Cincom®

Cincom Smalltalk™

VisualWorks®

Release Notes 8.1

P46-0106-23

SIMPLIFICATION THROUGH INNOVATION

Part Number: P46-0106-23

Software Release 8.1

This document is subject to change without notice.

Cincom Systems, Inc.

55 Merchant Street

Cincinnati, Ohio 45246

Phone: (513) 612-2300

Fax: (513) 612-2000

World Wide Web: http://www.cincom.com

Contents

1

Introduction to VisualWorks 8.1

These release notes outline the changes made in the version 8.1 release of VisualWorks. Both Commercial and Non-Commercial releases are covered.

These notes are not intended to be a comprehensive explanation of new features and functionality nor are they intended to be used in lieu of the product documentation. Refer to the VisualWorks documentation set for more information.

Release notes for 7.0 and later releases are included in the doc/ directory (e.g., 7.2.1 release notes cover 7.2 as well).

For late-breaking information on VisualWorks, check the Cincom Smalltalk website at:

http://www.cincomsmalltalk.com/

Product Support

Support Status

Basic support policies for the current release are described in the licensing agreement. As a product ages, its support status changes. To find the support status for any version of VisualWorks and Object Studio, refer to this web page:

http://www.cincomsmalltalk.com/main/services-and-support/support/

ARs Resolved in this Release

The Action Requests (ARs) resolved in this release are listed in doc/fixed_ars.txt.

Additional ARs may be discussed in individual sections of these release notes.

Outstanding ARs and limitations are noted throughout these release notes, as appropriate.

Items of Special Note

Distribution Designations and Non-crypto distribution

In compliance with U.S. security requirements, VisualWorks now has a commercial version without encryption code.

Installation directories names are also being updated, as follows:

`<user-files>/vw8.1`

Full commercial distribution

`<user-files>/vw8.1pul`

Personal use license (non-commercial)

`<user-files>/vw8.1ne`

No encryption distribution

where **`<user-files>`** is the selected installation directory.

VisualWorks on Vista or Windows 7

Microsoft Vista and Windows 7 operating systems impose additional restrictions on file permissions and applications. Accordingly, there are a few special considerations when using Windows.

Installing VisualWorks

You can install VisualWorks either as a regular user or an administrator, but only users belonging to the administrator group have write acess to the **Program Files** directory. Note that you can always install VisualWorks to other directories without complication.

When installing as an administrator, Windows will open a slightly cryptic prompt to confirm whether to run the Installer. Simply click **Continue** to proceed.

Note also that, when installing on 64-bit Windows 7, a dialog may display noting that the program might not have installed correctly. Our experience has been that the installation is correct, so click the option **This program installed correctly**, unless you know otherwise.

Uninstalling VisualWorks

If you installed VisualWorks to the **Program Files** directory, the install-uninstall shortcut in the **Start** menu will not work correctly. In this case, the Installer on the VisualWorks distribution CD must be used to uninstall the product.

Saving your Work

Since all directories under **Program Files** are write-protected, when working as a non-administrative user, your VisualWorks image files must be saved somewhere you have write privileges, such as your own **My Documents** directory.

Editing VisualWorks.ini

The `bin/VisualWorks.ini` file is installed as writable. After installation, all paths to all releases of VisualWorks in this file begin with the directory containing the VisualWorks home directory that you set when installing (so if you do not keep all your releases of VisualWorks within the same overall directory, you may wish to edit this file).

On a machine with more than one copy of VisualWorks installed, `.im` files will be assigned to the `VisualWorks.exe` of a given installation (unless they are not assigned to any). Double-clicking on a `.im` file opens the engine assigned by the `VisualWorks.ini` file of that `VisualWorks.exe`. If you already have an assigned version of VisualWorks on your machine, you can copy the more recent lines from your newly-installed `VisualWorks.ini` file to the existing one. Alternately, you can reassign the `.im` file type to your newly-installed `VisualWorks.exe`. We recommend reassigning the file type by editing its path by hand. Using the **Browse** function the dialog provides to navigate to the new `VisualWorks.exe` often fails to work (the change of a file type from one program to another of the same name is not recognised as a change).

In addition, if VisualWorks is installed in the **`Program Files`** directory structure (as is done by the Typical installation), you must have administrator privileges to save changes. Being logged on to an administrator account is not sufficient, and you must **Run as administrator** the editor program, which might be VisualWorks or any text editor.

An alternative is to use the Launch Pad (Project Manager), which maintains its own version of **`VisualWorks.ini`** to select the appropriate object engine for an image.

Xtreams

Xtreams is a generalized stream/iterator framework providing simple, unified API for reading from different kinds of sources and writing into different kinds of destinations (Collections, Sockets, Files, Pipes, etc).

Xtreams is now included in the **`xtreams`** directory of the standard distribution. The code in this directory is supported by Cincom, while the code that remains in **`contributed/xtreams`** is not.

For instructions on usage, refer to the documentation at:

http://code.google.com/p/xtreams/

Known Limitations

While a large number of ARs (Action Requests) have been addressed in this release, a number remain outstanding.

Known Limitations sections are provided throughout this document, pertaining to specific product areas.

Purging Undeclared

When a parcel is only partially loaded, the purge of Undeclared will detect references in unloaded code. For example, if a loaded parcel named MyParcel contains the method:

```
FirstClassDefinedElsewhere>>extendingMethod
    ^SecondClassDefinedElsewhere
```

and both FirstClassDefinedElsewhere and SecondClassDefinedElsewhere are not loaded, then:

- extendingMethod will also not be loaded but it will be in the parcel's unloaded code

- Undeclared *will not* contain FirstClassDefinedElsewhere (unless it is referenced in some other place)
- If SecondClassDefinedElsewhere and all loaded references to it are removed, and it gets added to Undeclared, then a purge of Undeclared will fail to remove it because of the reference in extendingMethod, even though that method is not loaded and there are no references to it in loaded code

This issue has existed since unloaded code was introduced, is recognised as undesirable and will be addressed.

Limitations on Windows of displayShape: methods

On more recent versions of Windows 7 it is not permitted to draw directly on the screen. Various methods in the Screen class in VisualWorks attempt to do so. This is implemented by creating an invisible window, drawing on it, and then destroying the window. However, this is a slow process, and methods that attempt to do animation using this primitive can become very slow and the animations can become effectively invisible. Use of these mechanisms is often used for animation of operations like dragging within a graphical editor. In such cases they can be replaced by graphics operations within that window, which will be much, much faster.

Operations like Rectangle>>fromUser: are not as slow because they can create the invisible window once, do numerous drawing operations, and then complete. But methods like the Screen>>displayShape:... family do not have this information. At the moment, use of these methods is discouraged while we examine possible solutions.

Publishing a Bundle as a Parcel

When **Publish as Parcel** is invoked on a bundle, we recommend selecting **Include bundle structure**.

If you choose not to select this, check that the bundle's own prereqs are what the parcel will need, since the prereqs of its subpundles will then not be assigned to the parcel (which may therefore show unexpected behaviour on load).

In the next release, the bundle structure will aways be saved when saving a bundle, and will no longer be an option.

Locating Shared Libraries for SQLite

The SQLite3Interface class has hard-coded values for the shared variable libraryDirectories. Depending upon your platform and installation of SQLite, these hard-coded directory locations may not contain your SQLite client library (.DLL on Windows, .so on Linux, etc.), and thus VisualWorks cannot find it. If your SQLite library is listed in the system path, VisualWorks could find it if these directories were not hard coded. In this case, you have two options:

- Add your SQLite directory to the libraryDirectories initialization method.
- Simply remove the entries and save the class definition. That is, use a code browser to change:

  ```
  #libraryDirectories #('.' '[win]$(windir)' ... )
  ```

  to:

  ```
  #libraryDirectories #()
  ```

  and save the class definition. Once this is done, the VM will use your system path to find and load the client library.

AppeX ServerMonitor needs manual refresh in IE

Late testing has uncovered a bug in the AppeX ServerMonitor: when it is first opened in Internet Explorer, the ServerMonitor page does not display correctly until the page is refreshed (or re-visited in the same tab). This problem is currently not observed in Firefox or Chrome.

2

New and Enhanced Features

This section describes the major changes in this release.

Base Image

Enhancements and Clean-up of Timestamps

VisualWorks 8.1 includes enhancements that make it possible to tell which Timestamp formats in TimestampPrintPolicy are "hintable" on a per-Locale basis. This allows the read Timestamp methods on TimestampReader to better interpret printed timestamps when reading them, for the purpose of creating new instances with the values read.

Additonally, new methods to create Timestamps have been added, which take as input an instance of CompositeLocale rather than the name of the Locale (which is prone to error in various ways). E.g., instead of:

```
TimestampReader class >>
   newFor: aSymbol
   customFormatString: aPrintPolicyFormatString
```

there is:

```
TimestampReader class >>
   newForLocale: aCompositeLocale
   customFormatString: aPrintPolicyFormatString
```

(AR 71405, AR 71825)

Better sensing of the system Locale on Unix and Linux Platforms

Changed both the old-fashioned "get locale" and the new one to use baseGetLocale(), a new C function that searches for defined environment variables rather than using setlocale() which hides what happens behind the scenes. This avoids inappropriately defaulting to the C Locale on some systems, notably modern Ubuntu distributions (AR 71111).

Handle OS X Cocoa VM vs. OS X X11 VM better in image code

Make the class hierarchy for graphics support mimic the system support hierarchy, but for Windows, X11 and Cocoa instead of Windows, Unix and OS X, in order to separate OS from GUI (eg: X11 on OS X vs Cocoa on OS X) (AR 72039).

Better checks for OS versions that might involve two or more levels

The approach used here was to encapsulate 10.8 as a SystemVersion object with major 10 and minor 8, and utilize "dictionary sort" behavior (AR 72105).

