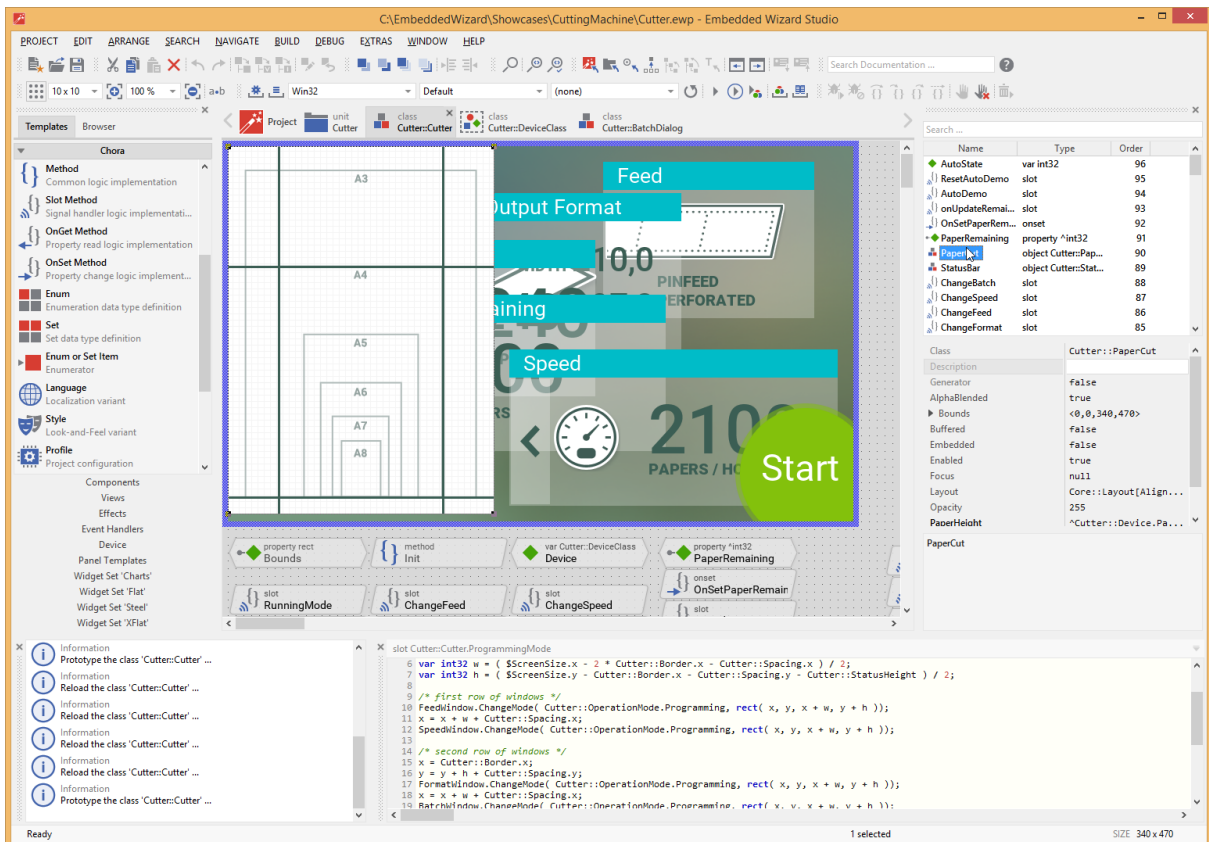




Embedded Wizard

Release Notes

- please read carefully -



The screenshot displays the Embedded Wizard Studio environment. The main workspace shows a UI prototype for a cutting machine control panel. The UI features a grid of paper sizes (A3, A4, A5, A6, A7, A8), a 'Feed' section with 'Output Format' and '10,0', a 'Speed' section with '2100 PAPERS / HOUR' and a 'Start' button. The software interface includes a menu bar (PROJECT, EDIT, ARRANGE, SEARCH, NAVIGATE, BUILD, DEBUG, EXTRAS, WINDOW, HELP), toolbars, a project browser, a class browser, and a code editor. The code editor shows the following code:

```
slot Cutter::ProgrammingMode
6 var int32 w = ( $ScreenSize.x - 2 * Cutter::Border.x - Cutter::Spacing.x ) / 2;
7 var int32 h = ( $ScreenSize.y - Cutter::Border.y - Cutter::Spacing.y - Cutter::StatusHeight ) / 2;
8
9 /* first row of windows */
10 FeedWindow.ChangeNode( Cutter::OperationMode.Programming, rect( x, y, x + w, y + h ) );
11 x = x + w + Cutter::Spacing.x;
12 SpeedWindow.ChangeNode( Cutter::OperationMode.Programming, rect( x, y, x + w, y + h ) );
13
14 /* second row of windows */
15 w = Cutter::Border.x;
16 y = y + h + Cutter::Spacing.y;
17 FormatWindow.ChangeNode( Cutter::OperationMode.Programming, rect( x, y, x + w, y + h ) );
18 x = x + w + Cutter::Spacing.x;
19 BatchWindow.ChangeNode( Cutter::OperationMode.Programming, rect( x, y, x + w, y + h ) );
```

Version 8.00
26 September 2016

Authors: Dipl. Ing. Paul Banach, Dipl. Ing. Manfred Schweyer

Release Notes

The following chapters contain a complete version history of Embedded Wizard and describe all news and changes. Please read carefully.



The following versions have been released:

- Embedded Wizard V8.00 (preview release)
- Embedded Wizard V7.10
- Embedded Wizard V7.02 (preview release)
- Embedded Wizard V7.00 (preview release)
- Embedded Wizard V6.60
- Embedded Wizard V6.51
- Embedded Wizard V6.50
- Embedded Wizard V6.41
- Embedded Wizard V6.40
- Embedded Wizard V6.30
- Embedded Wizard V6.20
- Embedded Wizard V6.10
- Embedded Wizard V6.00
- Embedded Wizard V5.40
- Embedded Wizard V5.30
- Embedded Wizard V5.20
- Embedded Wizard V5.10
- Embedded Wizard V5.00
- Embedded Wizard V4.40
- Embedded Wizard V4.30
- Embedded Wizard V4.20
- Embedded Wizard V4.11
- Embedded Wizard V4.10
- Embedded Wizard V4.01
- Embedded Wizard V4.00
- Embedded Wizard V3.30
- Embedded Wizard V3.20
- Embedded Wizard V3.00
- Embedded Wizard V2.30
- Embedded Wizard V2.20

- Embedded Wizard V2.10
- Embedded Wizard V2.00
- Embedded Wizard V1.01
- Embedded Wizard V1.00
- Embedded Wizard V0.92
- Embedded Wizard V0.91

Embedded Wizard V8.00 (preview release)

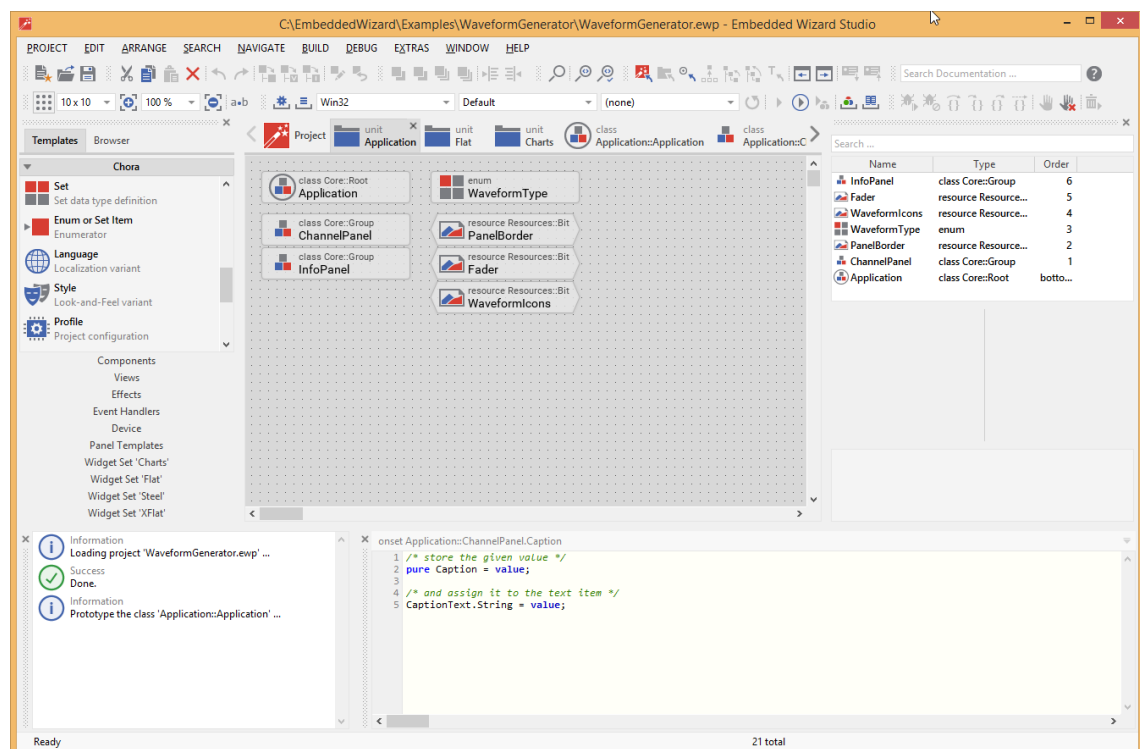
This version is a preview release. This means, that all new features and improvements are completed and well tested – just the documentation is not completed. Please note, the included user manuals and the tutorial documentation are not yet updated.

Version V8.00 contains the following changes and improvements:

Redesign of Embedded Wizard Studio

We are happy to present a new look and feel for Embedded Wizard Studio!

With the new version V8.00 we introduce a completely new user interface with a modern and flat appearance.



All program icons and all brick symbols have been redesigned in order to have a modern face and to be more intuitive. The entire application and all dialogs have a flat and homogeneous appearance.

The toolbars have been improved, useful menu shortcuts were added and the tooltips are enhanced. The handling of the dockable toolbars and windows has been simplified in order to avoid accidentally lost toolbars.

The Code Editor has an improved syntax highlighting with a better readability and shows line numbers.

The outdated feature 'AspectRatio' has been removed.

Support of HiDPI Displays

Embedded Wizard Studio is now ready for HiDPI displays. The entire user interface of Embedded Wizard will adapt automatically according the current Windows display DPI settings.

Additionally, the list of zoom factors (used to zoom the Composer content) has been enhanced (from 25% to 800%). The user can select the appropriate zoom factor more convenient according the current DPI setting. 100% zoom means 1 pixel in the GUI corresponds to 1 pixel on the screen.

The workspace settings for different DPIs are saved now separately within the Registry to avoid conflicts when the DPI settings are changed.

Documentation System

The entire Embedded Wizard documentation is going to be reworked. As a result, the different separate documents (Embedded Wizard User Manual, Chora Reference Manual, Mosaic Reference Manual, Tutorial, HowTo-Documents,...) will be completely reworked and integrated into one all-embracing knowledge base.

The new documentation system is available on <http://doc.embedded-wizard.de> and will be enhanced continually.

The screenshot shows the Embedded Wizard documentation interface. At the top left is the Embedded Wizard logo. A search bar is located at the top right. On the left side, there is a navigation menu with the following items: Welcome to Embedded Wizard, Basic concepts, Working with Embedded Wizard, Embedded Wizard Studio, Views, Event handlers, Animation effects, Project members (highlighted), Profile, Macro, Language (highlighted), Style, Unit, Inline code, Enum, Set, Enum/Set Item, Constant, Constant Variant, Bitmap resource, Bitmap resource variant, Font resource, Font resource variant, Class, Class variant, Autoobject, Autoobject variant, Variable, Array, Property, and Embedded objects. The main content area is titled 'Project members: Language'. It contains the text: 'With this definition you can determine all languages available for the localization of your application. For example, if the end-user should be able to switch the GUI between English, German and Spanish you will need three corresponding language members in your project. Language members are represented in **Composer** by following bricks:'. Below this text is a diagram of a 'language' brick with the text 'Language_Name'. A note states: 'Please note, language members can exist only at the root level of the project.' Below the note is the heading 'Add new language' followed by a list of four steps: 1. First switch to the Composer containing the project main page. 2. Then, in the Gallery window, ensure that the folder Chora is opened. 3. Look in the folder for the template named Language. 4. With the mouse, select the template and drag it into the Composer. 5. Drop the template within the Composer. At the bottom of the screenshot is a screenshot of the Embedded Wizard software interface. It shows a 'Templates' browser window with a list of templates including 'Set', 'Enum or Set Item', and 'Language'. A red arrow labeled '1' points to the 'Project' window in the background. A red arrow labeled '2' points from the 'Language' template in the browser to the 'Language' brick in the 'Composer' window.

The old documents 'QuickTour', 'Embedded Wizard User Manual' and 'Tutorial' are still available until the documentation system is not fully completed.

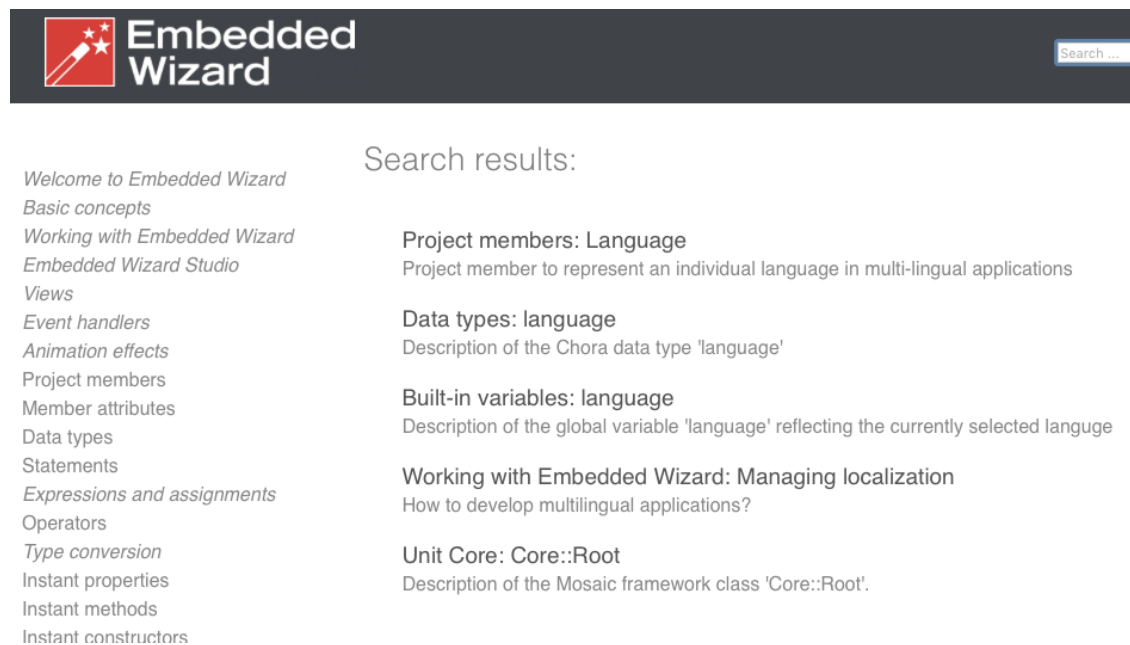
Please note: Since this version is a preview release, only parts of the help documents are available.

Context sensitive help

Embedded Wizard Studio supports now a context sensitive help. For this purpose a new documentation system is accessed.

Whenever a user presses the key 'F1', Embedded Wizard analyzes the current context (which item is selected and what is currently focused) and creates a search request for the documentation system.

As a result, a browser window is opened to present the user a list of related documents.



Embedded Wizard

Welcome to Embedded Wizard
Basic concepts
Working with Embedded Wizard
Embedded Wizard Studio
Views
Event handlers
Animation effects
Project members
Member attributes
Data types
Statements
Expressions and assignments
Operators
Type conversion
Instant properties
Instant methods
Instant constructors

Search results:

Project members: Language
Project member to represent an individual language in multi-lingual applications

Data types: language
Description of the Chora data type 'language'

Built-in variables: language
Description of the global variable 'language' reflecting the currently selected language

Working with Embedded Wizard: Managing localization
How to develop multilingual applications?

Unit Core: Core::Root
Description of the Mosaic framework class 'Core::Root'.

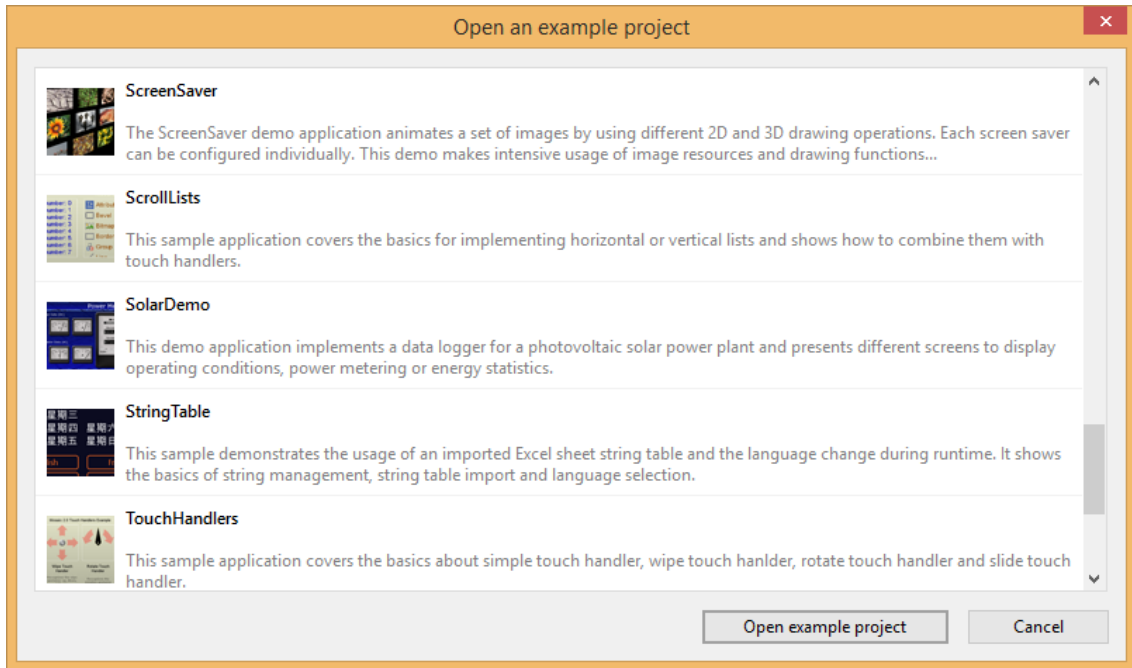
Please note: Since this version is a preview release, only parts of the help documents are available.

'Open Example'

Embedded Wizard Studio contains a new menu item to have a quick access to the provided examples.

The entire task is covered within a single 'Open Example' dialog.

In the dialog the user can select the desired sample project from a list of prepared / installed examples:



New Sample: Watches

A couple of analog watches have been implemented – they can be seen within the new 'Watches' sample.



New Setup System

Embedded Wizard and all Platform Packages are now installed by using a new setup system. Each setup is provided as Microsoft software installation (*.msi). Additionally, all setups are digitally signed to confirm the software manufacturer and to guarantee that the software is not modified.

New color formats RGB565 and RGB888

The set of supported basic color formats has been extended by the new formats called RGB565 and RGB888.

In this case, the screen color format fits exactly to the physical color format of the display / framebuffer, although Embedded Wizard still operates internally with RGBA8888 color format in order to support alpha blending and highest UI quality.

This means, the RGB565 Platform Package works internally with two color formats: a RGB565 screen color format and a RGBA8888 native color format.

Similar, the RGB888 Platform Package supports two color formats: a RGB888 screen color format and a RGBA8888 native color format.

With this new color format Embedded Wizard supports now the following configurable color formats: RGBA8888, RGB888, RGBA4444, RGB565, RGB565A8, RGB555A8, LumA44, Index8, YUVA8888

Support for 64-bit systems

The new instant data type '**handle**' is added to the programming language Chora. The data type '**handle**' represents a target specific pointer, etc. important for 64 bit systems.

The Prototyper, the Debugger and the Code Generators have been adapted to support the new '**handle**' type.

The Mosaic class library has been reworked in order to use '**handle**' for all references to target specific resources.

Runtime Environment and Graphics Engine are reworked to work on 64-bit targets by using the 64-bit model LP64.

The OpenGL and OpenGL ES 2.0 based Platform Packages are reworked to operate correctly on 64-bit.

The iOS application components are adapted to latest XCode and iOS version and support for 64-bit is added.

The OSX application components are adapted to latest XCode and OSX version. Support for 64-bit is added as well.

Remark: This 64-bit enhancement does not include int64 data types.

Improvements

Additionally, the following changes and improvements have been done:

Mosaic

- With Embedded Wizard V8.00 the old class library 'Mosaic 1.0' is no more supported. As a result, Mosaic20 is renamed to Mosaic, Templates20 is renamed to Templates and Examples20 is renamed to Examples.
- The class `Core::Root` contains now the method `GetFPS()` - helpful to measure the current frames per seconds of an UI animation.
- The set of timing functions within effect classes have been enhanced. A couple of new fancy timing functions are added to implement attractive effects - similarly to the easings of jQuery.

Bug Fixes

The following bugs have been solved:

Graphics Engine

- Bug fix in `EwDrawText()` function when the text ends with glyphs having negative x-offset. This caused the text to appear clipped at its right edge. Accordingly the function `EwGetTextExtent()` is adapted to calculate correctly the extent area with text that contains glyphs having negative x-offset at the end.

Inspector

- The Members List of the inspector is improved to not sort new members at the bottom end of the list. The members are sorted now accordingly to their Z-order.

Embedded Wizard V7.10

Version 7.10 contains updated documentation "Embedded Wizard User Manual", "Chora User Manual" and "Mosaic 2.0 Tutorial".

Embedded Wizard V7.02 (preview release)

This version is a preview release. Please note, the included user manuals and the tutorial documentation are not yet updated.

Version V7.02 contains the following changes and improvements:

Evaluation Edition

With the new version 7.02 we introduced a new Evaluation Edition of Embedded Wizard. This edition is intended primarily for test and evaluation purpose.

Unlike the old DEMO version, the evaluation edition allows you to generate code for your particular target system as well as to save modifications made on your project.

On the other hand, the evaluation edition is limited regarding the maximum size of your project. In other words, you can enjoy the entire functionality of Embedded Wizard for free – as long as the complexity of your project doesn't exceed predetermined limits.

The calculation of project complexity is based on the number of project members, implemented methods, code lines, etc. For your convenience Embedded Wizard shows the current project complexity as percent value in the Log window every time you save the project.

For more details regarding the evaluation edition please visit: <http://www.embedded-wizard.de>.

New color format LumA44

The set of supported basic color formats has been extended by the new format called LumA44. With this color format every native pixel is composed of a 4-bit luminance and 4-bit opacity (alpha) values. Like other basic color formats the order of the color channels and the pre-multiplication with the alpha value are configurable.

The new color format is intended for all kinds of gray-scale and monochrome displays. In particular it is appropriate for applications to be designed for e-ink paper devices.

With this new color format Embedded Wizard supports now following configurable native (frame-buffer) formats:

RGBA8888, RGBA4444, RGB565A8, RGB555A8

LumA44, Index8, YUVA8888

If you are interested please ask for the appropriate Platform Package.

Configuration of display orientation

With the new version 7.02 Embedded Wizard provides a possibility to configure whether bitmap resources should be generated in rotated orientation.

In the usual (not rotated) case the origin of the frame-buffer does correspond to the top-left corner of the display. Your product design, however, may require the display to be installed in any other orientation, e.g. a landscape display can be installed in the portrait position by rotating it by 90 degrees.

In such case you design the GUI application in the desired portrait format and you inform the Embedded Wizard about the real orientation of the display in the product. Embedded Wizard will then apply the necessary rotation steps on all bitmap resources when these are generated.

Using this option allows the graphics hardware in the target system to access the bitmaps without any intermediate rotation steps. The resulting performance will correspond to the system without any rotation.

If available for your particular Platform Package, the new option appears automatically in the corresponding profile member. The option is called `ScreenOrientation` and can assume one of the values `Normal`, `Rotated_90`, `Rotated_180` or `Rotated_270`.

Configuration for the format of bitmap resources

With the new version 7.02 Embedded Wizard provides a possibility to configure whether bitmap resources should be generated in the compressed format or uncompressed in the format valid for the particular target system.

This option is available for target system allowing the bitmaps to be accessed directly from the ROM code. In this manner no intermediate steps to allocate the video memory and decompress the bitmap are necessary. The RAM footprint is significantly reduced. On the other hand the ROM footprint is increased.

This option is intended primarily for low-end systems providing few RAM (e.g. 128kB). Target systems with sufficient amount of main memory should continue using the compressed bitmap format.

If available for your particular Platform Package, the new option appears automatically in the corresponding profile member. The option is called `FormatOfBitmapResources` and can assume either the value `Compressed` or `DirectAccess`.

Configuration for the format of string constants

With the new version 7.02 Embedded Wizard provides a possibility to configure whether string constants should be generated in the compressed format or uncompressed.

This option is available for target system allowing the strings to be accessed directly from the ROM code. In this manner no intermediate steps to allocate the memory and decompress the string block are necessary. The RAM footprint is reduced. On the other hand the ROM footprint is increased.

This option is intended primarily for low-end systems providing few RAM (e.g. 128kB). Target systems with sufficient amount of main memory should continue using the compressed string format.

If available for your particular Platform Package, the new option appears automatically in the corresponding profile member. The option is called `FormatOfStringConstants` and can assume either the value `Compressed` or `DirectAccess`.

Bug Fixes

The following bugs have been solved:

Assistants

- The `FontRanges` assistant could cause Embedded Wizard to crash when the user tried to open the assistant for a `FontRanges` attribute containing a very long list of glyph codes.

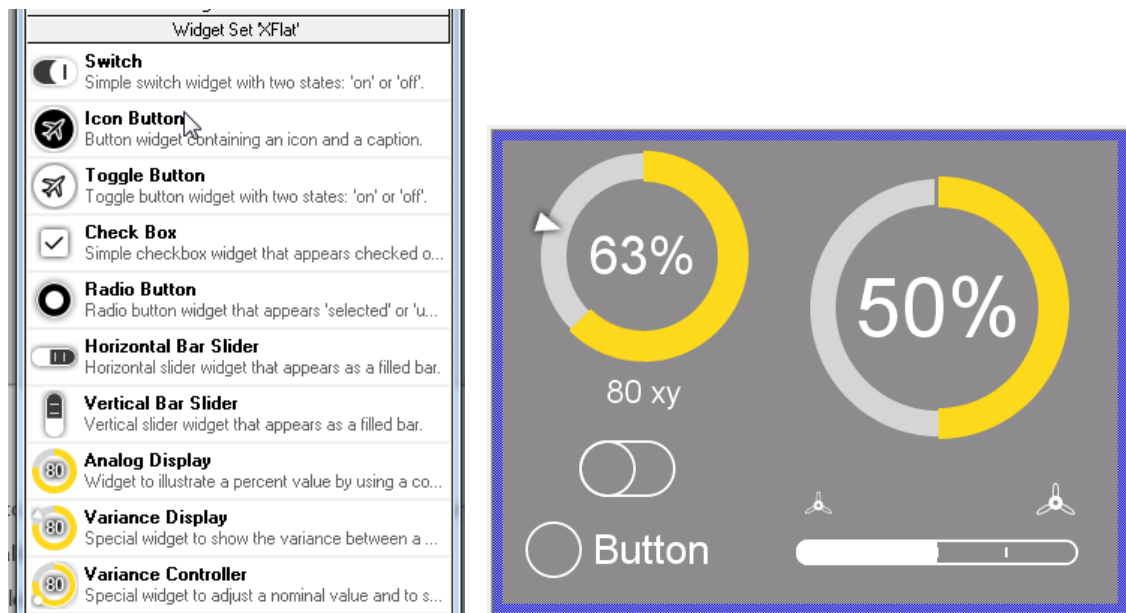
Embedded Wizard V7.00 (preview release)

This version is a preview release prepared for the trade fair 'Embedded World 2016'. Please note, the included user manuals and the tutorial documentation are not yet updated.

Version V7.00 contains the following changes and improvements:

New Widget Set 'XFlat'

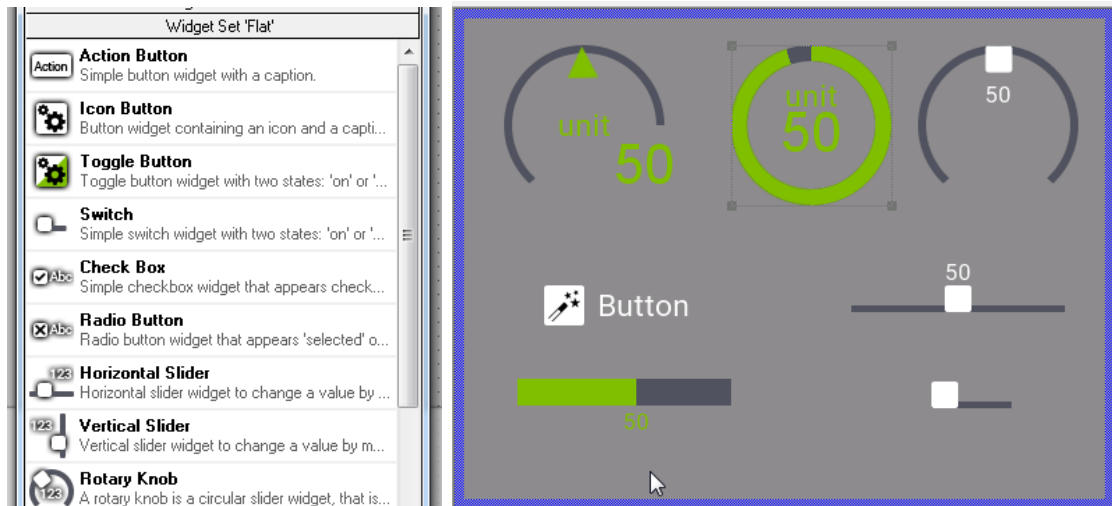
A new set of ready-to-use modern looking widgets is available now. You will find the widgets in the Gallery folder 'XFlat'. Simply drag the desired widget to your GUI component, arrange it and configure its properties. The following images show the templates in the Gallery and some of the new widgets in action:



You can use the widgets in your Embedded Wizard projects without any license limitations.

