Cincom®

Cincom Smalltalk™

VisualWorks®

Release Notes 8.0

P46-0106-22

SIMPLIFICATION THROUGH INNOVATION

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 8.0

These release notes outline the changes made in the version 8.0 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>/vw8.0`

Full commercial distribution

`<user-files>/vw8.0pul`

Personal use license (non-commercial)

`<user-files>/vw8.0ne`

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

Text2

Text2 is a new multilingual text layout and rendering framework for VisualWorks. In release 8.0, it has been premoted from “preview” status and is now part of the base image. Text2 comes with a model layer which supports new capabilities not previously supported in the product, such as:

- Adornments — strike through, jagged underlines, double underlines and more
- Actions — clicking on a piece of text could activate a hyperlink
- Annotations — pop up extra information as the mouse moves over a section of text
- Variable font sizes in the same document
- Vertical alignment within lines
- Bidirectional content with support for embedded direction marks
- Unicode font measuring and rendering
- Images in documents
- Lists both bulleted and numbered, with international numbering schemes
- Extensible design to add new features

- Sparse editing history, to support large editing and unlimited undo/redo
- Multilingual unicode word and sentence detection rules

The model comes with new widgets too:

DocumentView

View document objects rendered to the screen or a printer.

DocumentEditor

An interactive multi-line text editor widget with extensive customizability.

InputEditor

A single line input editor to compliment the document editor.

Details on how to use these new features are expanded on in class comments found on Text2.Document, Text2.Flow and UI.DocumentView.

For customers familiar with original Text framework, they know that to get only some of these features they must use the unsupported ExtraEmphasis package. Text2 offers these same kind of capabilities plus a great deal more and it's part of the product, ready to use from the UIPainter today.

In VisualWorks 8.0, the following enhancements have been made to the Text2 framework:

- Bidirectional formula made consistent with other text layout engines
- Added the ability to split internal Document nodes
- Added the ability to diff two documents and merge changes
- Added mouse cursor as a style
- Added annotations as a style
- Added “Sparklines" as document content
- Action styles are now objects subclassed off of DocumentAction rather than being a straight BlockClosure
- Added Paragraph background color
- Added paragraphSpacing and lineSpacing as styles
- Added highlight ranges for decorating a range in a document

- Background color is now settable on widgets
- Paragraph line numbers displayable in margins
- The Text2 widget now hides the mouse cursor when editing is taking place
- Pluggable Find & Replace tools for widget
- New simplified Find & Replace similar to modern editor popups

UTF-8 encoding for URI path components

(AR 70110) This release includes support for UTF-8 encoding and decoding of path components in the base image class OS.URI. The VisualWorks implementation follows the specification described in RFC 3986.

When a new URI scheme defines a component that represents textual data consisting of characters from the Universal Character Set, the data should first be encoded according to the UTF-8 character encoding; then only those octets that do not correspond to characters in the unreserved set should be percent-encoded.

When creating a URI from a string using #asURI or #fromString:, the old implementation parsed the string and encoded special characters in path components with %xx. If a string included non-ASCII chars it was necessary to first use URLEncoder to encode the string.

The new, VisualWorks 8.0 implementation parses a string, encoding non-ASCII characters using UTF-8 and special characters using %xx.

The new implementation assumes that the string provided to the #asURI and #fromString: methods is not encoded.

The revised API allows the creation or addition of already-encoded components, e.g.:

```
OS.URI fromEncodedString: 'http://localhost:9999/a%C4%B1b'
OS.URI fromString: 'http://localhost:9999/aıb'
```

For details, consult the *Internet Client Developer's Guide*.

Changes to External Library load errors and Search order for class ExternalInterface

VisualWorks 8.0 includes enhancements and changes in the error-handling mechanism for loading external libraries (AR 67852), and re-instates the ability to locate external libraries using explicit paths (AR 68928).

The first enhancement is to bring VisualWorks' error handling for external library loading into line with the information that is available from the host operating system. As it turns out, VisualWorks is unable to discern the difference between a library which is not found versus a library that fails to load for another reason, because this information is not provided by all operating systems. For this reason the class representing the error LibraryNotFoundError was replaced by the more general case LibraryNotLoadedError. At the time of the introduction of this change, the consensus was that all libraries should be found by the operating system using its internal resolution mechanism, and that custom specifications of library locations were not only unnecessary, but possibly dangerous. After this code was introduced, however, beta testers found that it removed flexibility which some customers found absolutely necessary to their applications — namely, the ability to specify explicitly where to look for an external library.

