Effectively moves the highlighted box down one entry. Does NOT automatically perform a "hit" method.
• **previousEntry()** - Highlighted box of a list box widget moves to previous entry in the list. Effectively moves the highlighted box up one entry. Does NOT automatically perform a "hit" method.
• **reappear()** - Makes the List visible and touchable on the LCD. Counteracts the disappear() method.
• **refresh()** - Redraws the List on the LCD. Will not redraw if currently in a disappeared state.
• **setMethod(m)** - Changes the method originally specified by the List's href parameter; only valid when the originally specified method is a single function.
• **setUARTMethod(m)** - Changes the UART method originally specified by the List's href parameter; only valid when the originally specified method is a single function.
• **setVariableNumber(x)** - Changes the variable number originally specified in the href function to x. Can only be used if the List href uses byte(y), word(y) or string(y). In all three of the cases, the value y is replaced with the value x. Only valid when the originally specified method is a single function.
• **setX(x)** - Sets the x-coordinate of the topleft corner of the List to the coordinate specified by the word x.
• **setY(x)** - Sets the y-coordinate of the topleft corner of the List to the coordinate specified by the word x.
• **value()** - Sends the intrinsic value of the List to the calling widget. The calling object then uses that value as its input. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.

Regarding Method Parameters

• **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
• **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

META Refresh Tag

Methods

• **disappear()** - Stops the META Refresh Tag URL from being launched.
• **forceHit()** - META Refresh Tag will perform its "hit" method without user input. The "hit" method will invoke all href functions of the META Refresh Tag.
• **forceUpdate()** - Forces the META Refresh Tag to call its trigger variable function immediately.
• **maskedValue(y)** - Sends the intrinsic value associated with the META Refresh Tag to the calling widget. The calling object then uses that value, ANDed with the mask y, as its input. This method is only valid if the intrinsic value is a byte or word. The mask y can be either a byte or word. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.
• **reappear()** - Allows the META Refresh Tag URL to be launched. Counteracts the disappear() method.
• **reset()** - Resets timers in the META Refresh Tag.
• **setValue(x)** - The META Refresh Tag intrinsic value is changed to x.
• **setMethod(m)** - Changes the method originally specified by the META Refresh Tag's URL parameter; it is only valid when the originally specified method is a single function. If the URL method uses the UART, then you must use the setUARTMethod() method.

• **setOnVarMethod(m)** - Changes the method originally specified by the META Refresh Tag's trigger variable parameter; it is only valid when the originally specified method is a single function. If the ONVAR method uses the UART, then you must use the setOnVarUARTMethod() method.
• **setOnVarUARTMethod(m)** - Changes the UART method originally specified by the META Refresh Tag's trigger variable parameter; it is only valid when the originally specified method is a single function. If the ONVAR method does not use the UART, then you must use the setOnVARMethod() method.
• **setOnVarVariableNumber(x)** - Changes the variable number originally specified by the META Refresh's onVar parameter. Can be used only if the META Refresh onVar has a byte(y), word(y) or string(y). In all cases, the y gets changed to the argument specified in setOnVarVariableNumber(x).
• **setTrigger(x)** - Changes the trigger value originally specified by the META Refresh's **trigger**, **trigger.gt or trigger.lt** parameter to the byte value x.
• **setUARTMethod(m)** - Changes the UART method originally specified by the META Refresh Tag's URL parameter; it is only valid when the originally specified method is a single function. If the URL method does not use the UART, then you must use the setMethod() method.
• **setUpdateRate(f)** - Changes the update rate originally specified by the META Refresh Tag, the argument being a floating point number, specifying time in **seconds.**
• **setVariableNumber(x)** - Changes the variable number originally specified by the META Refresh's URL parameter. Can be used only if the META Refresh URL has a byte(y), word(y) or string(y). In all cases, the y gets changed to the argument specified in setVariableNumber(x); it is only valid when the originally specified method is a single function.
value() - Sends the intrinsic value associated with the META Refresh Tag to the calling widget. The calling object then uses that value as its input. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.