New Widget Set 'Flat'

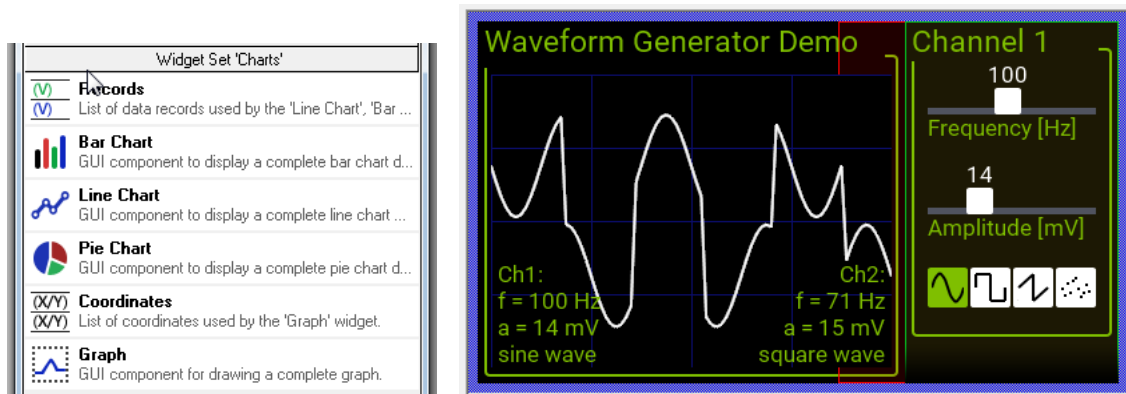
A new set of ready-to-use modern looking widgets is available now. You will find the widgets in the Gallery folder 'Flat'. Simply drag the desired widget to your GUI component, arrange it and configure its properties. The following images show the templates in the Gallery and some of the new widgets in action:



You can use the widgets in your Embedded Wizard projects without any license limitations.

New Widget Set 'Charts'

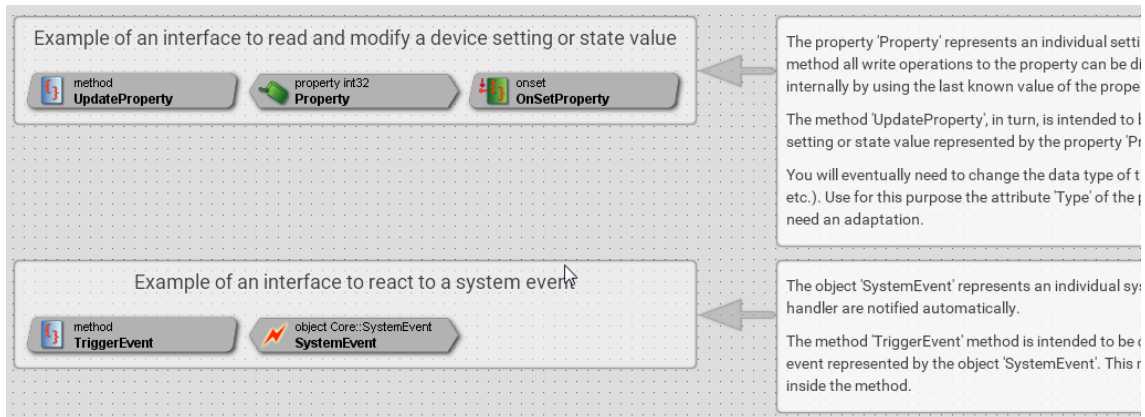
A new set of ready-to-use chart specific widgets is available now. You will find the widgets in the Gallery folder 'Charts'. Simply drag the desired widget to your GUI component, arrange it and configure its properties. Of course you will also need to specify the data to show in the chart. The following images show the templates in the Gallery and some of the new widgets in action:



Annotations

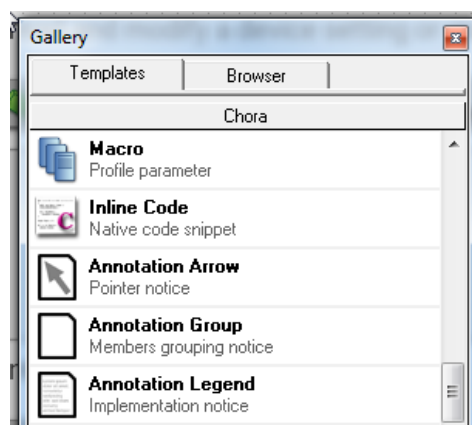
With annotations you are able now to add additional description to the content you are editing in the Composer. This should allow you to simply document your implementation. On the other hand the annotations are a new tool we intend to use in order to describe examples, project templates, and much more.

There are three different annotation types: 'Legend', 'Group' and 'Arrow'. With Legend you can simply leave some description within the Composer area. With Group you can visually group together and name project members. With Arrow you can simply point to something. The following screenshot demonstrates the three annotations used within the Composer:



Annotations are part of the Chora language. Anyway they have no function in the resulting application. You can add as many annotations as you want to every Composer you are editing your application members. If you are using templates or examples you can also select and delete the provided annotations without worrying to lose any application function.

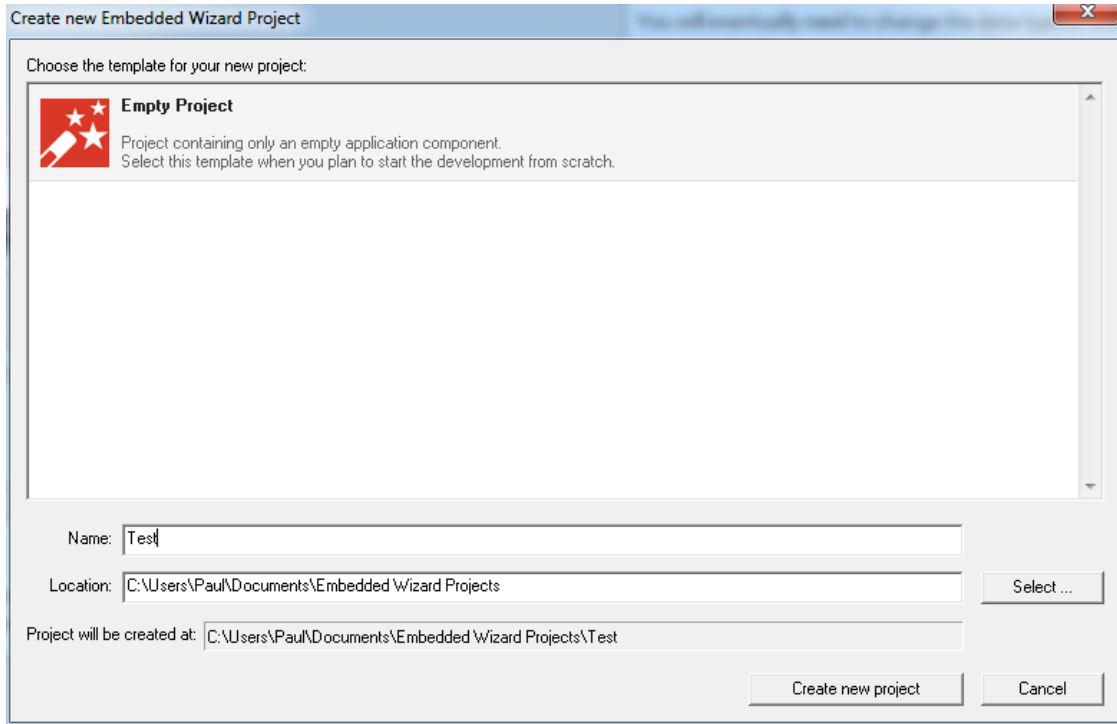
To add a new annotation use the templates from the Gallery folder 'Chora':



Creation of New Projects

We redesigned completely the parts of Embedded Wizard affecting the new project creation. The entire task is covered within a single 'New Project' dialog.

In the dialog the user can select the desired project type from a list of prepared project template. The dialog provides also a convenient way to enter the new project name and the location where to create the project:



Please note, new projects are created consequently within individual subdirectories now. You will need to explicitly confirm the creation of a project within an already existing folder. This should avoid multiple projects to be stored within the same directory.

Target Device Integration

To simplify the integration of Embedded Wizard designed GUI applications with underlying devices the Gallery provides now a new set of prepared templates:



With these templates you can add to your project new components to act as interfaces to the target device, to receive system generated events and more.

Improvements

The following improvements have been implemented:

Composer

- When new members are added to the Composer (e.g. by dragging them from the Gallery or simply by Copy/Paste) Embedded Wizard evaluates these members and if necessary completes automatically the project with units required due to dependencies caused by the new members.
- Copy/Paste of views together with some related methods or variables have been improved. The target class is reloaded automatically in such case to avoid eventual confusing error messages because of incomplete class definition.

Gallery

- The Gallery has been redesigned in order to show for every template a short description row. The description appears in the second text row of the template.
- Tooltip functionality has been added to the Gallery in order to show more detailed description of a template when the user remains for a while with the mouse pointer over it.
- The 'Rename Template' dialog has been redesigned. The new version allows the user to enter besides its name also the description for the template. In this manner the user can document its own created templates.
- All templates in the Gallery are reworked and usefully adapted. Some templates are inline documented by the feature 'Annotations'. Description in every template should help you to understand better for what the template does exist. The description appears in a tooltip when you remain with the mouse pointer over the template for a while.
- The sorting order of Gallery folders and templates are adapted to better group similar items together. Previously all items were sorted by their name. It should help you to find templates much faster.

- The original set of 'Steel Widget' component templates has been removed from the Gallery. The corresponding components are now stored directly within a 'Steel' unit. Instead of component templates, the gallery contains ready-to-use widget for every Steel widget. With this modification you can simply drag a Steel widget from the Gallery to your GUI component, arrange it and configure its properties.

Inspector

- All 'file open' and 'select folder' Inspector Assistants are adapted to use now the actual (Explorer-like) Windows file dialogs. This simplifies significantly the access to the user own directory.
- The 'object selection' Inspector Assistant has been adapted to show also all objects embedded within `autoobjects`. This is just convenient when you use `autoobjects` a lot.
- When new members are added to the Inspector (e.g. by dragging them from the Gallery or simply by Copy/Paste) Embedded Wizard evaluates these members and if necessary completes automatically the project with units required due to dependencies caused by the new members.
- Copy/Paste of views together with some related methods or variables have been improved. The target class is reloaded automatically in such case to avoid eventual confusing error messages because of incomplete class definition.

Chora

- The default behavior of the statements `postsignal` and `idlesignal` has been changed. Now when the signal is delivered, the parameter sender refers to the object, which has requested the signal delivery as the last one.
- The statements `signal`, `postsignal` and `idlesignal` has been extended by an additional, optional sender parameter. This allows you to specify explicitly the value to pass as the sender to the target slot method.
- The set of Chora instant data types has been extended by the type `any`. `any` is used exclusively to build typeless references to properties regardless of the property actual data type definition.

Mosaic

- New class `Core::TaskQueue` and `Core::Task` added. These classes provide functionality to enqueue short jobs (task) to be executed accordingly their queue order (FIFO queue).

- New class `Core::SystemEvent` added. An object of this class represents a system event source – this means the source of events generated by the target device.
- New class `Core::SystemEventHandler` added. Objects of this class serve as handler for system related events. You can simply place the handler wherever you want in your project and connect it with an object of the class `Core::SystemEvent`.
- New class `Core::PropertyObserver` added. Objects of this class react to notifications generated when a property is modified. You can simply place the observer wherever you want in your project and connect it to the desired property.
- The unit Resources has been cleaned up. All image resources needed by the 'Steel' widget component templates have been moved to a separate 'Steel' unit. Old projects using such resources will be updated automatically to use the 'Steel' unit when such project is opened in Embedded Wizard 7.0.

Other

- The 'Project Open' dialog has been adapted to use now the actual (Explorer-like) Windows open file dialog. This simplifies significantly the access to the user own directory.

Bug Fixes

The following bugs have been solved:

Composer

- In seldom cases resizing or moving views within the Canvas area of the Composer could cause Embedded Wizard Studio to crash.
- Breakpoints set in very frequently executed Chora code lines (e.g. within a long running for-loop) could cause Embedded Wizard to crash when the code was executed in context of the Composer.
- Editing a component (or switching between Composers) while the Prototyper window is opened and lot of animations are executed in the Prototyper could cause strange error message to appear in the log (e.g. 'Internal error in Graphics Engine ...'). In seldom cases Embedded Wizard crashed.
- The functionality to select a member at the clicked position within the Composer has been corrected. Now the user can select lines much simpler.

Gallery

- After creating a new project or opening an existing one the Gallery window showed randomly one of its template folders. Now the Gallery shows always the first folder (with Chora templates) when you open a project.
- When the user tried to create a copy of a template within one of the preinstalled template folders (e.g. Chora), the new template was incorrectly named causing name conflicts with the existing, preinstalled templates.

Inspector

- The 'short info' in the lower area of the Inspector has been corrected to not show the special sign `@`. The sign `@` is relevant for the documentation generation only.

Other

- All dialogs are corrected to allow the `Ctrl+A` shortcuts for the selection of text when the dialog contains a text edit field.
- The 'Pause' function to break an application running in an endless loop worked unreliable. In particular when a lot of animations were running in the Prototyper it was difficult to close the Prototyper window. Thereupon we decided to completely redesign this function.

The 'Pause' function remains now disabled for the first 3.5 seconds of continuous running code. After this time is elapsed, the simple clicking on title bar of one of Embedded Wizard windows or double clicking within one of the windows causes the current executed Chora code to pause. So you can recover the control over your application in a very simple manner. Once paused you can investigate the reason for the long code execution.

Embedded Wizard V6.60

Version V6.60 contains the following changes and improvements:

Bug Fixes

The following bugs have been solved:

Code Generation

- In case of a class variant the code generated for a `super()` call have under seldom circumstances missed a typecast to satisfy the C Compiler. The necessary typecast is generated now correctly.
- The optimizer stage of the code generation was inclined to exclude a class of an autoobject from being generated if no other of the project members did reference this class. This could produce erroneous C code. The problem is solved now.

Embedded Wizard Studio

- Several problems within Embedded Wizard Studio have been fixed to improve the overall stability.

Mosaic

- The redraw functionality of `Core::Group` was inefficient when the buffering mode (property `Buffered` is `true`) has been enabled for the group and the target system was configured to perform entire screen updates only.

The Mosaic caused is such cases all buffered groups to completely redraw their content. Now, only the modified (dirty) areas of a buffered group are redrawn even if the target system forces the Mosaic to perform the full screen update of the frame-buffer.

- A touch handler component was not able to deflect the current touch interaction to itself.

When processing touch events, a touch handler can ask the application to find another handler, which from now should continue with the event processing. This find-operation, however, ignored the original touch handler itself preventing the handler from being able to remain the active touch handler when no better candidate was available.

The transition is controlled by the both methods `DeflectCursor()` and `RetargetCursor()` implemented in the class `Core::Root`.

Improvements

The following improvements have been implemented:

Code Generation

- The layout of generated classes has been changed to arrange the 64, 32, 16 and 8-bit variables in groups accordingly to this order. This adaptation avoids eventual misalignment of generated entities on a target systems where the C compiler doesn't take care of this aspect leading the application in the target to raise a bus error.

Mosaic

- The component `Core::SlideTouchHandler` is improved to automatically speed up the slide animation when the handler is configured to stop only on predetermined snap positions. Usually the slide handler performs animations with the speed resulting from the user slide gesture. In case of active snap positions, the component can increase the speed in order to reach the next snap position in a shorter time.
- The find algorithm in the method `FindViewInDirection()` available in the classes `Core::Group` and `Core::Outline` has been improved to be more selective when searching for a view in a predetermined direction.
- The limitation for the minimal effect duration has been reduced from 0.1 sec to 0.015 sec. This allows the GUI to perform animations shorter than 100 msec.

Chora

- The set of mathematical operations has been enhanced by the following new Chora functions:

```
float math_round( float aValue );  
float math_ceil( float aValue );  
float math_floor( float aValue );
```

These functions are useful for common mathematical calculations to round floating-point values. The new function correspond to the C function `round()`, `floor()` and `ceil()`. For more details see Chora User Manual.

Embedded Wizard Studio

- The menu Help contains now a new menu item 'Visit Ask Embedded Wizard support page'. User can select this item to open the 'Ask Embedded Wizard' web page in the web browser.

With the web page we intend to provide a new service where Embedded Wizard programmer can interchange, search for problem solutions and put new questions we can answer.

- The Copy/Paste function in Embedded Wizard has been improved. Now it tries to take in account the origin of the copied members when these are pasted within a composer window.
If necessary it adjusts automatically within the pasted content all affected global names to ensure that these fit the new target unit where the user intends to paste the members.
- The **Profile** Chora template has been corrected to use the highest optimization level per default.
- DUMP file functionality has been added to Embedded Wizard allowing it to protocol exceptional situations. In case of a fatal error raised within Embedded Wizard IDE, a DUMP file is created and the user is asked to send this file to TARA Systems allowing us to analyze and fix the cause of such seldom and unexpected crashes.

Embedded Wizard V6.51

Version V6.51 contains the following changes and improvements:

Mosaic 2.0 Bug Fixes

The following bugs have been solved:

- The dispatching of keyboard events to list items didn't work if the affected item was located outside the list's boundary. For example after selecting an item and then scrolling the list. In such case the list discarded the invisible items from its internal cache preventing them from being able to receive events.

Now, the class `Core::VerticalList` and `Core::HorizontalList` manage a temporal item instance to send the keyboard events if the selected item is currently not available in the internal list cache.

- The dispatching of keyboard events to keyboard handler didn't work if the `OnPress` property of the affected handler was initialized with the value `null`. In such case the handler ignored both the press and the release keyboard events.

Now, the class `Core::KeyPressHandler` handles keyboard events regardless of whether there is a slot method assigned to its `OnPress` or `OnRelease` property.

- Touch events were ignored in some circumstances. This issue arose with an optimization of how touch events are evaluated in Mosaic of Embedded Wizard 6.50. Older Embedded Wizard versions are not affected by this problem.

Now, the touch events are handled correctly.

- Update of list components could lead to an endless loop in the execution of the application if the list content has been scrolled out. This problem arose with an optimization of how list components are updated in Mosaic of Embedded Wizard 6.50. Older Embedded Wizard versions are not affected by this problem.

Now, the class `Core::VerticalList` and `Core::HorizontalList` manage correctly its update.

Embedded Wizard V6.50

Version V6.50 contains the following changes and improvements:

Mosaic 2.0 Improvements

- The implementation of the class `Core::SlideTouchListener` has been reworked. Primarily the new version supports the definition of snap positions where the scrolled content should stop at the end of the scroll animation. In this manner you can avoid, that the animation stops in the middle of a list item, text row, menu item, etc.

To configure the distance between the consecutive snap positions the new property `SnapNext` is available. With the both properties `SnapFirst` and `SnapLast` a differing snap distance for the first and the last snap position can be specified.

The further adaptation of the class `Core::SlideTouchListener` is the property `SpeedLimit`. With this property you can specify the maximum speed in pixel the associated content can be scrolled by user touch gestures.

Additionally the integration of the `SlideTouchListener` with other components (list, text, outline, image, etc.) has been improved. The previous implementation impeded the usage of the properties `OnStart`, `OnSlide` and `OnEnd` when the `SlideTouchListener` was combined with another component (list, text, outline, image, etc.).

- The class `Views::Text` has been extended by the property `RowDistance`. With this new property you can explicitly specify the distance between consecutive text rows. Per default this property is 0, which means that the row distance is derived from the metrics of the used font resource.
- The animation classes `ColorEffect`, `FloatEffect`, `Int32Effect`, `PointEffect` and `RectEffect` from the unit `Effects` have been extended to allow the specification of the playback direction. The playback direction is controlled by the new property `Reversed`. You can set this property either before the effect is started or alternate it while an effect is running.

Changing this property while the effect is running will cause the playback direction to be changed immediately. In this case all already traversed animation cycles are rewind till the origin of the effect is reached.

If this property is set `true` before the effect has been started then the effect behaves as if it has been configured with exchanged start and end values and a mirrored timing function.

- The implementation of the class `Core::VerticalList` has been reworked. The item loading and the caching algorithms have been optimized. Furthermore optional padding above the first and below the last item can be specified now.

The padding is an area you can reserve for separate list header, footer or another decoration components. The both padding values are controlled by the properties `PaddingTop` and `PaddingBottom`.

To maintain the position of the additional decoration components synchronously with the current scroll position within the list the new property `onUpdate` has been added. You can assign a slot method to this property and the slot method is called each time the list is rearranged or scrolled. Within the slot method you can update the position of all decoration components.

- The implementation of the class `Core::HorizontalList` has been reworked. The item loading and the caching algorithms have been optimized. Furthermore optional padding left to the first and right to the last item can be specified now.

The padding is an area you can reserve for separate list header, footer or another decoration components. The both padding values are controlled by the properties `PaddingLeft` and `PaddingRight`.

To maintain the position of the additional decoration components synchronously with the current scroll position within the list the new property `onUpdate` has been added. You can assign a slot method to this property and the slot method is called each time the list is rearranged or scrolled. Within the slot method you can update the position of all decoration components.

Mouse/Touch Event Dispatching

- The Mosaic 2.0 method `Core::View.CursorHitTest()` has been extended by an additional parameter `aFinger`. This parameter identifies the mouse button or in case of the multi-touch screen the finger which has generated the event leading to the invocation of the method `CursorHitTest()`. Now the cursor hit test can involve the pressed mouse button or the finger number into the decision whether to accept or not the interaction.
- `RotateTouchHandler`, `SimpleTouchHandler`, `SlideTouchHandler` and `WipeTouchHandler` components from the Mosaic 2.0 unit `Core` have been extended by the new property `LimitToFinger`. Now you can explicitly restrict the handler to react to mouse/touch events generated only by the particular mouse button or finger. Per default this property is set to the value `-1` what means that no limitation should take place.

- The dispatching of mouse/touch events while an active animation lock has been redesigned. Originally all mouse/touch events were ignored while the animation lock was active. This could lead the application to loose mouse/touch release events when the user has touched the screen before the animation lock begun and released the screen again while the lock was active. As consequence the affected GUI components remained in the pressed state.

Now, the animation lock affects only events for interactions started while the lock is active. When the user has begun the touch/mouse interaction before the animation lock has been activated, the associated events are still processed in the application – even during an active animation lock.

The animation lock is controlled by the **BeginAnimation()** and the **EndAnimation()** methods of the class **Core::Root**.

Key Event Dispatching

- The Mosaic 2.0 class **Core::Root** has been extended by the new method **DriveKeyboardHitting()**. From now, this method is the correct entry point to integrate the GUI application with your OS or target (similarly to the method **DriveMultiTouchHitting()**, etc.)

Originally the keyboard events were fed by using the method **DispatchEvent()**. For compatibility reasons with older projects and target systems this approach is still supported. The new approach, however, treats the events in a more intelligent way.

The method **DriveKeyboardHitting()** follows now the 'grab cycle' concept. Similarly to the approach how mouse and touch events are dispatched, the key release event is delivered directly to the component, which has previously reacted to the corresponding key down event. In other words, when a GUI component handled a key down event, it is ensured that this component will also receive the corresponding key release event.

The delivery of key release events doesn't succeed along the focus path as this is the case for the key down events. The original implementation treated the key down and key up events as two independent kinds of events. The new implementation treats these as a couple.

The new key event feed method **DriveKeyboardHitting()** is used currently in the Prototyper and the WebGL target system. All other platform package templates are still based on the older approach with **DispatchEvent()** method to avoid 'C' compiler errors and to ensure the best possible compatibility with already existing GUI applications. With further Embedded Wizard releases we will adapt the platform package templates accordingly.

- The Prototyper has been adapted to feed the tested application with key up events. In the previous version only key down events were handled and delivered to the application.
- The platform package templates for the targets Win32, WinCE, OSX and WebGL have been adapted to react and deliver the key up events to the hosted GUI application.
- The Mosaic 2.0 enumeration `Core::KeyCode` has been augmented to provide better support for common keyboards. Originally the enumeration was limited to key codes typical for consumer electronic devices (e.g. key `Play` or `ChannelUp`).

Now the enumeration covers the frequently used key shortcuts, like `CtrlKeyS`, `AltKey8` or `CtrlShiftRight`, etc. These are very useful for industry applications controlled by a keyboard or for any kind of Win32, OSX, Linux or WebGL desktop application.

The Prototyper has been adapted to feed the application with the new key codes. Platform package templates, in contrast, are still not updated to not affect the compatibility with older versions. With further Embedded Wizard releases we will adapt the platform package templates accordingly.

- The dispatching of key events while an active animation lock has been redesigned. Originally all key events were ignored while the animation lock was active. This could lead the application to loose key release events when the user has pressed a key before the animation lock begun and released the key again while the lock was active. As consequence the affected GUI components remained in the pressed state.

Now, the animation lock affects only events for interactions started while the lock is active. When the user has begun the keyboard interaction before the animation lock has been activated, the associated events are still processed in the application – even during an active animation lock.

The animation lock is controlled by the `BeginAnimation()` and the `EndAnimation()` methods of the class `Core::Root`.

Font Resources and Font Converter

- The Graphics Engine has been reworked to allow font resources with height up to 256 Pixel. In the preceding version the maximum font height was limited to 127 Pixel.
- The Font Converters have been reworked to allow the generation of fonts with the height up to 256 Pixel.
- The font resource attribute `Ranges` has been improved to allow the user to specify a file from where the font resource converter should read the codes of font characters to involve into the code generation. Of course the character codes can still be specified directly in the attribute `Ranges`.

The file is a simple text file using the same syntax as defined for the attribute **Ranges**.

Support for UHD (Ultra High Definition)

- The Graphics Engine has been reworked to support displays with resolution up to 4096 x 4096 pixel.
- The Prototyper has been optimized to handle with large application contents. In particular the zoom function of in the Composer window has been redesigned.

Examples

- The following new examples have been added to Mosaic 2.0:

Mosaic3D

This example demonstrates how complex 3D scenes consisting of several objects are created.

MultiScreens

This example demonstrates how an application can be built of several components and how to implement touch gestures to switch between these components.

- With the version 6.50 we removed from the setup all Mosaic 1.0 examples. Mosaic 1.0 exists for compatibility with older projects only. Few of the old examples have been ported to the current version of Mosaic 2.0:

Aviation Demo

This example demonstrates how gauges and other complex indicating instruments are implemented.

Oscilloscope

This example demonstrates how to implement control panels.

Clock

This example demonstrates how to implement a 7-segment display.

Game

This example demonstrates how simple computer games are implemented.

Bug Fixes

The following bugs have been solved:

Embedded Wizard IDE

- The project save function verifies now, whether the unit files (***.EWU**) have been modified outside of Embedded Wizard. In such case a dialog window appears and the user has to decide how to proceed with the affected file.
This additional verification ensures, that unit files are not overwritten accidentally. This modification also affects the project auto save function.
- The naming of files within a unit configured as split (the unit's attribute `split` is `true`) is changed in order to avoid file name conflicts. Such conflicts could cause the project information to be lost if the unit contains members with names differing in their upper and lower case notation only.
Now in case of such conflict situation the affected file names are extended by unique appendix to make them different.
- The compatibility with the Windows version Vista and above has been improved to take in account the current Windows DPI settings. Now Embedded Wizard IDE appears scaled accordingly to these DPI settings.
- The compatibility with the Windows version Vista and above has been improved to function correctly with more than one monitor. Now all active monitors and their size as well as their arrangement is respected by Embedded Wizard main window and its dockable windows.
- In seldom cases the Prototyper could crash the Embedded Wizard IDE if during the creation phase of a Chora object an error is occurred. The problem didn't affect the applications running in the target system.
- In seldom cases a bitmap resource used in the project could crash Embedded Wizard IDE if the bitmap file was corrupted.
- The pause function in the Prototyper to break an application running in an endless loop failed in seldom cases. In such case it was not possible for the user to abort the running application. The entire Embedded Wizard IDE needed to be terminated explicitly. You can activate the pause function by clicking on the 'Close' button in the Prototyper window.
- In the Color Assistant window the text input fields where the user can enter the red, green, blue and alpha values have been corrected. The preceding version limited the entered text to two digits on Windows Vista and above.
- The maximal text length of the search pattern the user can enter in the dialog 'Search in the project for ...' has been increased. Now long search patterns can be entered.

Mosaic

- The method `Core::Root.RetargetCursor()` failed if it has been invoked with the root object itself as the potential new target for the cursor events. The root object and all nested components were simply ignored.
- The method `Core::Group.ObtainFocus()` failed if applied to a group having currently the modal state. Modal components become focused directly from the root object and not through the focus path.
- The method `Core::Group.InvalidateArea()` failed if used in a target system preferring full screen updates and there was a nested group with an active buffered mode (the property `Buffered` of the group was `true`). This could result in an incomplete screen redraw.

Platform Packages for target Win32 and WinCE

- The size of the viewer window corresponds now to the value stored in the attribute `ScreenSize`. This attribute is specified in the profile used for the code generation. The preceding version wrongly ignored this value and used instead the size of the application component.
- The usage of Win CE timers is fixed.

Documentation Generator

- The function 'Extract documentation' to generate from the project information an HTML documentation includes now also the `autoobject` project members.

User Manuals

- Minor errors have been corrected in the document 'Tutorial – Mosaic 2.0'.

Embedded Wizard V6.41

Version V6.41 contains the following changes and improvements:

Embedded Wizard Development

The complete development toolchain used for creating Embedded Wizard and the Platform Packages was updated. This helps us to maintain and develop the next generations of Embedded Wizard and to enable debugging on the latest Windows versions.

The feature set is identical to V6.40. Please see the corresponding release notes below.

Win32 Platform Packages

The old Win32 versions of the Graphics Engine (GE1.0) has been removed All Tara.Win32_GE10.<color format> platforms are no more supported.

Embedded Wizard V6.40

Version V6.40 contains the following changes and improvements:

WebGL Platform Package

With the Embedded Wizard 6.40 version the support for a new WebGL target platform has been added.

WebGL is a new web browser technology allowing an HTML page to dynamically render and display 2D and 3D contents. Technically seen WebGL does conform to the OpenGL ES 2.0 API. As its programming language JavaScript is used.

Thanks to this new adaptation it is now possible to use Embedded Wizard to create GUI applications optimized to run natively within a WebGL capable web browser. The benefits of this approach are:

- Significant reduction of platform dependence.
- Your GUI application can simply run within a web browser.
- Even already existing applications can run in a web browser.
- The application benefits from the WebGL hardware acceleration.
- The look-and-feel of the application is consistent over all web browsers.

All these aspects are covered by our new WebGL Platform Package. Its Code Generator translates the GUI application and its business logic in the corresponding JavaScript syntax. The WebGL Runtime Environment provides a common interface for the generated application. Additional code obfuscation makes difficult the reverse engineering of the generated JavaScript code.

The new WebGL Platform Package is available by Tara Systems. If you are interested in this new technology don't hesitate to contact our support team.