For this reason AR 68928 was introduced. Its code preserves the exception mapping and other changes introduced by 67852. It handles stripping platform designators from file names in the libraryFiles variable in class ExternalInterface when libraryDirectories is empty. It restores the ability to find dynamic libraries by explicit path and supports the alternative of letting the operating system find the library on its $PATH (or equivalent internal mechanism) when no specification is provided by the programmer.

COM

Add Warnings for Customer Classes that implement ClassInitializer

(AR 69299) In earlier versions of COM Connect, ClassInitializer methods where used to initialize classes. COM Connect 8.0 has been updated to use the more common VisualWorks initialization scheme. Classes now need to implement an initialize class method which should not be called explicitly. Please remember to remove all methods calling ClassInitializer.

While in previous versions the ClassInitializer method called the initialize method this is no longer appropriate. Instead, initialize methods usually have to call updateAndRegister. This applies to any kind of Interface-, as well as to COMStructure subclasses. In most other cases, the contents of the ClassInitializer method can be moved to the initialize method (if it does not do any initialize calls itself).

For additional details, consult the *COM Connect User's Guide*.

SendMessage (Win32UserDLL)

(AR 62871) SendMessage has been split into separate Unicode and ANSI versions: SendUnicodeMessage and SendANSIMessage. Users are strongly encouraged to update their code to use the Unicode version of the method since it supports the Unicode standard and is therefore fully compatible with modern Windows operating systems.

The ANSI version of the method is kept for backwards compatibility and corresponds to the original SendMessage method. It is nevertheless not recommended for use is as it does not support Unicode string parameters/results for standard system window messages. The Windows API will automatically interpret string parameters/results as ANSI and convert them. Passing Unicode string parameters will cause encoding exceptions or may lead to unexpected results later. Retrieving Strings containing Unicode characters will have all Unicode characters converted to question mark characters.

The following examples illustrate how ANSI SendMessage calls may be converted to use Unicode.

String parameters (in)

Input string parameters should be converted using the asUnicodeParameter method, e.g:

```
library SendUnicodeMessage: hwnd _: WM_SETTEXT _: 0 _: newTitle
asUnicodeParameter
```

String result buffers (out)

Buffer sizes

Please consult MSDN library whether the window message expects character or byte sizes. Please consider that in case of bytes, you will have to double the original buffer size to allow it to hold the same number of characters.

Buffer allocation

Depending on whether the size is specified in bytes or in the number of characters, there are two class-side utility methods in Win32ExternalInterface which allow easy allocation of buffers:

#unicodeStringBufferChars: numberOfCharacters

Allocate a Unicode string buffer which can contain the specified number of characters.

#unicodeStringBuffer: numberOfBytes

Allocate a Unicode String buffer of the given byte size.

Note that since Windows uses UTF-16 for Unicode character encoding, the buffer can only hold half as many characters (or less) as the specified byte size.

Passing buffers

Buffers need to be passed as pointers. This can be achieved using the method asPointerParameter, e.g.:

```
| numChars buffer |
numChars := 256.
buffer := Win32ExternalInterface unicodeStringBufferChars: numChars.
library SendUnicodeMessage: hwnd _: WM_GETTEXT _: numChars _:
buffer asPointerParameter
```

Database

PostgreSQL Enhancements

VisualWorks 8.0 contains two new drivers for the PostgreSQL database that use protocol 3.0. One driver uses the socket API (like the old protocol 2.0 driver). The other uses the C API, and so requires the appropriate library to be installed on the local machine.

If you have installed the optional component:

```
Contributed - PostgreSQL protocol 2.0
```

then you will also have the drivers used in VisualWorks 7.10.1 and earlier, which still work in 8.0. These drivers use protocol 2.0.

For details on how the new drivers differ from the old one, consult the PostgreSQL chapter in the *Database Application Developers Guide*.

In summary,

- Supporting protocol 3.0 future-proofs VisualWorks users of PostgreSQL

- Binding of values and reuse of statements are supported (the older driver accepted bound EXDI calls but then unbound the values back into the SQL stream; it did not bind)
- The new drivers are Cincom-licenced and supported

The new drivers are supported for use with PostgreSQL 9.* (and later) installations. They have also been tested with PostgreSQL 8.* installations and appear to work well. We do not recommend their use with PostgreSQL 7.* or earlier installations.