Regarding Method Parameters

• **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
• **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Numeric Field

Methods

• **disappear()** - Makes the Numeric Field not visible on the LCD.
• **forceUpdate()** - Forces the Numeric Field to call its href function immediately, regardless of the updateRate.
• **inverseRegionColor()** - Makes the entire Numeric Field (both the background and the text) display in reverse video. Counteracts the normalRegionColor() method.
• **inverseStringColor()** - Makes the Numeric Field's text string display in reverse video. Counteracts the normalStringColor() method.
• **normalRegionColor()** - Makes the entire Numeric Field (both the background and the text) display in normal video. Counteracts the inverseRegionColor() method.
• **normalStringColor()** - Makes the Numeric Field's text string display in normal video. Counteracts the normalStringColor() method.
• **reappear()** - Makes the Numeric Field visible on the LCD. Counteracts the disappear() method.
• **refresh()** - Redraws the Numeric Field on the LCD. Will not redraw if currently in a disappeared state.
• **setMethod(m)** - Changes the method originally specified by the Numeric Field's href parameter.
• **setUARTMethod(m)** - Changes the UART method originally specified by the Numeric Field's href parameter.

- **setUpdateRate(f)** - Changes the update rate originally specified by the Numeric Field, the argument being a floating point number, specifying time in seconds.
- **setValue(x)** - Numeric Field uses x as its input. This allows a Control Widget to provide the input to a Numeric Field.
- **setVariableNumber(x)** - Changes the variable number originally specified by an Numeric Field. Can be used only if the Numeric Field href is byte(y).value() or word(y).value(). In either case, the y gets changed to the argument specified in setVariableNumber(x).
- **setX(x)** - Sets the x-coordinate of the topleft corner of the Numeric Field to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the Numeric Field to the coordinate specified by the word x.
- **startUpdating()** - Numeric Field starts updating based upon its input data. Counteracts the stopUpdating() method.
- **stopUpdating()** - Numeric Field stops updating.
- **toggleRegionColor()** - Toggles the polarity of the entire Numeric Field (both the background and the text).
- **toggleStringColor()** - Toggles the Numeric Field's text string polarity. Does not affect the background color.
- **toggleUpdating()** - Changes current state of Numeric Field; either starts or stops updating.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
- **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Pulse Width Modulation

Methods

- **setPeriod(x)**[1] - Sets the period of the PWM object, in ms. Range is 0.01-2040. (MK-07C-HP Module range is 0.003-129000)
- **setPulseWidth(x)**[1] - Sets the pulse width of the PWM object, in ms. Range is 0-2040. (MK-07C-HP Module range is 0-129000)
- **start()** - Starts the PWM object, using the current period and pulse width settings.
- **stop()** - Stops the PWM object.

1. Regarding x: For static numbers, x is defined in milliseconds, but if the value is passed from a widget's intrinsic value, for example passed from a Slider widget, or from InternalRAM variables, this becomes microseconds.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.

• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Radio Button

Methods

• **disappear()** - Makes the Radio Button not visible or touchable on the LCD.
• **forceHit()** - Radio Button performs its "hit" method without user input. The "hit" method will invoke all href functions of that object. Unlike the CheckBox, a forceHit can only be imparted to an individual Radio Button, not a Radio Button group.
• **forceUpdate()** - Forces the Radio Button group to call its **initHref** function immediately. Only valid if **initialCondition** is FromInitHref. Updates the Radio Button group and performs a "hit". To forceUpdate a Radio Button group, use the **groupName** as the *widgetName* (rather than the individual Radio Button name).
• **maskedValue(y)** - Sends the intrinsic value of the Radio Button to the calling widget. The calling object then uses that value, ANDed with the mask y, as its input. This method is only valid if the intrinsic value is a byte or word. The mask y can be either a byte or word. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.
• **reappear()** - Makes the Radio Button visible and touchable on the LCD. Counteracts the disappear() method.
• **refresh()** - Redraws the Radio Button on the LCD. Will not redraw if currently in a disappeared state.
• **setMethod(m)** - Changes the method originally specified by the Radio Button's href parameter; only valid when the originally specified method is a single function.
• **setUARTMethod(m)** - Changes the UART method originally specified by the Radio Button's href parameter; only valid when the originally specified method is a single function.
• **setValue(x)** - The Radio Button intrinsic value is changed to x.
• **setVariableNumber(x)** - Changes the variable number originally specified in the href function to x. Can only be used only if the Radio Button href uses byte(y), word(y) or string(y). In all three of the cases, the value y is replaced with the value x. Only valid when the originally specified method is a single function. Can only be used on an individual Radio Button, not the entire Radio Button Group.
• **setX(x)** - Sets the x-coordinate of the topleft corner of the individual Radio Button to the coordinate specified by the word x.
• **setY(x)** - Sets the y-coordinate of the topleft corner of the individual Radio Button to the coordinate specified by the word x.
• **value()** - Sends the intrinsic value of the Radio Button to the calling widget. The calling object then uses that value as its input. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.