Database

Enhancements to PostgreSQL 3.0 Support

The PostgreSQL 3.0 socket and C API drivers (both new in 8.0) are now faster and more robust in 8.1. Sockets handle threaded interrupts robustly. The drivers handle cancellation better and disconnection faster, and are more robust in images being saved with active connections. Timeout duration can be set dynamically. Support for the Postgres types TimeWithTimeZone and TimestampWithTimeZone has been enhanced, and computes the image's local OS timezone (note that we assume Postgres' setting to distinguish Australian timezones will be set if in that region).

Enhancements to ODBC 2.0 Support

There have been various fixes to our support of ODBC 2.0, e.g., in MySQL, SQLite, DB2EXDI and OracleEXDI in the areas of external library loading, binding and bind templates, etc. In particular, Postgres stores Fractions at higher precision, MySQL now disconnects stale sessions without fuss, and the ODBC threaded API implements more threaded calls. (It also includes fixes to allow StoreForSqlAzure to run correctly.)

Binding NULLs when using ODBCEXDI and ODBC3EXDI

To bind nil to binary columns when using ODBCEXDI and ODBC3EXDI, it is necessary to use #bindTemplate: for the operation to work correctly. (AR 72418 and 72387)

Glorp

Enhancements to support for Dictionary keys

Glorp dictionaries can read and write keys in full generality, without needing keys to be simple objects and/or within the scope of the owner-value path; the whole area has been cleaned up. Limitations with comparing composite-key-mapped objects are gone: composite key objects can receive the in:, = and <> comparison operators, and subqueries can be parameters to them.

Improved support for Immutability

The longstanding clash between Glorp's optional use of immutability-exploiting WriteBarriers for auto-registration and Glorp's potential to walkback on reverting an immutable in other cases has been much eased by taking a range of non-immediate immutable literals (in the String and ArithmeticValue hierarchies) out of the problem area. When a single table holds rows for several concrete classes in a hierarchy filtered via a non-pirmary-key field(s), queries search the cache correctly. A query that is both limited and alsoFetch-ing a to-many-mapped object now returns the right number of objects.

Miscellaneous Fixes and Enhancements

Various other Glorp fixes have been made: more platform-robust substring functions, better SQLite compound queries, better DB2 meta-analysis, reuse of bound values better supported, etc. We've made minor improvements to Glorp performance by eliminating repeat parses for information in certain cases and repeat queries in others.

Both the Glorp and Database packages are now better at passing error strings up the chain to the tools. Glorp processes typical no-return-expected database statements to the explicit state of having completed execution (previously they could sometimes be left in an apparently-executing state, though in fact completed).

GUI

UISkinning

This feature gives widgets the ability to adapt their presentation to the native look and feel on Windows and OS X. It is our intention to get as close to a true native look and feel on Windows and Mac OS X as possible. The Native skin is only available on Mac OS X 10.5 and above, and Windows with theming support. In practice, this excludes Windows Server versions. This skin uses native drawing on Mac OS X or the Windows UxTheme DLL to provide a high-fidelity, native look for those platforms. On X11, where there is not a true “native” look and feel, the Native skin is replaced by a Default skin, which provides a platform-agnostic look. The Default skin is available on any platform.

UISkinning introduces two new groups of classes, Artists and Skins, which a View may use to control the rendering of its visual components.

Artists

An artist is responsible for all geometry and drawing for a view. A given artist might be shared between views, or there might be an artist per view. Artists conceptually define a structured appearance for the view that is usually a function of the model that the view presents (which might in fact be the transformation of one or more “real” models). The structured appearance is managed using the concept of “regions,” which is a dynamic mapping from points to objects, usually symbols (e.g., #body, #thumb or numbers such as the index of the tab in a tab bar visual that contains the point). Note that it is dynamic because the appearance probably changes according to the model value. Therefore, the regions are not fixed. Artists can also translate from a point to a model value (i.e., to provide the model value that corresponds to a given point) and from a point representing a distance to a model value delta (i.e., to adjust the scroll offset as a scrollbar thumb is dragged).

It’s important to understand that the artist knows nothing about events. It is responsible purely for the visual aspects of the view.

Artists all descend from AbstractArtist. Each artist is associated with a single view, and some contain instance variables that cache certain computations related to metrics.

Skins

Skins are artist factories. The current skin is available via UI.Skins.SkinFactory class>>current. Each view can request an artist from the skin at any time, and is required to do so when the skin or platform changes, which is notified via #newGraphicsDevice:.

Rather than subclassing artists to create different looks, each artist is initialized with parameters by the skin and delegates all functionality to the skin. It is done this way to avoid a proliferation of artist classes per skin. There is an artist type per widget type. So in practice, skins are a lot more than just artist factories — they contain most of the functionality of the skin.

All skins descend from UI.Skins.Default.DefaultSkin. This allows you to extend skins by adding methods to DefaultSkin, knowing that your default artist configuration and drawing/layout routines will be used for all skins. You can then progressively override the methods in each skin subclass.

You can add a new skin by adding a class side method to UI.Skins.SkinFactory that is marked with a pragma. See #nativeSkin for an example.

Views

Only those views that represent skinnable widgets have been converted to hold the artist in an instance variable. The artist is created in the #initializeArtist method, which is called both from the instance initializer and from #newGraphicsDevice:. The artist handles the view's visual representation, including the layout of its children. Views have no intrinsic geometry, appearance or layout. They appear to have all of those characteristics, but the view delegates their management to its artist.

You can continue to draw in Views. Just override #displayOn:, which will then not call the Artist. You can also provide your own controller. The same applies to geometry methods, provided that you follow the caveats described below.

UILayout

In support of the new platform-compliant skinning, our layout facilities have also undergone revision for 8.0. Existing GUIs that don't use custom components or layout should work unchanged, but client code that uses custom widgets and/or does custom layout will need to be updated.

The purpose of the changes are twofold.

- We wanted to provide better layout algorithms. Some of the requirements were to allow the layout of components to be determined by the relationship between components, such as baseline/top/bottom/center/leading/trailing alignment, as well as including gaps between components that depend on the functional relationship between components, as well as better control over the GUI resizing behavior.
- We wanted to clean up some of the confusing API in the GUI system, particularly in relation to component geometry.

The overall summary of layout changes is that CompositeParts are now responsible for sizing and positioning their children, using a LayoutManager delegate, whereas in the pre-8.0 architecture, children positioned and sized themselves within their parent CompositePart.

Component Geometry

Visual components in the VisualWorks GUI system have a number of geometric properties. The most obvious are the bounds that they are assigned, accessed using #bounds: and #bounds. Component bounds are typically computed and set by the component's container, and the computation of the bounds is the primary function of the layout mechanism described later in this document. Components have no way of specifying their preferred location, but they can specify their preferred extent, which is returned by #preferredExtent, as well as their minimum and maximum allowable extents, which are returned by #minimumExtent and #maximumExtent respectively.

Many components, such as labels, have minimum and maximum extents that are the same as the preferred extent. These are computed by measuring the text of the label. Thus, the default implementations of minimumExtent and maximumExtent simply return preferredExtent.

Along with the size, there are two further properties used for specifying alignment points for a component: alignment insets and baseline.

The alignment insets is a Rectangle returned by #alignmentRectInsets, which specifies insets from each of the four sides of the component. The default value is Rectangle zero. It can be used to specify that the visual alignment bounds of a component are different from the assigned bounds. The most obvious example of non-zero alignment insets is on Mac OS X, where controls include space for a highlighting ring around the outside of the control. If such a control is to be visually aligned with another control, then the algorithm has to adjust the bounds to account for the blank space that is included in the control's preferred, minimum and maximum extents.

The baseline is scalar value, returned by `#baseline`, specified as an offset from the top of the component. It specifies the baseline of the first line of text within a component, or where such a baseline would occur if the component had text. Alternatively, it can be `nil`, which indicates that the component has no natural concept of a baseline. In this case, the layout algorithm will most likely assume that the bottom of the component is the effective baseline.

One fact that may not be immediately obvious is that the geometric properties of a component (i.e., the minimum, preferred and maximum extents, baseline and alignment insets) must not be functions of the assigned bounds of the component or functions of any of the geometric properties of any of the component's containers. This is because the assigned bounds of a component and the geometric properties of the component's container are already potential functions of the geometric properties of the component by virtue of the layout architecture. In concrete terms, the container computes its preferred/min/max geometry by asking each of its children for their preferred geometry and applying a layout algorithm to the returned values. If any of the children compute their preferred geometry by asking their parent for its preferred geometry, then an obvious tight circularity arises. The same problem occurs if a component tries to compute its preferred geometry by asking its parent or itself for its actual bounds. These bounds are assigned by the layout architecture in a separate pass after all of the preferred geometries have been gathered, so this results in a looser, but still deadly circularity. This is a general rule and applies to all geometric characteristics of components, such as baselines.

In summary, it’s fine for a component to depend on the geometry of its children, but not that of its parents. In addition, a component cannot create “virtual” bounds for itself by moving its effective origin within its assigned bounds. The circular dependency created by violating these caveats will result in an incorrect layout at best or an infinite loop at worst.

Two examples of these restrictions and how to resolve the issues are as follows:

- `Labels` cannot center their text vertically within their assigned bounds, because that makes the label’s baseline a function of its bounds. The appropriate way to control the vertical alignment of labels is by positioning them correctly on the UI canvas.

- The elements in a sequence view cannot determine their width by referring to the bounds of their container (i.e., the sequence view itself). You might want to do this in order to change the text of the item depending on the width it has to display (e.g., eliding the content). In the current implementation, if the sequence view does not have a horizontal scrollbar, then the preferred width of the sequence view is not a function of its element's geometry. However, this is not guaranteed to consistently be the case. If the sequence view has a horizontal scrollbar, then the circularity is obvious because the sequence view's width will end up being the maximum of its element's widths. The solution in this case is to have the element's preferred extent be the full width of the text, but to adjust what is drawn in the #display: method depending on bounds of the container.

It is an error to set the extent of a component to be smaller or larger than its minimum or maximum extent respectively. If that happens, the component is free to exhibit bizarre behavior. However, it is recommended that custom components not throw an exception as a result; none of the supplied components do so.