Concerning their functional differences, the protocol 2.0 implementation automatically fully exhausted the answer streams in its driver undreneath the EXDI, whereas the new implementation uses the EXDI directly, not as a wrapper. So, if your application uses low-level cursor operations, be aware you must exhaust the set before closing or executing another query with any session sharing the same connection. That is, EXDI sessions must send `#answer` until the `#noMoreAnswers` is finally returned. This is the usual EXDI conformant behavior, but it wasn't necessary with the protocol 2.0 implementation. It is not an issue when using the standard higher level API from Glorp, which already conforms to the EXDI API. (However be aware that if you manipulate `GlorpCursoredStreams` directly, you must exhaust them, e.g. by sending `#upToEnd` or `#release`, where a stream obtained via the older driver could survive omitting this.)

The rest of this release note explains how to configure to use a given driver. If you have not installed the optional `old-PostgreSQL` component, you will not have to reconfigure to use PostgreSQL at all. If you have installed it, you will need to reconfigure when you are ready to upgrade to the new drivers.

One exception to this: `StoreForPostgreSQL` works with any driver but prereqs the new driver parcel, not the old. In your Store repositories list, you must change `PostgreSQLEXDIConnection` to `PostgresSocketConnection` or `PostgresLibpqConnection` (details on configuring the latter are provided below) for all your PostgreSQL-using Store profiles, or else you must arrange to have the old driver parcel also load when you wish to use Store on PostgreSQL.

To connect to PostgreSQL via any of these drivers, you need two things.

First, the relevant parcel must be loaded:

- `PostgreSQL3EXDI` for either of the new protocol 3.0 drivers

- PostgreSQLEXDI for the old protocol 2.0 driver.

Second, the relevant subclass of ExternalDatabaseConnection:

- PostgresSocketConnection for a protocol 3.0 connection over sockets
- PostgresLibpqEXDI for a protocol 3.0 connection via the C API
- PostgreSQLEXDIConnection for a protocol 2.0 connection over sockets

must be referenced in the calling code, or named in the relevant drop-down list of the Store **Connect to Repository...** dialog or the Ad Hoc SQL Tool's **Connect...** dialog.

Socket

Switching from use of the old driver to the socket driver needs no more than the above. Just change PostgreSQLEXDIConnection to PostgresSocketConnection in name-list or code reference.

Libpq

To use PostgresLibpqConnection, you must have (1) a suitable libpq library installed on your local machine and (2) visible to the interface class in your image.

1. Installed on your local machine: the library will inevitably be there if you have an entire local PostgreSQL installation. This *local* Postgres installation must be a 9.* one, even though the database to which you connect may be on a remote installation that is only 8.*.

   You can obtain up-to-date PostgreSQL components from www.postgres.org.

2. Visible to your interface class: the library may be located in a directory such as:

Windows	C:\Program Files\PostgreSQL\9.3\bin
OS X	/Library/PostgreSQL/9.3/lib
Linux	/usr/local/pgsql/lib

Sometimes, these directories contain both the specific library and softlinks to more generic names for it, e.g.:

Windows	libpq.dll
OS X	libpq.dylib -> libpq.5.dylib -> libpq.5.6.dylib
Linux	libpq.so.5 -> libpq.so.5.6

The important point is to ensure that the definitions in the subclass of PostgresInterface for your platform match what your image can see, either because you have added it to your PATH (the usual approach for production use) or because you have edited the definition, e.g.:

```
PostgresNTInterface
    ...
#(#libraryDirectories #('C:\Program Files\PostgreSQL\9.3\bin'))
```

(which may be simpler for initial exercise of the new driver). Ensure visibility *before* your first libpq connection attempt from that image, to avoid any wrong-initialization effects that could persist in that image (these would require that it be closed unsaved and reopened, or discarded if saved post-failed-attempt).

Support for DB2 EXDI array binding and fetching

DB2 allows clients to specify the number of rows that will be transferred between the server and the client in one logical bind or fetch. These features are referred to as array binding and array fetching, respectively. Using array binding and array fetching can greatly improve the performance of many applications by trading buffer space for time (network traffic). Starting with VisualWorks 8.0, Cincom has begun to support array binding in the DB2 wrapper.

For details, consult the *Database Application Developers Guide*.