Regarding Method Parameters

• **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
• **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Scribble

Methods

- **clearCanvas()** - Clears the scribble canvas completely, including any background images in the canvas.
- **disappear()** - Makes the Scribble Widget not visible or touchable on the LCD.
- **reappear()**[1] - Makes the Scribble Widget touchable on the LCD. Partialy counteracts the disappear() method. Should be immediatly followed by a paintBackground() or paintCanvas().
- **paintBackground()** - Redraws the background image from flash.
- **paintCanvas()** - Redraws the canvas image from flash
- **reset()** - Clears the canvas image currently on the Scribble Widget and redraws the canvas image stored in flash.
- **saveCanvas()** - The canvas displayed on the LCD is saved to flash, overwriting the previous canvas image.
- **setLinePattern(x)** - Changes the line pattern of the active freehand drawing line to x.
- **setLineWeight(x)** - Changes the line weight of the active freehand drawing line, in pixels, to x. Range is 1-15
- **setX(x)** - Sets the x-coordinate of the topleft corner of the Scribble Widget to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the Scribble Widget to the coordinate specified by the word x.
- **uploadImage()** - The canvas displayed on the LCD is uploaded to an external processor using xmodem crc protocol. The image will be sent in the Amulet Bitmap Format.

1. Regarding reappear(): If reappear() is selected, either paintCanvas() or paintBackground() should be called immediately afterwards, because the reappear() method for the scribble widget only allows redrawing, it does NOT redraw either the canvas or the background image. If it is desired to display exactly what was on the screen at the time of the disappear, they should first saveCanvas(), then call disappear(). Then when calling reappear(), also call paintCanvas().

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
- **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Slider

Methods

- **disappear()** - Makes the Slider not visible or touchable on the LCD.
- **forceHit()** - Slider performs its "hit" method without user input. The "hit" method will invoke all href functions of that object.
- **forceUpdate()** - Forces the Slider to call its **initHref** function immediately. Only valid if **initialCondition** is **FromInitHref**. Updates the Slider and performs a "hit".
- **maskedValue(y)** - Sends the intrinsic value of the Slider to the calling widget. The calling object then uses that value, ANDed with the mask y, as its input. This method is only valid if the intrinsic value is a byte or word. The mask y can be either a byte or word. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.
- **reappear()** - Makes the Slider visible and touchable on the LCD. Counteracts the disappear() method.
- **refresh()** - Redraws the Slider on the LCD. Will not redraw if currently in a disappeared state.
- **setValue(x)** - The Slider intrinsic value is changed to x.

• **setMethod(m)** - Changes the method originally specified by the Slider's href parameter; only valid when the originally specified method is a single function.
• **setUARTMethod(m)** - Changes the UART method originally specified by the Slider's href parameter; only valid when the originally specified method is a single function.
• **setVariableNumber(x)** - Changes the variable number originally specified in the href function to x. Can only be used only if the Slider href uses byte(y), word(y) or string(y). In all three of the cases, the value y is replaced with the value x. Only valid when the originally specified method is a single function.
• **setX(x)** - Sets the x-coordinate of the topleft corner of the Slider to the coordinate specified by the word x.
• **setY(x)** - Sets the y-coordinate of the topleft corner of the Slider to the coordinate specified by the word x.
• **value()** - Sends the intrinsic value of the Slider to the calling widget. The calling object then uses that value as its input. This allows a Control object to provide the input to a View Widget. This method is called from a View Widget href.