VisualComponent has been changed to use our new layout architecture. The protocol is regarded as largely private, and it will change.

Note that widgets that use artists delegate their geometry to the artist.

Convenience Methods

There are methods on Graphics.VisualComponent that return the x and y components of the preferred extent, namely #preferredWidth and #preferredHeight. These methods simply extract the appropriate axis from the preferredExtent, so it is almost always better to get the preferredExtent as a point and extract the x and y coordinates yourself to avoid computing the extent twice. There is also a #preferredBounds method that returns the extent as a zero-origin Rectangle. Note however that this is not necessarily or even likely to be the bounds of the component.

Such convenience methods are primarily due to the history of the API, and so they don't exist for minimumExtent and maximumExtent.

VisualComponent also has convenience methods that account for the alignment insets of the component, namely #alignmentBaseline, #alignmentMinimumExtent, #alignmentPreferredExtent, #alignmentMaximumExtent, #alignmentBounds and #alignmentBounds:. These are primarily for use by the layout mechanism.

Geometry Change Notification

Whenever the preferred geometry of a component changes, i.e., with changes to the preferredExtent, minimumExtent, maximumExtent, alignmentRectInsets or baseline, the component needs to broadcast that fact up the tree. It does that by sending #changedPreferredGeometryForComponent: to its container. A convenient way to trigger this is to send #changedPreferredGeometry to the component itself.

Transitioning Pre-8.0 Code

Any existing implementations of #preferredBounds, #preferredHeight or #preferredWidth should be replaced with a #preferredExtent method.

Custom components should implement #minimumExtent, #maximumExtent, #baseline and #alignmentRectInsets if appropriate. If you are subclassing a supplied component, then the implementation will probably already be appropriate.

Replace sends of #changedPreferredBounds: and #changedPreferredBounds:forComponent:. Instead use #changedPreferredGeometry.

Any use of Rectangle and Point as layout specifications should be changed to use a Layout object. The new code will now transform Rectangles and Points using #asLayout before assigning them. You should treat the layout instance variable as private and only use the #layout and #layout: accessors. This will affect not only layout specification, but also any code that dynamically modifies component position or size. Presuming that your container is using a TraditionalLayoutManager, which is the default, then you should change such code to set a new Layout object on the component in question, rather than moving the component manually. In particular, the movement and sizing operations are not functional before the layout has been resolved (e.g., in #postBuildWith:).

Layout Managers

The important components from the perspective of layout are the Container components. In the base image, they are of two types: CompositePart and PartPort. The PartPort layout isn't configurable — it makes each of its children the same size as itself and displays one of them at a time, according to a model value. It is used primarily for tab controls.

All other layout is the responsibility of CompositePart and its subclasses, the most significant of which is Panel. CompositePart is often used directly, and although it does have a fair number of specialized subclasses, apart from Panel, these aren't meant for generic layout.

Panel was an initial attempt to migrate away from our current Wrapper-based widgets. Unlike CompositePart, its children do not have to be wrapped in Wrappers. It has a pluggable layout mechanism that has nothing in common with our new layout mechanism, and won't be discussed further in this document.

Each CompositePart has a layout manager, which is implemented in the abstract by LayoutManager. The layout manager is responsible for laying out the children of a CompositePart as well as computing the resultant min/max/preferredExtent and baseline. The layout manager has a reference to its part, so therefore there is a 1:1 relationship between a CompositePart and the LayoutManager subclass.

CompositePart has a default layout manager that is used if no layout manager is explicitly set, provided by the #defaultLayoutManager method. This method is overridden in a number of subclasses to implement specialization to the layout algorithm. It is this method that is called to create a CompositePart's layout manager. Of course, you can set the layout manager yourself using the #layoutManager: method.

The most common layout manager, at least initially, will be the TraditionalLayoutManager, which implements the pre-8.0 layout behavior of CompositePart. Layout is generally not customized by defining new types of layout manager. Rather, a given layout manager is configured to lay out the children of its CompositePart.

The minimum, preferred and maximum extents and baseline of a CompositePart are computed by its layout manager and are cached in the part.

LayoutManager Model

LayoutManager has one important method: #layoutComponentForBounds:, which computes the bounds for each child; the minimum, preferred and maximum extent; and the baseline of its associated CompositePart.

Layout Invalidation

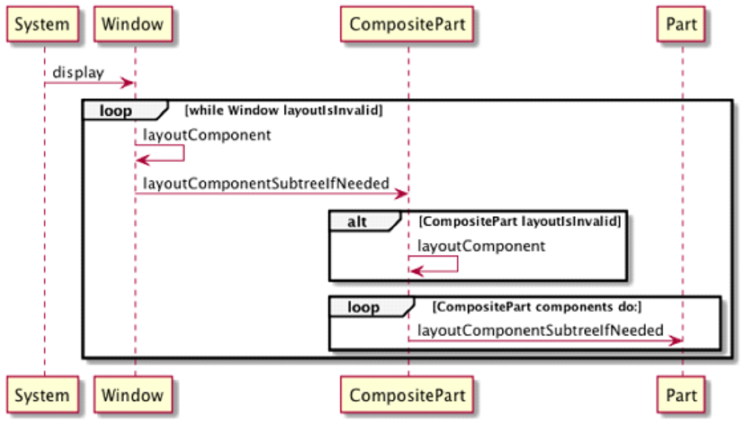
Layout is not triggered directly. Instead the layout of a CompositePart is marked as being invalid and needing to be recomputed, and the framework ensures that layout is done before the component is displayed. Layout can be invalidated on a CompositePart by sending invalidateLayout. In addition, the part will invalidate itself in response to a change in the preferred geometry of any child. When layout is invalidated, the component is also display-invalidated, which will eventually trigger a redisplay that will then trigger layout.

Layout Algorithm

Traditional layout in VisualWorks has the property that the layout of a component is independent of the layout of its siblings. That is, there is no interaction between sibling components when determining where they are placed. While this has a number of nice properties for implementation, it can present problems when building interfaces. The purpose of UILayout is to change this so that components can specify their layout in relation to their siblings (e.g., labels aligned with text field baselines, distances from label groups to field groups, etc.). None of which are constants in a world of changing looks and feels/ skins and internationalization.

However, a nice characteristic of the traditional layout is that a component can be positioned as soon as it is added to a container without needing to do any work on sibling positions. Thus, traditionally layout has been done when a component is added to a container. If, however, the layout needs to involve the entire container in a constraint-satisfaction manner (which most layout devolves to), then we want to defer layout. Deferring layout is easy. We have a flag on CompositePart, namely #layoutIsInvalid, which indicates that layout should be resolved before the window is displayed.

The control flow is represented in the following diagram, where obviously there can be nested containers when a Part is itself a CompositePart. Note that invalidating the layout on a CompositePart also sets the flag on the window as well, and the outer loop ensures that a layout that triggers further layout invalidation will be fully resolved. In practice, there is a limiting counter to ensure that an infinite loop of layout invalidation doesn't occur.


System
Window
CompositePart
Part
display
loop [while Window layoutIsInvalid]
layoutComponent
layoutComponentSubtreeIfNeeded
alt [CompositePart layoutIsInvalid]
layoutComponent
loop [CompositePart components do:]
layoutComponentSubtreeIfNeeded
System
Window
CompositePart
Part

One complication is that currently layout is triggered down through the view tree by the #bounds: method. But changes in preferred geometry also affect layout, and these are triggered up through the view tree by the #changedPreferredGeometryForComponent: method. What we don't want is for a component to be laid out, which changes its preferred extent, transmitting that change to its parent, which would think that it then needs to redo layout of that child.

Design your layout code so that the vertical layout of each widget is calculated after the horizontal layout, and the horizontal layout is entirely independent of the vertical layout. The vertical layout can (and probably will) be dependent on the horizontal layout (i.e., the standard text-layout methodology).

NativeGUIPolicy

One of the design principles for UISkinning is the idea that although the look of an application might change depending on the skin, the feel should always match the underlying platform. Users can adapt easily to a different appearance, but are distressed when their muscle-memory expectations aren't met. Thus skins are purely about look, and platform-compliant feel parameters are provided by Graphics.NativeGUIPolicy and its subclasses.

NativeGUIPolicy also provides access to some visual characteristics of the native platform, such as system fonts and sizes and selection colors, which the default skin uses. You can get an instance of the appropriate subclass for the current platform from NativeGUIPolicy class>>current. The values provided by this object are used to parameterize behavior as well as the skin.

Fonts, Sizes and Widget Alignment

One of the goals of UISkinning was to improve the look of existing applications without requiring customers to make extensive modifications to their applications. For many classes of applications, especially “form-based” UIs, this has been achieved. You can see the results in the VW development tools, which in most cases, haven't required any adjustment.

However, for some use-cases, especially where you wish to adjust the sizing, font or color, you will find that the widgets are no longer as customizable as they previously were. In some cases, this was necessary to achieve the aforementioned goals, and in other cases, it is simply work that is not yet complete.

Existing widgets were often customized without reference to a baseline configuration since there was really no reliable way of knowing if the widget was set to a “default” size or if it had been intentionally sized to certain dimensions. In other cases, widgets were customized in order to achieve a better platform look specific to a given version of a given platform. Sometimes sizing was used to achieve layout goals (i.e., making a label a certain size and expecting the text to center). In order to achieve a native look, widgets generally need to have a certain font, size and color. We found that the best way to make 99 percent of existing interfaces look good with respect to alignment with these new widget looks was to make them bottom-aligned.

The remaining issue is non-technical. People build UIs on one platform and want them to look right on all platforms. We do not encourage manually sizing your widgets, since that will likely interfere with their ability to adapt their presentation on multiple platforms. A widget without a manual size (or, as is the case with the current implementation, when widgets ignore the assigned size in a given dimension), can adjust its dimensions correctly according to the skin. So an unsized button has a better chance to look right on all platforms.

Removal of Graphics.ComposingComposite and Graphics.RecomposingComposite

Uses of these classes can be trivially replaced by Graphics.CompositePart using either a custom LayoutManager or a subclass of CompositePart that overrides #layoutComponent.