CTLibEXDI can return the version of the client library in use

In VisualWorks 8.0, the CTLibEXDI can return the version of the client library currently in use, e.g.:

```
conn := CTLibConnection new.
conn getClientVersion.
```

DotNET Connect

Enhanced Pointer Support

DotNET Connect now also supports pointers as method parameters and return values. Only pointers to native values are supported, no structs or value types.

64-bit Support

DotNET Connect can be used in 32- or 64-bit images. The base libraries come in both 32- and 64-bit versions.

Glorp

Glorp Enhancements

Along with bug fixes and some speed increases, the principal changes for VisualWorks 8.0 are:

1. More robust handling of complex conditions: in the past, where-clauses with multiply-nested #anySatisfy: expressions, or complex nesting of HorizontalTypeMapping to class hierarchies, etc., could loop until out-of memory.
2. Better generation from legacy: in GlorpAtlasSystemGeneration, the DescriptorSystem code generated from a legacy database now covers more cases and uses a better, terser API.
3. The schema of a Login was not by default made the schema of tables created using that Login when the tables' schema was not otherwise set. This violated the principle of least surprise and is now changed: after exhausting other means of determining the desired schema, the table creation will use that of the Login, if set.

GUI

New Source Code Editor

VisualWorks 8.0 includes an all-new source code editor that uses Text2. As a subclass of DocumentView, the source code editor now provides a semantic styling feature, rather than the older, independent content styling behavior. That is, the source code editor now auto-styles the content using a matched language (e.g.: Smalltalk) and a theme to provide syntax-based colour highlighting.

The editor is refactoring aware and has been integrated into the most important parts of the VisualWorks IDE, such as the System Browser, Debugger, Workspace, Inspector (Trippy) and Launcher windows. All of these tools now use a unified code editor that supports multiple languages, themes and plugins.

Some of the default plugins included are: Auto-Complete, Auto-Quote, Auto-Indent, Code Critic, and URL Highlighting. The framework provides a new API for developers to add their own plugins to the IDE.

Thanks to the new capabilities of Text2, annotations are used to display errors and warnings that include descriptions and actionable links. Unlike the old source code editor, errors and warnings no longer alter your source code, but appear on a separate line that disappears automatically. Errors and warnings can both independently be configured to show immediately, after a delay or when the source code is saved/evaluated (via the **Errors Level** and **Warnings Level** drop-down menus on the **Tools** > **Editor** page of the Settings Tool).

Theming has been added for the syntax-based code highlighting. Themes are designed to be language agnostic, matching a language to theme styling via a “decoration” plugin. The newly-improved incremental RBParser is used to parse Smalltalk code, then decorated with the new SyntaxHighlightDecoration. A host of different themes are included in the product and can be selected via **Tools** > **Editor** > **Theme** in the Settings Tool. New themes can be added by including a new method on the class-side of SourceCodeTheme using the <theme> pragma.

In addition to the annotation and decoration features, some plugins provide new behaviour. The Auto-Complete plugin, for instance, monitors text input to see if it has a suggestion for the developer. When it makes a suggestion, it inserts the text and selects it. Pressing the Escape key cancels its suggestion, as would backspace, delete or simply continuing to type. To accept the suggestion, press the Tab key. This also teaches the plugin your preferred suggestions. When a suggestion can expand into a multi-keyword selector, arguments are sampled from an existing implementor in the image. You can tab back and forth between suggested keyword arguments with tab and shift+tab.

Breakpoints have had a major overhaul. They are now inline content in a method, but the editor is smart enough to recognise the difference between the method code and breakpoints, as well as conditional breakpoints. Inserting a breakpoint without making any other changes to the method will automatically install the breakpoint. Breakpoints, by default, are now once-only. Clicking on the breakpoint can change it between **Disabled**, **Once** or **Infinite** modes. A shortcut key of Shift+Control/Command+B can be used to change modes as well. Breakpoints can be made conditional by placing the input cursor inside the breakpoint and typing out the conditional code. The breakpoint will expand its area as you type to include the contextual condition.

<Operate> menu actions have been re-organised into two categories: Contextual and Whole-of-Document. All whole-of-document actions can be found on the drop-down **Edit** menu, such as formatting the whole method, or performing a find. Contextual items can be found in both the **Edit** menu and also the right-click <Operate> menu in the document. Contextual menu actions are always associated with the current cursor position or selection, such as **Cut**/**Copy**/**Paste**, **Do it**/**Inspect it**/**Debug it**/**Print it**, etc.