Regarding Method Parameters

• **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
• **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
• **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

String Field

Methods

• **addCanvasX(x)** - Moves the StringField's canvas x pixels to the right.
• **addCanvasY(x)** - Moves the StringField's canvas x pixels down.
• **disappear()** - Makes the String Field not visible on the LCD.
• **forceUpdate()** - Forces the String Field to call its href function immediately, regardless of the updateRate
• **inverseRegionColor()** - Makes the entire String Field (both the background and the text) display in reverse video. Counteracts the normalRegionColor() method.
• **inverseStringColor()** - Makes the String Field's text string display in reverse video. Counteracts the normalStringColor() method.
• **normalRegionColor()** - Makes the entire String Field (both the background and the text) display in normal video. Counteracts the inverseRegionColor() method.
• **normalStringColor()** - Makes the String Field's text string display in normal video. Counteracts the normalStringColor() method.
• **reappear()** - Makes the String Field visible on the LCD. Counteracts the disappear() method.
• **refresh()** - Redraws the String Field on the LCD. Will not redraw if currently in a disappeared state.
• **setCanvasX(x)** - Moves the StringField's canvas to the X-coordinate x.
• **setCanvasY(x)** - Moves the StringField's canvas to the Y-coordinate x.
• **setValue(x)** - String Field uses x as an input. This allows a Control Widget to provide the input to a String Field.
• **setMethod(m)** - Changes the method originally specified by the String Field's href parameter.
• **setUARTMethod(m)** - Changes the UART method originally specified by the String Field's href parameter.
• **setUpdateRate(f)** - Changes the update rate originally specified by the String Field, the argument being a floating point number, specifying time in seconds.
• **setVariableNumber(x)** - Changes the variable number originally specified by a String Field. Can be used only if the String Field href is byte(y).value() or string(y).value(). In both cases, the y gets changed to the argument specified in setVariableNumber(x).

- **setX(x)** - Sets the x-coordinate of the topleft corner of the String Field to the coordinate specified by the word x.
- **setY(x)** - Sets the y-coordinate of the topleft corner of the String Field to the coordinate specified by the word x.
- **startUpdating()** - String Field starts updating based upon its input data. Counteracts the stopUpdating() method.
- **stopUpdating()** - String Field stops updating.
- **subCanvasX(x)** - Moves the StringField's canvas x pixels to the left.
- **subCanvasY(x)** - Moves the StringField's canvas x pixels up.
- **toggleRegionColor()** - Toggles the polarity of the entire String Field (both the background and the text).
- **toggleStringColor()** - Toggles the String Field's text string polarity. Does not affect the background color.
- **toggleUpdating()** - Changes current state of String Field; either starts or stops updating.

Regarding Method Parameters

- **Regarding x** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use x. The range for x is 0-255 (0x00-0xff) for a BYTE, 0-65535 (0x00-0xffff) for a WORD and strings in double quotes for STRINGs.
- **Regarding m** - When setMethod(),setOnVarMethod(),setOnVarUARTMethod() or setUARTMethod(), is the IWC method, the argument should be the name of the method you want to set. i.e. disappear() or byte.value(). Notice when dealing with a method that relies on a type (byte, word or string) you need to include the type separated by a dot and then the method (i.e. word.value()) instead of just the method by itself.
- **Regarding f** - For Control Widgets that have intrinsic values, such as lists and sliders, leave the argument field empty, since the intrinsic value of the selection will be sent out. Anchors, META Refresh Tags, Area Maps and Function/Custom Buttons should use f. Like the regular **updateRate**, use the floating point number to specify the update rate in seconds. The range for f is 0-655.35.

Restore Amulet OS

The Amulet OS needs to be restored when the following occurs:
1. When there is an OS mismatch between the display and the compiler or GEMstudio.
2. When the flash is corrupted.
3. When OS related settings are modified such as touch panel, init file or LCD settings.

To restore Amulet OS:
1. Flip respective dip switch (shown in the list below) to off position.

Part Number	Switch#	Dip#
EVK-TA-TM035KBH02	18	1
EVK-VX-COG-T350MCQV-03	18	2
EVK-TA-TM043NBH02	6	1
EVK-TA-TM047NBH01	6	1
EVK-HX-HDA570ST-VH	6	1
EVK-HX-HDA570ST-V	6	1
EVK-KA-TCG057VGLBL-C50	18	2
EVK-SY-SCA05711-BFN-LRA	1	2
EVK-KA-TCG062HVLBC-G20	18	2

EVK-SY-SCA07010-BFN-LRA	12	2
EVK-TA-TM070RBH10	12	1
GEMboard	6	1
GEMboard II	6	1
STK-480272C	6	1
MK-480272C	6	1

2. Power up the unit.
3. Plug in the USB cable.
4. In GEMstudio, open the project to be programmed.
5. Select the Program button
6. In the Program Flash pop-up window, check the Include OS Files check box
7. Select the Program Project button

The project and Amulet OS Files should then be programmed into the module.