Multi-Touch Support

Embedded Wizard 6.40 supports now the development of multi-touch GUI applications. Following parts of the Mosaic 2.0 framework have been modified in order to support multi-touch events:

- The class `Core::Root` provides two additional methods `DriveMultiTouchHitting()` and `DriveMultiTouchMovement()`.

These methods are intended to feed the GUI application with touch events received from the underlying target system. You will never need to invoke these methods directly from your GUI application. Usually the method will be invoked in response to touch events received in the `main()` message loop from the target specific touch screen driver.

For more details see "Mosaic 2.0 User Manual".

- The class `Core::SimpleTouchHandler` has been extended by the property `EnableMultiTouch` and variable `Finger`.

The property `EnableMultiTouch` determines how the touch handler should behave when the user interacts with it by using more than one finger. If this property is `false` and the handler is actually involved in a user touch interaction then the handler will ignore all additional touch events. Such events can be thus handled by other handlers lying e.g. behind this handler. If this property is `true`, the handler will receive all affected touch events. In order to distinguish the multiple taps and touches the variable `Finger` can be evaluated.

The variable `Finger` stores the number of the finger associated with the current touch event. This can be useful if the handler is intended to handle multi-touch events. The fingers are numbered with values lying in the range 0 .. 9.

For more details see "Mosaic 2.0 User Manual".

- The following handler components: `Core::RotateTouchHandler`, `Core::SlideTouchHandler` and `Core::WipeTouchHandler` have been adapted to handle single touch events only. One handler can thus react to events generated by one finger at the same time only. Several handlers can, however, react simultaneously on events generated by several user interactions.

For more details see "Mosaic 2.0 User Manual".

Chora

The project member `profile` supports now two additional attributes:

- The attribute `ApplicationName` allows you to assign a name to the application and thus identify it within an environment executing simultaneously multiple Embedded Wizard applications.

The effect of this attribute depends on the used Code Generator. Usually the attribute determines a namespace for the affected application. The attribute is ignored if the generator doesn't support it.

- The attribute `Obfuscation` controls the format of the generated code. If this attribute is `true`, the Code Generator will try to compress and obfuscate the generated code. If this attribute is set `false`, the code is not modified.

The effect of this attribute depends on the used Code Generator. If the generator doesn't support the code obfuscation, then the attribute has no effect on the generated code.

For more details see "Chora User Manual".

Further Mosaic 2.0 Modifications

- The class `Core::Root` has been extended by the new method `DoesNeedUpdate()`. This method returns a value indicating whether there is a screen area, which should be redrawn. If there is no invalid area to redraw, `DoesNeedUpdate()` will return `false`. This new method is intended for screen update optimizations you implement within the `main()` message loop. You will never need to invoke this method directly from your GUI application. For more details see "Mosaic 2.0 User Manual".

Bug Fixes

The following bugs have been solved:

Mosaic

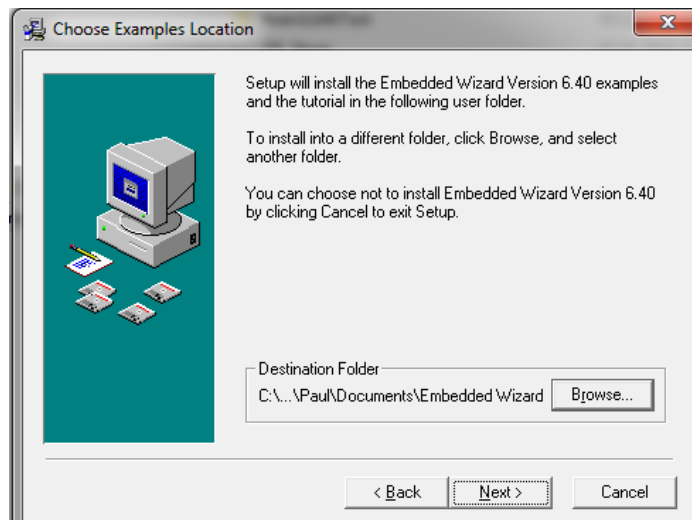
- The view `Views::WarpImage` has been modified to handle immediately the assignment of a new bitmap. In the previous version the update was performed after a short delay so that after assigning a new bitmap the view coordinates remained unchanged for this short period.
- The view `Views::WarpGroup` has been modified to handle immediately the assignment of a new group. In the previous version the update was performed after a short delay so that after assigning a new group the view coordinates remained unchanged for this short period.
- The component `Core::Outline` has been modified to handle correctly the displacement (scrolling or arrangement) of embedded views. In the previous version the displacement was performed always with the so-called 'fast reshape' mode. This could cause in seldom cases an incomplete screen redraw. Now the implementation takes in account whether the affected view supports the 'fast reshape' mode.

- Chora

- The implementation of the `parentthis` operator in the Embedded Wizard Prototyper has been corrected. In the previous version the operator delivered the value `null` when evaluated by an object embedded within a class variant. This erroneous behavior affected the prototyping environment only. Within the target system the operator behaved as expected.

Examples

- All examples and tutorials are installed now in the current user's `Documents` folder. You can determine the folder's name during the Embedded Wizard setup stage. See the screenshot below:



- Examples and tutorials containing Microsoft Visual C++ 6.0 projects have been updated to the Microsoft Developer Studio 2010. The support for the old Visual C++ 6.0 has been discontinued.

You can download the current Microsoft Developer Studio express version for free. Please visit <http://www.visualstudio.com>.

Documentation

- "Chora User Manual" and "Embedded Wizard User Manual" are delivered now as PDF files. In order to view these document you will need a PDF reader program, e.g. the Adobe Reader. Please visit <http://www.adobe.com>.
- Minor adaptation has been made to the document "Tutorial – Mosaic 2.0".

Embedded Wizard V6.30

Version V6.30 contains the following changes and improvements:

Core::Time

The Mosaic 2.0 class `Core::Time` has been improved: The property `Time` of the class `Core::Time` has been changed to from `int32` to `uint32`. With this modification the timer tick counter will overflow in year 2106 instead of year 2038.

Improvement of 'enum'

The representation of the 'enum' data type in the Chora programming language has been changed from 16 bit to 32 bit. Additionally, the usage of the 'enum' data types has been improved: Now, the programmer can assign any signed or unsigned 32 bit numeric value to the elements of an 'enum' definition.

Since Chora provides type cast operators which allows the conversion of 'enum' operators, the interpretation of the assigned 32 bit value depends on the used typecast (either `int32` or `uint32`).

Optimizations

The Code Generator of Embedded Wizard has been improved:

- `set.contains()` – The operation `set.contains()` which is used intensively by the Mosaic class library is now optimized by the code generator, in order to avoid unnecessary calls to the corresponding function within the Runtime Environment.
- Garbage Collector – The functions `EwMarkRef()`, `EwMarkSlot()` and `EwMarkObject()` are now implemented as macros. This improves the runtime of the Garbage Collector significantly.

Bug Fixes

The following bug has been solved:

Graphics Engine

In seldom cases, the Graphics Engine may cause a crash, when there are still pending graphics instructions for a bitmap, which has already been removed from the internal bitmap cache. This error happens only if the bitmap was deleted because the internal bitmap cache was full.

This problem was introduced with V6.20 and it is solved now.

Embedded Wizard V6.20

Version V6.20 contains the following changes and improvements:

Mosaic 2.0 Modifications

The Mosaic 2.0 class library has been optimized:

- **Draw() Method** – The method `Draw()` of the class `Core::View` and all derived classes has been enhanced by two additional parameters: `aOpacity` and `aBlend`. With these parameters the screen update can be optimized if semitransparent UI components are drawn. With this modification the allocation of temporary off-screen buffers for semi-transparent objects are avoided.

For more details see "Mosaic 2.0 User Manual".

- **Opacity Property** – The behavior of the property `Opacity` of the class `Core::Group` and all derived UI components has been changed. In previous versions a temporary off-screen buffer was allocated automatically, in case that the property `Opacity` was set to a value < 255 . Now, the `Opacity` value is propagated to the `Draw()` methods of all embedded UI objects. To achieve the previous behavior you have to set the property `Buffered` explicitly.

For more details see "Mosaic 2.0 User Manual".

- **AlphaBlended Property** – The behavior of the property `AlphaBlended` of the class `Core::Group` and all derived UI components has been changed. In previous versions a temporary off-screen buffer was allocated automatically, in case that the property `AlphaBlended` was set to `false`. Now, the `AlphaBlended` value is propagated to the `Draw()` methods of all embedded UI objects. To achieve the previous behavior you have to set the property `Buffered` explicitly.

For more details see "Mosaic 2.0 User Manual".

Applet Class

The new Mosaic 2.0 class `Views::Applet` implements an environment for the integration of external applications, already existing for the target system. For example an existing computer game, an image viewer or a teletext application written in 'C' may be integrated into the Embedded Wizard environment by deriving and customization of this `Views::Applet` class. This class is similar to the Mosaic 1.0 version.

For details see "Mosaic 2.0 User Manual".

Prototyper Window

The size and the position of the Prototyper Window is now stored and used as start value for the next prototyping session. The values are only kept as long as Embedded Wizard is running. If Embedded Wizard is restarted, default values are used to position the Prototyper Window.

Examples

The installation of Embedded Wizard contains two more examples:

Applet

This example demonstrate the usage of the class `Views::Applet` in order to integrate an external C application into an Embedded Wizard application.

HelpViewer

This example demonstrate the usage attributed strings and the corresponding Mosaic 2.0 classes `Views::AttrText` and `Graphics::AttrSet`.

Bug Fixes

The following bugs has been solved:

Views::Text

The Mosaic class `Views::Text` produced runtime warnings because of multiple postsignals. The problem should be solved now.

Bitmap Converter

If a bitmap resource contains undefined attributes `FileName` and `AlphaName`, the code generation could fail without any error message. The problem is solved now.

Navigation

The navigation shortcut to "Navigate to Unit" does not work if a Set is currently shown within the Composer window. The problem is solved now.

Code Generator

The Code Generator was adapted in order to avoid the generation of superfluous `_vthis` pointer variables. This should avoid 'C' compiler warnings.

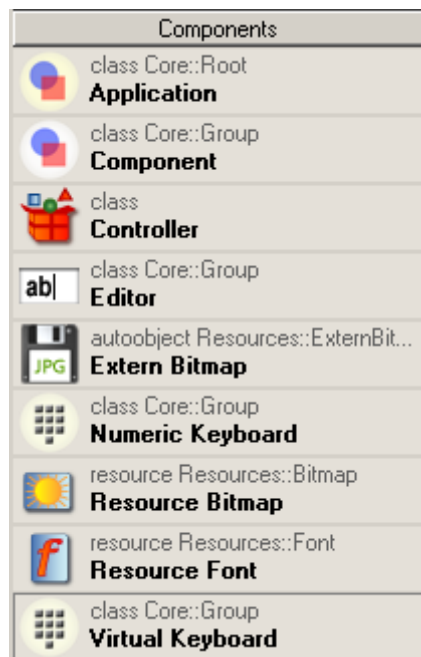
Embedded Wizard V6.10

Version V6.10 contains the following changes and improvements:

Templates

The set of delivered component templates has been enhanced. Within the Mosaic 2.0 Gallery folder 'Components', the following templates are now available:

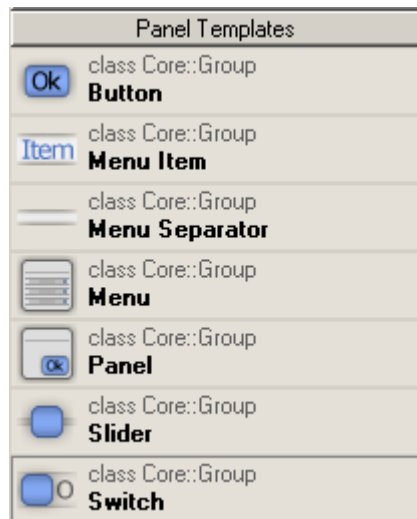
- Editor - This template implements a text editor component where the user can input and edit single-line or multiline text. The component supports keyboard and touch events to input and navigate within the text.
- Extern Bitmap - The class Resources::ExternBitmap provides the functionality to handle a bitmap loaded dynamically at the runtime from an extern image file. Bitmaps loaded in this manner can be used as regular bitmap resources. Please note, that this class provides only an interface for integrating your own image loader (e.g. PNG, JPG, GIF decoder).
- Numeric Keyboard - This template implements a numeric virtual keyboard GUI component.



Additionally the folder 'Panel Templates' has been added to the Gallery. It provides the following new templates:

- Panel - This template implements a simple panel component you can use to create your own message boxes or control panels.
- Button - This template implements a simple push button component.
- Slider - This template implements a simple slider component.

- Switch - This template implements a simple switch component.
- Menu - This template implements a GUI component you can use to create your own menus.
- Menu Item - This template implements a GUI component intended to be used within the menu described above.
- Menu Separator - This template implements a GUI component intended to be used as item separator within the menu described above.



All templates comes with a predefined look&feel. They are intended to be used as they are. However, you can adapt or modify these GUI component to your particular needs.

Mosaic 2.0 Enhancements

The class `Core::Group` has been extended by following methods:

- The method `HasViewState()` verifies whether a specified state is currently valid for the GUI component. It is useful e.g. if you need to test whether a component is really focused and thus it is able to receive user keyboard inputs, etc.
- The method `ObtainFocus()` establishes the focus path to the GUI component by switching the `Focus` properties of all superior `Owner` components. This ensures that the component may receive user keyboard inputs.

The class `Core::Root` has been extended by the following methods:

- The method `BeginAnimation()` deactivates temporarily the handling of keyboard and mouse/touch panel events. It is useful during animated screen transitions to avoid any interferences between the transition and user interactions.
- The method `EndAnimation()` reactivates the handling of keyboard and mouse/touch panel events.

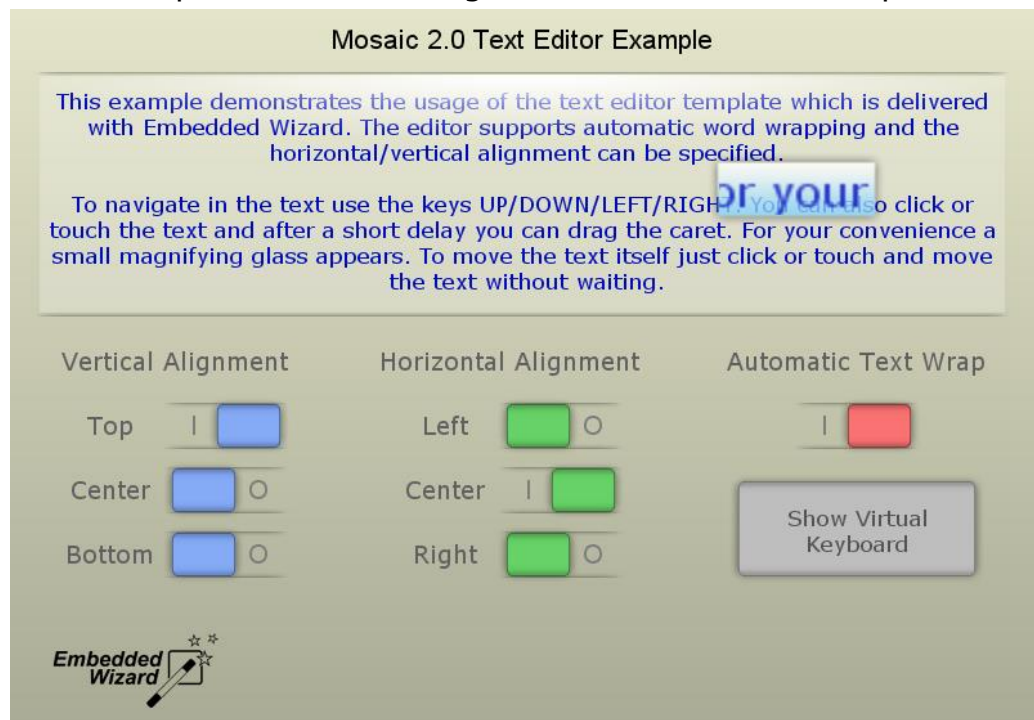
The class `Views::Text` supports beside the normal space sign also the so-called non-breakable space with the Unicode code `\xA0`. In case of justified text output the signs are stretched as usual space signs. However the non-breakable space is not used to automatically wrap text rows.

Examples

The set of Mosaic 2.0 examples has been extended:

Editor

This example shows the usage of the new 'Editor' template.



ExternBitmap

This example demonstrates the usage of the new class `Resources::ExternBitmap` and the integration of an extern bitmap loader.

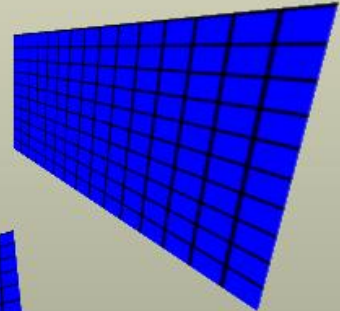
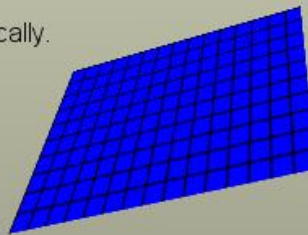
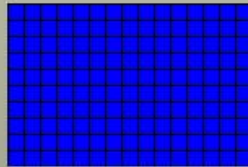
Extern Bitmap Example

This example demonstrates the usage of the class Resources::ExternBitmap and the implementation of an extern bitmap loader. An extern bitmap loader can be used to provide your GUI application with dynamically loaded images from an external image decoder (e.g. a PNG or JPG decoder).

In order to keep this example as simple as possible, we just fill the content of the extern bitmap with a color and a grid pattern. Please have a look into the file externbitmaploader.c which is part of this example.



A click on each button changes the property 'Name' of the extern bitmap and reloads the content by using the external bitmap loader. All views are updated automatically.



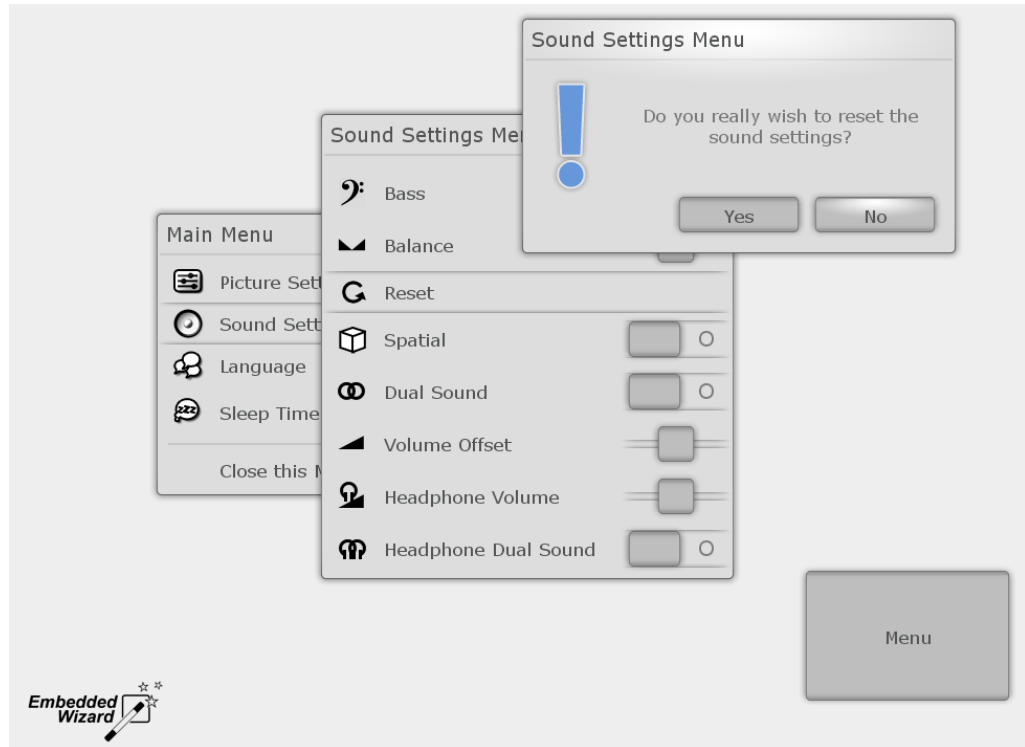
Lightweight3D

This example demonstrates the usage of lightweight 3D functionality of Embedded Wizard. Originally the example was developed for the Mosaic 1.0. Now we ported the example to Mosaic 2.0 and enhanced it by additional Mosaic 2.0 features:



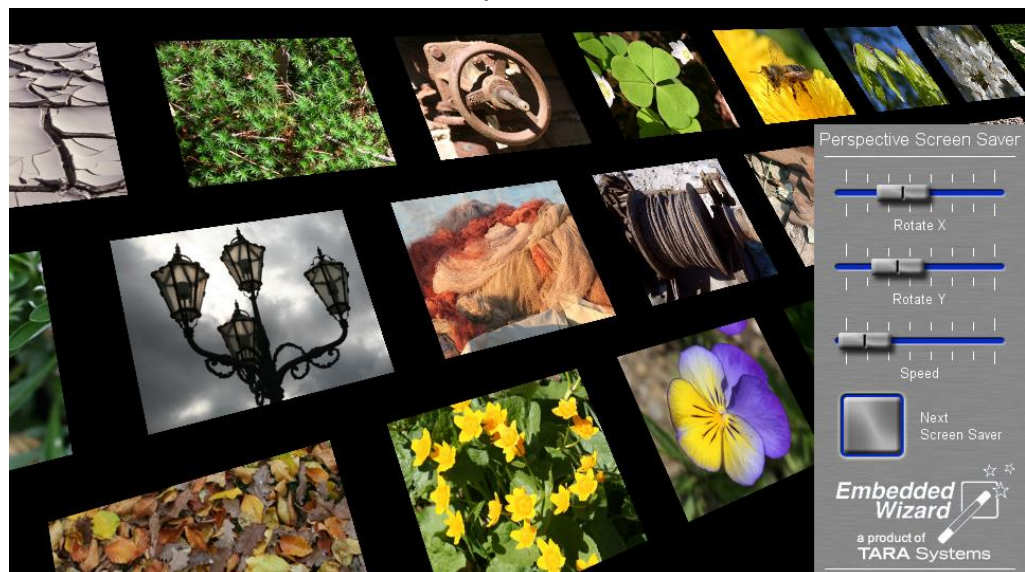
Menus

This example shows the usage of the templates from the new Gallery folder 'Panel Templates'. This example implements a menu system similar to the from the origin Mosaic 1.0 example Widgets. It should help you to understand how panels and menus are developed with Mosaic 2.0:



ScreenSaver

This example demonstrates the usage of lightweight 3D functionality of Embedded Wizard. Originally the example was developed for the Mosaic 1.0. Now we ported the example to Mosaic 2.0 and enhanced it by additional Mosaic 2.0 features:



Documentation and Tutorials

The document 'EmWi Tutorial - Mosaic 2.0' has been enhanced by additional chapters describing how to develop a classic user interfaces with menus and panels by using the templates mentioned above.

Bug Fixes

The following bugs has been solved:

Usage of JPGs, GIFs and BMPs under Windows7

If Embedded Wizard is running on a PC with Windows7, the import of JPG, GIF or BMP files as bitmap resources caused an error message. The problem is solved now.

Mosaic 2.0 class Views::Line

The automatic layout functionality failed in case of lines with thickness 2 or more pixel. The problem is solved now.

Mosaic 2.0 class Core::VertList and Core::HorzList

In some cases the content of a list is loaded twice. The problem is solved now.

Mosaic 2.0 class Views::Text

Text output with the alignment mode set to justified produced jitter artifacts. The problem is solved now.

IDE Crash

In seldom cases, the renaming of members in the Embedded Wizard IDE could produce a crash. The problem is solved now.

Embedded Wizard V6.00

Version V6.00 contains the following changes and improvements:

Mosaic 2.0

Embedded Wizard 6.00 comes with a fully redesigned version of the Mosaic framework. It provides a wide range of components and ready-to-use widget templates. Mosaic 2.0 introduces additional graphical effects and contains the necessary support for touch devices.

The Mosaic 2.0 framework provides a very simple programming model expecting less development experience and less time for training. The new Mosaic 2.0 framework should be selected as the development foundation for all new GUI projects.

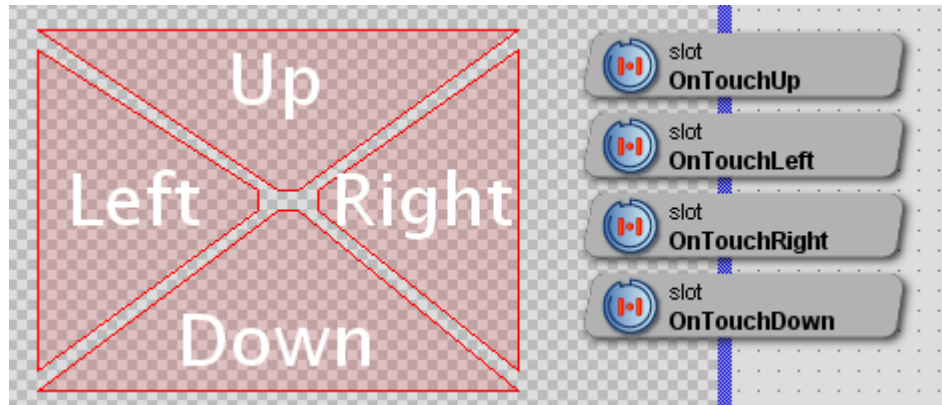
Please note, that due to the completely new approach of the Mosaic framework, Mosaic 2.0 is not compatible to Mosaic 1.0. Nevertheless, Mosaic 1.0 is still available and will be maintained in the future. With other words: Existing EmWi projects are still supported with Mosaic 1.0.

These are the main features of the new Mosaic 2.0 framework:

- Redesign of Mosaic class hierarchy – Similar classes have been unified in order to simplify the entire class architecture.
- Color and opacity gradients – All views can be drawn with color or opacity gradients, e.g. text views:

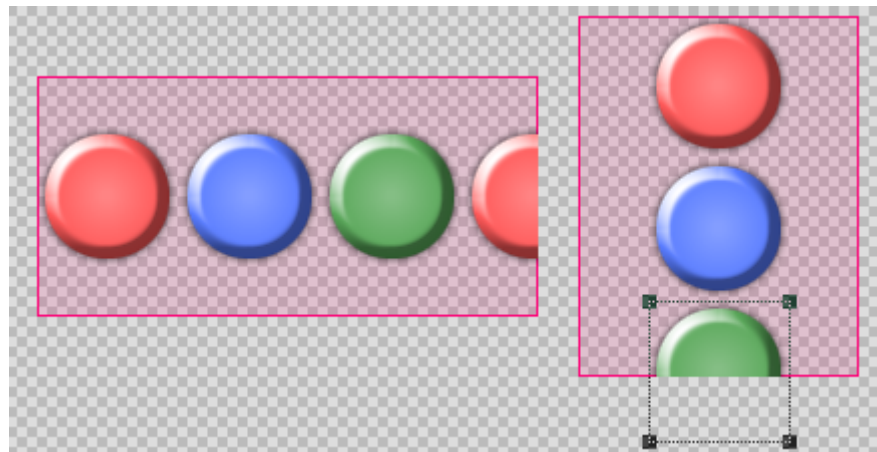
Mosaic 2.0

- Alpha8 bitmap resources - This kind of bitmap resources contains only the transparency information. They are intended to be used as patterns colored dynamically during runtime.
- Optimized screen update cycle.
- Keyboard handler – The new ready-to-use keyboard handler simplifies the processing of user keyboard inputs. All you need to do is to place the handler in your GUI component and connect it with slot methods, which are invoked as a response to user interactions.
- Touch event handler - A set of touch event handlers simplifies the processing of user interactions. Touch event handlers can be placed and arranged within GUI components. They appear as semitransparent areas and represent the touchable area:



Apart from these simple touch handler, more sophisticated gesture handlers are able to process touch screen gestures, like rotation or slide.

- Layout management – The position and size of all views within a component can be calculated automatically without writing a single line of code. Additionally, views can be arranged vertically or horizontally within a rectangular outline area:



- Scroll Lists – The new framework provides ready-to-use scroll lists, which can be easily integrated within a GUI application. Their purpose is the presentation of list items, which can be scrolled and selected by the user.

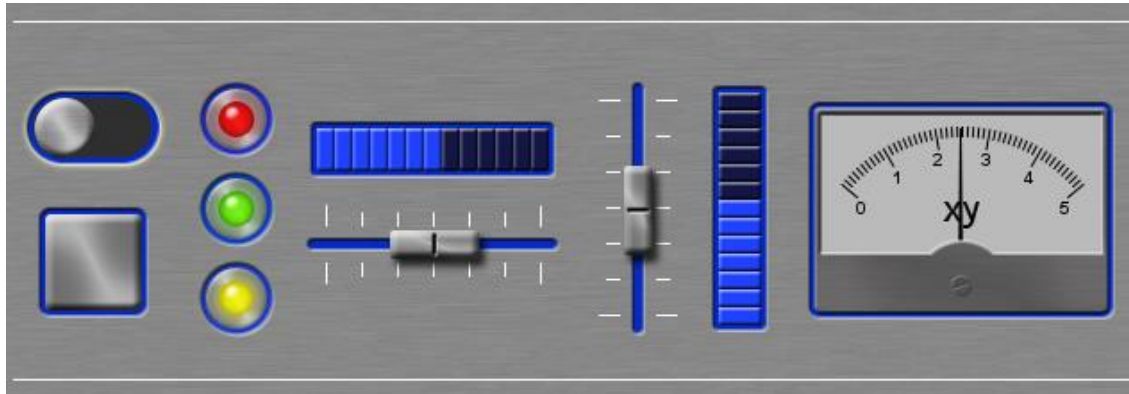
For more details, please have a look into the new Mosaic 2.0 User Manual.

Widget Templates

With Mosaic 2.0 framework, Embedded Wizard is delivered with a set of ready-to-use widgets. They are intended as templates, which can be either used "as is" or they can be adapted to your own needs.

Due to this component based approach, less object oriented know-how is needed for creating own widgets. Just copy the template to your project and adapt its implementations according to your needs.

The following image demonstrates the default appearance of the widget templates:



Documentation and Tutorials

Due to the new Mosaic 2.0 framework, the set of available documentation has been reworked.

“Embedded Wizard User Manual” – This document provides an introduction to Embedded Wizard and its integrated development environment. The chapter ‘Quick Tour’ has been completely rewritten and provides a new sample based on the Mosaic 2.0 framework. Please have a look into this chapter to get a first impression of the new feature set.