The **Explain** feature has been retired, and in its place the editor has become more aware of what the cursor is currently interacting with (or a mouse click). Almost all language features can be navigable now from two shortcuts, Control/Command+E to see a definition, and Shift+Control/Command+E to see references. For methods, this is the equivalent of **Browse Implementors** and **Browse Senders**. For classes this is the equivalent of browsing the specified class or browsing class references. The exact menu action or shortcut key effect changes dynamically based on the context.

Additionally, a new form of keyboard navigation called *lexical navigation* has been adde to the editor. This reflects off of a valid or semi-valid parse tree to move up/down the parse tree nodes, or left/right throughout he parse tree nodes. Use Alt+Control/Command+Left/Right/Up/Down to perform these navigations. An expert user can rapidly navigate Smalltalk or other language tree nodes with these keys.

Finally, the source code editor can now support different programming languages. While almost all code in a Smalltalk image is written in Smalltalk, there are a few other languages that are included. The ParserCompiler, for instance, is a custom grammar language for constructing parsers. It includes bits of Smalltalk inside of its grammar, but it itself is not Smalltalk. Similarly, the special <C: > and <COM: > pragmas introduce C language syntax inside of Smalltalk methods. Extending this even further, the AppeX framework allows a developer to write full Javascript inside the VisualWorks System Browser. Given the differences between these languages, the standard tools cannot be assumed to work exactly the same. Thus, all the decorators, plugins, parsers, stylings, etc. are now matched against the compilerClass of the currently-active source code, allowing each language to fully customise the editor.

Refactoring Browser

The new source code editor has been integrated into the following code tools of the System (Refactoring) Browser: pundle comments; pundle properties; class comments; namespace and shared variable editing; class definition editing; method editing.

Debugger

The new source code editor has been integrated into the Debugger, and the refactoring tools are also now available. The current code to be evaluated is now drawn using a Text2 highlight, rather than the current selection.

Support for breakpoints has been overhauled and enhanced. For details, see: *New Source Code Editor*, above.

Workspace

The new source code editor has been integrated into the Workspace window. It supports proper URL highlighting, and provides a new **Style as Smalltalk** function. When this is enabled, the contents of the Workspace are treated as Smalltalk code, rather than free form text.

In addition, several issues with the pre-8.0 Workspace have been resolved. In particular, when the image is launched and all workspace windows are refreshed, the contents of a workspace are no longer discarded if an external workspace file is not found.

Inspector (Trippy)

The new source code editor has been integrated into the Trippy Inspector.

Launcher

The new source code editor has been integrated into the Launcher window transcript view.

SliderView and ProgressWidgetView removed

(AR 70078) With native skins on both Windows and OS X, we no longer offer the two-dimensional version of the SliderView or ProgressWidgetView widgets. There is no native control from which the Artist can obtain supporting graphics with which to render it. Therefore these widgets are only available on X11, or with a Default skin on Windows or OS X.

UISkinning

UISkinning is a project that improves the look and feel architecture in VisualWorks. For VisualWorks 8.0, it has been promoted from “preview” status into the base image. UISkinning is the future of look and feel support for VisualWorks and will be the focus of all of our efforts in this area. It is our intention to get as close to 100% native look and feel on Windows and Mac OS X as possible.

UISkinning provides a look and feel named Skinned that can take on a number of 'skins'. The supplied skins are:

Native

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 the Mac OS X artfile or the Windows UxTheme DLL to provide a high-fidelity native look for those platforms

Default

This is a platform-agnostic look and is available on every platform. It has a flat, geometric design.

Default Red

Identical to Default except with the addition of a strong red tint to all elements. This skin is meant for use when you are running more than one instance of VisualWorks and you need to easily distinguish the windows.

Default Green

As per Default Red, but the tint is green.

UISkinning employs a delegate/strategy architecture to provide different looks, rather than inheritance. This means that widgets are not subclassed. Support for different feels is provided purely within the widget — each widget is a 'super widget' in this respect. There are some characteristics of the look and feel that are not determined by the skin, but rather by the underlying platform. For example, font selection is not part of the look, but is provided by a new mechanism in the base image: Screen>>nativeGUIPolicy. Similarly, feel is determined primarily by the underlying platform. Thus the Default skin looks slightly different on each platform w.r.t. fonts, menu accelerators etc, and obeys platform-native feel conventions.