GEMstudio Production Files

Once you are finished with your project, you can save the project files along with the latest OS files tuned specifically for your display in one production ready file. There are two production file types. The first is a .gem file, which is compressed and encrypted, but not easily parsable. A .gem file can be used if GEMstudio or GEMprogrammer will be used to program the .gem file into the Amulet module. The second is a .pdb file, which is neither compressed nor encrypted, but is easily parsable. A .pdb file is useful if you are going to program the Amulet module via your processor.

Production File Builds

There are three options in GEMstudio's File menu when choosing to save a production file:

Save as Production File With OS - Everything needed to run the GUI from scratch
Save as Update File With OS - Same as above except it leaves out the touchpanel calibration file to preserve the custom calibration already on the hardware. This is particularly useful with resistive touchscreens, whose default behavior is to force a recalibration whenever installing OS.
Save as Update File - This only contains the custom GUI, no supporting OS configurations. Use this only when you know the Amulet OS version you are updating on is the same.

Opening a Production File

You can open either a .gem or a .pdb file in GEMstudio by selecting File > Open Production File. Once the .gem or .pdb file has been loaded into GEMstudio, you can either Run the project in GEMplayer or you can Program the project into an Amulet module.

InField Programming

This feature enables the user to program a pre-compiled Amulet project file generated by GEMstudio through an external processor. See the next three topics to learn more.

Pass Thru Programming

Here are the basics you need to know to program the Amulet Color chip from an external processor over a serial port.

1. You need to save your compiled project as a .pdb file.

2. You need to extract the individual files from the pdb format so you have individual binary files which can be sent to the Amulet. Palm Database Format explains how to parse this file.

3. Those binary files are sent to the Amulet using a slightly modified 128-byte or 1K Xmodem (with 16-bit CRC) protocol. For 1k-Xmodem, the SOH byte is 0x02, and for 128-byte it is 0x01. See XMODEM Protocol regarding the protocol used and review the notes below.

Here are the details you need to know:

1. Bytes 0-5 of each binary file are the file header bytes. The Amulet uses these bytes to determine where to store the file. You can send the files in any order, the Amulet won't mind.

2. In standard Xmodem protocol the slave sends out a 'C' character once per second. In Amulet's color chip, we will instead output a character that indicates the size of the serial dataflash connected to first chip select CS0 of the SPI bus. This character is reffered to as the Flash Size Notifier, or FSN. The flash size to character mapping is as follows:

    a : 1Mbit
    b : 2Mbit
    c : 4Mbit
    d : 8Mbit
    e : 16Mbit
    f : 32Mbit (standard starter kit size)
    g : 64Mbit (GCC-2 size)
    C: eMMC (MK-07C-HP Module only)
    Z: None Detected

3. There is no time limit between files. The chip will wait about 2 seconds, and if it gets no response, then it just sends out the same thing again. There is a 500ms time limit between packets of the same file and between bytes of the same packet. If either of these 500ms time limits are met, then that specific file transfer is aborted, and the FSN will be sent out, and that file will need to be resent. Receiving just one byte of the next packet will reset that timer, so systems with excessive latency can build in an additional buffer because the first three bytes are predictable.

4. You can set the Amulet up to program either by flipping the Program Mode (pin 26) to HIGH and asserting the reset, or the much cleaner way of sending a "wake up" message (0xA0, 0x02, 0x00, 0x16, 0x48), which has to be at the same baud rate as the page you are currently on when trying to wake it up (4800, 9600, 19200, 38400, 57600, 115200, etc). By default, it will also program at this rate. Please contact Amulet if you would like to change the default programming rate to something other than the current baud rate. This requires a quick modification of the OS files.

5. When entering the programming mode from software "wake up" message, there will be an OS version String output after 3 seconds. The compiler uses this message to make sure the firmware and compiler version match, but you can ignore this if you do not want to use it. You can skip the 3 second wait by sending a 0x43 (ASCII 'C') after sending the "wake up" message. Additionally, the OS version string may contain characters which match the expected FSN, but are not valid FSN characters. The OS string is encapsulated with { and } brackets, or in hex: 0x7B and 0x7D respectively. This means if entering program mode with the software command, you should wait until after the closing } bracket before looking for the FSN to begin programming.