Depending on the framework you are using for your project, a different set of documents is relevant. For working with Mosaic 1.0, please have a look into the following documents:

- Mosaic 1.0 User Manual
- EmWi Tutorial – Basics
- EmWi Tutorial – Widgets

Both tutorials are located within the subdirectory \Tutorial_Mosaic10.

For working with the new Mosaic 2.0 framework, please see the following new documents:

- Mosaic 2.0 User Manual
- EmWi Tutorial – Mosaic 2.0

The new tutorial is installed in the subdirectory \Tutorial_Mosaic20.

Examples

Due to the new Mosaic 2.0 framework, the set of examples has been split and enhanced. All old examples (based on Mosaic 1.0) are now located within the subdirectory \Examples_Mosaic10.

Within the subdirectory \Examples_Mosaic20 several new examples are installed. They are based on the new framework and demonstrate the basic concepts and most important features.

The following examples are now available:

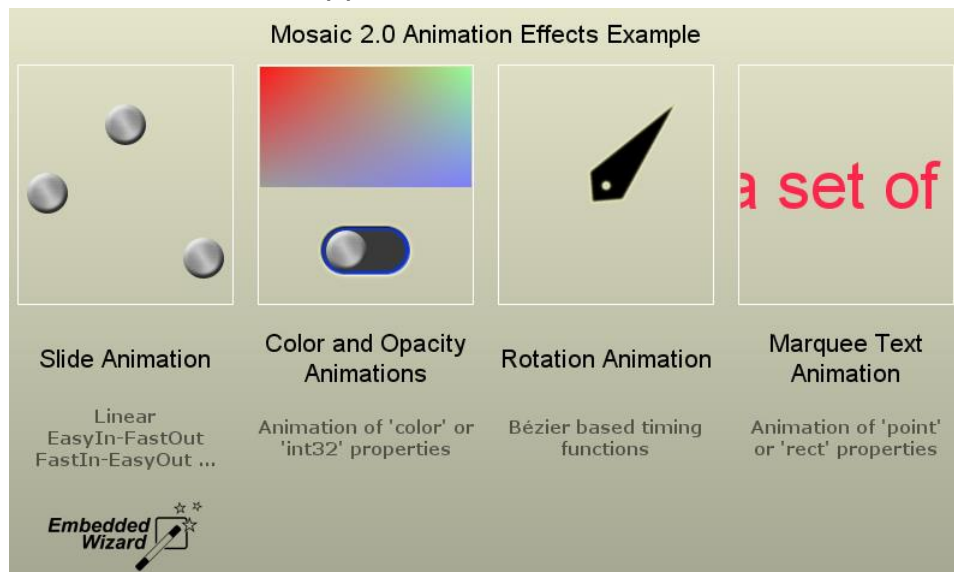
Quick Tour

The starting example 'Quick Tour' is completely redesigned and described in the Embedded Wizard User Manual. This Quick Tour is stored in the subdirectory \QuickTour.



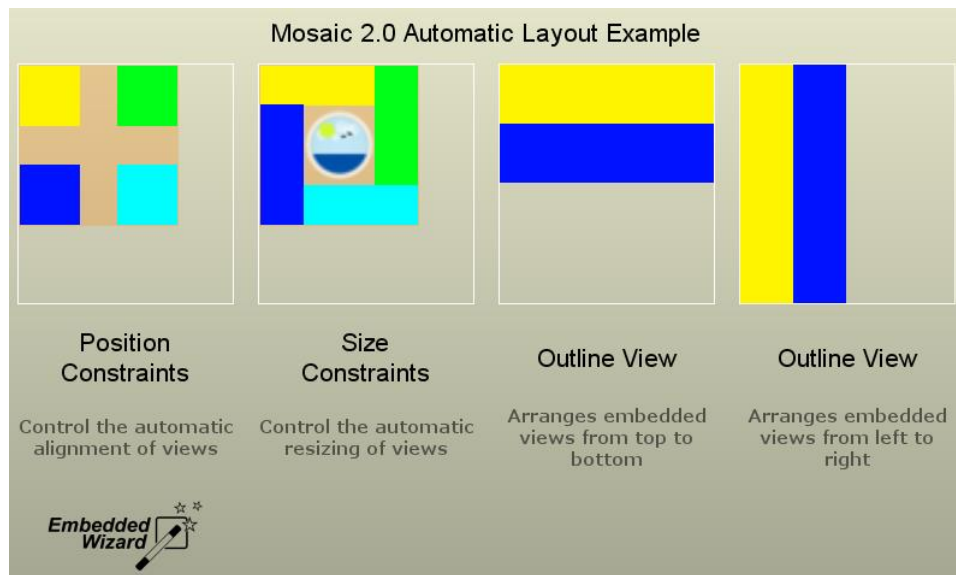
Animation Effects

This example demonstrates the usage of several animation effects within a GUI application.



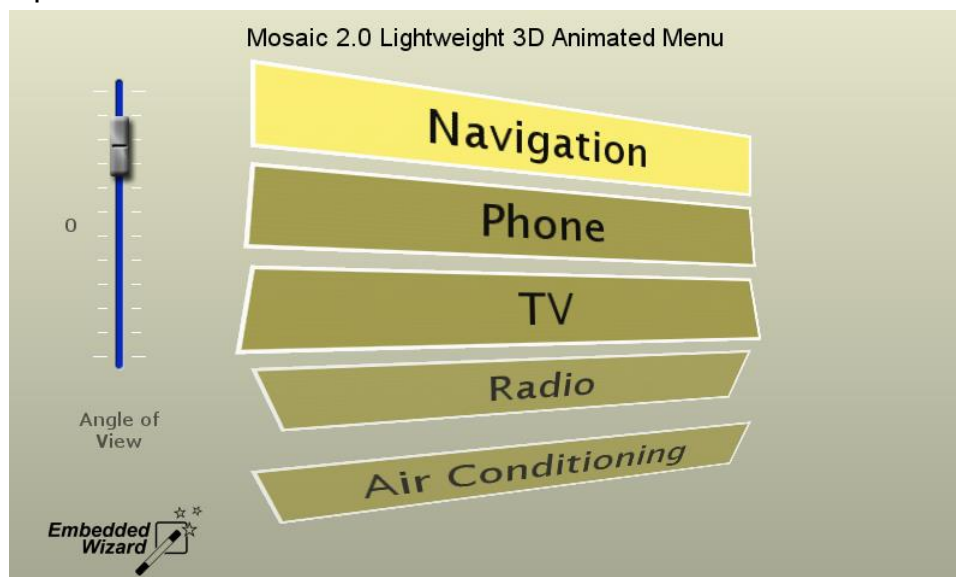
Layout

This example demonstrates the new automatic layout calculation and arrangement, introduced with Mosaic 2.0.



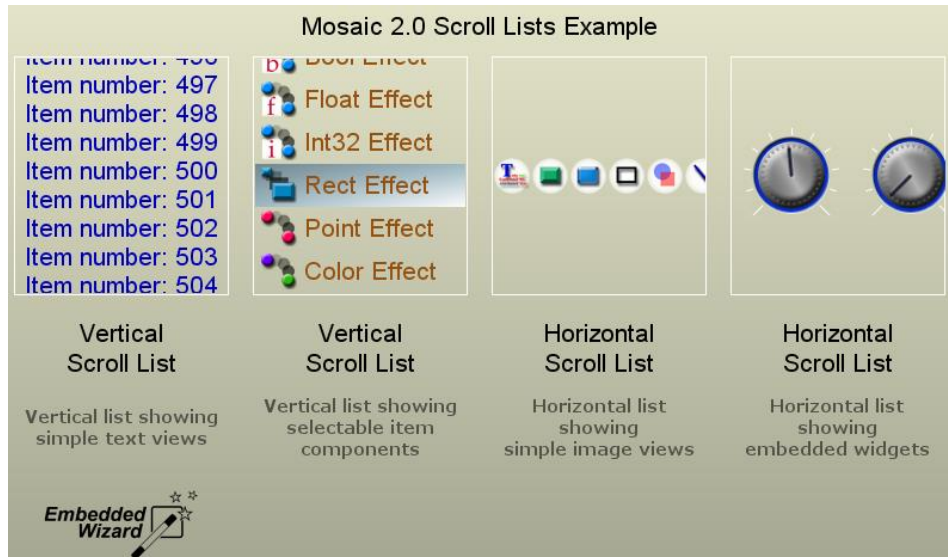
Menu3D

This example demonstrates a scene based approach of a simple 3D menu.



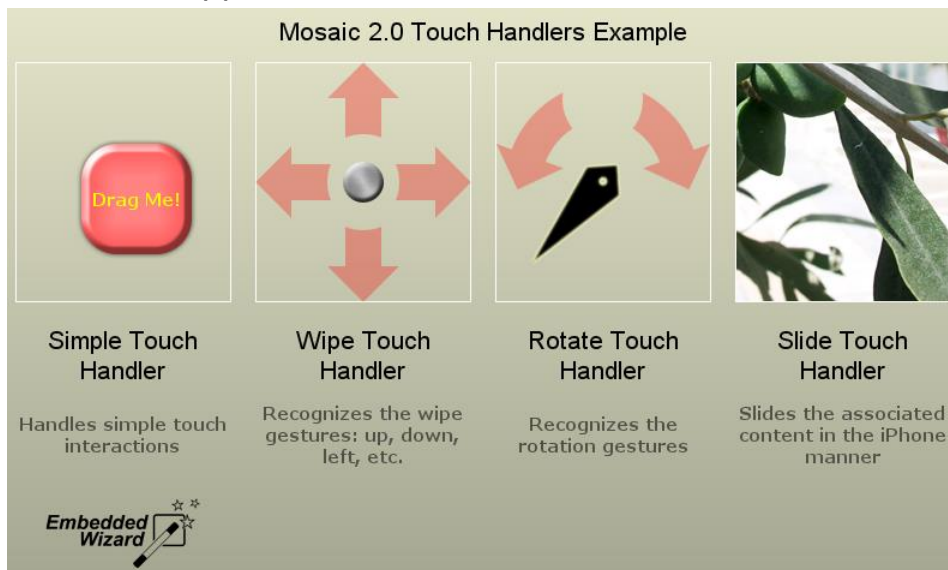
Scroll Lists

With Mosaic 2.0 new scroll list components are provided. This examples demonstrates several fancy, iPhone-like scroll list implementations.



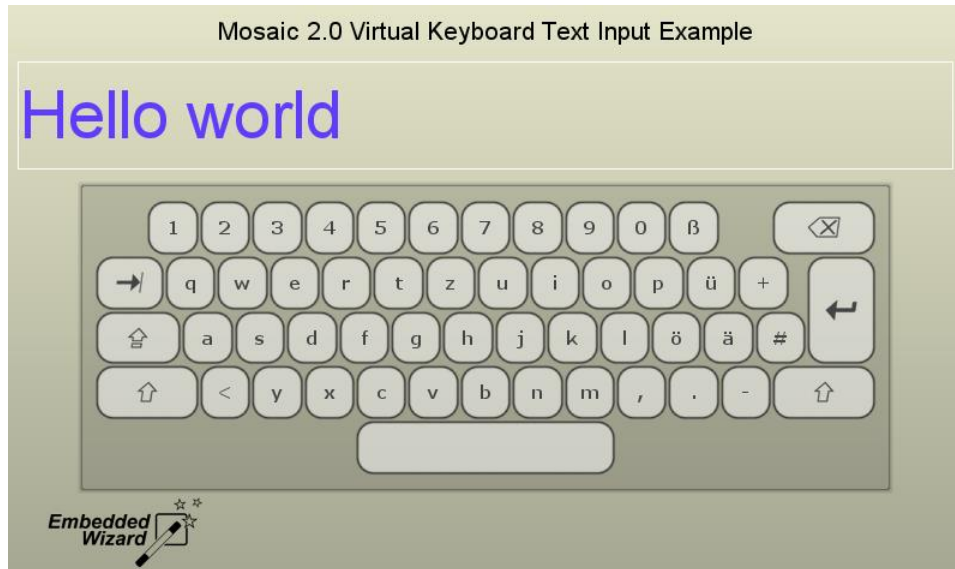
Touch Handlers

With Mosaic 2.0 a couple of ready-to-use touch and gesture handlers are introduced. This sample demonstrates their usage within a GUI application.



Virtual Keyboard

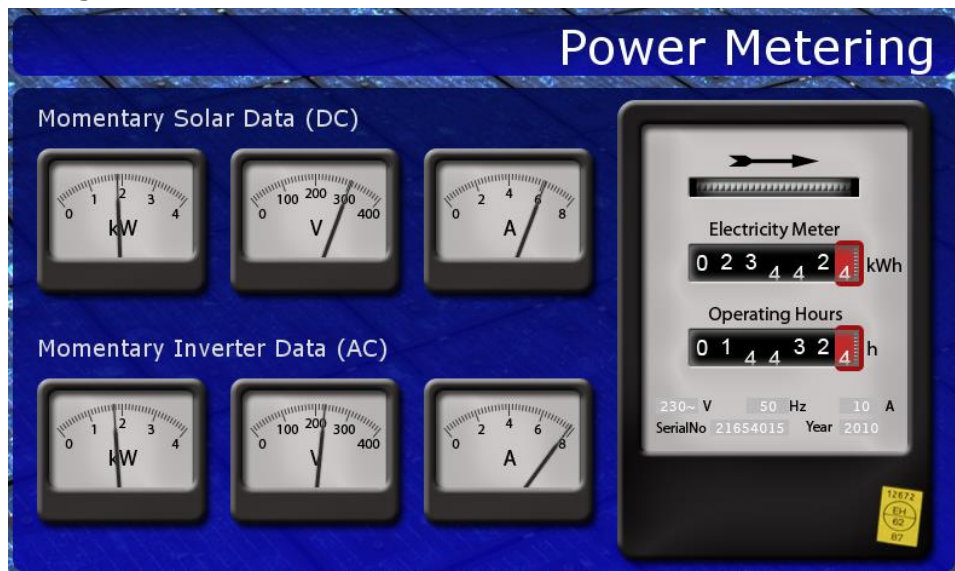
This sample application demonstrates how a virtual keyboard can be implemented and integrated within a GUI application.

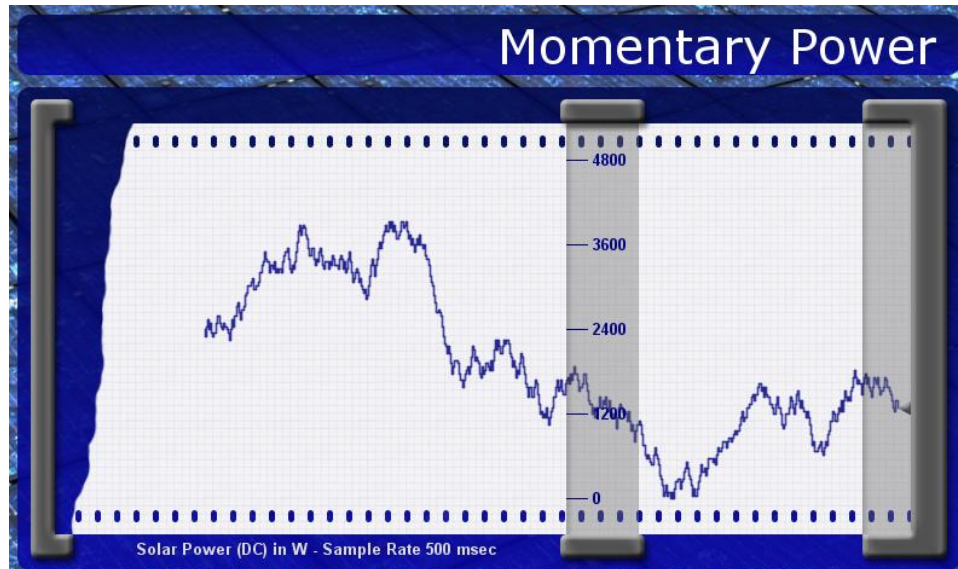


Solar Demo

This application demonstrates a couple of fancy analogue and digital instruments and data displays as they could be used in a modern solar power application (data logger).

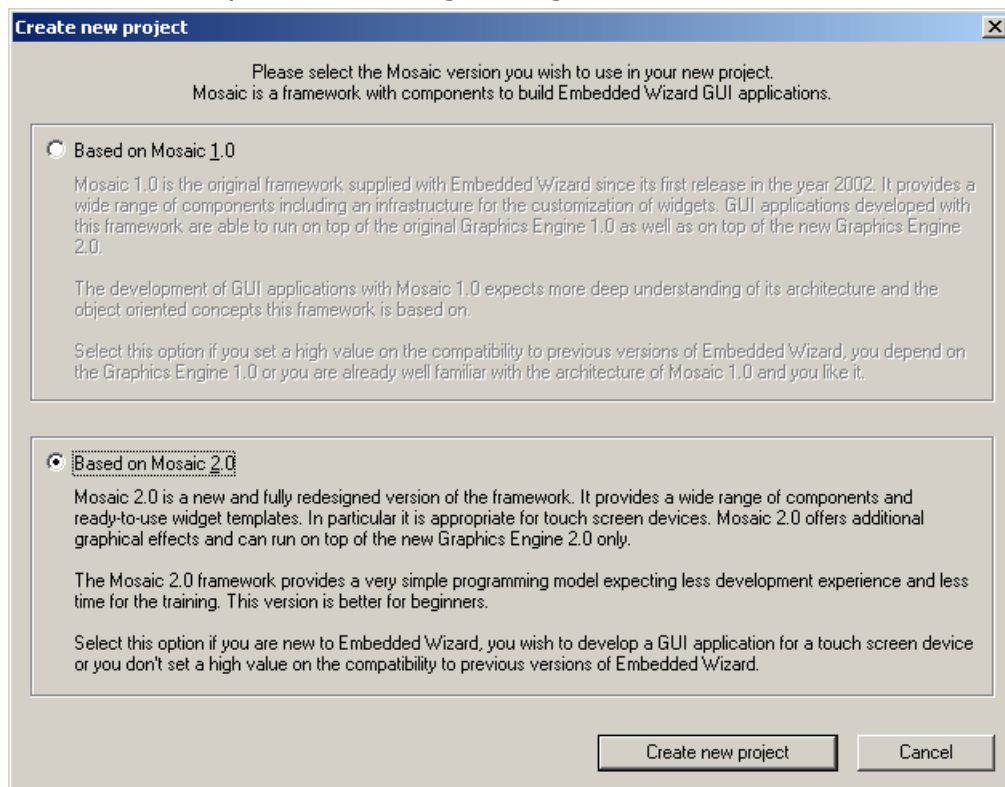
It was developed to demonstrate Embedded Wizard's suitability to design meters and gauges with rotating elements for various purposes. The solar demo will start in an auto demo mode. Additionally, it can be controlled with simple touch gestures.





New Project Templates

When the user creates a new project, the desired Mosaic framework can be selected by the following dialog:



Templates Gallery

Since Embedded Wizard supports the development of GUI applications based on Mosaic 1.0 as well as on Mosaic 2.0, the content of the Template Gallery is exchanged dynamically. Depending on the currently opened project, the content of the Templates Gallery is reloaded automatically according to the used Mosaic version.

Math Functions

The set of mathematical operations has been enhanced by the following two Chora functions:

```
float math_asin( float aValue );  
float math_acos( float aValue );
```

These functions are useful for common mathematical calculations. Please note, that these functions are only available in new Platform Packages based on the new Graphics Engine 2.0 architecture.

For more details see Chora User Manual.

Bug Fixes

The following bug has been solved:

Prototyper Screen Update

In case that the user scrolls the content of the Prototyper window while the screen update is in progress, some areas were not updated correctly. The problem is solved now.

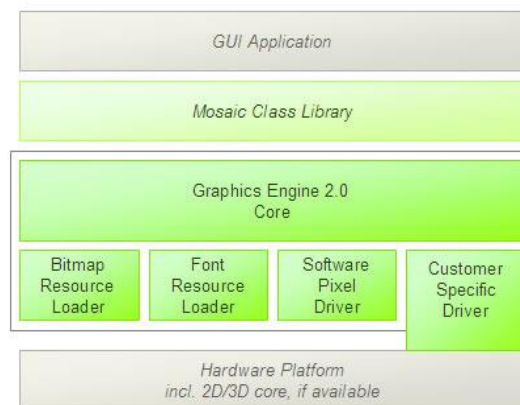
Embedded Wizard V5.40

Version V5.40 contains the following changes and improvements:

Graphics Engine 2.0

With Embedded Wizard version 5.40, TARA Systems introduces a newly implemented, improved Graphics Engine (GE) 2.0 as heart of the GUI development tool. The GE 2.0 provides the necessary architecture and interfaces to integrate the underlying hardware, e.g. a 2D/3D graphics core, very efficiently. If certain graphics functions are not available in hardware, the GE 2.0 performs the execution in software by the CPU.

Moreover, the new GE 2.0 enables more flexible adaptation of Embedded Wizard to various hard- and software APIs including OpenGL ES 1.1 and 2.0 or DirectFB and facilitates the dynamic loading of bitmap and font resources.



These are the most important improvements of GE 2.0:

Optimization

All drawing operations are now evaluated, reorganized and optimized within the GE 2.0 before they are executed. Thus, superfluous drawing operations can be eliminated. If possible, the order of drawing operations is changed in order to reduce the synchronization overhead between the hardware accelerator and software drawing routines.

Bitmap Formats

The set of supported bitmaps has been enhanced and unified. The GE 2.0 supports now three bitmap formats: Index8, Alpha8 and so called Native bitmaps. The Graphics Engine provides a rich set of drawing operations with bitmaps as a destination and source. Bitmaps can be copied, alpha-blended or even perspective correct projected as it is necessary for 3D effects.

The denomination 'Native' refers to the bitmap format supported natively by the particular target system (e.g. 32 bit RGBA8888). Native bitmaps can serve as the destination and/or as the source for a drawing operation.

Index8 bitmaps provide a generic format of an 8 bit / pixel bitmap with a color look-up table CLUT. When drawing an Index8 bitmap, its pixel values will be translated through the CLUT in the particular native color format. An advantage of Index8 bitmaps is the possibility to change its colors by modifying the color entries within its CLUT.

Alpha8 bitmaps provide a generic format of an 8 bit / pixel bitmap with a single alpha (A) channel - no color (RGB) information is stored there. Alpha8 bitmaps are suitable for drawing patterns modulated by externally specified colors. For example: text output.

In contrast to the Native bitmap format, Index8 and Alpha8 can serve as source only. It is not possible to draw to an Index8 or Alpha8 bitmap.

Drawing Functions

The new GE 2.0 provides an extended set of drawing operations: Lines, filled rectangles, bitmap copying, 3D perspective correct projection and text output. All operations can be performed with or without alpha-blending. Alpha-blending allows fine fading effects when several graphical objects are drawn one upon another.

Additionally all drawn pixel can be modulated by color or opacity values based on linear gradient calculation. In this manner, copied bitmap areas can, for example, be faded out on its edges. A gradient can be vertical, horizontal or both. In the last case for each corner of the gradient a different color value can be set, so 4 colors (or opacity values) can be mixed in the gradient area!

With 'Warp' operations, Graphics Engine provides 3D perspective correct projection of bitmaps. Beside the 3D effect, this functionality allows 2D scaling and rotating of bitmap areas. For best possible quality, all warp operations are performed with increased sub-pixel precision. Additionally, bi-linear filters reduce artifacts, when bitmaps are warped.

Please note, that GE 2.0 is a completely new implementation with a new interface, which is not compatible to the previous version of the Graphics Engine. However, the current implementation of the Mosaic class library is able to operate with both versions. Thus, all GUI applications, based on the Mosaic class library, are still working with both engines. In case that an application contains direct Graphics Engine function calls, some adaptation is necessary.

Win32 Platform Packages

Due to the new Graphics Engine 2.0, the entire set of Windows Platform Packages installed with Embedded Wizard 5.40 has been adapted. The following Win32 configurations are provided now:

- Tara.Win32.RGBA8888
- Tara.Win32.RGB565A8
- Tara.Win32.RGBA4444
- Tara.Win32.Index8

Apart from the implementation of the new Graphics Engine, the set of delivered Platform Package source and header files has been changed. In order to compile Win32 applications the Developer Studio projects need to be adapted accordingly. Please have a look to the delivered sample applications, which are adapted to the new Platform Package files.

Remark: The Runtime Environment and the Graphics Engine of all Win32 Platform Packages are delivered now as libraries. These are linked automatically with your Developer Studio project as soon as the corresponding `ewrte.c` and `ewgfx.c` are compiled. Of course, Platform Packages for targets are still provided as source code.

For compatibility reasons, the old set of Win32 Platform Packages based on the old Graphics Engine are still provided. In order to avoid conflicts, they are renamed to `Tara.Win32_GE10.<color format>`. If you still need to generated code for the `Win32_GE10` target, please change the attribute Platform of the Profile brick within your Embedded Wizard project.

Math Functions

With the introduction of Graphics Engine 2.0, the set of mathematical operations has been enhanced by the following Chora functions:

```
float math_sin( float aAngle );
float math_cos( float aAngle );
float math_pow( float aA, float aB );
float math_sqrt( float aValue );
float math_rand( float aValue1, float aValue2 );
int32 math_rand( int32 aValue1, int32 aValue2 );
```

These functions are useful for common mathematical calculations. Please note, that these functions are only available in new Platform Packages based on the new Graphics Engine 2.0 architecture.

For more details see Mosaic User Manual.

Mosaic

With the introduction of Graphics Engine 2.0, the functionality of the following Mosaic classes has been extended:

Views::AttrText

The class `Views::AttrText` supports now the standard Unicode soft-hyphen sign (0x00AD) for automatic word wrapping. In case of word wrapping, a hyphen sign is shown at the position of a soft-hyphen sign.

Views::MultilineText, Views::FlowText

The classes `Views::MultilineText` and `Views::FlowText` support now automatic word wrapping. This can be controlled by following special signs:

- `\xAD` (Unicode soft-hyphen) is used to mark within a word a potential wrap position. If the word has been wrapped at the given position, a hyphen sign is shown. Otherwise the sign remains invisible.
- `~` (Tilde) has the same behavior as the soft-hyphen and exists for compatibility purpose with the attributed strings.
- `^` allows automatic word wrapping without showing of any hyphen signs. If the word has been wrapped at the given position, no hyphen is displayed.

Placing a `%` (percent) sign in front of a special sign deactivates its word wrap signification; the sign is handled as an ordinary sign and is displayed.

Please note, that these extended functionality is only available in new Platform Packages based on the new Graphics Engine 2.0 architecture.

Prototyper Optimization

An optimization within the Prototyper increases the prototyping execution speed up to factor x 2.

Examples

The set of examples has been extended:

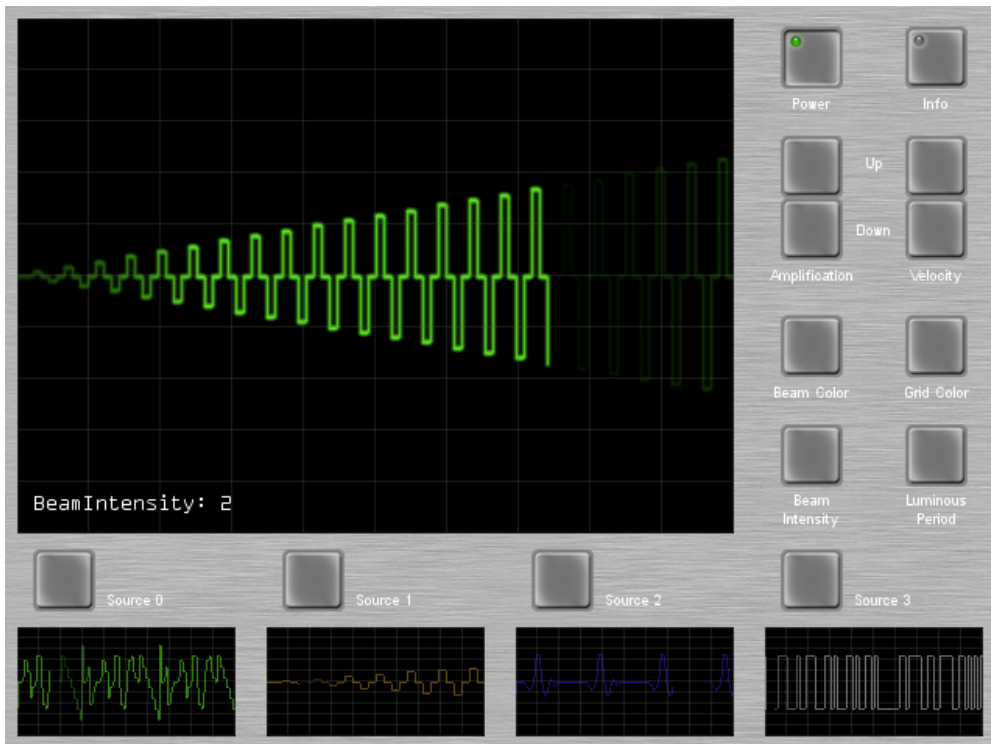
Aviation Demo

The Aviation demo shows a relatively lifelike replica of an airplane dashboard with several analogue instruments. It was developed to demonstrate Embedded Wizard's suitability to design meters and gauges with rotating elements for various purposes. The aviation demo will start in an auto pilot mode. It is absolutely safe, so just sit back and enjoy the flight. ☺



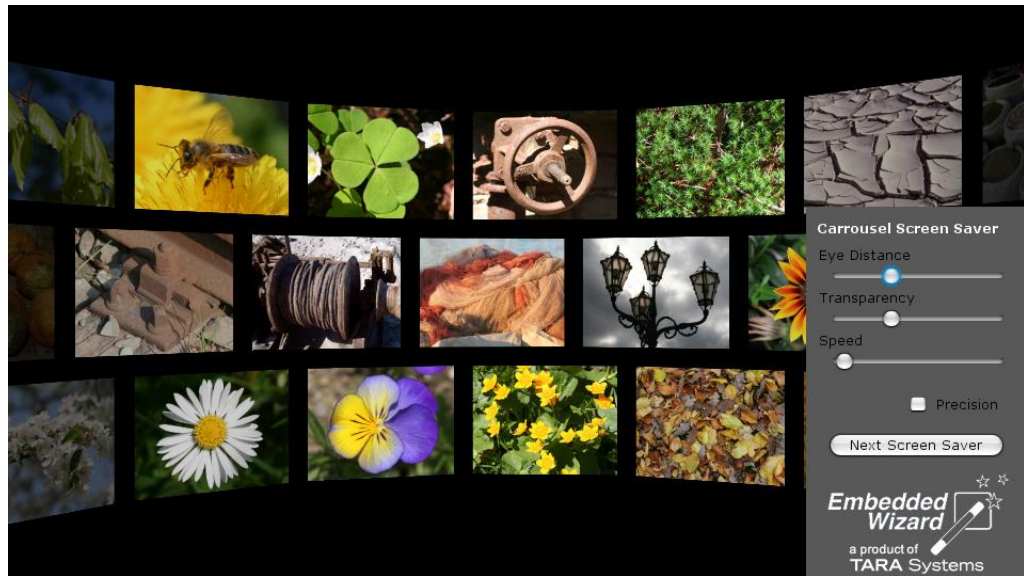
Oscilloscope Demo

The Oscilloscope demo shows a diagram of various signals, to demonstrate further areas of use of Embedded Wizard. The demo can be controlled with several buttons.