RecomposingComposite in particular is not needed because every CompositePart now has that behavior.

Replace Calls to #pressAction with #trigger

With the introduction of ViewEventController as the first step toward improving our widget event processing, the #pressAction method of various button controllers has been deprecated in favor of the more universally implemented #trigger that allows either the view or the controller to respond to the click event as appropriate.

The reasoning behind this is that the selector #pressAction describes a particular UI technique (pressing) rather than describing the desired effect (triggering).

Replace ComboBoxInputFieldView

Replace references to ComboBoxInputFieldView with ComboBoxView.

The ComboBox widget has been significantly restructured in order to, among other things, make its drop-down menu behave according to platform expectations. Classes have been renamed and accessing internal components of this composite widget have changed. Please review any extensions you have made to this widget or any code that references the internal components.

Progress Bar Widgets Don't Follow User Color Settings or Show the % as Text

Since the old look isn't platform-compliant and it doesn't correspond to any native look, it is no longer available. Currently you need to do a custom skin to get that appearance. We are looking into providing some additional custom widgets in future releases.

VisualWave UI Construction in a Headless Server

The changes for UISkinning have impacted VisualWave's ability to appropriately identify a row/column position for certain widgets (most notably text labels) when creating HTML tables to generate a web layout that approximates that of the native VisualWorks window.

Since this issue only shows up with a headless image, we believe that this is due to the absence of a reference font description that can be used to determine a widget's bounds in relation to its neighbors.

The workaround is to modify any widgets in the UIPainter so that any text (especially for labels) uses the HTML default font instead of relying on the System (Widget Text). Unfortunately, this is not easily automated, since it requires adding a new line to the spec to specify the font for the widgets in question, as opposed to changing an existing parameter in a spec.

Menu Loop

Prior to 8.1, menus used multiple nested event loops — one per menu invocation. The code was very difficult to reason about, which affected our ability to achieve platform-compliant menu behavior. We now have a single nested event loop for all menu interaction, with a per-platform subclass of UI.MenuActivation extending the generic handling to provide the platform-compliant behavior.

This shouldn't affect client code unless you have customized menu event handling. We suggest that you don't do that for the reasons stated above. However, if there is some additional behavior you think we should provide, please contact Cincom Support with your requirements.

HostGraphicsSupport

VisualWorks 8.1 includes the addition of HostGraphicsSupport, a new class hierarchy that reflects the graphics capabilities of the host environment, not just its operating system. This is relevant when there is more than one host graphics environment, eg: X11 and Cocoa. Be aware that you may need to move extensions from Win32SystemSupport and the like to their equivalent in the HostGraphicsSupport hierarchy.

Double-Buffering is now enabled by default

If you use a library that directly writes to a window or pixmap you may need to disable double buffering for your window.

Text2 Word-splitting Enhanced

A word with two different stylings in Text2 would previously be allowed to split across lines. This was considered a bug and will no longer happen in VisualWorks 8.1.

Internationalization

Enhancements to Class Locale

Use the "isHintable" attribute to exclude using hints for some Locales when reading printed Timestamps for the purpose of instantiating new Timestamp objects. This speeds up the parsing attempt in some cases (AR 71396).

Fixes to Message Catalogs

Clarify that the result of #defaultMessageText is a String rather than a UserMessage — this was causing errors. Fixed a typo of "self subclassResponsibility" that was obviously in error and causing problems (AR 71284, AR 71826).

OpenTalk

Profiler and debugger tools removed

The following parcels from preview/opentalk have been removed from the distribution:

Opentalk-Debugger-Remote-Monitor

Opentalk-Debugger-Remote-Target

Opentalk-Profiler-Core

Opentalk-Profiler-Tool

Store

Enhancements

The RBStoreExtensions add-on incleds fixes and improved display options.

Testing

Enhancements

The RBSUnitShowResult parcel appears in the **Testing** category in the Parcel Manager, though the more basic RBSUnitExtensions remains the utility presented in **Popular** category. The difference between them is that RBSUnitShowResult is more pluggable. I.e., it is easy to plug in suite classes and/or result classes to randomise SUnit test runs, or avoid test UI effects in very fast tests, or suppress TestResource interactions and show TestResource failures separately, or support additional test outcome classes, etc.

Loading the RBSUnitShowResults parcel adds a settings page, **Testing**, to the Tools settings section. This provides a UI to lets users set default TestSuite and TestResult subclasses.

Tools

Enchancements

- When you search for methods the find tool pops open to allow you to use the **Find Next** and **Find Previous** hot keys to move between search results in the method you've selected. This restores functionality from VisualWorks 3.
- You can now click on misspelt selectors and see a suggested correction in the menu. This also works on misspelt pragmas — and misspelt variables.
- The **Shared Variables** menu has been reorganised to put the commonly used functions at the top.
- Disabling all breakpoints now actually disables all breakpoints, not just ones which check DebugActive.
- Classes will now sort in to hierarchy by default (you can change this back to alphabetical in the settings). (RBHierarchicalClassesExtensions)
- Finding methods containing a 'string matching...' is now available from the **Browse** menu in the Launcher.
- The debugger now has more commonly used return types for **Execute** > **Return…** such as true and false.

- Auto-Complete now also suggests shared variables.
- You thought we'd gotten rid of conditional breakpoints? We hadn't, they just weren't obvious enough, so we've made it more obvious that you can write code between the a breakpoint, e.g.: `[aPerson name = 'bob']`.
- Watches are back, both for watching variables and also doing an arbitrary watch expression. You can either return an object inside the condition or you can write to the provided pseudo '`log`' variable, which is a regular `WriteStream`.
- Browsing references to a temporary variable highlights the variable uses in-place in your current method.
- Browsing references to an instance variable highlights the variable uses in-place in your current method.
- When creating a new method from the debugger, you will now be asked what protocol you want to put it in (`DebuggerProtocolPrompt`).
- Double-clicking on a quote now recognises a quote-quote as an escape and selects the whole string.
- You can now delete-next-word and delete-previous-word. Did you know you can also move and select by semantic meaning too? Try it, command+alt+(shift)+arrows or control+alt+(shift)+arrows.

Virtual Machine

Linux baseline version upgraded to Glibc 2.5, libx11 1.0.3, zlib 1.2.3

Updated our baseline versions of Linux libraries and the compiler, and moved to a more modern Linux platform for VM builds. This is a modest upgrade that keeps us current with versions commonly in use (AR 71357, AR 72130).

OS X baseline version upgraded to 10.8 (Mountain Lion), support for OS X on PPC discontinued

Another modest upgrade. Version updates to OS X are happening more frequently these days, this positions us to support modern versions of the operating system with our OS X VMs. PowerPC is no longer supported on modern versions of OS X (AR 71854, AR 71928).

Adapt to OS API changes on Solaris 11

On Solaris11, some sockets that answer FALSE to isastream() nonetheless can generate POLL signals using the stream-based ioctl(). The change is on AIX and Solaris to just unconditionally try the stream-based API, and if that fails, try the non-stream alternatives. This is more reliable and supports operation on more modern versions of Solaris (AR 71606).

WWW

User Authentication with LinuxPAM libraries

VisualWorks 8.1 includes support for the LinuxPAM libraries. The LinuxPAM parcels can be loaded using the Parcel Manager from the **Security** directory. More information about the libraries can be found at:

http://linux.die.net/man/3/pam

or:

http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_ADG.html

The LinuxPAM parcel includes:

- LinuxPAMInterface, an interface to the libraries.
- Class PAM, which provides #authenticate:with:using: utility to authenticate a user. The method uses a callback that allows communication between a loaded module and the application.

The PAM>>authenticate:with:using: utility is used in the AppeX demo to authenticate a user. To see the demo, load the parcel: AppeX-Examples-LinuxPAM and start the TestLinuxPAM server. The web page asks to enter a service name, user id and password. The policy for the service will be read from the file **`/etc/pam.d/service_name`** or, if that file does not exist, from **`/etc/pam.conf`**. The PAMAuthFilter authenticates the user credentials for a specified service and then allows sending messages to the server in an authenticated session.

Service name

Specifies the name of the service to apply. The policy for the service will be read from the file **`/etc/pam.d/service_name`** or, if that file does not exist, from **`/etc/pam.conf`**.

User name

The name of the target user.

Password

The value to verify.

Clicking on **Authenticate the user** sends a request to the server to verify the user password using LinuxPAM library.

Support for Reusable CSS Themes

AppeX web applications can now include reusable CSS themes, with a Javascript API to dynamically change the UI theme based on application preferences. For example, developers can create themes for certain types of users, a certain locale, or a type of device (desktop vs. mobile).

Support for Scaffolding and ActiveRecord

In VisualWorks 8.1, AppeX now includes parcels implementing the ActiveRecord interface and a web-based GUI Scaffolding framework.

The AppeX-ActiveRecord parcel includes an API that streamlines the delivery of JSON-encoded ActiveRecord objects to the client side of AppeX via a REST-ful interface using the HTTP GET, POST, PUT and DELETE requests.

The AppeX-Scaffolding parcel provides a mechanism to render HTML5 elements in a web browser from JSON-encoded ActiveRecord objects. With minimal coding, developers can build a working Single Page Web Application from an existing database schema. A GUI tool to support the creation of such applications is included in the AppeX-Scaffolding-Tool parcel, which is now in preview.

Enhanced support for RequestFilter

Class RequestFilter has been enhanced such that a filter can now stop request processing by creating a response and raising ResponseReady exception. The exception can be raised for error and non-error responses.

Documentation

Application Developer’s Guide

Discussion of VMs and debugging strategies updated.

COM Guide

Updated in this release. Added discussions of some new functionality.

Database Guide

VisualWorks 8.1 includes support for the ODBC 3.0 APIs in the ODBC3EXDI parcel, located in the preview directory. For details and examples illustrating the use of these APIs, refer to the discussion of "ODBC 3.0 Support" in chapter 1 of the *Database Application Developer's Guide*. Going forward, in VisualWorks 8.2, the 3.0 API functionality will be merged into the ODBCEXDI package, so that it will no longer be necessary to load a separate package.