The ability to have e.g a Windows look and feel running on Mac OS X or X11 is not available for obvious technical reasons (the native support is not available). The focus of UISkinning is on adaptation, not simulation.

Regardless of the above caveats, UISkinning now covers most of widgets and is usable for development. The most obvious exclusion is support for table header elements being buttons, which affects the sort order controls.

Internationalization

Import of updated CLDR Locales and Removal of Legacy Locale support

“CLDR” is an acronym for the Common Locale Data Repository project of the Unicode Consortium.

VisualWorks 8.0 includes locales based upon the 2.0.0 release of the CLDR. (In upcoming releases, VisualWorks will be again updating locales to newer CLDR versions. After version 2.0.0, the numbering convention for CLDR releases changes. For details, see: www.unicode.org/cldr.)

Additionally, the previous ability to switch back to VisualWorks' original locale implementation has been removed from the image. It is still available in the **Obsolete** parcels directory of the distribution, however, and its name is **Legacy-Locales**.

When loading this support, be aware that at the conclusion of the parcel loading process, the image will be running with the legacy locale set, and the locale will be the C locale. Upon saving and restarting the image, however, the locale in use should be the one reported by the current setting of the operating system.

Once Legacy Locale support is reinstalled, it is again possible to change the locale setting back and forth between Legacy and CLDR, as described in the current version of the *Internationalization Guide*.

You should also be aware that if you subsequently unload the Legacy Locale support from your image, it will reset your working locale set to CLDR-based locales. Since it is expected that anyone needing this support would be using it for the deployment of legacy code bases of their own, it is not considered a common use case to again unload the Legacy Locale support from the image. However, if you do, please be aware that the Base VisualWorks and Internationalization components will show "dirty" in your image at that point, and you should consider reloading Base VisualWorks.

Net Clients

Updated Cookie support based on RFC6265

(AR 66218) VisualWorks 8.0 includes enhanced support for cookies that conforms to the specification in RFC 6265.

In brief, the enhancements are:

- Class Cookie2Field, CookieValue, SetCookie2Field, SetCookie2Value and UnknownCookieParameter classes have been deprecated
- Class CookieField now represents the "Cookie" header field. This class is derived from CollectionField. The method CookieField>>value returns a collection of associations, where each key represents a cookie-name and each value a cookie-value. Notice that the cookie attributes are not returned.
- Class SetCookieField represents the "Set-Cookie" header field. The class implementation has not been changed since the previous release of VisualWorks.
- Parsing the "Set-Cookie" field doesn't raise an UnknownCookieParameter error. Any invalid cookie attributes are skipped.
- Class SetCookieField supports all cookie attributes including extensions, e.g.:

```
cookie-av =expires-av / max-age-av / domain-av /
           path-av / secure-av / httponly-av /
           extension-av
```

For example, to create a Set-Cookie header field:

```
header := SetCookieField new.
value :=header addName: 'Customer' value: 'WILE_E_COYOTE'.
```

```
value
   secure: true;
   httponly: true;
   domain: 'cincom.com';
   maxAge: 0.
```

To create a Cookie header field:

```
cookie := CookieField new.
cookie addName: 'Customer' value: 'WILE_E_COYOTE'.
cookie addName: 'Part_Number' value: 'Rocket_Launcher_0001'.
```

This creates:

```
Cookie: Customer=WILE_E_COYOTE;
Part_Number=Rocket_Launcher_0001
```

To parse the following cookie:

```
header := HeaderField readFrom: 'Cookie: SID=31d4d96e407aad42;
lang=en-US' readStream.
header isKindOf: CookieField.
(val := header value first) isKindOf: Association.
val key = 'SID'.
val value = '31d4d96e407aad42'.
```

For additional details, refer to the topic on “Cookie Support” in the VisualWorks *Internet Client Developer's Guide*.

Web Application Server

SiouX: A New Web Server

VisualWorks 8.0 includes a new high-performance web server, SiouX. This new framework is meant to supplant the existing HTTP servers in VisualWorks, and finally enable the ability to run different kinds of services from the same IP address and port, as well as the ability to serve multiple ports at the same time. It also brings the capability to directly serve connections secured by SSL/TLS protocol (HTTPS). SiouX includes a configuration tool for servers in the image (SiouX-Tools).