Screensaver Demo

The Screensaver Demo contains a set of 3D image animations and demonstrates the possibilities of Embedded Wizards Lightweight 3D features.



Bug Fixes

The following bugs have been solved:

HTML Documentation Generator

In some cases the generated HTML documents contained erroneous HTML tags. The problem is solved now.

Chora Block Comments

Due to an error in the Chora compiler, `/* */` block comments could cause confusing error messages if there were long sequences of `*` signs enclosed. The problem is solved now.

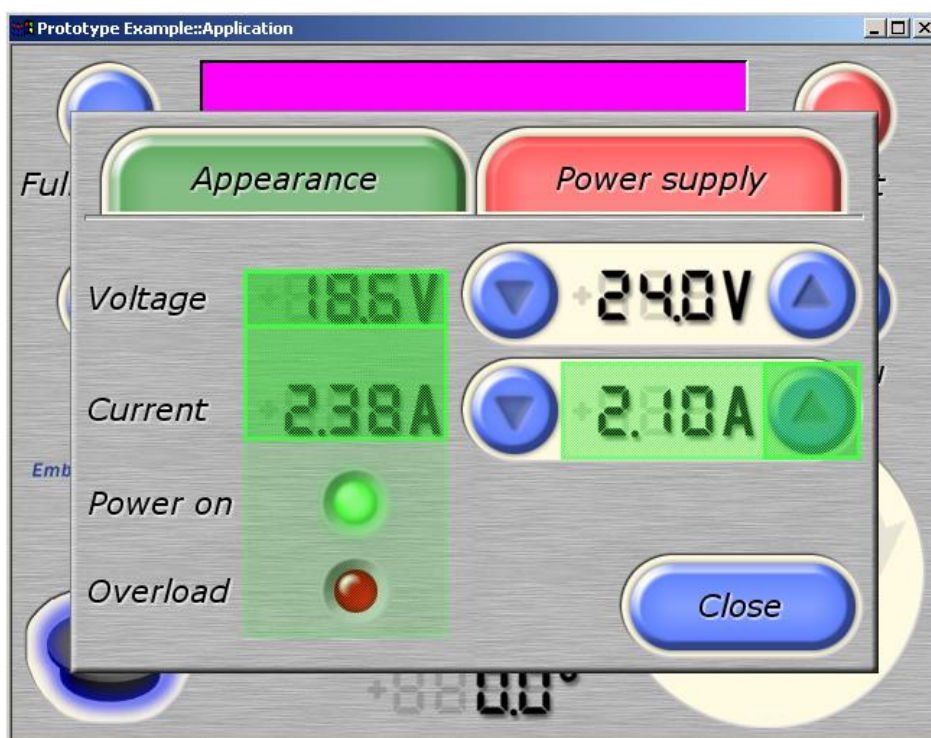
Embedded Wizard V5.30

Version V5.30 contains the following changes and improvements:

Track Screen Updates

The Prototyper of Embedded Wizard provides now a new debugging mode called "track screen updates". If enabled, this feature highlights all areas redrawn during the screen update. This function can help you to analyze and optimize the screen updates in your GUI application.

The following image demonstrates this feature used within a demo application:



All update areas are highlighted by a green rectangular area, which will be faded out within a short period.

The feature can be activated by using the menu item 'Debug' – 'Track screen updates'.

Font Import Improvement

The font converter within Embedded Wizard provides now a cache feature. This font cache avoids, that all used fonts resources are converted every time the Prototyper starts or the Composer page is changed. This features reduces the loading time especially for large character sets (e.g. Far East fonts).

Please note, that in cases you are editing a true type font by using an external font editor, the font modifications are not visible within Embedded Wizard until it has been restarted again.

Prototyping of Application Class

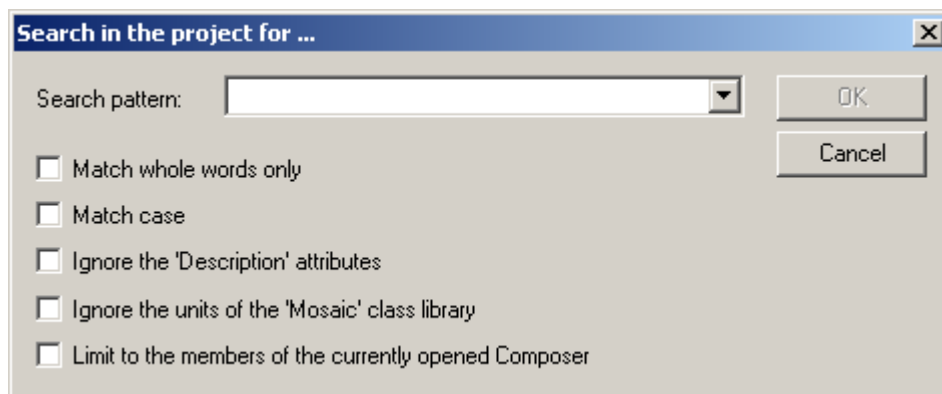
The Embedded Wizard IDE provides now an easy way to launch directly the application class of your GUI project into the Prototyper. Thus, it is no more necessary to switch to the Composer page of the application class in order to prototype the complete GUI application.

The new feature is available via the new menu item 'Debug' – 'Start prototyper with application class' or simply the shortcut 'Ctrl' + 'F5' can be used.

The name of the application class has to be defined within the new profile attribute **ApplicationClass**.

Search in Project

The search dialog within Embedded Wizard has been enhanced by an additional option to exclude the units of the Mosaic class library. By selecting this option, only your project members are considered for the search operation.



Chora

The new instant property **size** can be used now to query the number of elements within an array. This instant property returns the maximum number of elements which can be stored within the array.

Example:

```
array string a[4];
array string b[4,2];
var int32 c;

c = a.size;    // c = 4
c = b.size;    // c = 8
```

This instant property is read only.

Minor Improvements

Additionally, Embedded Wizard has been improved by the following features:

Auto Save

Embedded Wizard supports now an automatic project save. If enabled, your project modifications are saved every 20 minutes. The feature can be controlled by the new menu item 'Extras' - 'Enable auto save'.

Prototyper

In cases your application remains in an endless loop, the 'Close' button of the Prototyper Window can be used now to obtain the control of the running application. After that you can analyze the reason of the endless loop by using the integrated debugger.

Please note, that a running application within the Prototyper can also be interrupted by pressing the key 'Pause'.

Multi-Monitor Support

The Embedded Wizard IDE is now capable to run on PCs with more than one monitor. All dockable windows (e.g. Assistants, Inspector) can now be placed on the entire workspace. The arrangement is also saved.

EwNewStringUtf8()

The Runtime Environments (RTE) of all Platform Packages provide now the function `EwNewStringUtf8()`, which converts a given UTF-8 string into an Embedded Wizard Unicode string.

Code Editor Assistant

The Code Editor has been enhanced in order to support the preprocessor Assistant window when editing inline code. The Assistant window appears when the user begins to enter a Chora preprocessor directive or macro, which start with the \$ character.

Bug Fixes

The following bugs have been solved:

Panels::EditField

In cases the edit field contains already the maximum number of characters, the text input fails in overwrite mode.

The problem is solved now.

Prototyper

The internal string buffer used for intrinsic function calls, was limited to 4k characters, which was not enough for applications working with large strings. Now the buffer limit is set to 64k characters. Please note, this limitation had no influence concerning the generated code.

Animation Effects

Due to an error within the Mosaic effect classes (like **Effects::Int32Effect**, **Effects::ColorEffect**, ...) the end value of animation was not reached, if the property **Mode** was set to **Effects::Mode.Asymptotic**.

The problem is solved now.

Embedded Wizard V5.20

Version V5.20 contains the following changes and improvements:

Introduction of new "Lightweight 3D" design approach

GUI development for consumer electronics and mobile appliances is more and more influenced by the demand for stylish interface designs with rich effects and 3D-like appearance – driven by trend setting products, e.g. the Apple iPhone or iPod Touch. In order to support GUI developers to realize such fancy and compelling GUI concepts on embedded platforms, Embedded Wizard introduces the "Lightweight 3D" design approach in V5.20.

The term "Lightweight" was chosen, to point out two important goals of this new design approach: First its suitability for dedicated embedded systems. For each embedded platform, which supports this new feature, an optimal, dedicated adaptation to the specific hardware capabilities, e.g. available 2D/3D graphic accelerator, will be realized within the underlying platform package.

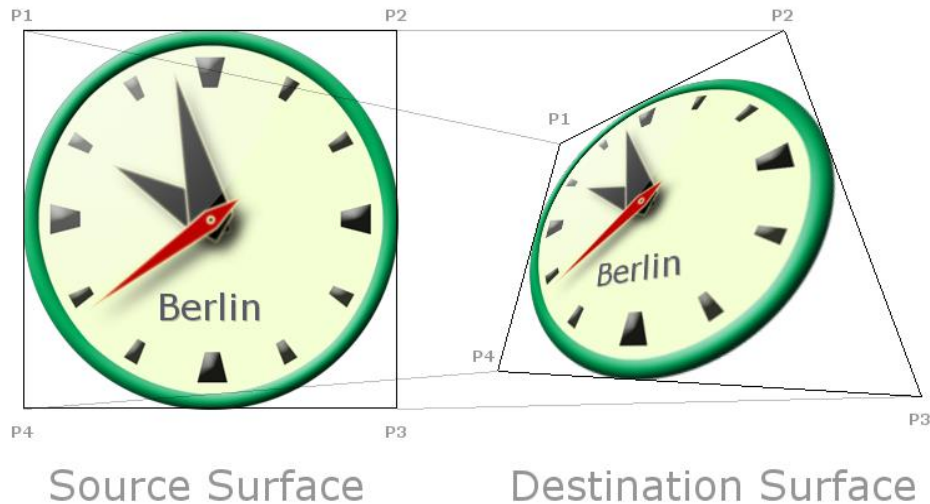
Second, the creation and modification of 3D appearance is seamlessly integrated into the familiar and established workflow of Embedded Wizard, to facilitate the design process.

The lightweight 3D approach enables various new design concepts and effects for moving, rotating or flipping objects, but also for development of analog controls and indicators like rotary knobs, dashboard tachometers, etc. Even simple 2D/3D games can be developed now with Embedded Wizard.



"Image Flow" example with 3D animation and reflection.

The lightweight 3D effects are solely based on perspective correct projection of an image to a four-point polygon. Depending on the shape of the polygon the original image can be scaled, rotated and deformed in any other desired way in order to e.g. create the impression of an image tilted backward in a three-dimensional space. The following illustration demonstrates this idea:



One advantage of this approach is that the developer is free to warp the destination polygon immediately with the mouse and thus is enabled to create any desired perspective or distortion easily. Furthermore, for complex operations and animations such as scaling and rotating objects along X/Y/Z-axis, certain functions can be used, which automatically calculate the corresponding polygon coordinates.

From the developers point of view the 3D functionality is covered by the new Mosaic classes `Views::WarpImage` and `Views::WarpGroup`. The first one implements a special kind of image view, which is able to display 3D transformed (warped) images, bitmaps, etc. Beside the additional 3D functionality this class acts similarly to an ordinary Image view.

The second class `Views::WarpGroup` is specialized to handle the appearance of a whole group. It allows you to simply transform a control or even an entire menu in the 3D space. Moreover, the transformed control or menu remains able to react to user inputs, e.g. the user can still click the warped button surface!

Please note, the new 3D effects are supported only for dedicated target systems, which offer e.g. sufficient CPU power or provide a dedicated 3D hardware engine! In coordination with the responsible semiconductor manufacturers, TARA will reserve the decision for which target system the lightweight 3D functionality is offered. In case of a non-3D capable target system, the GUI application will still be executable, but no 3D output will appear on the screen. For details about the supported target systems please contact TARA Systems GmbH.

For further details see:

- the Chapter 'Lightweight3D' in 'EmWi Tutorial – Basics'
- the supplied example Lightweight3D
- and the 'Mosaic User Manual'

Minor Improvements

Additionally, Embedded Wizard has been improved by the following features:

Refactoring

The feature Refactoring was improved in order to be able to deal with name conflicts in the inheritance of class members. In the previous version members affected by the name conflicts were simply ignored by the Refactoring feature.

Support of Microsoft Visual Studio C++ 2005, 2008

The Win32 Platform Packages supplied with Embedded Wizard and Embedded Wizard DEMO have been adapted to support the Microsoft Visual Studio C++ 2005 and 2008.

Please note we still deliver all supplied examples with project files for the Microsoft Developer Studio 6.0. In case of the new Microsoft Visual Studio C++ 2005 or 2008, the Microsoft development environment will convert the project files automatically.

Graphics::Canvas ScaleBuffer

The class `Graphics::Canvas` has been enhanced in order to scale a platform independent 8 bit bitmap, used as off-screen buffer for external applications. The content of this bitmap can be drawn with a new size and/or aspect ratio into a Canvas by using the new method `Graphics::Canvas.ScaleBuffer()`.

Bug Fixes

The following bugs have been solved:

Chora Compiler

The missing restriction in the designation of members has provoked confusing run-time errors if the name `Class` has been assigned to a member within a Class definition.

The current version of the Chora compiler treats the name `Class` as a reserved identifier.

Chora Optimizer

An error introduced in the previous version 5.10 could provoke overeager eliminations of Enum and Set definitions if the Optimization level attribute has been set to the value `High`.

This issue is solved now.

Code Generator

The conversion of space sign literals ' ' (hex code 0x20) during the code generation was erroneous in seldom cases. Consequently the resulting 'C' code could not be compiled successfully.

This issue is solved now.

Embedded Wizard V5.10

Version V5.10 contains the following changes and improvements:

Improvement of the 'enum' data type

The support of 'enum' data types in the Chora programming language has been improved. Now, the programmer can assign explicitly numeric values to the elements of an 'enum' definition. These values will then appear in the generated code artifacts of the corresponding 'enum' definitions. This should simplify the mapping between Chora 'enum' type definitions and other 'C' enumerations already existing in the target system or in the underlying middleware.

All 'enum' elements, which have not been signed explicitly with numeric values are numbered automatically by the Chora compiler. The numbering starts with the value 0 for the first element. Please note, in the previous versions this automatic numbering was started with the value 1!

Additionally, Chora provides now new type cast operators, which allow the conversion of 'enum' operands into the corresponding numeric values and backwards. By using these operators, it is possible to calculate with 'enum' operands.

For more details see Chora User Manual.

Support of the 'set' data type

The Chora supports now the definition of so called 'set' data types. With a 'set' the programmer defines a list of elements, which can be combined together at the runtime. This means that a variable of type 'set' may contain any combination of elements of the appropriate definition, including an empty 'set' (containing no elements).

Beside the new type definition Chora also supports operators predestined especially for handling with 'set' operands, e.g. operators to unify or intersect of sets.

Analog to the 'enum' definitions, the elements of a 'set' also allow an explicitly specification of the corresponding numeric values. The type cast operators, already mentioned in the section above, allow then the conversion between 'set' operands and the numeric values. In this manner calculation with sets are possible.

For more details see Chora User Manual.

Platforms

The platform packages have been adapted:

Code Generator

The generation of code artifacts for language, style, enum and set definitions has been changed. Now, all generated enumeration items use consistently the 'C' enum notation instead of the 'C' macros. This avoids possible name conflicts and confusing 'C' compiler error messages due to identifiers overridden by such 'C' macros.

Font Converter

The font conversion is improved and supports additionally the conversion of few OpenType fonts, which could not be converted in the previous versions.

Mosaic

The Mosaic class library has been extended:

Core::Group

The class `Core::Group` provides now an additional property `Buffered`. It determines whether the affected group does store its own aspect in an internal canvas. Such buffered groups are handled more faster, when they are moved or faded-in/out because the drawing of the enclosed group members can be omitted.

Additionally the new variable `Buffer` permits the access to the internal buffer of a buffered group. As long as the group is not buffered, the variable is `null`.

Please note, buffered groups may occupy a lot of display memory. Use this feature temporary, during animations only.

For more details see Mosaic User Manual.

Minor Usability Improvements

Additionally, Embedded Wizard has been improved by the following features:

Code Editor Assistants

The 'Directives und Macros' Assistant has been extended. Now the Chora directives `$if`, `$endif`, etc. can be selected directly from the assistant window.

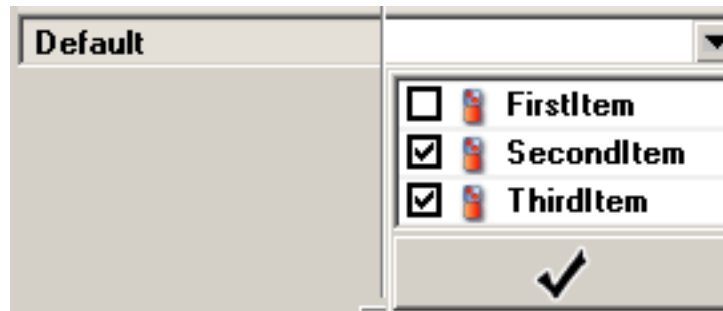
Code Editor Assistants

The behavior of the assistants has been improved. Now, the text entered in a Code Editor Assistant can be simply taken over by hitting the Enter key, even if no item within the assistant does match the entered text.

Inspector Assistants

Due to the support for the new data type 'set', a new

Inspector Assistant window has been implemented in order to allow the user the selection of 'set' literals in a more convenient way:



Browser Window

Analog the Browser Window has been extended in order to display the 'set' definitions.

Bug Fixes

The following bugs have been solved:

Mosaic

The implementation of the property `HelpString2` in the class `Panel::PushButtonClass` was erroneous. The problem is solved now.

Variants

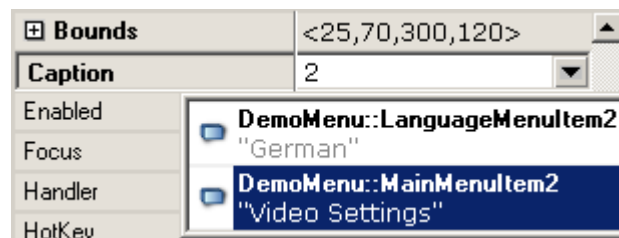
Due to an error in the saving routines of Embedded Wizard project files, the `$variant` directive (also known as the `VariantCond` attribute) has ignored the value `false`. The problem is solved now.



Embedded Wizard V5.00

Version V5.00 contains the following changes and improvements:

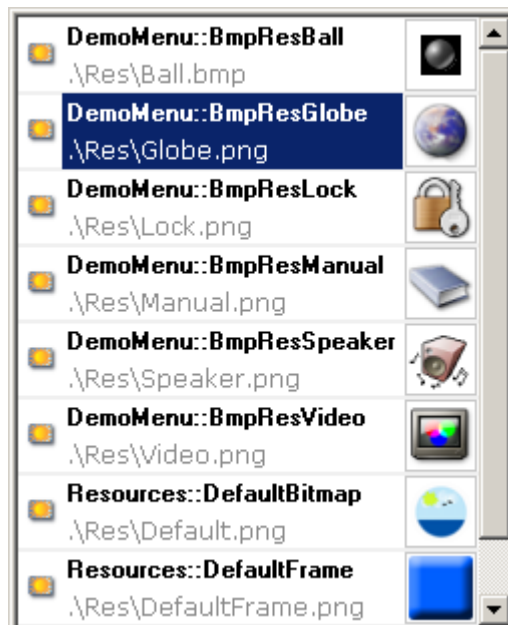
Inspector Assistants

The Inspector contains now a set of assistants to simplify and speed up the application development. Assistants provides a comfortable way to modify the content of a property or an attribute. Thus, the value of a property or an attribute can be changed easily by selecting the new content from a list instead of writing the new value manually.

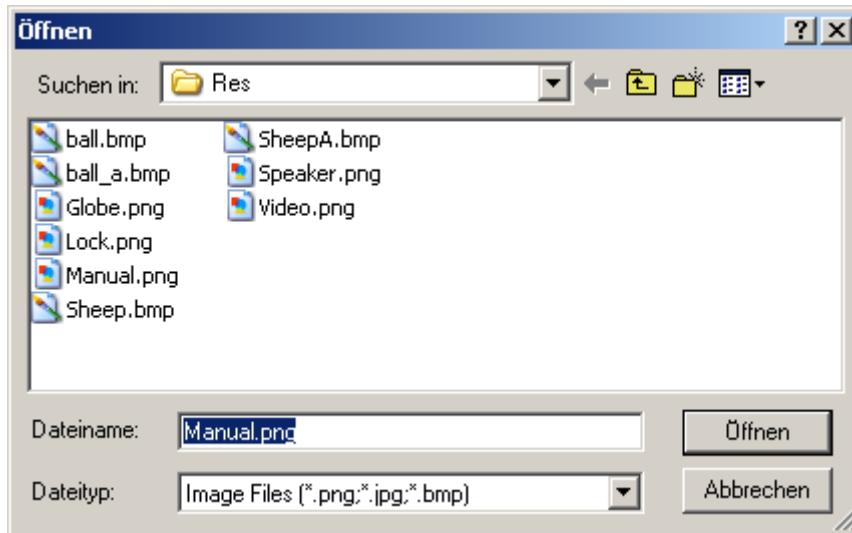


To open an assistant click on the small button  or  on the right hand side of the selected attribute or property.

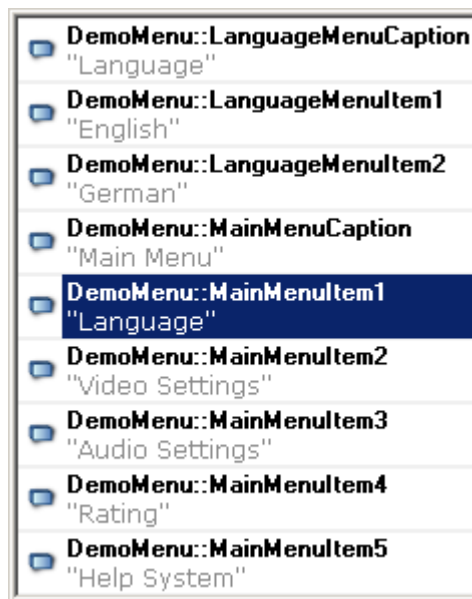
The kind of the displayed assistant corresponds to the data type of the selected attribute or property. For example the Resource Assistant allows you to determine a bitmap, which should be displayed in front of a menu item by selecting it from a list of existing bitmaps of your project.



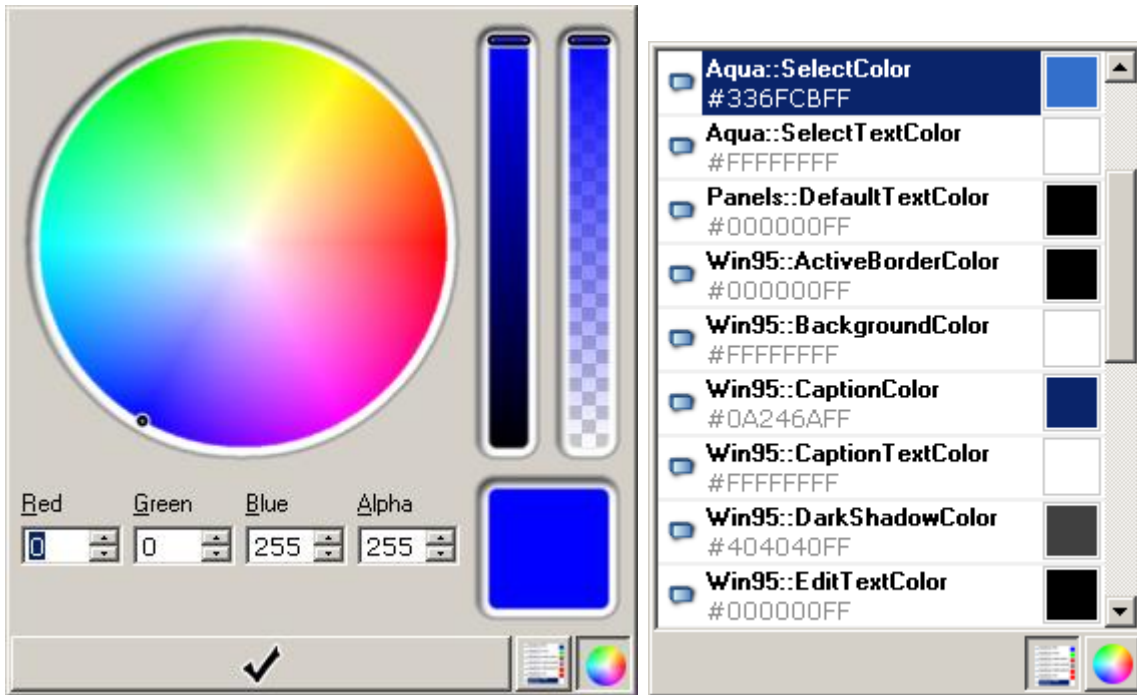
Dialog assistants include the 'File Open' dialog and the 'Select Directory' dialog. They appear, for example, when you are editing the *FileName* attribute of a bitmap resource and you wish to select the desired bitmap file.



Popup list assistants display a list of entries, which fit to the edited attribute or property. For example, if you are editing a **string** property, the assistant will provide you with the list of all **string** constants of your project.



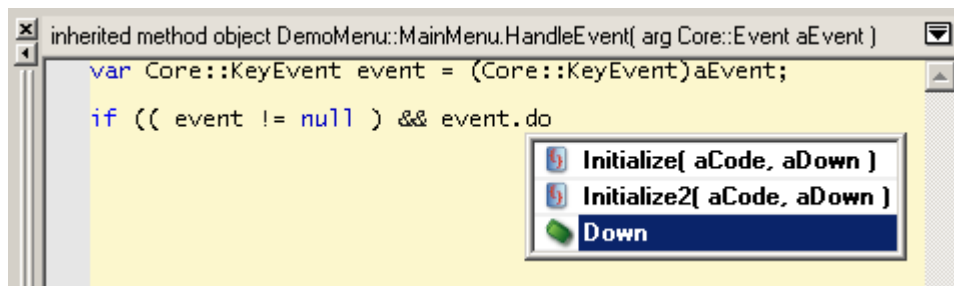
The color assistant is a special kind of the popup list assistant. It provides two views: the list view with all available color constants of the project and the color wheel view, where the desired color can be directly determined in a very convenient way.



Please note, the usage of the assistants requires an upgrade of your Embedded Wizard license.

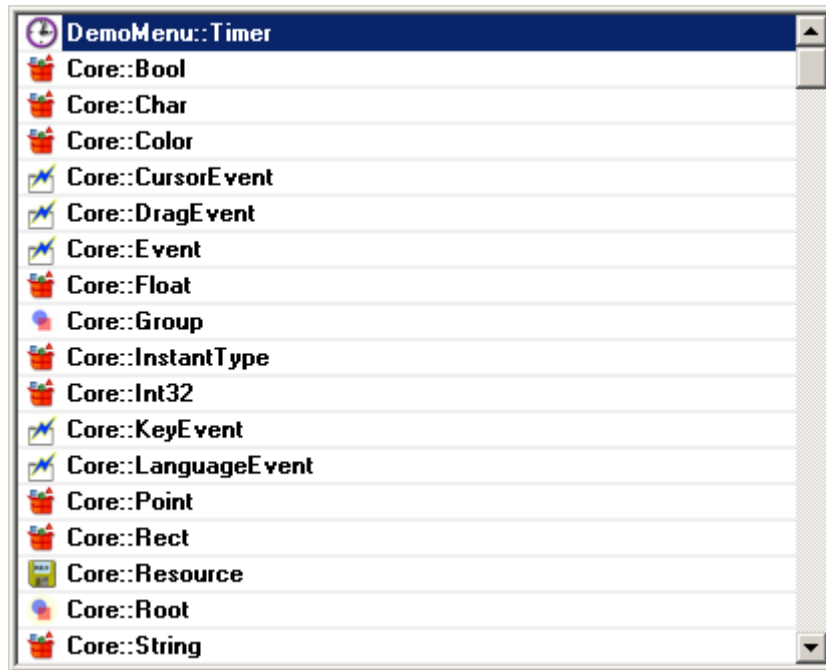
Code Editor Assistants

Similar to the Inspector, the Code Editor also provides a set of assistants. They appear while the user edits expressions within the Code Editor. For example the Scope Assistant displays a list of members, which are valid in scope of an edited expression:



Beside the Scope Assistant, the Editor also provides other assistants. For example, the Macro Assistant which appears after entering the \$ sign. Or the Attributed Text Assistant, which appears, when the user has entered a brace { sign within a string literal.

By pressing the keys *Ctrl+Space* you can open scope independently a global assistant, which displays all project members and the members of the currently edited class. In this manner you get access to all members at once:



Please note, the usage of the assistants requires an upgrade of your Embedded Wizard license.

Refactoring

Embedded Wizard supports now refactoring for renaming. When renaming a member within the Inspector, Embedded Wizard evaluates the entire project and searches for references to the renamed member in order to adapt them to the new name. This, so called, refactoring take in account the inheritance of classes, the initialization expressions of members, the source code of methods, etc.

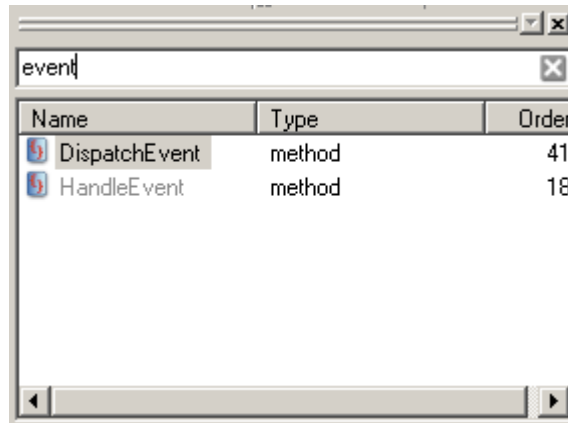
Additionally, when renaming a parameter within the Code Editors declaration area, Embedded Wizard evaluates the body of the method and the body of all derived methods in order to adapt all expressions, where the parameter has been used.

To suppress the automatic refactoring, press the keys *Shift+Return* instead of the key *Return* when closing the inplace edit field.

Please note, the usage of the refactoring requires an upgrade of your Embedded Wizard license.