DLL & C Connect Guide

Minor corrections and updates in this release.

Installation Guide

Updated in this release.

Internationalization Guide

Updated in this release.

Opentalk Guide

The *Opentalk Opentalk Communication Layer Developer's Guide* has been revised to reflect the obsolescence of Opentalk support for HTTP, CGI, SOAP, and XML. Opentalk-HTTP(S) has been replaced with Opentalk-Web(-Secure), and Opentalk-CGI has been replaced by SiouX.

Source Code Management Guide

Updated in this release. Clarify discussion of setup and configuration of Store with SQLite.

Tools Guide

Updated in this release.

Walk-Through

Completely updated in this release.

Web Application Developer's Guide

Updated in this release to cover support for scaffolding. Added an index. Miscellaneous updates.

Web Server Developer's Guide

Updated in this release.

Web Service Developer's Guide

Minor corrections in this release.

Update URLs in Old Release Notes

Previous Release Notes have been updated to include current URLs for the Cincom Smalltalk web site.

This affects the older Release Notes for VisualWorks 7.1, 7.3, 7.4, 7.5, 7.6, 7.7, 7.7.1, 7.8, 7.8.1, 7.9, and 7.10.

3

Deprecated Features

By deprecating certain features, we remove them from the system. These are made available for a limited time as parcels in the obsolete/ directory, to provide you the opportunity to port applications away from using the features before they are removed altogether. This directory is on the default parcel path.

Virtual Machine

Obsolete VM command line switches

(AR 69111) The VM command line switches **`-h`**, **`-z`** and **`-nologo`** have been obsoleted. Instead, use the switches **`-m6`**, **`-m5`**, and **`-noherald`**, respectively.

IGC primitive changes

(AR 69147) In this release, primitive 324 (primFinishedIGCIsInterruptible:objects:bytes:) has been deprecated. If your application depends on this primitive, use primitive 449 instead (primMeasureIGCIsInterruptible:objects:bytes:).

Support for short compiled methods has been removed

(AR 69339) With this release, the VM no longer supports short compiled methods (that is, compiled methods that encode bytecodes using small integers). The image no longer creates short compiled methods, which in turn causes a small image size increase. If this image growth is a concern, consider using the UniqueObjectTable and UniqueObjectTable-BytecodeArray (in the **`parcels`** directory).

Removed primitives

(AR 69340) With this release, the VM no longer implements primitives 255, 460, 470, 770, 771, 997, 1202 and 1215.

Replaced I/O primitives with error holder arguments, errorCode instance variable

(AR 69341) In this release, I/O primitives taking an error holder argument have been deprecated. Equivalent primitives that do not take error holder arguments are now available. Specifically, the replaced primitives are as follows:

```
615 -> 1615
616 -> 1616
617 -> 1617
650 -> 1610
670 -> 686
681 -> 684
```

Moreover, primitives 652 through 655 have been modified not to take an error holder argument.

In general, I/O primitives no longer write to the `errorCode` instance variable of `OSErrorHolder` and its subclasses. Instead, the error is passed on the stack via a pseudo-temporary variable upon primitive failure.

Base Image

Class MemoryPolicy is obsolete

(AR 69086) In this release, the class `MemoryPolicy` is now provided as an obsolete package only. Users should migrate their custom memory policies to the default memory policies provided with the system.

DotNET Connect

DotNETMarshaler removed

(AR 68510) In this release, the class DotNETMarshaler has been removed. It was used in older versions of DotNETConnect to marshal parameters of type System.Object. This means that proxy code that was generated before VisualWorks 7.7 cannot be used in VisualWorks 8.0 anymore. It needs to be regenerated.

Application Server

VisualWave GUI Development No Longer Supported

(AR 70629) VisualWorks 8.0 does not support VisualWave GUI development, namely the UIPainter extensions for VisualWave are henceforth obsolete. Existing VisualWave applications, however, will continue to work in this release.

Naturally, the UIPainter for VisualWave can be used in a pre-8.0 release, while VisualWave applications would still work in the 8.0 release. This functionality could be reinstated if it were to become economicaly viable. For details, please contact the VisualWorks Product Manager.

4

Preview Components

Several features are included in a preview/ and available on a “beta test,” or even pre-beta test, basis, allowing us to provide early access to forthcoming features. Several are described in the following sections. Browse the directory contents for last minute inclusions.

Database

Support for ODBC 3.0

Load the preview parcel ODBC3EXDI to have the option of accessing ODBC databases via the ODBC 3.0 protocol instead of our current standard ODBC 2.0. We expect to release this ODBC 3.0 driver in VisualWorks 8.2 as part of the existing ODBCEXDI parcel, possibly as its default with the ODBC 2.0 option being made secondary. Thus, the preview parcel will be dropped from 8.2.

Glorp

Reading DescriptorSystems from Existing Databases

Several new packages are available to assist with reading descriptor systems from existing databases.

GlorpAtlasSystemsInVW

The GlorpAtlasUI is a simple VisualWorks tool that kickstarts fresh application development from a legacy database application. It lacks the full power of the ObjectStudio mapping tool (its UI employs lists and checkboxes, not a garphical model of a schema), and of course none of the mapping tool's integration with the modeling tool. However, it does provide VisualWorks' users a UI with which they can get started on Glorp schema for a legacy database if they lack the ObjectStudio suite (e.g. if they are on a non-Windows platform.

If a Glorp model has been prepared in ObjectStudio's Modelling and Mapping tools and saved to a parcel or package, load this utility to enable loading that component into a VisualWorks image.

GlorpAtlasClassGeneration

This package extends the GlorpActiveRecord utilities. Like Active Record, it automatically reads the schema to create a descriptor system that maps between them, but it can read from schemas that do not adhere to the Active Record pattern, and it can generate classes that match the tables.

GlorpAtlasSystemGeneration

This supports generating a DescriptorSystem subclass, with classFor<Name>:, descriptorFor<Name>: and tableFor<Name>: methods, from a descriptor system. (Typically, this will be a descriptor system created when a DB's metadata is read by GlorpActiveRecord or GlorpAtlasSystemGeneration.)

For detailed API descriptions, see their package comments. The GlorpAtlasTests package provides some usage examples. Load them in VisualWorks only; ObjectStudio's modelling and mapping tools provide a UI on a more powerful version of the same capability.

Universal Start Up Script (for Unix based platforms)

This release includes a preview of new VW loader that runs on all Unix and Linux platforms. This loader selects the correct object engine for an image, based on the image version stamp. Formerly, the only loader of this sort was for Windows.

The loader consists of two files and a readme in preview/bin. Installation and configuration information is provided in the readme.

This loader is written as a standard shell script which allows it to be used to launch VW on virtually any Unix based platform. This opens up the possibility of having a centrally managed site-wide installation of an arbitrary set of VW versions allowing users to simply run their images as executables without any user specific setup required. The loader figures out which version of VW and which specific VM is needed to run the image being launched, using information provided in the INI file).

For installations using only final releases (not development build releases), a single entry line in the INI file for each VW version will suffice to serve any Unix based platform for which a VM is available at the specified location.

Base Image for Packaging

/preview/packaging/base.im is a new image file to be used for deployment. This image does not include any of the standard VisualWorks programming tools loaded. The image is intended for use as a starting point into which you load deployment parcels. Then strip the image with the runtime packager, as usual.

BOSS 32 vs. BOSS 64

The current implementation of BOSS (boss32, version 7), does not accomodate 64-bit small integers and small doubles natively. Also, it does not support extremely large objects that are outside the implementation limits for 32 bits. Furthermore, since the implementation of identityHash is not equal in 32 and 64 bit images, identity based collections may require special handling when moving them across images of different bit size.

A preview implementation of boss64 (version 8) has been implemented for this purpose. This implementation is an addition to the existing BOSS parcel, and is called BOSS64.

The new BOSS implementation has been structured so that there is a new factory class that takes care of choosing the proper reader for either boss32 or boss64 without user intervention, and a similar factory arrangement that chooses either boss32 or boss64 as the output format depending on the image BOSS is running on.

More concretely, until now application code would have referred to BinaryObjectStorage to write BOSS streams in boss32 format:

```
BinaryObjectStorage onNew: aStream
```

Referencing the class BinaryObjectStorage64 instead will result in BOSS streams in boss64 format:

```
BinaryObjectStorage64 onNew: aStream
```

Finally, referencing AbstractBinaryObjectStorage will choose either boss32 or boss64 depending on the image in which the code is running:

```
AbstractBinaryObjectStorage onNew: aStream
```

Moreover, referencing the abstract factory class for reading,

```
AbstractBinaryObjectStorage onOld: aStream
```

will automatically determine the format of the stream and choose the appropriate reader accordingly:

Execution environment	**Selected reader**
32-bit image, 32-bit BOSS stream	BOSSReader
64-bit image, 32-bit BOSS stream	BOSSReader32
64-bit BOSS stream	BOSSReader64

Existing code making reference to classes already present before these changes will not be affected, and they will still rely on existing boss32 behavior.

Also, although boss64 streams can be written by 32 bit images, 32 bit images should write BOSS streams in 32 bit format because 64 bit images can read these BOSS streams while doing all the necessary conversions.

64-bit Image Conversion

The ImageWriter parcel is still capable of converting arbitrary 32- bit images to 64-bit images. However, due to an unresolved race condition, occasionally it may create an image that brings up one or more error windows. These windows can safely be closed, and if the 64- bit image is saved again, they will not return.

However, they may be problematic in a headless image or an image that has been produced by the Runtime Packager. For such cases, re-saving or recreating the original 32-bit image, and then converting it again may avoid the race condition. Alternatively, converting the image to 64 bits before applying the Runtime Packager or making the image headless may also be helpful.