SiouX also serves as a platform for our re-implementation of HTTP using Xtreams (SiouX-Http). However it also provides a fully supported HTTP back-end based on the proven HTTP implementation from the Net framework (SiouX-Net-Http). This implementation is largely equivalent to what used to be provided by Opentalk-HTTP, just without the complications imposed by the Opentalk architecture. Note that both back-ends can be used simultaneously in the same server. Even on the same connection requests can be handled by either back-end, the Responder selected to handle a request determines which back-end is used.

Seaside and Web Services are already re-hosted on top of SiouX-Net-Http. With Seaside it is largely transparent. In Web Services the tooling generates a lot of additional infrastructure, which would previously be based around Opentalk. With SiouX the infrastructure is a lot simpler, but also very different. It is still possible to run Web Services created with previous releases in the 8.0 release, relying on Opentalk-HTTP (now moved to **`/obsolete`**). However, it should be fairly straightforward to regenerate the service infrastructure in the new release to get a SiouX-based solution that can take advantage of the benefits (like being able to run multiple services off the same address/port). Main goal of preserving the ability for the two solutions to coexist in the same image was to allow for an easier, incremental transition.

SiouX also facilitates creation of custom data services, web APIs, etc. Just create your own subclass of the SiouX Responder that takes an incoming request and creates a response and register it with a server under a specified url prefix. The Responder superclass determines which back-end is used (Net or Xtreams) and which kind of request/response objects will be used.

For complete documentation on SiouX, refer to the *Web Server Developer's Guide*.

AppeX: A New Framework for Developing Web Applications

VisualWorks 8.0 includes AppeX, a new framework designed to simplify the development of web applications that make use of client-side Javascript. It is based on SiouX, our new general purpose web server.

AppeX is a full-featured framework providing automatic web session management, asynchronous client/server AJAX communication, server to client push notification of events using the HTML5 EventSource object, and more.

AppeX is completely library-neutral; there are no requirements to use any particular Javascript library. Rather, it enables developers to use whichever library they wish. Developers who prefer direct manipulation of the client DOM can do so right out of the box. Those who prefer to work with more friendly DOM API libraries (jQuery, Mootools, Twitter Bootstrap, etc.) can easily include them if they wish, and write the Javascript code using the library of their choice. One of our goals is to provide web developers with the ability to write web applications in the scripting language and API with which they are already familiar, while integrating them with the rich set of tools included in the VisualWorks IDE.

For server-side code, developers use the Smalltalk classes they are already familiar with. Writing web request-handling methods is as simple as tagging a method with a pragma declaring the path to the request, while the framework takes care of the rest, including session management and event push notifications. The request handlers can serve various content types, from plain text, to JSON-encoded binary data, to the content of both static and dynamically-generated files.

For client-side code, developers can write object-oriented Javascript code directly in the VisualWorks System Browser, and the AppeX framework pushes the resulting Javascript code library to the client’s web browser at runtime.

AppeX and SiouX are intended to supercede the older VisualWorks Application Server. They build on the experience and insights gained from our previous web application frameworks. We recognize the importance of our products' ability to support modern web application development, and this represents our commitment to provide developers with the tools and frameworks to remain competitive in the web application marketplace.

For complete documentation on AppeX, refer to the *Web Application Developer’s Guide*.

Web Services

Enhance WSDL wizard and X2O tool for selecting simple types

(AR 69747) To simplify the selection of unsorted simple types in the Web Services Wizard and X2O tool, an option to sort the simple types has been added. The method WebServices.XMLToolModel class>>sortDatatypeBlock: allows setting a block that can change the order of the types. By default, the simple marshalers are sorted by name.

Enhance RestrictionMarshaler to support and validate all possible simpleType constraints

(AR 69884) In VisualWorks 8.0, in addition to enumeration the Web Services framework also validates all possible XML Simple type restrictions such as: length, minLength, maxLength, pattern, whiteSpace, minInclusive, minExclusive, maxInclusive, maxExclusive, totalDigits, and fractionDigits. If a simple type description includes a pattern constraint, it is also necessary to load the Regex11 package. If the package is not loaded, a RequiresRegex exception is raised, and the pattern validation is skipped. By default, validation is enabled. To change the default value, use the method:

```
SimpleObjectMarshaler class>>validate:
```

or use **Settings > XMLObject Marshaling > Validate simple types restrictions.**

This enhancement includes a new API in class BindingBuilder that enables X2O binding modification:

forClass: *aClass* **element:** *elementName* **addEnumValues:** *collectionOfValue*