Inspector Search Filter

The upper area of the Inspector provides now an additional 'Search' edit field. This edit field can take a filter condition. If the filter is set, the Inspector will show only the members, whose name does fit the given filter condition:



The condition is fulfilled, if the filter text does appear anywhere within the members name. The filter evaluation is case insensitive. For example, if the filter is set to `event`, the members `HandleEvent` and `DispatchEvent`, etc. are displayed.

JPEG Image File Support

Embedded Wizard contains now an infrastructure to support the loading of image files dynamically during runtime. This feature allows you to show images, which are stored on the hard-disc or which are received from a broadcaster.

The generic infrastructure is implemented within the new class `Vfs::ImageFile`. The concrete support for loading of JPEG files is implemented in the derived class `Vfs::JpegFile`. Images loaded by the new infrastructure can be shown directly by using one of the classes `Views::Image`, `Views::Wallpaper` or `Views::Frame`.

Please note, the necessary JPEG decoder is not part of the Embedded Wizard delivery. For demonstration purposes, the Win32 Platform Packages contains the TARA JPEG decoder library. Please contact TARA Systems for a ported target version of the JPEG decoder.

The usage of the JPEG image file support is described in the Mosaic User Manual and the new How To document 015 "Using Runtime Loadable Image Files as Resources".

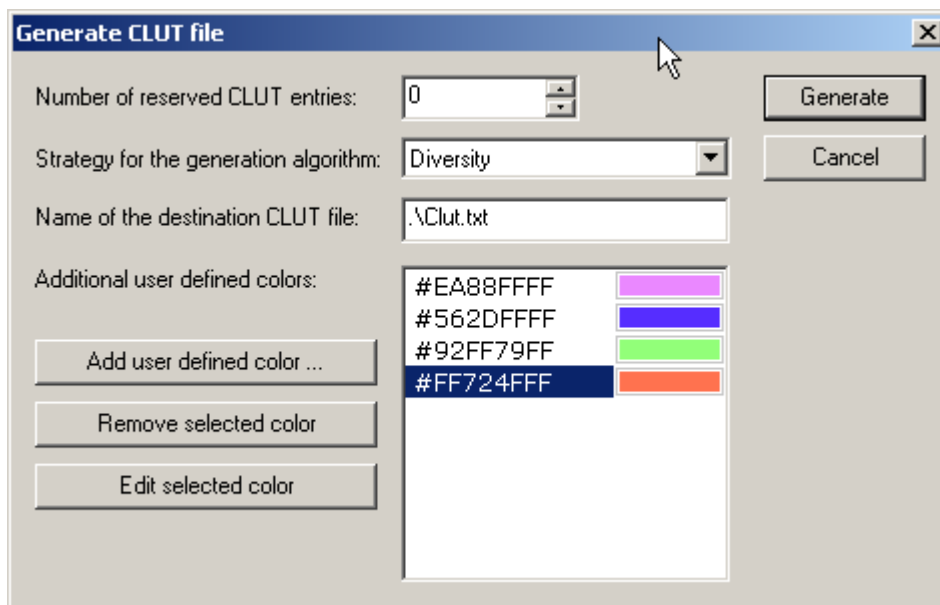
CLUT Generator

Embedded Wizard contains now an automatic CLUT generator, which creates the global palette for any Index8 platform containing the best fitting colors for a certain project.

The generation of the CLUT is done in two steps:

Within the first step, during a prototyping session, a color statistic file is generated, which contains a histogram of all colors and their occurrence. This prototyping session with color statistic can be seen as a learning phase for the following CLUT generation: the more GUI screens are displayed during the prototyping session, the more color values are added to the color statistic and the better results can be created from the CLUT generator. To start the color statistic generation, select the new menu item *'Start prototyper with color statistic ...'* from the *'Build'* menu.

Within the second step, the CLUT generator converts the color statistic of the current project into a CLUT file containing the 256 best fitting colors. This CLUT file can be used later within an Index8 Platform Package as a global color palette. To start the CLUT generation, select the new menu item *'Generate CLUT file ...'* from the *'Build'* menu. The CLUT generation can be configured within the CLUT generator dialog:



Within the second line of the dialog, the strategy for CLUT generation can be selected:

- The *'Majority'* strategy takes care on colors with a high occurrence, whereas colors with low occurrence are merged.
- The *'Diversity'* strategy takes care that all different colors are kept within the CLUT, independent from their real occurrence.
- The *'Popularity'* strategy takes care on all different colors, which are often used, and combines very different colors only if they are used rarely. This strategy can be seen as a compromise between Majority and Diversity.

Please note, the usage of the CLUT Generator requires an upgrade of your Embedded Wizard license.

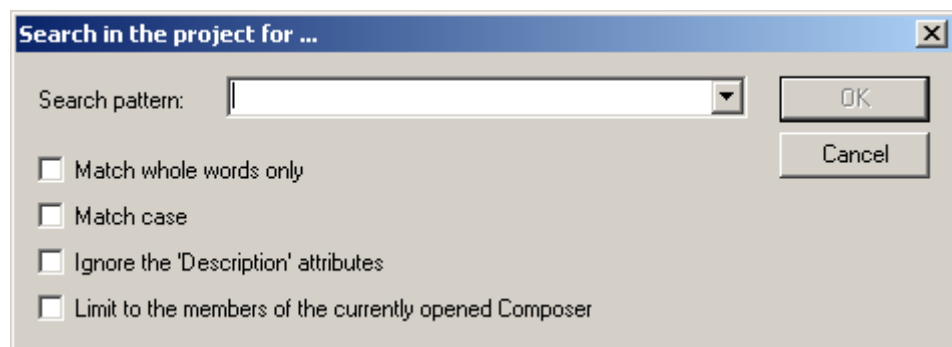
Minor Usability Improvements

Additionally, Embedded Wizard has been improved by the following features:

Search in Project

The search dialog within Embedded Wizard has been enhanced by additional options:

- The content of the **Description** attributes can be excluded from the search operation.
- The search operation can be limited to the members of the currently opened Composer.
- Additionally, the search dialog provides a history list with the recently used search patterns.



Description of Resource Attributes

The description area of the Inspector is now able to display the description of resource attributes, like **FontName**, **Quality**, etc.

Gallery Shortcuts

The navigation within the Template view of the Gallery window has been improved.

As long as the Templates view is active, you can press a key in order to open a folder, whose name starts with the pressed key. For example, if you want to open the folder *Panels*, press the key *P*. If there are more folders, which start with the sign *P*, press the key *P* more times until the desired folder has been reached.

In the same manner you can select a template within the opened folder. In order to distinguish between the shortcuts for the folder selection and the shortcuts for the template selection, the templates selection has always to start with the *Space* key. For example, you wish to select a *Push Button* template in the currently opened folder. In this case press one after another the keys *Space* and *P*. If there are more templates, which starts with *P*, press the key *P* more times until the desired template has been reached.

Browser Improvement

The list view of the Browser window and its Search edit field can now be controlled simultaneously without the necessity to toggle between them. This means, while a filter condition is edited, the *Up* and *Down* keys can be used to navigate within the list.

Log Window

The Log Window provides now the following additional features:

- The new shortcuts *F4* and *Shift+F4* allow you to quickly navigate to the next or previous log window message in order to inspect it.
- The log window supports now a copy to clipboard functionality. This command copies the entire content of the Log Window as text into the clipboard.

Icon Redesign

The set of delivered class icons has been enhanced. For example, the classes `Core::Root`, `Core::View`, `Core::Group`, `Core::Event`, etc. appear now with their own icons, instead of using the generic class icon.

Application Class Template

For your convenience, the Gallery contains now the template 'Application Class'. This template can be found in the folder *Forms*.

Mosaic

The Mosaic class library has been extended:

Views::Frame

The class `Views::Frame` provides now four additional properties `DrawLeftEdge`, `DrawRightEdge`, `DrawBottomEdge` and `DrawUpperEdge`. These properties determine whether the corresponding edge of the frame is drawn or not.

Panels::Label

The class `Panels::Label` has been enhanced by the property `AlignV`. This property determines optionally the vertical alignment of the label's caption or icon.

Documentation

The documentation has been extended:

Embedded Wizard User Manual

Beside of regular updates, the document contains a reworked 'Quick Tour' explaining the first steps with Embedded Wizard.

Tutorial Basic

The Tutorial Basic has been reworked. The document describes beside of some minor aspects, the following new topics:

- Usage of cursor events.
- Development of user defined controls. For this purpose, a LED view, a LCD view, a touch sensitive joystick control and a touch sensitive wheel control are developed and explained step by step.

Tutorial Widgets

The Tutorial Widgets describes now the usage of cursor events within the development of widgets.

How To's

The set of How To documents have been reworked. The new HowTo 015 describes the usage of runtime loadable JPEG images as resources.

Examples

The set of examples has been extended:

Robot Demo

The subdirectory \Examples\Robot contains a new example application demonstrating the development of sophisticated custom controls, like a joystick, a wheel control or a seven-segment display. The development steps for such kind of controls are described in detail within the Tutorial Basic.

Vfs Browser

The subdirectory \Examples\VfsBrowser contains a new example application demonstrating the usage of the VFS (virtual file system switch) functionality. With this application, the filenames of your hard-disc can be displayed within a list.

Platforms

The platform packages have been adapted:

extern const

All `const extern` declarations have been replaced by `extern const` declarations to avoid compiler warnings.

Unused Argument Warnings

The code generators have been adapted to avoid C compiler warnings concerning unused arguments.

Bug Fixes

The following bugs have been solved:

FontRange

Converting a font with a font range containing character code 0xFFFF, caused a hang-up of Embedded Wizard. The problem is solved now.

Mosaic

Due to an erroneous implementation of the method `Forms::Form.GetLowerCtrl()`, the navigation within a panel was not working properly in some cases. The problem is solved now.

Variant of class with no super-class

If a variant is derived from a class, which has no super-class, the instantiation of the affected class was not possible during runtime on the target. The problem is solved now.

Dongle Access

The access to the Embedded Wizard license key (dongle) is improved. In seldom cases, Embedded Wizard was unable to detect the USB dongle, caused by an internal defect within the dongle. The routines for accessing the dongle protection have now been improved, to furthermore reduce the probability of such effects. Moreover, we are constantly doing further analysis, to fully eliminate such errors in the future.

Embedded Wizard V4.40

Version V4.40 contains the following changes and improvements:

Cursor Events

Embedded Wizard supports now Cursor Events. This makes it possible to create UI applications which can be controlled by pointing devices like mouse or touch screen.

Event delivery

For this purpose, the Mosaic class library has been enhanced by two new event classes: `Core::CursorEvent` and `Core::DragEvent`. These events are sent to UI objects as soon as the user has clicked them with the mouse or touched them with the finger on the touch screen.

The dispatching and the delivery of cursor events is controlled by the class `Core::Root`. It implements the following new methods:

`DriveCursorHitting()` and `DriveCursorMovement()`, which have to be invoked from your main loop, whenever the user clicks with the mouse or touches with the finger on the screen. The Win32 Platform Package contains already the appropriate implementation to feed the UI application with the cursor events.

Widgets

The ready to use widgets (like Buttons, EditFields, MenuItems, etc.) have been adapted to be able to react on the cursor events. In this manner they already provide a standard behavior typical for the appropriate widget. If desired, this behavior can be modified by deriving new classes. For details see the description of the particular widget within the Mosaic User Manual.

Virtual keyboard

In case of devices, which do not provide a real keyboard, the Mosaic class library allows you to register a so called 'virtual keyboard'. Unlike a real keyboard, the virtual keyboard does only appear on the screen and can be used with the pointing device by the user. The registration and deregistration of a virtual keyboard is done by the new methods `AttachKeyboard()` and `DetachKeyboard()` of the class `Core::Root`. Alternatively it is possible to use an ordinary panel as virtual keyboard. For this purpose the panel's method `OpenAsKeyboard()` has to be used. For more details see the Mosaic User Manual.

Example

The subdirectory \Examples\CursorEvents contains a new widget based example application demonstrating the usage of cursor events and the virtual keyboard.

Additionally, the existing example applications (ChannelList, Game, HelpViewer, ListForm, Tetris and Widgets) have been adapted in order to operate with cursor events.

Windows Platform Packages

The set of Windows Platform Packages has been extended. The supported graphic formats are now RGBA 8:8:8:8, 8 bit gray with 8 bit alpha, 8 bit index format and the new RGBA 5:5:5:1 color format.

Update Service

Embedded Wizard checks now automatically, whether a newer version of the tool is available on the Download Center and informs the user by a message-box. This feature can be enabled and disabled by using the menu item 'Help' – 'Always check for latest version'.

During this version check, no customer related information is transferred (only current Embedded Wizard version number and the license number are sent in readable format).

Bug Fixes

The following bugs have been solved:

Auto objects

The generated code for auto objects was erroneous if all of the following conditions were true: (1) the class of an auto object is defined within a separate unit and (2) the auto object is not used within this unit and (3) there are no other relationships between the unit where the auto object is defined and the unit where the class of auto object is defined.

The problem is solved now.

Embedded Wizard V4.30

Version V4.30 contains the following changes and improvements:

Tutorial - Widgets

The new tutorial "Using Widgets, Panels and Menus in the Development of GUI Applications" is now available and part of the Embedded Wizard setup. The document and all examples are installed in the \Tutorial subdirectory.

Code Generator Improvement

The Code Generator of Embedded Wizard has been improved in order to write only the modified *.c and *.h files on the hard-disc. This avoids the complete recompilation of all C modules each time the code has been generated.

Chora

The Chora language has been adapted:

OutputDirectory attribute

The new profile attribute '**OutputDirectory**' can be used to define a destination directory for the generated code. The output directory can be either an absolute path or a location relative to the current project directory.

OutputPrefix attribute

The new profile attribute '**OutputPrefix**' specifies an optional prefix for the generated file names. The value of this attribute is only a suggestion for the Code Generator how the generated files should be named. If the Platform Package is not able to evaluate this attribute, the prefix will be ignored.

PostProcess attribute

The new profile attribute '**PostProcess**' can be defined to execute an external batch file or command directly after the code generation.

For more details see Chora User Manual.

Mosaic

The Mosaic class library has been extended:

Overwrite property

The classes `Panels::EditField` and `Menu:EditFieldItem` support now additionally the text input in overwrite mode. The new property `'Overwrite'` is used to switch between the modes: If this property is set `'true'`, the edit field will overwrite the signs at the caret position with the new text. If this property is set `'false'`, the new text is inserted at the caret position.

OutletOnApply property

The widgets of the Mosaic class library provide now the new property `OutletOnApply`. The property `OutletOnApply` determines when a widget will transfer its content to the controller specified in the property `Outlet`.

For more details see Mosaic User Manual.

Bug Fixes

The following bugs have been solved:

Escape sequences in string and char literals

The evaluation of `string` and `char` literals in Chora was erroneous for the following escape sequences: `\x0022` (double quote sign), `\x0027` (single quote sign) and `\x005C` (backslash sign). The usage of these escape sequences caused Chora compiler errors.

The problem is solved now.

Embedded Wizard V4.20

Version V4.20 contains the following changes and improvements:

Observer Infrastructure

Chora contains now a simple and powerful infrastructure for registering of observers and delivering of notifications to them when the observed subject has been signaled. This technique provides the basis for the Controller/View model and is used in the development of complex GUI applications.

For this purpose the following new statements are provided: **attachobserver**, **detachobserver** and **notifyobservers**.

For more details see Chora User Manual.

Auto Objects

Chora supports now the concept of auto objects. These auto objects are instantiated automatically on-request and are released, when they are not in use anymore. Unlike the ordinary Chora objects, auto objects do exist in the global scope of a unit, so they can be directly accessed from anywhere of the Chora code.

Usually auto objects do serve as 'controller' in the Controller/View model and are used in the development of complex GUI applications.

To create an auto object, drag and drop the objects' class into a unit while holding down the 'Ctrl' + 'Alt' keys.

For more details see Chora User Manual.

Mosaic

The Mosaic class library has been extended:

Outlet property

The widgets of the Mosaic class library provide now the new property `Outlet`. The property `Outlet` provides an interface for the Controller/View model. In the Controller/View model, the widgets (views) and the application logic (controllers) are always kept apart.

An automatism behind this model ensures, that widgets are notified automatically as soon as the affected controller has changed its state. On the other hand, user interactions on a widget cause the affected controller to execute the application logic. Usually, a controller is a simple Chora object containing several properties and the implementation of `onget/onset` method.

AutoActivate property

The behavior of the **AutoActivate** property of the widgets **EditField**, **Slider** and **SelectBox** has been enhanced. The automatic activation is performed only, if the property is set to **true** and the key events handled by the widget do not conflict with the key events necessary for the navigation within the Panel.

Typically, such conflicts occur when a widget reacts on the key events **'Left'**, **'Right'**, **'Up'**, **'Down'**, which are also used for navigation within a panel. Before a widget is able to become auto activated the arrangement of widgets within the panel is evaluated.

The property **AutoActivate** is now also available for the items **EditFieldItem**, **SliderItem** and **SelectBoxItem**.

AddBehind() and RestackBehind() methods

The class **Core::Group** has been enhanced by the methods **AddBehind()** and **RestackBehind()**. These functions simplifies the insertion and restacking of views within a group.

For more details see Mosaic User Manual.

Runtime Environment

The following improvements have been implemented:

Signals and timers

The function **EwProcessTimers()** returns now a value, which indicates whether at least one timer is expired. The new function **EwNextTimerExpiration()** evaluates the list of pending timers and returns the time span until the next timer expiration (please see **ewtimer.h** for details). The function **EwProcessSignals()** returns now a value, which indicates whether at least on signal has been delivered or not.

This improvement helps to optimize the platform integration.

Profiling

The profiling feature has been enhanced in order to monitor all memory allocations within the Platform Package: Chora objects, string objects, bitmaps, fonts and internal buffers. This feature is helpful to get an overview of all currently created objects and the resulting memory occupation within the target. To activate the profiling feature, recompile your application with the C-define **EW_ENABLE_PROFILER**.

To print the statistic report call the function **EwPrintProfilerStatistic()**.

Version check

The Graphics Engine (GE) and the Run Time Environment (RTE) contains now a version define. Within the generated code a preprocessor macro compares now the version of generated code with the version of GE / RTE. This helps to avoid version mismatch between the generated code and the used version of GE and RTE.

Platform Package ReadMe

The Platform Packages are now delivered with a read-me-file, containing the version history of Runtime Environment and Graphics Engine.

Bug Fixes

The following bugs have been solved:

Alignment of classes with float members

The usage of float variables or properties within a multi-variant class caused an access violation within the target system, if the resulting size of the class was not a multiple of 8 byte.

The problem is solved now.

Navigation within panels and menus

The navigation within panels and menus has been improved. In some cases, the behavior of the up, down, left, right navigation was unexpected.

Optimizer

If the optimization level was set to **Low** or **Medium**, unused wrapper functions were eliminated by the optimizer. This optimization causes C-compiler errors, if these functions were directly accessed from some native C-code.

The elimination of unused wrappers as well as members is performed only if the optimization level is set to **High**.

float to string conversion

The string formatted by the `string(float)` instant constructor was erroneous in the prototyping environment (leading zeros were missing).

The problem is solved now.

Documentation Generator

The search feature within the automatic generated HTML documentation did not work.

The problem is solved now.

string array memory leak

The elements of string arrays were not released correctly in the following case: If an object contains an array of strings and the object was disposed, only the first member of the string array was deleted - all other members of the string array were still kept in the memory.

The problem is solved now.

Embedded Wizard V4.11

Please read first all of the release notes of version 4.10, which contains a lot of features and improvements since the last official release 4.01!

Version V4.11 contains the following changes and improvements:

Warnings

The code generator has been adapted in order to avoid some compiler warnings:

`'break'` warning

If a `'case'` statement is exit by a `'return'` statement, then no `'break'` statement is generated. Also nested expressions are evaluated.

`'const static'` warning

All `'const static'` declarations are replaced by `'static const'` declarations.

`'typecast'` warnings

Additional typecasts are added.

Chora

The Chora language has been adapted:

`'parentthis'` operator

The new Chora `'parentthis'` operator allows an embedded object to access its superior object. If the object has not been embedded, the operator returns null.

[Remark: This operator replaces the `'parent'` operator introduced with V4.10. Due to some name conflicts, with existing projects, the operator is renamed.]

`'Generator'` attribute

If a property is defined without an `onget` or `onset` method, the automatically generated `onget/onset` methods inherit the value of the `'Generator'` attribute of the corresponding property now.

Mosaic

Following existing Mosaic classes has been extended:

Core::Root

The class has been extended by the method **EndModalOf()**. This method works similar to the already existing method **EndModal()**. The difference is, that the new method allows the user to exit the modal state of any modal form – not only the currently topmost modal form.

Documentation Generator

The arrangement of the classes within diagrams has been changed. Now, the derived classes appears on the left side, the base classes on the right side.

Intrinsic Loader

The loading of Embedded Wizard extension modules (so called Intrinsic) has been adapted: Now, the user can define an optional condition in order to determine for which environment the module is loaded. The condition is enclosed within brackets []. The condition can consist of the identifiers 'composer', 'prototyper', or a profile name.

Example:

Inaris[composer].ewi

→ loaded only into the Composer environment

Inaris[prototyper].ewi

→ loaded only into the Prototyper environment

Inaris[Win32].ewi

→ loaded in case of active 'Win32' profile

The name of the profile and identifiers 'composer' or 'prototyper' may be combined together by a plus sign '+' (logical AND condition). Several conditions can be separated by a comma sign ','.

Example:

Inaris[Win32+composer].ewi

→ loaded only into composer in case of active 'Win32' profile.

Inaris[Win32,Target].ewi

→ loaded in case of active 'Win32' or active 'Target' profile.

Bug Fixes

The following bugs have been solved:

Language and Styles IDs

Due to an error in the code generator, sometimes the IDs of Languages and Styles within the file ewlocale.h have not been counted beginning from 0. The problem is solved now.

EmWi access violation

In some seldom cases, the Embedded Wizard IDE causes an access violation when the currently active language or the style has been changed by the user. The problem is solved now.

Embedded Wizard V4.10

Version V4.10 contains the following changes and improvements:

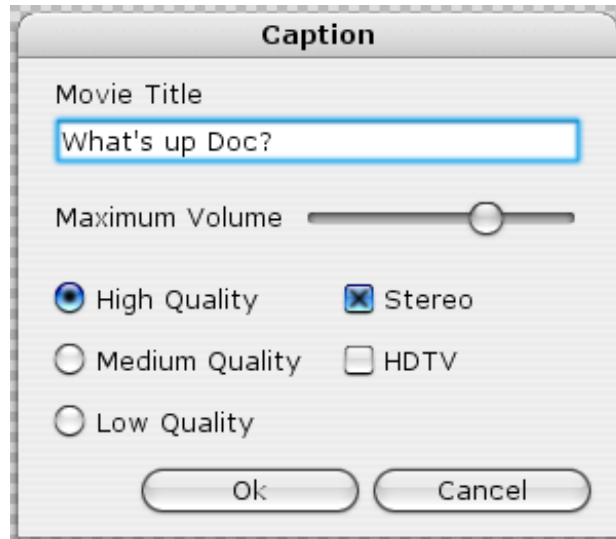
Widgets – Panels

The new unit 'Panels' of the Mosaic class library contains a set of predefined and ready-to-use controls and panels. These controls can be combined and arranged within a panel in an easy and comfortable way in order to build dialogs, control panels, message boxes, etc. The following widgets are implemented:

- Push Button
- Check Box
- Radio Button
- Edit Field
- Select Box
- Vertical and horizontal Slider
- Vertical and horizontal Scrollbar
- Label
- Panel
- Message Panel

The unit 'Panels' contains the common, appearance and behavior independent infrastructure for developing your own widgets. Hereby the infrastructure implements a generic functionality of the most important widget features, without a predefined appearance or behavior. By deriving variants from these widgets a concrete customer specific look&feel can be implemented. The infrastructure can also be used in order to implement new kinds of widgets. For details see 'Mosaic User Manual'.

The following picture gives you an impression of a panel with some controls:

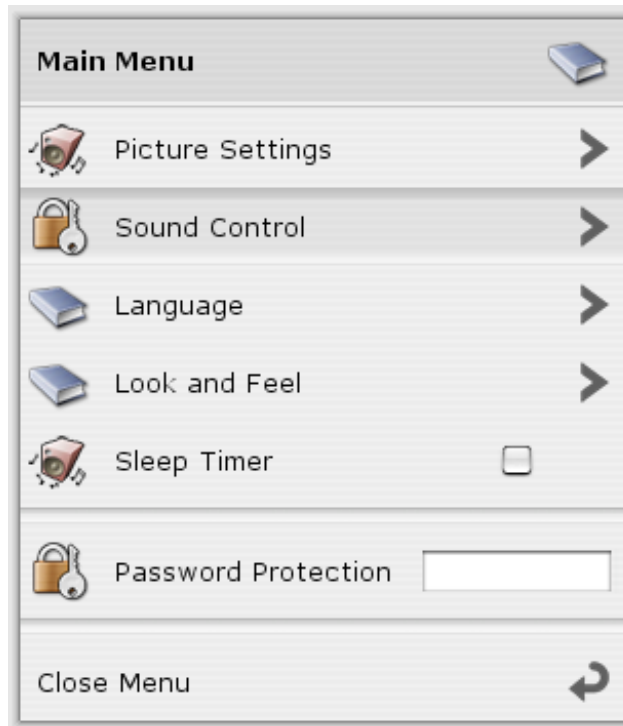


Widgets – Menus

The new unit 'Menus' of the Mosaic class library contains a set of predefined and ready-to-use menu items and menus. Unlike panels, menus perform an automatic arrangement of the menu items. The Gallery folder 'Menus' contains a set of templates for building menus. In order to build a menu, just drag the desired menu item templates from the gallery and drop them into the menu. All members of the menu items are then arranged accordingly to their Z-order. The following items and menus are available:

- Check Box Item
- Edit Field Item
- Select Box Item
- Command Item
- Open Menu Item
- Close Menu Item
- Separator Item
- Panel Menu
- Popup Menu

The unit 'Menus' contains the common, appearance and behavior independent infrastructure for developing your own menus. Hereby the infrastructure implements a generic functionality of the most important menu features, without a predefined appearance or behavior. By deriving variants from these menus and items a concrete customer specific look & feel can be implemented. The infrastructure can also be used in order to implement new kinds of menus and items. For details see 'Mosaic User Manual'. The following picture gives you an example of a menu with some menu items:



Widget skins 'Aqua' and 'Win95'

The Mosaic class library contains two widget implementations:

'Aqua' – an Apple OSX Aqua look and feel

'Win95' – a Microsoft Windows look and feel

Please note, that both skins are implemented only for learning and demonstration purposes to show the flexibility of widgets and variants. They can be used as an example and adapted to own designs. They are NOT intended for product usage!

The subdirectory 'Examples\Widgets\' contains a sample menu application with both skins.

Mosaic

Following existing Mosaic classes has been extended in order to provide additional functionality for the widgets:

Forms::Form

The class has been extended by the methods `GetLeft()`, `GetRight()`, `GetLower()`, `GetUpper()`, `SelectLeft()`, `SelectRight()`, `SelectLower()` and `SelectUpper()`. These methods exist for navigation purpose within panels and menus.

Forms::Ctrl

The class has been extended by the properties **Enabled**, **HotKey** and **Label**. The property **Enabled** determines, whether the control is able to react to user inputs or not. The property **HotKey** specifies an optional key code, the control may react to automatically. With the property **Label**, a label object can be assigned to the control. This label can then be used to display a caption for the control.

Additionally the class **Forms::Ctrl** has been extended by the method **Activate()** and **UpdateCtrlState()**. The method **Activate()** is invoked, when the control become activated by the user. The method **UpdateCtrlState()** Is invoked, when the control's state is about to be changed. The available states are defined in the enum **Forms::CtrlState**.

Core::View

This class has been extended by the method **HandleSelection()**. This method is invoked automatically, when the view is about to become selected or unselected. Usually, this occurs, when the view obtains or loses the focus. But unlike **HandleFocus()** the view retains the selected state although the focus state is lost.

Core::Group

The class has been extended by the method **GetHelpString()** and **UpdateHelpString()**. These methods provide a simple protocol for displaying and updating of help strings within menus or panels.

Core::Root

The behavior of the root class has been changed. When an event is send to a modal form, and the event has not been handled by the form, the event is passed to the **HandleEvent()** method of the root class. If this behavior is unwanted, adapt the **HandleEvent()** method and evaluate the variable 'forms' in order to suppress further event processing.

Core::KeyCode

The enumeration has been extended by the code for ASCII characters A – Z, Space: **KeyA** – **KeyZ**, **Space**. These key codes are predestined to specify hot keys of controls.

Since these enhancements of the Mosaic classes may conflict with some methods or properties of your derived classes, the previous version of the Mosaic class library is installed additionally in the directory 'MosaicV400'. Please verify your software with the new version. In case of name conflicts, we recommend to resolve these conflicts by renaming the affected members within your classes. Alternatively, the previous version of the Mosaic class library can be used for existing projects – without using the new features like menus or panels. To use the old version, set the attribute 'Directory' of all Mosaic units to '\$MosaicV400' instead of '\$Mosaic' and reload the project.

Chora

The Chora language has been enhanced:

'parent' operator

The new Chora '**parent**' operator allows an embedded object to access its superior object. If the object has not been embedded, the operator returns **null**.

'class' data type

The '**class**' data type is able to store a reference to a Chora class. This reference can then be used in order to create objects of this class dynamically by using the '**new**' operator.

'classof' operator

The '**classof**' operator determines the class of an object at the runtime. The determined class can then be passed to a '**new**' operator in order to create an object of the same class.

For details see 'Chora User Manual'.

New Features of Bitmap Resources

The bitmap resources have been enhanced by the following features:

FrameSize

The new attribute **FrameSize** is useful in case of a bitmap, which consists of several small images (frames). Multi-frame bitmaps can serve as image-lists or they can contain short animation sequences. Within a bitmap, the frames are arranged in tiles, side by side. This attribute specifies the size of a single frame in pixel.

FrameDelay

The new attribute **FrameDelay** is useful in case of a bitmap, which contains an animation sequence. The attribute specifies the delay in milliseconds between two animation frames.

Accordingly to this extension the class `Resources::Bitmap` has been extended by the new properties `FrameSize`, `FrameDelay`, `NoOfFrames` and `IsAnimated`.

To use these features, it is necessary to replace an existing bitmap resource brick with a new bitmap resource brick from the Gallery. For more details about this new feature see 'Chora User Manual'.