6. Xmodem has a CRC, but there is an optional secondary CRC that checks the data that was actually written to the flash. After sending the EOT byte (0x04) after the last packet in each file, you can send what we have dubbed the EOTCRC (0x16) This will cause the Amulet color chip to calculate a new CRC based off of the data in the flash. A match is indicated by an ACK and a non-match by a NACK. The FSN will begin transmission after the ACK or NACK. A NACK'd file should be reprogrammed. Every 500ms during the flash read and CRC calculation a 0x14 will be sent to let the host know we're still alive, but working on the CRC verification. This would only happen on larger files.

7. When there are no more files to send the Amulet, send an ETB (0x17) and the Amulet will reboot and start running at the new homepage that you just programmed.

For LabVIEW users, this process is implemented in the "Deploy PDB" VI that is part of the Amulet LabVIEW driver.

Palm Database Format

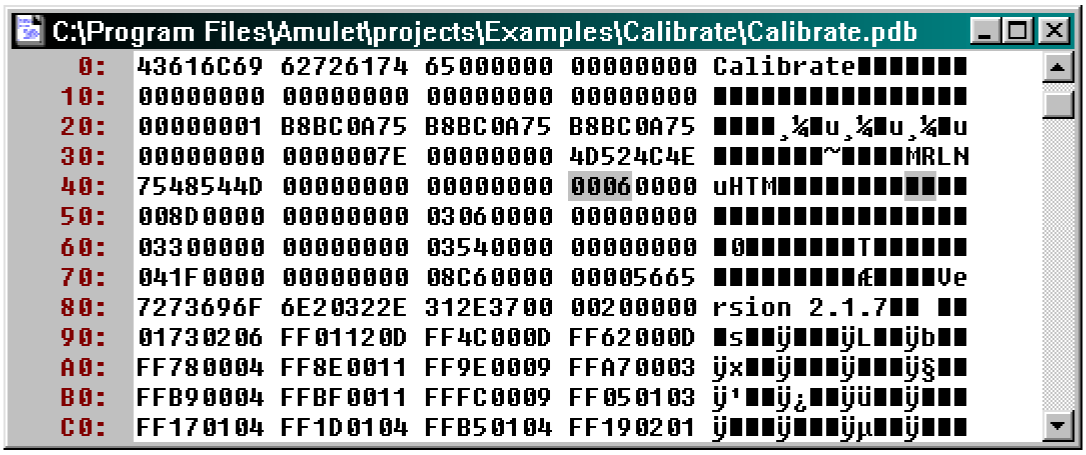
Explaining the entire Palm database format in detail is beyond the scope of this document, so only the key components will be discussed. A PDB file is made up of the following key components:

1. A header which describes the database.
2. A list of record entries, each describing a chunk of raw data.
3. And the raw data itself which is stored linearly.

Since a PDB can consist of many files, depending on the user interface, one must know how to get to all the files in the PDB in order to remotely program the whole user interface. Each file of a user interface has its own record entry in the PDB as well as the raw data that pertains to it.

Parsing the PDB

This section will use a hex editor such as CodeWright in order to show the steps involved in parsing through a PDB file for remote programming. The following image is a snapshot of an example PDB opened in CodeWright in hex format:



```
C:\Program Files\Amulet\projects\Examples\Calibrate\Calibrate.pdb
  0: 43616C69 62726174 65000000 00000000 Calibrate███████
 10: 00000000 00000000 00000000 00000000 ████████████████
 20: 00000001 B8BC0A75 B8BC0A75 B8BC0A75 ████¸¼█u¸¼█u¸¼█u
 30: 00000000 0000007E 00000000 4D524C4E ███████~████MRLN
 40: 7548544D 00000000 00000000 00060000 uHTM████████████
 50: 008D0000 00000000 03060000 00000000 ████████████████
 60: 03300000 00000000 03540000 00000000 █0███████T██████
 70: 041F0000 00000000 08C60000 00005665 █████████Æ████Ve
 80: 7273696F 6E20322E 312E3700 00200000 rsion 2.1.7██ ██
 90: 01730206 FF01120D FF4C000D FF62000D █s██ÿ███ÿL██ÿb██
 A0: FF780004 FF8E0011 FF9E0009 FFA70003 ÿx██ÿ███ÿ███ÿ§██
 B0: FFB90004 FFBF0011 FFFC0009 FF050103 ÿ¹██ÿ¿██ÿü██ÿ███
 C0: FF170104 FF1D0104 FFB50104 FF190201 ÿ███ÿ███ÿµ██ÿ███
```