ImageWriter empties all instances of HandleRegistry or its subclasses. Since these classes have traditionally been used to register objects which must be discarded on startup, emptying them during the image write is safe. But if your code is using HandleRegistry or a subclass to hold objects which are intended to survive across snapshots, ImageWriter may disrupt your code. Running ImageWriter before initializing your registries may solve this problem. We would also like to know more about how you use HandleRegistry, in order to improve ImageWriter's ability to transform images without breaking them.

Tools

With the Trippy basic inspector now being much more robust, work was done on integrating this with the debugger. Nicknamed Diggy, it has not been finished yet, but may be loaded from **`preview/parcels`**.

Note: Given the ongoing renovation of tools in VisualWorks 8.1, Diggy won't be updated for 8.1. If Cincom updates Diggy again it'll be to integrate it in to the product fully. It might go a different direction, though still with the intention of unifying the inspector tools, especially in the debugger.

At this writing, this causes the inspectors located in the bottom of the debugger (the receiver and context fields inspectors) to be basic trippy inspectors (not the entire diving/previewing inspector, just the basic tab). This makes operations between the two the same, and provides the icon feedback in the debugger. The stack list of the debugger also shows the receiver type icons.

Cairo

Overview

Starting in release 7.8, VisualWorks includes a preview of ready-to-use Cairo libraries for MS-Windows (Windows 2000 and newer) and Apple OS X (10.4 and newer, PowerPC or Intel processors). The prebuilt libraries are built from 1.8.8 stable release sources. It also includes version 7.7.1 - 1 of the CairoGraphics parcel which binds to said libraries.

Developers on MS-Windows and OS X, should be able to simply load the CairoGraphics parcel and begin using Cairo.

Developers on X11 platforms may also use the CairoGraphics parcel, but will need to make sure VisualWorks have libcairo.so available in their library path. Most up to date Linux versions ship with Cairo binaries.

What is Cairo?

The main project page found at http://cairographics.org/ states:

> Cairo is a 2D graphics library with support for multiple output devices. Currently supported output targets include the X Window System, Quartz, Win32, image buffers, PostScript, PDF, and SVG file output.
>
> Cairo is designed to produce consistent output on all output media while taking advantage of display hardware acceleration when available (e.g. through the X Render Extension).
>
> The cairo API provides operations similar to the drawing operators of PostScript and PDF. Operations in cairo including stroking and filling cubic Bézier splines, transforming and compositing translucent images, and antialiased text rendering. All drawing operations can be transformed by any affine transformation (scale, rotation, shear, etc.).
>
> Cairo is implemented as a library written in the C programming language, but bindings are available for several different programming languages.
>
> Cairo is free software and is available to be redistributed and/or modified under the terms of either the GNU Lesser General Public License (LGPL) version 2.1 or the Mozilla Public License (MPL) version 1.1 at your option.

The CairoGaphics parcel is a VisualWorks bridge to the Cairo graphics library. It is maintained as an open source project, hosted in the Cincom Public Repository.

Drawing with Cairo

This section describes a simple overview of drawing with Cairo with VisualWorks. It is not exhaustive, but rather demonstrative.

Getting a Cairo Context

A Cairo context is the object which defines transactions that draw things on a given surface. Cairo may be used to draw on 3 different types of VisualWorks surfaces: Windows, Pixmaps, and Images. For the first two one usually has a VisualWorks GraphicsContext object in play for the surface. To help manage resources efficiently, we use a while: aBlock pattern to create Cairo interface objects and release them efficiently.

```
aVWWindowGC
   newCairoContextWhile: [:cr | "...cairo work goes here..."].
aVWPixmapGC
   newCairoContextWhile: [:cr | "...cairo work goes here..."].
aVWImage
   newCairoContextWhile: [:cr: | "...cairo work goes here..."].
```

By convention, in the Cairo community in large, as well as in VisualWorks code, a Cairo context variable is always called a cr. Using this convention increases the likelihood that other Cairo programmers (both Smalltalk and for other language bindings) will understand your code.Cairo also has its own built in Surface type called an ImageSurface. These are like VisualWorks Pixmaps, but are managed by Cairo. They are created with a format code and an extent.

```
cairoImage := ImageSurface
   format: CairoFormat argb32
   extent: 100@100.
cairoImage
   newCairoContextWhile: [:cr | "...cairo work goes here..."].
```

Note: The dual-threaded VMs for 10.5 and 10.6 are not compatible with the CairoGraphics on OS X. The problem involves drawing on Pixmaps. On 10.6, the Pixmaps are blank, and on 10.5, this operation wll quickly crash the image.

Setting the Source

In VisualWorks parlance, the source is somewhat analogous to the paint of a GraphicsContext. Cairo operators will draw pixels from the source onto the target surface. Sources may be simple color values, linear and radial gradients, and other cairo surfaces.

Setting a simple ColorValue

```
cr source: ColorValue red
```

Setting an alpha channel weighted color

```
cr source: (ColorBlend red: 0.9 green: 0.2 blue: 0.3 alpha: 0.5).
```

Setting a vertical green to blue linear gradient

```
gradient := LinearGradient from: 0 @ 0 to: 0 @ 10.
gradient addStopAt: 0 colorBlend: ColorValue green.
gradient addStopAt: 1 colorBlend: ColorValue blue.
cr source: gradient.
```

Setting a radial translucent orange to yellow gradient

```
gradient := RadialGradient
            from: 0 @ 0
            radius: 0
            to: 0 @ 0
            radius: 100.
gradient addStopAt: 0 colorBlend: (ColorBlend orange alpha: 0.5).
gradient addStopAt: 1 colorBlend: (ColorValue yellow alpha: 0.2).
cr source: gradient.
```

Setting the file background.png as the soruce

```
surface := ImageSurface pngPath: 'background.png'.
cr source: surface.
```

Defining Shapes

Shapes in Cairo are defined by paths. A path is more than a simple polyline. A path is composed of a series of move, line, bezier curve, and close commands. They do not need to be contiguous. Defining a path does not actually cause it to be rendered to the context.

The following examples are a sample of the path creation methods found in the paths and handy paths method protocols of the CairoContext object.

Simple line from 0,0 to 40,50

```
cr
   moveTo: Point zero;
   lineTo: 40 @ 50.
```

Two disjoint rectangles

```
cr
   rectangle: (10 @ 20 corner: 30 @ 40);
   rectangle: (110 @ 120 extent: 40 @ 40).
```

Closed right triangle with leg length of 30

```
cr
   moveTo: 5 @ 5;
   relativeLineToX: 0 y: 30;
   relativeLineToX: 30 y: 0;
   closePath.
```

Cincom Logo in unit coordinate space

```
cr
   moveTo: -1 @ 0;
   arcRotationStart: 0
      sweep: 0.75
      center: 0 @ 0
      radius: 1;
   relativeLineTo: 0 @ -2;
   arcRotationStart: 0.5
      sweep: 0.25
      center: 1 @ 0
      radius: 1;
   lineTo: 1 @ 0.
```

Filling and Stroking Shapes

With a source set and a path defined, you can stroke and/or fill the shape. The messages stroke and fill can be sent to a cr. In both cases, the path is reset by the call. The messages strokePreserve and fillPreserve, cause the path to remain in effect even after the operation.The stroke width may be set with the strokeWidth: aNumber method. Stroking is a solid line unless a dashes: anArrayOfLengths offset: aNumber is set.

Draw a blue rectangle with a thick dashed red outline

```
cr
   rectangle: (10 @ 10 extent: 40 @ 40);
   source: ColorValue blue;
   fillPreserve;
   source: ColorValue red;
   strokeWidth: 3;
   dashes: #(1 2 3 4) offset: 0;
   stroke.
```

Additional Operators

Stroke and fill are the most common operators performed on a context. There are others that may be used:

paint

Like fill, but requires no path. Simply fills the entire clip region.

paintAlpha: *aNumber*

Like paint, but applies a uniform alpha adjustment during the operation.

maskSurface: *aSurface*

Paints the current source using the alpha channel of aSurface.

clip

Renders nothing, but intersects the current clip with the current path and clears the path. Use clipPreserve if path resetting is not desired.

Affine Transformations

Cairo uses a transfomation matrix at all levels of drawing. The matrix is described as:

$$\begin{array}{ccc} XX & XY & Xo \\ YX & YY & Yo \end{array}$$

Given two input values, Xi and Yi, the new values Xn and Yn are computed as follows:

$$Xn = Xi \bullet XX + Yi \bullet XY + Xo$$
$$Yn = Xi \bullet YX + Yi \bullet YY + Yo$$

The easist way to manipulate the matrix of a context is to use the modifyMatrix: method which takes a single argument block as its argument. For example, to adjust the matrix to be a centered unit coordinate space of the receiver view:

```
cr modifyMatrix:
      [:matrix |
      matrix
         scale: self bounds extent;
         translate: 0.5 asPoint].
```

Matrices may be modified with methods such as translate:, rotate:, scale:, etc. See the transformations method category of class Matrix. Any of these modifications are cumulative to the receiver.

Individual elements may be set as well using accessors such as xx:. The matrix can be returned to an initial unity state by sending initIdentity to it.

Patterns (source surfaces, gradients, etc) have their own matrices as well and also respond to the modifyMatrix: method. It is important to remember when using a pattern's matrix to modify its appearance, the matrix is applied on the source side, where as the context matrix is applied on the target side. In other words, pixels are extracted from the source pattern through the inverse of the matrix. One might translate: 10@20 a cr context to cause things to shift to the right 10, and down 20. But to achieve the same end result by modifying the source's matrix, and leaving the cr's untouched, one would use a translate: -10@-20. Use the reciprocal of any scaling factors in the same way.

The Context Stack

Cairo supports a drawing context stack. This allows one to take a “snapshot” of the current context stake, make changes for further operations, and then at some point “rollback” to the snapshot. The API used is saveWhile: aBlock. These may be nested.