The classes `Views::Image`, `Views::Wallpaper` and `Views::Frame` has been extended to the ability to display frames of bitmaps and animated sequences. For this purpose, the new properties `Animated` and `FrameNumber` have been added to these classes. For more details about this new feature see 'Mosaic User Manual'. The old classes `Views::AnimatedImage` and `Views::ImageList` are obsolete now, but still available for compatibility.

Due to the new image file decoder (see below) the bitmap resource converter supports JPG and GIF image file formats now. These can be set in the attribute `FileName` of a bitmap resource. (Please note, that these additional formats are used as an additional source for the bitmap resource converter.)

Optimizations

The following optimizations have been done:

Invalidate()

The `Invalidate()` method of `Core::Group` has been optimized to avoid superfluous update requests.

Wallpaper and Frame

The drawing of Wallpapers and Frames has been accelerated.

Composer Navigation

The navigation within the Composer Window has been improved:

After closing a Composer Page, the Composer returns to the previously opened page automatically.

If necessary, the Composer content is scrolled automatically in order to ensure, that a member appears within the visible area of the Composer. This helps e.g., to find a member within the composer after selecting the corresponding member within the Browser.

Bug Fixes

The following bugs have been solved:

Long Enum Names

Due to an error in the Code Generator, enumerators with long names have been translated in erroneous `#define` directives. The problem is solved now.

'char + string' concate operator

If the operator has been applied on an empty string, the operands have not be concatenated properly and an erroneous string has been returned. The problem is solved now.

New properties in class variants

Properties added to a class variant have not been handled correctly, if the `onget` or `onset` method of the property was not implemented explicitly. The problem is solved now.

New image file decoder

The used PNG decoder was not able to decode every PNG file (especially, small PNG images with transparency causes sometimes errors). The entire decoder is replaced and the problems should be solved now.

Embedded Wizard V4.01

Version V4.01 contains the following changes and improvements:

Bug Fixes

The following bugs have been solved:

Font Ranges and Aspect Ratio

If a font resource consists of several font ranges and if the aspect ratio was set to a value different than 1, the Prototyper or the code generator of Embedded Wizard V4.00 may cause an access violation. The problem is solved now.

Font Cache

The Graphics Engine V4.00 did not find the cached glyphs within the font cache. Thus, all glyphs are loaded even they are already available within the cache. The problem is not critical since it only reduces the performance. The problem is solved now within all Platform Packages.

Embedded Wizard V4.00

Version V4.00 contains the following changes and improvements:

Variants of classes, resources and constants

Embedded Wizard supports now a very powerful feature, called 'Variants', which is very helpful to create OSD applications for different screen resolutions, to implement different skins or themes, or to simplify the customization of an existing reference software. The concept of variants is based on the concept of object-oriented inheritance and can be applied to classes, resources or constants. Each of them can exist in several versions, with a modified or added content or implementation. The selection of the appropriate version is either done at code generation time ("static variant") or dynamically during runtime ("dynamic variant"). The advantage of this approach is, that a software still accesses an origin class, although a variant of it is used. This makes it possible to add a new appearance or behavior to an existing software without any modifications on it!

The derivation of variants is as simple as the derivation of classes: To build a variant of a resource, constant or class just hold the keys 'Ctrl' + 'Shift' + 'Alt' while dragging the affected brick. In the Composer and Inspector all variants are visualized by an additional 'V' icon:

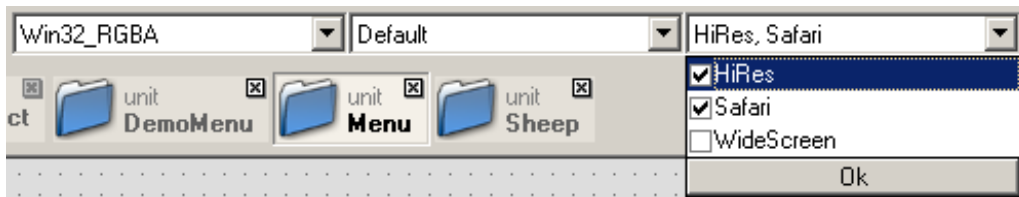


The selection of a variant is controlled by the `VariantCond` attribute in the Inspector. If this attribute is set to `true`, the variant overrides the origin class, resource or constant statically and permanently ("static variant"). If this attribute is set to the name of a profile, the variant is only used in case of generating code for this profile ("static variant"). However, if the attribute contains the name of a style, the selection is performed during runtime if the affected style is active ("dynamic variant").

Styles can be compared with flags which can be set during runtime to control the selection of dynamic variants. The user can define several styles to control for example the screen resolution, the currently active skin or to activate special features. For this purpose, a new template 'Style' is available within the Gallery. Styles can be defined only within the project Composer, in the same way as Languages.



To switch the different styles on or off, the new built-in variable `'styles'` can be set during runtime. In the Embedded Wizard IDE, the selection of the currently active styles can be done at design time within the new combo-box `'Styles'`.

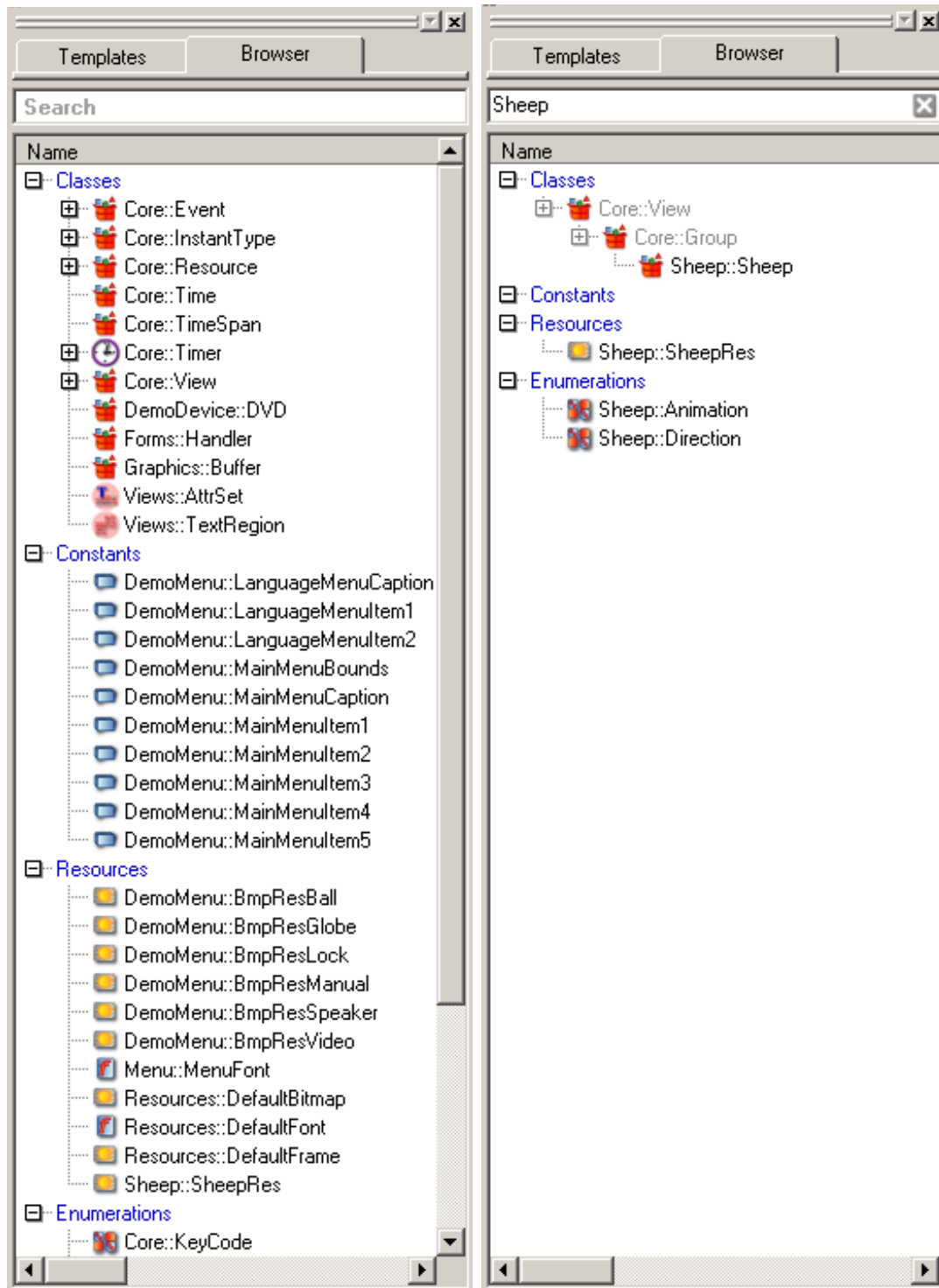


For more details see Chora User Manual.

Browser

The Gallery window is enhanced by a new view called `'Browser'`. The Browser gives an overview about the project structure. Here you can find a list of all constants, resources, enumerations and a tree of all classes. The Browser can be used to navigate within the project. Additionally, the members displayed within the Browser can be used for any `'Drag & Drop'` operation, e.g. to derive a new class or variant, or to create an object of a class. Additionally, the Browser view supports a filter. For this purpose the edit field `'Search'` can be used to define a filter condition. Wildcards are supported.

For details see `"Embedded Wizard User Manual"`.



Debugger

The Debugger is adapted to the new concept of 'Variants'. All members of class variants are grouped together. The applied class variants are accessible and can be inspected in all Debugger windows.

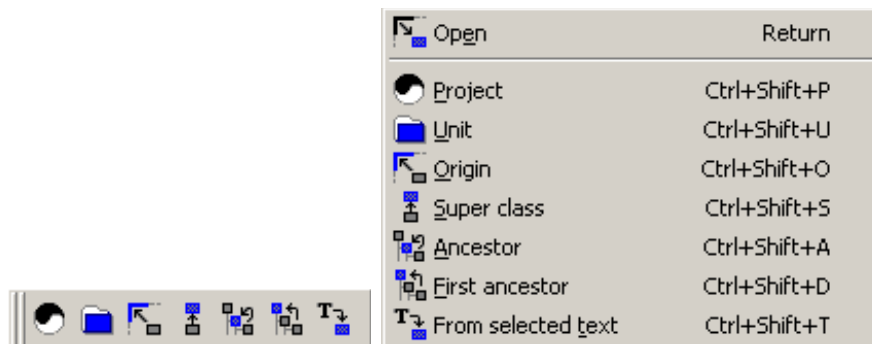
Macros

The usage of macros in the derivation of classes or in the type definition of objects, variables etc. is no more supported. For this purpose the new feature 'Variants' is available now. To ensure the compatibility with older versions, the macros are still accepted in these special cases. However, the evaluation of the macros is done during the code generation only. Additionally, a warning is reported.

For all other purposes (e.g. initialization of variables, within a method, ...) macros are still supported and can be used without any limitation.

Navigation

The navigation within Embedded Wizard is improved and enhanced according the concept of 'Variants'. The old set of 'PUSDITC' navigation commands is replaced by the following set of navigation commands:



The navigation commands can now applied also to unit members in order to find origins of variants or super classes, etc.

For details concerning the new navigation see "Embedded Wizard User Manual".

The command 'Open' can now also be used to inspect the class of an object. A simple double-click on the brick of an object navigates also to the implementation of the object's class.

Edit menu

The edit menu has been enhanced by two new commands 'Override class member' and 'Restore default value'.

For details see "Embedded Wizard User Manual".

Composer and Inspector

The visualization of members within the Composer and Inspector is improved:

Erroneous derived classes or variants are displayed with red text color to indicate the problem. In case of resources, a question mark icon is displayed. Initialization values for non-existing properties or language variants are displayed also in red within the Inspector.

Up to now, all derived classes and their objects are visualized with a standard icon. To improve the recognition within the composer different icons are used. In case of derived classes and variants, the icon of the base class is used automatically.

Font resources with 'Italic' and 'Bold'

To simplify the usage of the fonts, the font resources contains now the new attributes '**Italic**' and '**Bold**'. These attributes can be set to '**true**' or '**false**' in order to import italic and/or bold versions of a font. To use this feature, it is necessary to replace an existing font resource brick with a new font resource brick from the Gallery.

Font quality

If a font contains several language variants and these variants uses different quality levels, the resulting quality of the converted font is determined by the best quality. In previous versions only the quality of the default version was used.

Core::**Root**

The class **Core::**Root**** contains now the property '**Styles**'. This property corresponds to the build in Chora variable '**styles**' and can be also used to change the currently active styles. When using this property, a **Core::**StylesEvent**** is sent additionally via broadcast to all Views in order to inform them about the new settings.

Views::**Frame**

The new mosaic class **Views::**Frame**** can be used to display a decorated bitmap frame. The basis for the decorated frame is a single bitmap resource, which consist of three rows and three columns. The single tiles are used to draw the corners, the edges and the interior area. To draw the decorated frame with the desired size, the different areas are filled with the corresponding tile of the bitmap. The new class **Views::**Frame**** can be used to build buttons, backgrounds of menus or dialogs, etc. For your convenience the Gallery folder 'Views' contains a new template. The following image shows an example of the bitmap resource and the resulting frame.



Welcome splash screen

The welcome splash screen is no more a topmost window, the user can continue working with other applications during the start of Embedded Wizard.

Examples

The installation of Embedded Wizard contains two more examples:

Applet

This example demonstrate the usage of the class `Views::Applet` in order to integrate an external C application into an Embedded Wizard application.

ChannelScan

This example demonstrate the implementation of a channel scan dialog.

Optimizations

The following optimizations have been done:

Duplicates of resources and constants

During code generation, the language variants of resources and constants are eliminated if they are equal to their default variant.

Prototyper

The performance of the Prototyper is increased by 10% to 20%.

Bug Fixes

The following bugs have been solved:

Undo within Gallery

The undo after rename or delete operation did not work properly in previous versions. The problem is solved now.

Renaming of objects

In some cases, the name of an object within a class was not changed properly after renaming the object within the Inspector.

Endless recursion in initialization of constants

If a constant refers to itself within the initialization expression, an error message is reported to avoid endless recursion.

Embedded Wizard V3.30

Version V3.30 contains the following changes and improvements:

Code Generator Optimization

Embedded Wizard supports now several levels of optimization for the generated source code. Depending on the set optimization level, Embedded Wizard analyses the dependencies between the members of the project. It evaluates the logic of methods and determines, which members are unused or can be accessed in a more efficient way. Then in the second pass the unused members are eliminated and the logic of the methods is adapted.

Depending on the complexity and structure of your application, the binary size of the generated code may reduce to 90% ... 50%. To activate the optimization, set the new attribute `'Optimization'` of the profile to one of the values `'Low'`, `'Medium'` or `'High'`. By default, the optimization is set to `'None'`.

Please note, that during the optimization parts of the generated code are modified or eliminated. This may cause problems, if your C application accesses directly the generated Chora code (methods, variables, constants, classes...). To avoid these problems, you should set the attribute `'Generator'` of all affected members to `'true'`. Otherwise, the C compiler or linker may report confusing errors due to missing declarations or implementations.

For details see "Chora User Manual".

Generator Attribute

The attribute `'Generator'` is used to force the code generation of a class, method, variable, etc. (even it is not used from the Chora application). Now, the attribute is extended to evaluate more complex conditions instead of only `'false'` or `'true'`: It is allowed to set the attribute to a profile name to make the code generation depending on the selected profile. Additionally, the attribute accepts a list of conditions consisting of profile names or macros. In this case, the code generation is forced, if already one of the conditions is fulfilled.

For details see "Chora User Manual".

Runtime Environment Optimization 1

The global Variable `'EwThis'` within the file `ewrte.h` has been eliminated. Instead, the object pointer is now passed in the first argument `'_this'` of the invoked method. This modification makes the generation of optimized code (see above) possible and simplifies the wrapper functions. The compatibility to older version is provided by the new C macro `'EwThis'`.

In seldom cases, when your C application directly calls a Chora method without the usage of the corresponding wrapper function, the method call has to be enhanced by the additional argument `'_this'`. Please note, that calling a Chora method without using the wrapper function is generally not recommended.

Runtime Environment Optimization 2

The declaration of the data types `XBool` and `XEnum` within the file `ewrte.h` has been changed to be stored more compact within 1 or 2 bytes.

If the optimization level is set to `'Medium'` or `'High'`, the layout of the generated Chora objects is optimized by arranging all small variables close together. This results in a reduction of the binary size of Chora objects to approx. 80%.

Fonts with Aspect Ratio

The new attribute `'AspectRatio'` can be used to change the width-to-height ratio of a font resources. To use this feature, it is necessary to replace an existing font resource brick with a new font resource brick from the Gallery.

For more details see Chora User Manual.

ScreenSize and Clut

The profile definition has been extended by the new attributes `'ScreenSize'` and `'Clut'`. These attributes serve as replacements for the old macros `$ScreenSize` and `$Clut`. In this manner, it is now possible to change the settings more comfortable directly in the Inspector window, when the appropriate profile brick is selected.

The old macros are converted automatically into the new attributes, when an old project is loaded.

The size of the Prototyper window is now adapted automatically to the size defined in the `'ScreenSize'` attribute of the currently selected profile.

Additionally, the code generator writes now a global constant `'EwScreenSize'`, which can be used directly from the C application for the display initialization. This constant is declared in the module `Core.h`:

```
const XPoint EwScreenSize = { width, height };
```

Abort of Code Generation

Now, the code generation process can be aborted immediately after the `'Cancel'` button is pressed.

Scrolling in Log Window

The scrolling behavior within the Log Window has been improved.

Profiling within Target System

The runtime environment provides now profiling capabilities directly in the target system. The profiler monitors all existing Chora objects and prints a statistic report. This feature is helpful to get an overview of all currently created objects and the resulting memory occupation. To enable this feature, recompile your application with the C-define **EW_ENABLE_PROFILER**.

Head and Foot Templates

User defined templates head.ewt and foot.ewt can now be stored directly within the project directory. These templates are copied by the code generator at the beginning and the end of each generated source code file. A default version of these templates is available within the \platforms subdirectory.

CLUT Management

In all index8 target systems it is now possible for an external C application to allocate colors for private usage. This feature is useful for external applications like a computer game or a teletext application to store a set of private colors within the hardware CLUT. Since the CLUT has to be shared with the Embedded Wizard global palette, only color entries can be allocated which are not already occupied by the Embedded Wizard application. The Embedded Wizard application itself is not affected from the usage of these private colors. A set of functions provides the new feature:

```
GenAllocUserColor()  
GenFreeUserColor()  
GenSetUserColor()  
GenFindUserColor()
```

The declaration and the description of these functions can be found in the file 'GenColor.h'.

Display Initialization

The module 'GenDisplay.c' contains now a template for the initialization of the framebuffer for the particular target system. The new functions **GenInitDisplay()** and **GenDoneDisplay()** contains a default initialization and deinitialization of the framebuffer.

Embedded Wizard V3.20

Version V3.20 contains the following changes and improvements:

Attributed Strings

Embedded Wizard supports now attributed strings. Unlike simple text drawing operations, attributed text may appear with multiple fonts, colors and images - like HTML. The appearance of the text is controlled by attributes inserted within the text. A set of special attributes allows the arrangement of the text within columns and paragraphs.

This new service is available by using the new classes `Views::AttrSet` and `Views::AttrText`.

For details see "Mosaic User Manual".

To display attributed strings on the target system, the new module 'GenAttrText.c' has to be added to your project.

postsignal statement

The new `postsignal` statement records a pending signal to a selected slot of an object. The signal is delivered always immediately before the screen update is performed. Unlike the ordinary `signal` statement, `postsignal` is suited for deferred code execution, for example to finalize some initialization steps.

For details see "Chora User Manual".

idlesignal statement

The new `idlesignal` statement records a pending signal to a selected slot of an object. The signal is delivered always after the screen update is performed and no user inputs are waiting for execution. Unlike the ordinary `signal` statement, `idlesignal` is suited for deferred code execution and for performing of idle tasks.

For details see "Chora User Manual".

main message loop

The new statements '`postsignal`' and '`idlesignal`' expect a small adaptation of the main message loop within the OSD task to process the pending signals. If the adaptation is missing, your application still works - however, the pending signals are not delivered.

In most cases, it is enough to add a call to the new function `EwProcessSignals()` immediately before the update of the screen is performed. This function is responsible to process all pending signals.

The following example demonstrates the usage of **EwProcessSignals()** within a message loop, which is endless executed:

```
void OSD_Task()
{
    ... init
    // The message loop
    for (;;)
    {
        ... get the IR command
        ... dispatch the IR command

        EwProcessTimers();

        EwProcessSignals();    <-- the new function!!!

        ... update screen

        EwReclaimMemory();
    }
    ...
}
```

In case of an OSD task, which suspended by the operating system until the next event is arrived, the adaptation should correspond to the following example:

```
void OSD_Task()
{
    ... init
    // The message loop
    for (;;)
    {
        ... wait for the next IR command or timer events
        ... dispatch the IR command

        EwProcessTimers();

        do <-- a new do/while loop to handle 'idle' signals
        {
            EwProcessSignals();

            ... update screen

            EwReclaimMemory();
        }
        while ( EwAnyPendingSignals() && !( any IR commands )
            && !( any timer events ) );
    }
    ...
}
```



```
}

```

The second function `EwAnyPendingSignals()` determines whether any signals are pending or not. As long as the function returns a nonzero value waiting signals are pending. In this case, the OSD task should not be suspended. This is handled within the condition of the do/while loop.

The following example demonstrates the adaptation within the Win32 environment:

```

/* Process MS-Windows messages */
while( GetMessage( &msg, NULL, 0, 0 ))
{
    TranslateMessage( &msg );
    DispatchMessage( &msg );

    /* Process expired timers */
    EwProcessTimers();

    /* The following do/while loop is responsible for the
       idle processing. This loop is executed as long as any
       signals are pending and no other user inputs are received
       by the Viewer application. */
    do
    {
        /* Process the pending signals */
        EwProcessSignals();

        /* Refresh the screen, if something has changed and draw
           its content in the viewer window */
        EwUpdateViewer();

        /* After each processed message start garbage collection */
        EwReclaimMemory();
    }
    while ( EwAnyPendingSignals() && !GetQueueStatus(QS_ALLEVENTS));
}

```

native statement

The new **native** statement allows the programmer to insert native code, for example calls to 'C' functions, directly into the body of a method. In this manner, the Chora statements and the native statements can be mixed together within one and the same method. For details see "Chora User Manual".

Tutorial

The installation contains now the new tutorial "Developing GUI Applications with Embedded Wizard".

Examples

Embedded Wizard is now delivered with some additional examples. They can be found within the subdirectory \Examples.

Applet Class

The new class `Views::Applet` implements an environment for the integration of external applications, already existing for the target system. For example an existing computer game, an image viewer or a teletext application written in 'C' may be integrated into the Embedded Wizard environment by deriving and customization of this `Views::Applet` class.

For details see "Mosaic User Manual".

Buffer Class

The new class `Graphics::Buffer` implements an platform independent 8 bit bitmap, to be used as off-screen buffer for external applications (integrated by using the `Views::Applet` class). The content of this bitmap can be drawn into a Canvas by using the new method `Graphics::Canvas.DrawBuffer()`.

Update Optimization

The classes `Forms::Form`, `Forms::ScrollForm` and `Forms::ListForm` have been adapted to the new Chora statement `postsignal`. This modification improves the performance, the update of attached scrollbars, the scrolling within `Forms::ScrollForm` and the reloading of `Forms::ListForm`. Unnecessary updates are avoided now. As a consequence, the following methods are eliminated:

```
Forms::Form.UpdateScrollBar()  
Forms::ScrollForm.UpdateScrollBars()  
Forms::ListForm.DoMoveCursor()  
Forms::ListForm.UpdateIndicators()
```

ListForm Initialization

The first loading of the items within a `Forms::ListForm` is now triggered automatically after the initialization of the `Forms::ListForm`. Thus, it is no more necessary to implement an `Init()` constructor to force the first `Reload()` call. Since the class `Forms::ListForm` contains now its own `Init()` implementation, please remove your version of the `Init()` method - otherwise, a Chora warning is displayed.

Chora Macros

The usage of an undefined (non-existing) macro within the Chora code does not cause a Chora error message. Instead, a warning is displayed and Embedded Wizard assumes, that the macro is empty.

Init() constructors

The new version of the Chora compiler accepts now `Init()` constructors without the `uint32` argument. This argument is only for internal purposes. It is recommended to implement new `Init()` constructors without the additional argument. Of course, Embedded Wizard is still compatible and accepts the old declaration.

Boxing classes

The boxing classes to store the basic Chora data types (e.g. `int32`, `string`, `rect`, `color`, ...) are renamed and moved from the unit `Forms` to the unit `Core` to be used for general purposes. For example the class `Forms::StringCourier` is now available as `Core::String`.

\$ScreenSize

The new macro `$ScreenSize` can be defined within a profile to specify the size of the prototyping window. By default, the size is currently set to `<720, 576>`. This macro expects a point literal, e.g. `<1024, 768>`.

Bug Fixes:

The following bugs have been solved:

`char - char' operator

The operator was implemented erroneously within `Prototyper`.

ReInit()

The `ReInit()` re-constructor was not generated properly, which causes compiler errors.

EwNewStringAnsi()

The Runtime Environment function did not convert character in the range from 0x80 to 0xFF correctly.

Fonts within Prototyper

The problem, that sometimes incorrect characters are displayed within the prototyping environment, is solved now.

Statusbar

In some cases, the update of the status bar on bottom of the Embedded Wizard IDE was missing. The problem is solved now.

Core::Time

Within the Win32 environment and within the Prototyper, the current time did not take the time zone and the daylight saving mode into consideration. The problem is solved now.

Access Violation

Embedded Wizard causes an access violation if a string constant exceeds 8192 character. The size of string constants is unlimited now.

const string

In some cases, the release of string constants was handled erroneously by the garbage collector: If an OSD application is using more than 64 kByte of Chora string constants, it was possible, that a loaded string constant was discarded undesirably by the garbage collector.

Core::Root

The transparent rectangle 'Background' within the class `Core::Root` is now adjusted to the size of the root object automatically.

Composer

Within the Composer all views are displayed independent from their 'visible' property. Thus, the user is able to select and arrange also all invisible views with the mouse. However, this behavior was also used for all nested (embedded) views – even they can not be selected by using the mouse. Since this might confuse the user, the behavior is now limited to all views placed directly within the Composer.

XFloat

The declaration of the `XFloat` data type in the runtime environment for the Win32 platform was changed from float to double, to avoid 'C' compiler warnings.

Embedded Wizard V3.00

Version V3.00 contains the following changes and improvements:

Debugger

Embedded Wizard contains now a powerful integrated debugging environment. Breakpoints can be set, applications can be executed in single step mode and program execution can be interrupted every time. Several monitor windows informs you about the call stack, the local variables and all existing objects.

During prototyping the 'Garbage Collector' window can be displayed. It shows a statistic of the used objects, how many objects are alive and what happened during last garbage collection. This is very useful to observe the amount of dynamically created objects during runtime. The debugger settings are stored within the project session file.

For details see "Embedded Wizard User Manual".

Key Events

The support of Mosaic key events is extended. The enumeration `Core::KeyCode` contains now a rich set of predefined key events commonly used in CE devices (e.g. Play, Record, EPG, Power, Help,...). For all other purposes a set of user defined key codes (User0 – User19) is available. The usage of these key codes makes it possible, that the complete application can be executed and tested within the Prototyper. For this purpose, the Prototyper translates PC keyboard inputs into Mosaic key events. Please note, that some of the previously used keyboard shortcuts are changed. A list of all available key codes and corresponding shortcuts appears after holding down the *Ctrl* key within the Prototyper for a few seconds.

Additionally, the class `Core::KeyEvent` is extended by the variable `CharCode`, which can be used to feed the application with alpha numeric inputs.

For details see 'Mosaic User Manual – Core::KeyEvent'.

Key Events in Win32

The Win32 platform packages contains now the modules `ewapp.c` and `ewapp.h`. These modules covers some common software parts used for all Embedded Wizard applications running on the Win32 target. This simplifies the creation of a Windows application and ensures, that the same key codes and corresponding shortcuts are supported as in the prototyping environment. Thus, the necessary manually written code for a complete Windows application can be reduced to the following lines of code:

```
#include "ewapp.h"
#include "DemoMenu.h"

int APIENTRY WinMain( HINSTANCE aInstance,
    HINSTANCE aPrevInstance, LPSTR aCmdLine, int aCmdShow )
{
    XPoint          size = { 720, 576 };
    MSG             msg;
    DemoMenuApplication rootObject;

    EwOpenConsole(); // opens the console window
    GenInitViewer( size.X, size.Y );
    GenInitGlyphCache( 10000 );

    rootObject = EwNewObject( DemoMenuApplication, 0 );
    EwLockObject( rootObject );
    CoreRoot__Initialize( rootObject, size );

    EwOpenViewer( rootObject ); // opens the viewer window

    while( GetMessage( &msg, NULL, 0, 0 ) )
    {
        TranslateMessage( &msg );
        DispatchMessage( &msg );

        EwProcessTimers();
        EwUpdateViewer(); // updates the viewer window
        EwReclaimMemory();
    }

    EwUnlockObject( rootObject );
    EwReclaimMemory();
    GenDoneGlyphCache();
    GenDoneViewer( 0 );

    return msg.wParam;
}
```

For details see inline comments in ewapp.h and ewapp.c.

Display Update Optimization

The display update cycle after each key or timer event was optimized. The display areas that have to be updated are divided now in up to three separate areas instead of one big update area. This mechanism improves especially the update cycle if only small areas have to be updated, that are far away from each other.

Chora Extensions

The Chora data type `'rect'` was extended by the following operators:

Instant Property `rect.center`

The new instant property `rect.center` returns the center of a rectangle.

Operator `'&&'`

The `rect && rect` operation builds an intersection of two rectangles. Unlike the already available `rect & rect` operator, no intersection is build if one of the rectangles is empty and the operator returns the second rectangle. This avoids unnecessary Chora `if` statements.

Operator `'=='` and `'!=='`

The behavior of the existing operators `rect == rect` and `rect != rect` have changed. Now, rectangles with negative width or height are supported. The previous implementation handles them as empty rectangles.

Prototyper Optimization

An optimization within the prototyper increases the prototyping execution speed up to 25%.

Shortcuts

The shortcuts for 'Build this profile' and 'Build in batch mode...' have been changed from `F9` to `F8` and from `Shift+F9` to `Shift+F8`.

Ellipsis (...)

The classes `Views::Text` and `Views::MultilineText` support now the property Ellipsis. If the property is set to true, the leading or trailing part of a displayed text is replaced by "... " signs in order to ensure, that the displayed text fits into the views' bounds.

Dockable Windows

The behavior of all dock-able windows has changed. The moving of floating windows does not force them to dock into the Embedded Wizard main window automatically. To dock them, it is necessary to press the `Ctrl` key.