The total number of record entries (user interface pages) is stored as a 16-bit number in the header at location 0x4C. In this example, there are 6 unique record entries. Immediately following the total number of record entries are the record entry arrays, consisting of 8 bytes per array, for each of the 6 record entries. In the case of the example in Figure above, the first record entry array equals "00,00,00,8D,00,00,00,00". The important part of the record entry array is the starting offset of the raw data which is a 32-bit number which can be found in bytes 1 through 4 (0x0000008D). As you can see from this example, data is stored Big Endian style, meaning the bytes go from most significant to least significant.

The starting offset is from the start of the PDB file (0x00). You can calculate the size of the raw data by subtracting the starting offset from the the next chunk's starting offset, if one is available. If not, then use the end of the file to calculate the last entry's raw data size. To help clear this up, see the table below:

File#	Starting Offset of Current Record	Starting Offset of Next Record	Length of Raw Data
1	0x0000008D	0x00000306	0x0279
2	0x00000306	0x00000330	0x002A
3	0x00000330	0x00000354	0x0024
4	0x00000354	0x0000041F	0x00CB

5	0x0000041F	0x000008C6	0x04A7
6	0x000008C6	End of File (0x08F3)	0x002D

It is the raw data of the individual record entries which need to be transmitted via xmodem over to the Amulet chip. Now that you know how to parse a Palm database file, you can implement the Xmodem Protocol in your microprocessor firmware and have the flexibility of reprogramming any Amulet module via your controller.

XModem Protocol

Introduction

The Xmodem protocol was created years ago as a simple means of having two computers talk to each other. With its half-duplex mode of operation, 128- byte packets, ACK/NACK responses and CRC data checking, the Xmodem protocol has found its way into many applications. In fact most communication packages found on the PC today have a Xmodem protocol available to the user.

NOTE: Amulet uses a slightly modified 128-byte or 1K Xmodem (with 16-bit CRC) protocol. For 1k-Xmodem, the SOH byte is 0x02, and for 128-byte it is 0x01.

Theory of Operation

Xmodem is a half-duplex communication protocol. The receiver, after receiving a packet, will either acknowledge (ACK) or not acknowledge (NAK) the packet. The CRC extension to the original protocol uses a more robust 16-bit CRC to validate the data block and is used here. Xmodem can be considered to be receiver driven. That is, the receiver sends an initial character “C” to the sender indicating that it is ready to receive data in CRC mode. The sender then sends a 133-byte packet, the receiver validates it and responds with an ACK or a NAK at which time the sender will either send the next packet or re-send the last packet. This process is continued until an EOT is received at the receiver side and is properly ACKed to the sender. After the initial handshake the receiver controls the flow of data through ACKing and NAKing the sender.

Table 1. XmodemCRC Packet Format

Byte 1	**Byte 2**	**Byte 3**	**Bytes 4-131**	**Bytes 132-133**
Start of Header	Packet Number	(Packet Number)	Packet Data	16-bit CRC

NOTE: Bytes 0-5 of each binary file are the flash header bytes. The Amulet uses these bytes to determine where to store the file within the flash. You can send the files in any order, the Amulet won't mind.

Definitions

The following defines are used for protocol flow control.

Symbol	**Description**	**Value**
SOH	Start of Header	0x01 for 128-byte protocol 0x02 for 1K protocol

EOT	End of Transmission	0x04
ACK	Acknowledge	0x06
NAK	Not Acknowledge	0x15
EOTCRC	End of Transmission CRC (Amulet Chip will recalculate the CRC based off of the data in the flash)	0x16
ETB	End of Transmission Block (Return to Amulet OS mode)	0x17
CAN	Cancel (Force receiver to start sending C's)	0x18
C	ASCII “C”	0x43

Byte 1 of the XmodemCRC packet can only have a value of SOH, EOT, CAN or ETB anything else is an error. Bytes 2 and 3 form a packet number with checksum, add the two bytes together and they should always equal 0xff. Please note that the packet number starts out at 1 and rolls over to 0 if there are more than 255 packets to be received. Bytes 4 - 131 form the data packet and can be anything. Bytes 132 and 133 form the 16-bit CRC. The high byte of the CRC is located in byte 132. The CRC is calculated only on the data packet bytes (4 - 131) .