They are particular useful with transformation operations. Consider the following example, which decides a 12-sided equilateral polygon centered around the point 50,50.

```
cr translate: 50 @ 50.
0 to: 1
   by: 1 / 12
   do: [:rotation | cr saveWhile:
      [cr
      rotation: rotation;
      lineTo: 50 @ 0]].
cr closePath
```

Grouping Operations

A final pattern of interest is the groupWhile: aBlock pattern. A group in Cairo terminology refers logically to a series of operations that are buffered to a temporary surface, which then may be used as source for a one time paint operation. This can be used to implement “double buffering” but may also may be used to assemble complex graphis that require multiple surfaces to piece together (e.g. two overlapping linear gradients, one in the vertical direction, one in the horizontal direction).

Deploying VisualWorks with Cairo Support

MS-Windows

The Cairo library is contained in a single cairo.dll file which is placed alongside the VisualWorks virtual machine. Simply include this dll along with your virtual machine executable, and you should have access to the Cairo library.

Mac OS X

Cairo is embedded in the application bundle. It is a set of 3 dylib's placed in a Frameworks directory which is coresident with the MacOS directory found in the application bundle directory structure. If you simply deploy the visual.app application bundle, you shouldn't need to do anything. If you assemble your own application bundle, you will need to ensure that the Frameworks directory contains the 3 dylibs and is a parallel directory to whatever directory the virtual machine executable exists in.

Ongoing Work

The CairoGraphics package is an ongoing work. It is maintained in the Cincom Public Repository. If you find bugs or want to provide enhancements, please do so, publishing your work on a branch version. In the future, Cincom hopes to be able to support Cairo prebuilt libraries on all of its various supported platforms, making it a true piece of the VisualWorks cross-platform strategy, and allowing the VisualWorks IDE to begin to take advantage of the possibilities Cairo offers.Cairo is a 2D vector graphics library. It has rudimentary support for rendering character glyphs from platform fontsets. But it does not pretend to offer any of the higher level CairoGraphics preview services one usually needs to work with text (layout, measuring, etc). A sister project to Cairo called Pango is under investigation by Cincom engineers to also be used in a cross platform fashion, in the same way Cairo is being considered.

Grid

A Grid widget combines elements of the Table and Dataset widgets for a simpler and more flexible interface of viewing and editing tabulated forms. This release includes a preview of a new Grid, based on the Grid from the Widgetry project. It currently supports the following features:

- Multiple row sort by column with or without a UI
- Multiple and single selection options by row or individual cell
- Interactive row or column resize
- Scroll and align column, row, or cell to a particular pane position (e.g., center, left, right top, bottom)
- UIBuilder canvas support

Planned features include:

- A SelectionInGrid model. Currently one may directly access, add, remove, and change elements of the Grid. Direct access will always be available.

- Drag-and-drop rows or columns to add, remove, or sort elements
- A tree column
- Completion of announcements, trigger events, or callbacks
- Specific OS look support for column headers. Currently only a default look is supported.
- The column and row headers may be set to not scroll with the Grid.

Further information on usage and supporting classes with examples appears in /preview/Grid/grid.htm.

Store Previews

Internationalization

MessageCatalogManager supports multiple locales simultaneously

The PerProcessCatalogs package, in preview (preview/parcels), extends the existing MessageCatalogManager facilities to support simultaneous use of multiple locales. This is mainly needed for application servers where different clients may require server side processing to use different catalogs depending on the client locale.

For languages with different variants in different territories, e.g. different english dialects #en_US, #en_GB, etc. the application may require some messages to be localized differently, e.g. 'color' vs. 'colour'. At the same time many other message are common among the territories. To support this efficiently, the Locale searches for the translations in the most specific catalogs first and then looks for less specific ones if that fails. For example, an #en_US Locale may subsequently look into catalogs for en_US, en and finally C.

Opentalk

The Opentalk preview provides extensions to VisualWorks and the Opentalk Communication Layer. It includes a remote debugger and a distributed profiler.

For installation and usage information, see the readme.txt files and the parcel comments.

Distributed Profiler

The profiler has not changed since the last release and works only with the old AT Profiler, shipped in the `obsolete/` directory.

Installing the Opentalk Profiler in a Target Image

If you want to install only the code needed for images, potentially headless, that are targets of remote profiling, install the following parcel:

- Opentalk-Profiler-Core

Installing the Opentalk Profiler in a Client Image

To create an image that contains the entire Opentalk profiler install the following parcels in the order shown:

- Opentalk-Profiler-Core
- Opentalk-Profiler-Tool

Opentalk Remote Debugger

This release includes an early preview of the Remote Debugger. Its functionality is seriously limited when compared to the Base debugger, however its basic capabilities are good enough to be useful in many cases. The limitations are mostly related to actions that open other tools. For those to work, we have yet to make the other tools remotable as well.

The remote debugger is contained in two parcels.

The Opentalk-Debugger-Remote-Monitor parcel loads support for the image that will run the remote debugger interface. The monitor is started by sending:

```
RemoteDebuggerClient startMonitor
```

Once the monitor is started, other images can “attach” to it. The monitor will host the debuggers for any unhandled exceptions in the attached images.

To shutdown a monitor image, all the attached images should be detached first and then the monitor should be stopped, by sending:

```
RemoteDebuggerClient stopMonitor
```

The Opentalk-Debugger-Remote-Target parcel loads support for the image that is expected to be debugged remotely. To enable remote debugging this image has to be “attached” to a monitor, i.e., to the image that runs the remote debugger UI. Attaching is performed with one of the “attach*' messages defined on the class side of RemoteDebuggerService. Use detachMonitor to stop forwarding of unhandled exceptions to the remote monitor image.

A packaged (possibly headless) image can be converted into a “target” during startup by loading the Opentalk-Debugger-Remote-Target parcel using the -pcl command line option. Additionally it can be immediately attached to a monitor image using an -attach [host][:port] option on the same command line. It is assumed that the Base debugger is in the image (hasn't been stripped out) and that the prerequisite Opentalk parcels are also available on the parcel path of the image.

Opentalk CORBA

This release includes an early preview of our OpentalkCORBA initiative. Though our ultimate goal is to replace DST, DST will remain a supported product until OpentalkCORBA matches all its relevant capabilities and we provide a reasonable migration path for current DST users. So, we would very much like to hear from our DST users, about the features and tools they would like us to carry over into OpentalkCORBA.

For example, we do not intend to port any of the presentation-semantic split framework, or any of the UIs that essentially depend upon it, unless there is strong user demand. Please contact Support, and ask them to forward your concerns and needs to the VW Protocol and Distribution Team.

This version of OpentalkCORBA combines the standard Opentalk broker architecture with DST's IDL marshaling infrastructure to provide IIOP support for Opentalk. OpentalkCORBA has its own clone of the IDL infrastructure residing in the Opentalk namespace so that changes made for Opentalk do not destabilize DST. The two frameworks are almost capable of running side by side in the same image. The standard base class extensions, however, like 'CORBAName' can only work for one framework, usually the one that was loaded last. Therefore, if you want to load both and be sure that DST is unaffected, make sure it is loaded after OpentalkCORBA, not before.

This version of OpentalkCORBA already offers a few improvements over DST. In particular, it supports the newer versions of IIOP, though there is no support for value types yet. A short list of interesting features and limitations follows:

- supports IIOP 1.0, 1.1, 1.2
- defaults to IIOP 1.2
- does not support value types
- does not support Bi-Directional IIOP
- doesn't support the NEEDS_ADDRESSING_MODE reply status
- system exceptions are currently raised as Opentalk.SystemExceptions
- user exceptions are currently raised as Error on the client side
- supports LocateRequest/LocateReply
- does not support CancelRequest
- does not support message fragmenting
- the general IOR infrastructure is fleshed out (IOPTaggedProfiles, IOPTaggedComponents, IOPServiceContexts) and adding new kinds of these components amounts to adding new subclasses and writing corresponding read/write/print methods
- the supported profiles are IIOPProfile and IOPMultipleComponentProfile, and anything else is treated as an IOPUnknownProfile
- the only supported service context is CodeSet, and anything else is treated as an IOPUnknownContext
- however it does not support the codeset negotiation algorithm yet; correct character encoders for both char and wchar types can be set manually on the CDRStream class
- the supported tagged components are CodeSets, ORBType and AlternateAddress, and anything else is treated as an IOPUnknownComponent

IIOP has the following impact on the standard Opentalk architecture and APIs:

- there is a new IIOPTransport and CDRMarshaler with corresponding configuration classes
- these transport and marshaler configurations must be included in the configuration of an IIOP broker in the usual way
- the new broker creation API consists of the following methods

- #newCdrIIOPAt:
- #newCdrIIOPAt:minorVersion:
- #newCdrIIOPAtPort:
- #newCdrIIOPAtPort:minorVersion:
- IIOP proxies are created using Broker>>remoteObjectAt:oid:interfaceId:
- there is an extended object reference class named IIOPObjRef
- the LocateRequest capabilities are accessible via
- Broker>>locate: anIIOPObjRef
- RemoteObject>>_locate
- LocateRequests are handled transparently on the server side.
- A location forward is achieved by exporting a remote object on the server side (see the example below)

Examples

Remote Stream Access

The following example illustrates basic messaging capability by accessing a stream remotely. The example takes advantage of the IDL definitions in the SmalltakTypes IDL module:

```
| broker stream proxy oid |
broker := Opentalk.BasicRequestBroker newCdrIiopAtPort: 4242.
broker start.
[   oid := 'stream' asByteArray.
    stream := 'Hello World' asByteArray readStream.
    broker objectAdaptor export: stream oid: oid.
    proxy := broker
        remoteObjectAt: (
            IPSocketAddress
                hostName: 'localhost'
                port: 4242)
        oid: oid
        interfaceId: 'IDL:SmalltalkTypes/Stream:1.0'.
    proxy next: 5] ensure: [broker stop]
```

Locate API

This example demonstrates the behavior of the “locate” API:

```
| broker |
broker := Opentalk.BasicRequestBroker newCdrIiopAtPort: 4242.
broker start.
[   | result stream oid proxy found |
    found := OrderedCollection new.
    "Try to locate a non-existent remote object"
    oid := 'stream' asByteArray.
    proxy := broker
        remoteObjectAt: (
            IPSocketAddress
                hostName: 'localhost'
                port: 4242)
        oid: oid
        interfaceId: 'IDL:SmalltalkTypes/Stream:1.0'.
    result := proxy _locate.
    found add: result.
    "Now try to locate an existing remote object"
    stream := 'Hello World' asByteArray readStream.
    broker objectAdaptor export: stream oid: oid.
    result := proxy _locate.
    found add: result.
    found
] ensure: [ broker stop ]
```