Out of Memory Panic

If the creation of a resource is failed, the system does not hang in the function `EwPanic()` anymore. The error is reported by the Graphics Engine and the execution can be continued.

Bug fixes:

The following bugs have been solved:

Char literals

The char literals `'\0'` and `'\x0'` are now handled correctly by the Chora compiler.

Warnings

Unused arguments within `Init` constructor methods are no more reported as warnings.

Reload class

The Composer update problem concerning the command `'Reload Class'` is solved now.

Embedded Wizard V2.30

Version V2.30 contains the following changes and improvements:

Language Selection

To simplify the implementation of multi-lingual OSD applications, Embedded Wizard supports now an automatic re-initialization of existing objects after the language has been changed.

The Chora compiler is able to generate the necessary re-initialization code. The language switch will then trigger the objects to re-evaluate their language dependent initialization expressions and to redraw themselves. This is done automatically in a way fully transparent to the programmer.

For details see 'Chora User Manual – Language Selection'.

Unit Splitting

The unit splitting improves the simultaneous development of large units within a team of developers. A unit file can be splitted in several include files, one file for each constant, resource, class, etc. definition. Include files, which belong to a unit are stored within a directory with the name of the affected unit. While loading a project, Embedded Wizard determines, whether a unit is stored as a single file or as a directory consisting of several include files. The unit splitting is controlled by the unit attribute 'Splitted'.

Documentation Generator

Embedded Wizard contains now a full automatic documentation generator. Embedded Wizard evaluates the content of your project, the dependencies between the classes, constants, resources, etc. and stores it together with the associated inline documentation within a set of HTML files. In a second step a single MS-Windows help file is created. This help file contains detailed documentation of your project and it can be used or distributed as the 'software reference manual'. To start the documentation generator, select the menu item '*Extract Documentation...*' from the '*Build*' menu.

For details see "Chora User Manual – Comments".

Zoom-In

The Zoom-In feature of the Composer has been improved. If the upper left corner of the working area is visible, it is used as center of the zoom operation.

Switch/Case Statement

The verification of **switch/case** statements is improved now – repeated case expressions are now handled as errors.

Tracestack

The new **tracestack** debug statement can be used within the prototyping environment of Embedded Wizard to evaluate the calling stack at runtime.

Ewext.c

The new file 'ewext.c' contains a set of functions to customize the adaptation of the Platform Package to the underlying hardware. The file is stored in the Runtime Environment and has to be added to your projects or makefiles. It contains functions to allocate or free memory, to output debug or error messages and to access the hardware timer. The module can be regarded as a template - if necessary, the implementation of these functions can be changed by the user, e.g. to access a different memory manager or to use a different printing function. Since this module is installed by a setup program (and all modifications will be overwritten by an update), it is recommended to use a copy of this file within the project directory and to exclude the original 'ewext.c' file from the make file.

EwClearStringCache()

The new function **EwClearStringCache()** is very useful for a safe shut-down of the runtime environment. The function forces the cache to discard all string constants which are loaded but not in use anymore.

EwStringToAnsi()

The new function **EwStringToAnsi()** can be used to convert a Chora string into an ANSI string. All character codes greater than 255 are replaced by the given default character.

Rename of Members

Members can be renamed now directly from the composer window by pressing 'F2'.

Virtual File System Switch

Embedded Wizard is now delivered with a Virtual File System Switch. The Virtual File System Switch is an abstraction of a generic file system. It is responsible for the management and the access distribution to volumes registered (mounted) during the runtime. It hides the differences between the registered volumes and allows an uniform access to the files stored on them. The Virtual File System Switch is installed in the directory `\VFS`. The unit `Vfs.ewu` implements a set of classes to access the virtual file system. The adaptation to the respective file system has to be done by the user. For details see the inline documentation of the delivered `*.c` and `*.h` files and the sample implementation of the Win32 file system interface (`vfswin32.h` and `vfswin32.c`).

Core::Time, Core::TimeSpan

The new Mosaic classes `Core::Time` and `Core::TimeSpan` can be used to store time and date, to format time and date strings and to calculate time spans.

Locale Unit

The new unit `'locale.ewu'` contains a set of language dependent constants (names of months and days) for the Mosaic class `Core::Time`. If your application uses the `Core::Time` class, the new unit `'locale.ewu'` has to be added to your project. A template of this unit is stored in the subdirectory `'\templates'`. This template should be copied into your project directory and adapted to the languages of your project. All new projects created with Embedded Wizard V2.30 already contains a copy of this unit file.

Build Settings

The build settings are stored in the session file now.

Excluding of non-used definitions

Classes, constants or resources which are not used from an application are excluded from the code generation, if the attribute `'Generator'` of the affected definition is set to `'false'`. The Mosaic class library is adapted to this feature. This helps to reduce the generated code size.

Alpha blending

All drawing operations with alpha-blending have been optimized to speed up these operations up to x 1.5.

Bug fixes:

The following bugs have been solved:

Missing FileName Attribute

The code generation was stopped without any message, if the `'FileName'` attribute of a bitmap resource was empty.

AutoSize Property in Views::Image

In some cases the `'AutoSize'` Property of Image objects was ignored.

Initialization of Properties, Variables, etc.

Expressions within the initialization of properties, variable, etc. which start with an open parenthesis `'('` caused an unexpected compilation error.

Warnings

The comparison expression with an `uint32` operand caused unexpected warning.

Compiler Error

The following kind of expression caused a Chora compiler error: `^((typecast)operand).Property`

Embedded Wizard V2.20

Version V2.20 contains the following changes and improvements:

Font Ranges

Embedded Wizard is now able to handle fonts with an unlimited number of character ranges. This means, it is now possible to select single characters or character ranges out of a large font (e.g. Far East fonts). To use this feature, it is necessary to replace an existing font resource brick with a new font resource brick from the Gallery. The new font resource brick replaces the attributes `'FirstChar'` and `'LastChar'` by a new attribute `'Ranges'`. However, the old attributes `'FirstChar'` and `'LastChar'` are still supported (to be compatible). The new attribute `'Ranges'` can be used to define all the character ranges for a font resource. For more details see Chora User Manual.

Compression

All bitmaps, fonts and strings are now compressed during the code generation. Thus, all resources and strings need less code memory. They are decompressed automatically during runtime. It is necessary to update the Platform Packages.

Chora SDK

The Embedded Wizard Setup has been enhanced to install the Chora Software Development Kit. This is necessary to develop own intrinsic modules. Intrinsic modules are simple extensions of the Chora language, which are loaded dynamically into Embedded Wizard, to add user defined functions into the Prototyper. The files are stored within the directory `\Chora\Sdk`.

Bug fixes

The following bugs have been solved:

Backslash

During the code generation the single backslash sign `'\'` was not converted correctly to ANSI C. The problem is solved now.

Comboboxes

The comboboxes for language and profile selection can be scrolled properly now.

Name conflicts after derived paste

In some cases embedded objects receive a non unique name, when they were embedded in the derived paste mode. This caused error messages. The problem is solved now.

Embedded Wizard V2.10

Version V2.10 contains the following changes and improvements:

Animated Image

The class `Views::AnimatedImage` is redesigned: The properties `'NoOfFrames'` and `'FramesPerSecond'` are replaced by the new properties `'FrameSize'`, `'Begin'` and `'Period'`. The property `'FrameSize'` defines the dimension of a single image; the delay time between the different frames are controlled by `'Begin'` and `'Period'`.

Additionally, the class `Views::AnimatedImage` contains the slots `'Activate'` and `'OnFinished'`, which can be used (similar to timer objects) to chain effects.

Effects

The `'Effect'` classes contain now the new property `'Endless'`, which can be used to run effects repetitively (instead of connecting the `'OnFinished'` property with the `'Activate'` slot).

The effect class `Effects::BoolEffect` is now derived directly from `Core::Timer`, to avoid the unused property `'Mode'`.

Session manager

The session manager has been enhanced to store the currently Editor content and state. Now, the content of the Editor is reloaded automatically, when the user changes the class inside the Composer.

Navigation bar

The Navigation bar on top of the Composer has been improved. A click with the right mouse button opens a context menu with a list of all currently opened composers.

Navigation

The navigation function `'C'` within the Navigate menu has a new meaning: Navigate to the selected content. It is now possible to navigate to the name, which is currently selected within the Editor or Inspector.

Additionally, the navigation functions `'S'`, `'D'` and `'I'` can be used within the Editor to navigate to the base implementation of a derived method (`'S'`), to navigate to the first implementation (`'D'`) or to locate the brick of the method within the Composer (`'I'`).

Gallery

The template 'class' within the Gallery folder 'Default' has been changed. It represents now an empty class without base class. To get a template of a `Core::Group` class, please use the template 'Simple Group Class' within the folder 'Forms'.

Sounds

The animation sounds can be changed like other system events, within the sound preferences dialog of the Windows system control.

Forms, Multiline and Flow Text

The Mosaic class library is enhanced by a set of new classes. It contains the new unit 'Forms'. This unit contains a set of useful classes to build menus, dialogs, scrollboxes, lists, ... The Gallery contains a new folder called 'Forms', where you can find the classes.

The following classes are added to the Mosaic class library:

Views::ImageList

The class `Views::ImageList` is responsible for drawing an image from an image list. All images are stored in a single bitmap where each image has the same width and the same height. Using the `ImageNumber` property the desired image can be selected from the bitmap and displayed within the view.

Views::MultilineText

The class `Views::MultilineText` is responsible for drawing a multiline text. The user can specify a rectangular area around the text, an alignment of the text within this area, the color and the font to draw the text. Additionally a shadow color and the shadow size can be specified. The shadow is always drawn behind the text. `Views::MultilineText` tries to arrange the text within the view. If necessary automatic word wrapping is performed. The wrapped text will not exceed the view's bounds in horizontal direction.

Views::FlowText

The class **Views::FlowText** is responsible for drawing a special case of the multiline text. By overriding this class the user can specify one or more regions to cover the text. The text will then 'flow' from one region to the next region. In this manner complex text filled areas are possible. Beside the regions the user can specify the color and the font to draw the text, the shadow and the shadow color and the alignment to align the text within the regions. The shadow is always drawn behind the text. **Views::FlowText** tries to arrange the text within the regions as good as possible. If necessary, automatic word wrapping is performed. The wrapped text will not exceed the region's area.

Views::TextRegion

The **Views::TextRegion** class is used by the **Views::FlowText** class to describe the area to be filled with the text. To define complex areas, multiple **Views::TextRegion** can be used by a single **Views::FlowText** view. Note: when using **Views::TextRegion** ensure, that regions are initialized before the corresponding **Views::FlowText** object become initialized.

Forms::Ctrl

The class **Forms::Ctrl** is the base class for the development of user defined controls. Simplified said, controls are special kind of groups one might use to create menu items, dialog buttons, volume level sliders, radio buttons, etc. Usually controls are used within forms only. The form is responsible for the management of these embedded controls, for theirs initialization, focus management, the scrolling, update of scrollbars, etc.

Forms::Form

The class **Forms::Form** implements the basis for the development of small menus and dialogs. The term 'small' does mean, that the menu's or dialog's content is small enough to be shown completely without the need for scrolling or moving of this content. Usually a menu or dialog consists of one or more menu items, buttons, radio buttons, sliders, etc. The form is thus responsible for the management of these so called 'controls'. Beside the controls, a form may contain any number of ordinary views and groups. To distinguish between controls and ordinary views, the class of the affected object is important. Only objects instantiated from the **Forms::Ctrl** or derived class are recognized and managed by the form as controls. Unlike groups and views, a form can become modal. As long as a form is modal, all user input events are passed directly to this form. When the form exits the modal state, the previous event flow and the focus path are restored automatically. To enter the modal state, the **BeginModal()** method of the **Core::Root** object should be called. To exit the modal state the **EndModal()** method should be called. This feature is very useful for modal dialogs, message boxes, etc.

Forms::ListForm

The class **Forms::ListForm** extends the simple form to a 'list box'. Within this list box a set of list items appear and the user can scroll these items in order to explore the whole list. The list form can be used, for example, to display the names of all channels within a TV set. The **Forms::ListForm** is very flexible and can be used to display very long lists, consisting of hundreds and thousands of items. To save the memory, the **Forms::ListForm** does not store the whole content of the displayed list - only the currently visible items are stored. To access the content of other items the **Forms::ListForm** calls the method **GetItemData()**. This method belongs to the interface the programmer has to implement in derived classes in order to supply the form with the items. Like other forms the **Forms::ListForm** uses controls - they will be used to display the items. During the update of the **Forms::ListForm** the content of the currently visible items is assigned automatically to the corresponding controls. To keep the **Forms::ListForm** and the displayed list items synchronous, the **Forms::ListForm** uses the 'cursor' - a pointer to the list item which appears in the currently focused control. When the **Forms::ListForm** content is scrolled by the user, the method **MoveCursor()** is called to update the position within the list. This is the second method the programmer has to implement in order to supply the form with items.

Forms::ScrollForm

The class **Forms::ScrollForm** extends the simple form by an additional scrolling behavior. This allows the programmer to implement large menus or dialogs. The term 'large' does mean, that the menu's or dialog's content is bigger than the area reserved for the form itself. For this reason it may be necessary to move the whole content when the user wants to reach one of the embedded controls.

Forms::ImageForm

The class **Forms::ImageForm** is a special kind of the **Forms::ScrollForm** which allows the programmer to display a large image within a small form. The user can then scroll this large image in order to explore it. The **Forms::ImageForm** can be compared to the **Views::Image** class with the additional feature of scrolling the bitmap content.

Forms::TextForm

The class **Forms::TextForm** is a special kind of the **Forms::ScrollForm** which allows the programmer to display a large text block within a small form. The user can then scroll this large text block in order to explore it. The **Forms::TextForm** can be compared to the **Views::MultilineText** class with the additional feature of scrolling the text content.

Forms::ScrollBar

The class **Forms::ScrollBar** implements a generic scrollbar OSD object. This scrollbar object can be used, for example, to show the user which part of a large content is currently displayed on the screen. This **Forms::ScrollBar** class does not define the appearance of the scrollbar. You can derive **Forms::ScrollBar** in order to define your own appearance. For this purpose the method **ArrangeTracker()** should be overridden in order to update the position and the size of the displayed scrollbar tracker.

Forms::Indicator

The class **Forms::Indicator** implements a generic indicator OSD object. This indicator object can be used, for example, to show the user whether the content of a **Forms::ListForm** can be scrolled or not. An indicator can take one of two states: enabled or not enabled. Depending on the enabled state the appearance of the indicator should change. To do this the method **UpdateIndicator()** has to be overridden in the derived classes. The **Forms::Indicator** itself does not define the appearance of the indicator. You can derive Indicator in order to define your own appearance or you can use the **Forms::ImageIndicator** class which displays a bitmap as indicator.

Forms::ImageIndicator

The class **Forms::ImageIndicator** extends the generic indicator to display bitmaps depending on the state of the indicator. If the indicator is enabled, the bitmap attached to the property **EnabledBitmap** is displayed. If the indicator is disabled, the **DisabledBitmap** appears.

Core::Time

The class **Core::Time** is used for date and time operations. Objects of this class can store the time, convert it and format strings with the time, etc.

Bug fixes

The following bugs have been solved:

Unit-Reload

Reloading of a changed unit causes crash in some cases in previous versions. The problem is solved now.

\$if / \$elseif

The directives **'\$if'** and **'\$elseif'** are now evaluated correctly. The order of the operators **'&&'** and **'||'** are evaluated correctly. Additionally, it is now possible to use macros without **'=='** or **'!=='** operators.

Fatal Out of Memory

The out of memory error which occurs some times after working for a long time with Embedded Wizard, was caused by a memory leak after **'Drag & Drop'** operations and is solved now.

Log Window

If a return type or a argument of a method is erroneous, it was not possible to open the corresponding editor with a double-click on the error message of the Log Window. The problem is solved now.

Class reload

The reloading of a class (F7) did not reload changed code inside the currently opened Editor. The problem is solved now.

Code Generator

Local variables or arrays, which are not declared on the beginning of the method, have been generated erroneous in some cases. The problem is solved now. Additionally, it is now possible to use empty blocks {} after **if**, **while** or **for** statements.

Drag & Drop

The 'Drag & Drop' feature has been improved. The Inspector does not modify the selected item immediately after dropping a new value or expression – this avoids an error if the user wants to change the dropped value or expression.

Chora Interpreter

The instant methods `string.right()`, `string.left()`, `string.middle()`, `string.remove()` and `string.insert()` returned in some cases un-initialized strings. The code generation was not affected.

Index Operator

A read access to the last character of a string caused a runtime error instead of returning the termination character 0. Now, the index operator returns 0, if the last character is accessed.

Session file

The error message 'Can not save session file' is avoided now.

Ctrl + Enter

The key commands 'Ctrl' + 'Enter' transfers the content between Inspector and Editor. If the user presses 'Ctrl' + 'Enter' after changing the content within the Inspector, the modification was lost. Now, the latest modification is used.

Composer

Empty Views with the size <0, 0> have been handled as selected, even they were outside the selected area. The error is corrected now.

Log Window

Ugly scrolling behavior is solved.

Dongle

Shut down problem under Windows XP is solved.

Embedded Wizard V2.00

Version V2.00 contains the following changes and improvements:

User Interface

The complete Embedded Wizard User Interface has been redesigned to improve the usability. The old approach with 'Multiple Document Interface (MDI)' has been replaced by a 'Single Document Interface (SDI)' architecture with tab bar to switch between the different Composer windows. The advantage is, that the user is not confronted with many different windows at the same time. Now, the arrangement is clear and all necessary information is accessible fast and comfortable.

Navigation

To improve the productivity, a powerful navigation between the different Composer windows is integrated. Additionally, an intelligent navigation (Navigate menu) helps to find definitions and dependencies within your project.

Session file

Embedded Wizard stores now the current workspace settings in a session file. When a project is loaded, all previously opened Composers and their settings are reloaded automatically. You can continue working at the point you finished the last 'session'.

Composer

The Composer is unified and simplified – the 'Ingredients' page and the 'Appearance' page are now combined to one common page. The advantage is, that the user gets a better view of the whole class implementation.

The canvas area (where all graphical objects are drawn) appears now within a thick blue frame, which can be easily resized to define the size of an object's viewable area.

Additionally, the 'Inherited Methods' area has been removed. To override a method, you have to drag the method from the Inspector into the Composer while holding *Ctrl+Shift* keys.

Visual Programming

The usage of 'bricks' as visual representation has been redesigned: Now, only those bricks are displayed, which are overridden in an inherited class or which are newly added to a class. The advantage is, that the user is not confused by hundreds of bricks, which are only inherited from one of the base classes. Objects, which are displayed inside the canvas area (rectangles, images, text objects, ...) are not displayed as bricks, because they are already visible. All inherited, not overridden members can be still accessed within the Inspector.

Zoom, Aspect Ratio, Grid

The new feature Zoom helps to arrange graphical objects more precise and comfortable. The Aspect Ratio feature is useful to adapt the graphical appearance to the real geometry of the target display. Additionally, the new Grid feature helps to arrange bricks and graphical objects within a adjustable grid.

Prototyper

The Prototyper window appears now as stand-alone top-level window.

Editor

The Editor is now placed as a dock-able window to see the content of a Composer and any source code inside the Editor at the same time. So, 'Drag & Drop' from the Composer into the Editor is possible.

Inspector

The Inspector is enhanced by the Description Area to display the description of members, attributes and properties (defined by the user in the '**Description**' attribute).

The content of complex Chora data types can be edited more comfortable now: You can access directly the components of **color** (**red, green, blue, alpha**), **rect** (**x, y, w, h, x1, y1, x2, y2**) or **point** (**x, y**).

Additionally, when editing a class, a new alias member called '**this**' appears inside the Inspector. This member represents the interface of the class and its initialization values.

'Connect Editor with Inspector'

This new feature synchronizes the content of edit field inside the Inspector with the Editor windows. So, you are able to view and edit the content of a selected attribute or property much more comfortable within a larger Editor instead of a single line field.

'Drag & Drop'

The complete Embedded Wizard User Interface can be controlled now by simple 'drag & drop' operations. For this purpose, the windows are mouse-sensitive. This means, that if the mouse cursor is hold for a short time during a 'drag & drop' operation over a member or window the corresponding member or window is selected, opened, expanded, ...

Enhancement of Project Search Feature

The project search has been improved to find better definitions inside your project, e.g. to find all inherited methods in the project you can search for the string "inherited method".

Language and Profile selection

The language and profile selection influences now immediately the currently edited class inside the Composer.

Restacking and Initialization Order

Changing the z-order of members has been redesigned, to avoid problems during the initialization of objects and their members. Now, the initialization is done strictly in the same order as the members appear in the Inspector (z-order).

The z-order column of the Inspector is now updated automatically.

Since the previous implementations (Embedded Wizard V1.x) was erroneous, the changes may affect existing projects. In worst case, you have to restack some of the members inside your classes.

Chora 2.00

The specification of the Chora language has been simplified. All inherited members do not expect any more the definition of the data type, method arguments or array sizes. This information is derived from the first definition of the member. This avoids any inconsistencies and conflicts, when the definition inside the base class is modified.

Example:

```
class Menu : Core::Group
{
    inherited property Visible = true;

    inherited method HandleEvent()
    {
        var Core::KeyEvent e = (Core::KeyEvent) aEvent;
        if ( e != null )
    }
}
```

```
... // do something in response to the key event
return super( aEvent );
}
}
```

The **onget** and **onset** Methods do not expect the type definition of the corresponding property. The type definition taken from the definition of the property.

Example:

```
class Menu : Core::Group
{
    property string Caption = "Hello world";
    onset Caption
    {
        pure Caption = value;
        InvalidateView();
    }
}
```

Additionally, it is no more necessary and possible to specify the data type for a preset within an object.

Example:

```
object Views::Rectangle Background
{
    preset Bounds = <10,10,200,200>;
    preset Color = #000000FF;
}
```

Old projects are accepted and converted by Embedded Wizard automatically.

Mosaic Bug Fix

Some minor bugs inside the Mosaic class library has been fixed: The effect classes reach now the defined end value of the effect. The **Core::Group.Add()** method invalidates the area of the new added member correctly.

Code Generation

The generation of the currently selected profile can now be started by simple pressing 'F9' without any dialog.

Embedded Wizard V1.01

Version V1.01 contains the following changes and improvements:

Support of user CLUT

The platform packages for the 8 bit index format supports now user defined colors. To define a new color palette, the following steps are necessary:

- Open the Profile brick and add a new macro from the Template Gallery.
- Change the name of the macro from 'NewMacro' into 'Clut'
- Set the content of the macro (field 'Default') to the filename of your color table (e.g. 'MyClut.txt').
- Create a textfile within your project directory (e.g. 'MyClut.txt') and fill it with your colors in the following way:

```
$version 1.0
clut
{
  #000000FF, // black
  #FF0000FF, // red
  #00FF00FF, // green
  #0000FFFF, // blue
  #FFFFFFFF, // white
  ...
  #00000000 // transparent
};
```

- Colors are defined in Chora notation (#RRGGBBAA). Up to 256 colors can be defined within the CLUT.
- During the code generation, a new C-file with the same name as your text file is generated (e.g. 'MyClut.C'). You have to include this file into your project.

The selection of the colors has to be done carefully! If your application or your resources contains colors, which are not stored in the CLUT, they will be displayed with similar colors. If there are no similar colors available in the CLUT, the results may appear ugly.

To use the default CLUT (to be compatible with earlier versions), it is necessary, to add the file 'DefaultClut.C' to your project.

Embedded Wizard V1.00

Version V1.00 contains the following changes and improvements:

Windows Platform Packages

The set of MS Windows Platform Packages has been extended. The supported graphic formats are now RGBA 8:8:8:8, 8 bit gray with 8 bit alpha, and 8 bit index format.

Mosaic Class Library Enhancement

A set of new useful classes has been added to the Mosaic Class Library, e.g. `Views::AnimatedImage`, `Views::Bevel`, `Views::Wallpaper`, ... and several effect classes defined in the new unit `Effects`. Some of the existing classes have been enhanced by additional properties or behavior. The class library is still compatible to earlier version.

Mosaic Documentation

A Mosaic reference manual is now available as compiled HTML file.

Template Gallery Adaptation

After the Mosaic Class Library enhancement, the set of templates within the Gallery has been adapted.

\$Mosaic

The new built-in macro `$Mosaic` makes projects independent from the directory, where the Mosaic files are installed. This macro can be used within the `Directory` attribute of a Mosaic unit only.

Selection frame in Composer

The update problems concerning the selection frame within the Composer are solved now.

Log List

The update problems within the Log List (only on some PCs) are solved now.

Prototyper Shut Down

The complete prototyping environment is now terminated correctly, when the Prototyper window is closed (no further background activities).

Column 'Order' within Inspector

The content of the column 'Order' within the Inspector can be updated by a simple click on the column header.

Property Value Apply in Inspector

When the user changes a value in the Inspector and then clicks into another window the value was lost sometimes. The problem is solved now.

Chora Documentation

The documentation of the language 'Chora' is completed now.

OnGet / OnSet Methods

`OnGet` and `OnSet` methods are now optional. If a property does not need any special processing, the `OnGet` and/or `OnSet` method are not necessary anymore. The Chora compiler will add the missing methods automatically.

Embedded Wizard V0.92

Version V0.92 contains the following changes and improvements:

Embedded Wizard Optimization

The performance of the Embedded Wizard application is improved. Especially, the code generation is now up to 35 times faster.

Code Generation Hang-Up

The hang-up during code generation is solved now. All generated files are stored in the correct directory.

Unexpected Access Violations

The access violations while editing or prototyping a class should be avoided now.

Instant Property Access

The access to an instant property in scope of another instant property causes no more runtime errors in the Prototyper.

Dongle Access

The access to the Embedded Wizard license key (dongle) is improved.

Screen Update

The screen update in the target system is improved and optimized. Unnecessary redraws of screen areas are avoided now. This improvement requires a few changes within your C software:

1.) Add the following function to the module containing the EmWi message loop:

```
/* UpdateRoot() forces the root to redraw its invalid areas. */
static void UpdateRoot( CoreRoot aRoot, XGenDisplay* aDisplay )
{
    XGenRect clip;

    /* Redraw the invalid area and get the origin and the size
       of this area. */
    XRect updateRect = CoreRoot__Update( aRoot );

    /* No area has been redrawn. */
    if ( EwIsRectEmpty( updateRect ) )
        return;
}
```

```
/* The area to update within the display */
clip.X1 = updateRect.Point1.X;
clip.Y1 = updateRect.Point1.Y;
clip.X2 = updateRect.Point2.X;
clip.Y2 = updateRect.Point2.Y;

/* convert into the display */
GenConvertDisplay( aDisplay,
    (XGenBitmap*)aRoot->canvas->Super1.bitmap, &clip );
}
```

2.) Insert a call to the new function within you EmWi message loop.

Example:

```
/* Let's loop in the message loop ... */
while ( 1 )
{
    ... Read the message and handle it ...

    /* Process pending timer events */
    EwProcessTimers();

    /* Refresh the display if something has changed */
    UpdateRoot( (CoreRoot)Application, &display );

    /* Activate garbage collector */
    EwReclaimMemory();
}
```

3.) Change the initialization of the root object (application object) and remove the creation of the canvas object.

Example:

```
/* Create the applications root object... */
Application = EwNewObject( DemoMenuApplication, 0 );
EwLockObject( Application );

/* ... and initialize the root object */
CoreRoot__Initialize( Application, size );
```

Whole Application Prototyping

It is now possible to execute a complete application within the Prototyper, if the application is derived from class `Core::Root`.

Unit reload

The request to reload a unit appears only if the unit file has changed on the drive. The read access to the unit file (for example from the MS Windows Explorer) does not cause this request anymore.

Data Type 'rect'

The behavior of the instant data type `rect` within the Prototyper is now compatible to the target system.

Short-Circuit-Evaluation

The operators '`||`' and '`&&`' are now evaluated according to the 'short-circuit-evaluation' within the Prototyper.

Language Switch

The wrong behavior of the language switch is corrected now.

Font Converter

The font conversion is optimized now. No character equal to the 'default character' are stored as copies of the default character.

Embedded Wizard V0.91

Version V0.91 contains the following changes and improvements:

Language Switch

The currently active language can be changed now directly in Chora. For this purpose, the new data type `language` is integrated. The currently active language can be changed by a simple assignment to the 'global' variable `language`. The `language` variable can be written and read.

Example:

```
// change the language...
language = English;

// or query the language...
if ( language == German )
    ...

// create your own variable with the datatype 'language'
var language l = German;

if ( l != French )
    l = French;
```

After changing the language, all further accesses to any language dependent constant or resource are done with the new language selection.

Language Switch and Mosaic

The class `Core::Root` contains a new property `language`. The currently active language can be changed by setting this property. The result is the same as setting the global `language` variable. Additionally, all Mosaic objects receive a `Core::LanguageEvent` event. Thus, each object can handle the new language selection.

Example:

```
var Core::Root root = ...

// change language and send a 'Core::LanguageEvent'
// to all Mosaic objects
root.Language = Greek;

// or query the language
if ( root.Language == Greek )
    ...
```

Memory leaks

All memory leaks inside the Embedded Wizard application are removed.

Error messages

The error message (initializing variables, properties, attributes,...) corresponds now exactly to the erroneous variable, property, attribute, etc.

Log Window

A double click onto an error message guides you now always to the error location. After a double click onto a message, the message is displayed gray, to indicate that the message is handled by the user.

Garbage Collector in Prototyper

The Prototyper includes now a Garbage Collector to delete all unused objects.

Undo on deleted units

The 'Undo' operation after deleting a unit from a project does now work correctly.

Dongle integration

The dongle detection is improved. In older versions it was sometimes necessary to disconnect and reconnect the dongle before starting the Embedded Wizard application.

Delete key

The 'Delete' key does now work correctly in the Inspector and the Editor.

Restack

The restack function does now work in Composer and Inspector.

Alpha-Blending

The mode `Graphics::Mode.Blend` does now produce proper results and objects can be combined with each other by an alpha-blending operation. The mode `Graphics::Mode.Blend` is now the default setting for all Line, Rectangle, Border, Image and Text objects.

DrawPixel

The class `Graphics::Canvas` contains now a function for drawing a single pixel. This could be helpful for your own drawing functions.

Win32 RGBA8888 platform package

The delivery contains a new platform package for Windows with the color format `RGBA8888`. Please change within your projects the attribute setting `platform package` in the `Win32` profile from `Tara.Win32` to `Tara.Win32.RGBA8888` for 32 bit color format or to `Tara.Win32.Index8` for the 8 bit color format. Please adapt the path for GE and RTE within your Visual C++ project.

BMP/PNG converter

PNG/BMP conversion problems are solved now.