Xmodem has a CRC, but there is an optional secondary CRC that checks the data that was actually written to the flash. After sending the EOT byte (0x04) after the last packet in each file, you can send what we have dubbed the EOTCRC (0x16) This will cause the Amulet color chip to calculate a new CRC based off of the data in the flash. A match is indicated by an ACK and a non-match by a NACK. The FSN will begin transmission after the ACK or NACK. A NACK'd file should be reprogrammed.

Synchronization

In the standard Xmodem Protocol, the receiver starts by sending an ASCII “C” (0x43) character to the sender indicating it wishes to use the CRC method of block validating. In Amulet's color chip, we will instead output a character that indicates the size of the serial dataflash connected to first chip select CS0 of the SPI bus. This character is referred to as the Flash Size Notifier, or FSN. The flash size to character mapping is as follows:

a : 1Mbit
b : 2Mbit
c : 4Mbit
d : 8Mbit
e : 16Mbit
f : 32Mbit (standard starter kit size)
g : 64Mbit (GCC-2 size)
C: eMMC (MK-07C-HP Module only)
Z : None Detected

After sending the initial FSN character, the receiver waits for either a 2 second time out or until a buffer full flag is set. If the receiver is timed out then another FSN character or an ACK is sent to the sender and the 2 second time out starts again. This process continues until the receiver receives a complete 133-byte packet.

Receiver Considerations

This protocol NAKs the following conditions: 1. Framing error on any byte 2. Overrun error on any byte 3. Duplicate packet 4. CRC error 5. Receiver timed out (didn't receive packet within 1 second) On any NAK, the sender will re-transmit the last packet. Items 1 and 2 should be considered serious hardware failures. Verify that sender and receiver are using the same baud rate, start bits and stop bits. Item 3 is usually the sender getting an ACK garbled and re-transmitting the packet. Item 4 is found in noisy environments. And the last issue should be self-correcting after the receiver NAKs the sender.

Sender						**Receiver**
					<---	FSN Character
						Times out after 2 seconds
					<---	FSN Character
SOH	0x01	0xFE	Data	CRC	--->	Packet OK
					<---	ACK
SOH	0x02	0xFD	Data	CRC	--->	(Line hit during transmission)
					<---	NACK
SOH	0x02	0xFD	Data	CRC	--->	Packet OK
					<---	ACK
SOH	0x03	0xFC	Data	CRC	--->	Packet OK
(ACK gets garbled)					<---	ACK
					<---	ACK
SOH	0x04	0xFB	Data	CRC	--->	(UART Framing Error on Any Byte)
					<---	NACK
SOH	0x04	0xFB	Data	CRC	--->	PACKET OK
					<---	ACK
SOH	0x05	0xFA	Data	CRC	--->	(UART Overrun Error on Any Byte)
					<---	NACK
SOH	0x05	0xFA	Data	CRC	--->	Packet OK
					<---	ACK
EOT					--->	Packet OK
					<---	ACK
ETB					--->	Finished
Finished					<---	ACK

Sample crc calculation code

```
int calcrc(char *ptr, int count)
{
    int  crc;
    char i;

    crc = 0;
    while (--count >= 0)
    {
        crc = crc ^ (int) *ptr++ << 8;
        i = 8;
        do
        {
            if (crc & 0x8000)
                crc = crc << 1 ^ 0x1021;
            else
                crc = crc << 1;
        } while(--i);
    }
    return (crc);
}
```

Embedded GUI Video Tutorials

If you haven't already done so, please take the time to look at the Online Video Tutorials. The tutorials will provide you with an overview of how to create, compile and simulate your GUI. To get to the tutorials, pull down the menu for Help when you open a GEMstudio window and select "Online Video Tutorials". For additional examples, please go to your GEMstudio directory and click on the .gemp files.
What will I learn in the tutorials?

- Creating and Compiling a GUI.
- Simulate the GUI
- Basics of the GEMscript scripting language

Click here for the videos.

GEMstudio Change Log

Go to the GEMstudio Help menu and select "View Change Log" to see all new features, enhancements, and bug fixes organized by the GEMstudio version number when they were added.