Transparent Request Forwarding

This example shows how to set up location forward on the server side and demonstrates that it is handled transparently by the client.

```
| broker |
broker := Opentalk.BasicRequestBroker newCdrIiopAtPort: 4242.
broker start.
[   | result stream proxy oid fproxy foid|
    oid := 'stream' asByteArray.
    stream := 'Hello World' asByteArray readStream.
    broker objectAdaptor export: stream oid: oid.
    proxy := broker
        remoteObjectAt: (
            IPSocketAddress
                hostName: 'localhost'
                port: 4242)
        oid: oid
        interfaceId: 'IDL:SmalltalkTypes/Stream:1.0'.
    foid := 'forwarder' asByteArray.
    broker objectAdaptor export: proxy oid: foid.
```

```
        fproxy := broker
            remoteObjectAt: (
                IPSocketAddress
                    hostName: 'localhost'
                    port: 4242)
            oid: foid
            interfaceId: 'IDL:SmalltalkTypes/Stream:1.0'.
        fproxy next: 5.
    ] ensure: [ broker stop ]
```

Listing contents of a Java Naming Service

This example provides the code for listing the contents of a running Java JDK 1.4 naming service. It presumes that you have Opentalk-COS-Naming loaded. To run the Java naming service, just invoke 'orbd -ORBInitialPort 1050' on a machine with JDK 1.4 installed.

Note that this example also exercises the LOCATION_FORWARD reply status, the broker transparently forwards the request to the true address of the Java naming service received in response to the pseudo reference 'NameService'.

```
| broker context list iterator |
broker := Opentalk.BasicRequestBroker newCdrIiopAtPort: 4242.
broker passErrors; start.
[   context := broker
        remoteObjectAt: (
            IPSocketAddress
                hostName: 'localhost'
                port: 1050)
        oid: 'NameService' asByteArray
        interfaceId: 'IDL:CosNaming/NamingContextExt:1.0'.
    list := nil asCORBAParameter.
    iterator := nil asCORBAParameter.
    context
        listContext: 10
        bindingList: list
        bindingIterator: iterator.
    list value
] ensure: [ broker stop ]
```

List Initial DST Services

This is how you can list initial services of a running DST ORB. Note that we're explicitly setting IIOP version to 1.0.

```
| broker dst |
broker := Opentalk.BasicRequestBroker
        newCdrIiopAtPort: 4242
        minorVersion: 0.
```

```
broker start.
[   dst := broker
        remoteObjectAt: (
            IPSocketAddress
                hostName: 'localhost'
                port: 3460)
        oid: #[0 0 0 0 0 1 0 0 2 0 0 0 0 0 0 0]
        interfaceId: 'IDL:CORBA/ORB:1.0'.
    dst listInitialServices
] ensure: [ broker stop ]
```

International Domain Names in Applications (IDNA)

RFC 3490 “defines internationalized domain names (IDNs) and a mechanism called Internationalizing Domain Names in Applications (IDNA) which provide a standard method for domain names to use characters outside the ASCII repertoire. IDNs use characters drawn from a large repertoire (Unicode), but IDNA allows the non-ASCII characters to be represented using only the ASCII characters already allowed in so-called host names today. This backward-compatible representation is required in existing protocols like DNS, so that IDNs can be introduced with no changes to the existing infrastructure. IDNA is only meant for processing domain names, not free text” (from the RFC 3490 Abstract).

Limitations

The current implementation in VisualWorks:

- doesn't do NAMEPREP preprocessing of strings (currently we just convert all labels to lowercase)
- doesn't properly implement all punycode failure modes
- needs exceptions instead of Errors
- needs I18N of messages

Usage

You can convert an IDN using the IDNAEncoder as follows:

```
IDNAEncoder new encode: 'www.cincom.com'
    "result: www.cincom.com"
```

or

```
IDNAEncoder new encode: 'www.cìncòm.com'
    "result: www.xn--cncm-qpa2b.com"
```

and decode with

```
IDNAEncoder new decode: 'www.xn--cncm-qpa2b.com'
   "result: www.cìncòm.com"
```

This package also overrides the low level DNS access facilities to encode/decode the hostnames when necessary. Here's an example invocation including a Japanese web site.

```
host := (String with: 16r6c5f asCharacter with: 16r6238 asCharacter),
'.jp'.
address := IPSocketAddress hostAddressByName: host.
   "result: [65 99 223 191]"
```

The host name that is actually sent out to the DNS call is:

```
IDNAEncoder new encode: host
   "result: xn--0ouw9t.jp"
```

A reverse lookup should also work, however at release time we were unable to find an IP address that would successfully resolve to an IDN. This functionality remains untested, and even our example gives only the numeric result:

```
IPSocketAddress hostNameByAddress: address
   "result: 65.99.223.191"
```

MatriX

Polycephaly2 renamed to MatriX

In VisualWorks 7.10, a new name was chosen to avoid unnecessary discomfort caused by the original name. The first version of Polycephaly is not being renamed as it will be obsoleted in favor of MatriX.

MatriX

MatriX (formerly known as Polycephaly) provides simple mechanisms for spawning multiple running copies of a single image and using those to perform various tasks in parallel. This is primarily useful when attempting to utilize hosts with multi-core CPUs. The images are spawned headless and are connected with the main controlling image through their standard I/O streams, which are wrapped with BOSS so that arbitrary objects can be sent through.

There are two versions of MatriX in preview at this time. MatriX (polycephaly2.pcl) is a direct descendant of Polycephaly 1 (polycephal.pcl), however there are some key differences between the two frameworks.

Under the hood, Polycephaly 1 used BOSS to marshal objects, while MatriX uses the Xtreams ObjectMarshaler. These two marshalers may handle object edge cases differently.

Polycephaly 1 used pipes to communicate with local images, while Polycephaly 2 uses sockets. On Unix platforms, these are domain sockets, which should be secure — but on Windows they may be TCP sockets, which may NOT be secure.

For more information and examples, refer to the package comments.

WWW

AppeX Scaffolding Tool

VisualWorks 8.1 includes a new AppeX Scaffolding Tool for creating web applications from existing database tables. The tool requires loading the AppeX-Scaffolding-Tool and GlorpMapping parcels. With this framework loaded, a web application can be created using **Create Web Application** from the Launcher **Tools** menu, or programmatically using class WebAppBuilder.

To create a web application from tables, your working image must contain a subclass of ActiveRecordDescriptorSystem. This subclass can be created by the Glorp Mapping tool, as described below.

The tool creates two classes and an extension to class SiouX.Server:

- A subclass of AppeX.Application
- A subclass of AppeX.ApplicationClient
- An extension method for class SiouX.Server that defines the server name, port and connection type

Creating a web application using class WebAppBuilder

To illustrate, let's consider an example that uses database tables from AppeX-Examples-Scaffolding. The system descriptor is class DriverSchema. E.g.:

```
builder := AppeX.WebAppBuilder new.
```

Set the builder options:

```
builder
   activeRecordSystemLogin:  (Glorp.Login new
               database: Glorp.SQLite3Platform new;
               username: 'example';
               password: String new;
               connectString: 'driverTest.sqlite';
               yourself);
   initializeDescriptorSystem: AppeX.DriverSchema;
   initializeApplicationNamed: 'DriverTest';
   namespaceName: 'DriverTest';
   package: 'DriverTest';
   firstRequestPath: 'driver';
   baseUrl: '/DriverTest';
   port: 6767.
```

Create a web application:

```
builder createApplication
```

The builder creates:

- The DriverTest package
- DriverTest.DriverTestApplication is the application class. It includes two services for two tables: Driver and Vehicle.

The application services include: display a list of all table records; add a new record; update existing records; and remove records. See the implmentation of DriverTestApplication>>processDriver:

```
<GET> “Display all Driver table records”
<POST> “Update a record”
<PUT> “Add a new record”
<DELETE> “Remove a record”
<activeRecord: 'driver' class: #{AppeX.Driver}>
```

To test the application execute:

```
builder testApplication
```

This opens a default web browser and displays a list of drivers.

Creating a web application using the GUI Tools

To open the Appex Scaffolding tool, slect **Create Web Application** from the Launcher window's **Tools** menu.

The first page of the tool prompts you for login details, to connect to the database. Provide the required parameters. (For details on this interface, consult the release notes on the Glorp Mapping tool.)

On the second page, specify an application name, descriptor system class, package and name space, as follows:

Application Name

The HTML <title> of the web application. By default, this string is also used to create the application client and responder class names.

Descriptor System

A subclass of ActiveRecordDescriptorSystem. The class can be created by Glorp Mapping tool. For details on how to do this, refer to the Glorp Developer's Guide.

Package

The package in which client and responder classes are placed. If the named package does not exist in the working image, it will be created.

Namespace

The name space in which new classes are placed.

Advance to the next page, to provide settings for the Responder and Client classes, as follows:

Client Name

The name for creating the subclass of ApplicationClient.

First request path

This path with the base URL that defines the URL to open the first application page. For example/ the URL: /myBaseURL/driver opens the first page with the path corresponding a class model with the #driver path.

Responder name

The name for creating the subclass of AppeX.Application.

Base URL

The path is used as a base URL to all requests.

Server id

The name of a SiouX.Server, either pre-existing or new.

Port

The port to start the server.

Advance to the next page, to map class models to database tables, and to define the application services, as follows:

Active Record Class table

The table allows selecting Active Record classes and changing a class path. The tool will generate Application responder services for selected classes.

Click on the **Create Classes** button to create the client and responder classes, and proceed to the last page.

The last page allows us to test the application. Click **Test Application** to start a server and open a default Web Browser on a new responder.