Modify the marshaler for <aClass>. The type marshaler for the specified element <elementName> shall ensure the use of enumeration values in <collectionOfValue>.

forClass: *aClass* **element:** *elementName* **addRestriction:** *restrictionType* **value:** *aValue*

Modify the marshaler for <aClass>. The type marshaler for the specified element shall guard the given restriction.

forClass: *aClass* **element:** *elementName* **refineSimpleTypeAs:** *newSimpleTypeName*

Modify the marshaler for <aClass>. The type marshaler for the specified element shall use a refined simple type named <newSimpleTypeName>, based on the current type. Answer the new refined type (a SimpleTypeMarshaler).

Examples:

```
binding := XMLObjectBinding loadFrom: x2oSchema readStream.
builder := BindingBuilder new.
builder binding: binding.
```

Create a simple type marshaler with the #kind tag:

```
builder
   forClass: Address
   element: 'kind'
   refineSimpleTypeAs: 'addressType'.
```

Add restrictions to the simple type marshaler #kind:

```
builder
   forClass: Address
   element: 'kind'
   addEnumValues: #('delivery' 'invoice' 'other').
```

Create a #streetType simple marshaler and add restrictions:

```
builder
   forClass: Address
   element: 'street'
   refineSimpleTypeAs: 'streetType'.
builder
   forClass: Address
   element: 'street'
   addRestriction: 'minLength'
   value: 1.
builder
   forClass: Address
   element: 'street'
   addRestriction: 'maxLength'
   value: 30.
builder
   forClass: Address
   element: 'street'
   addRestriction: 'whiteSpace'
   value: 'collapse'.
```

To commit the changes, use:

```
builder finish.
```

Documentation

This section provides a summary of the main documentation changes.

Basic Libraries Guide

Minor updates. Update links to support and documentation Wiki. Enhance pixel API doc.

Tool Guide

Minor updates. Update links to support and documentation Wiki. Updated discission of the Rewrite Tool. Updated screen shots.

Application Developer's Guide

Several updates. Update links to support and documentation Wiki. Add new discussion of VM debugging. This was rescued from the old Cincom Smalltalk Wiki, revised and updated by the VM team, and added to the ADG chapter on the virtual machine. Updated discussion of Memory Policy for the Runtime Packager. Fix some errors in the discussion of MS-Windows deployment. Refresh many screen shots, and make numerous minor corrections to reflect changes in the VisualWorks 8.0 tool set, e.g., the System Browser. Update discussion of editing bundle specifications to match current tools.

COM Connect Guide

No changes. Update links to support and documentation Wiki.

Database Application Developer's Guide

Several changes and additions. Update links to support and documentation Wiki. Document DB2 EXDI array binding and fetching support. Add a new chapter on the PostgreSQL EXDI. Revise the chapters "Developing a Database Application" and "Configuring Database Support".

DLL and C Connect Guide

Minor changes. Update links to support and documentation Wiki. Revise notes on compiler compatibility. Update the discussion of where 32/64-bit libraries should be located on MS-Windows platforms. Update the discussion of Error Checking in Appendix D.

DotNETConnect User's Guide

No changes.

DST Application Developer's Guide

No changes.

Glorp Developer's Guide

No changes.

GUI Developer’s Guide

No changes.

Installation Guide

Update platform requirements, as usual.

Internationalization Guide

No changes.

Internet Client Developer’s Guide

Update links to support and documentation Wiki.

Opentalk Communication Layer Developer's Guide

No changes.

Security Guide

Update links to support and documentation Wiki.

Source Code Management Guide

Update links to support and documentation Wiki.

Walk Through

No changes.

Web Application Developer’s Guide

This is a new guide to document AppeX, the replacement for the Web Toolkit (WTK). Henceforth, the WTK is considered a legacy subsystem. When optionally installed, the old documentation is available as *WebAppDevGuide-Legacy.pdf*.

Web GUI Developer’s Guide

Removed from the basic documentation set, but still available via an install option. Henceforth, the VWAS, VisualWave, and WTK are considered to be legacy subsystems. The old documentation is still available as an install option.

Web Server Developer’s Guide

This is a new guide to document SiouX, the replacement for the VisualWorks Application Server and Web Toolkit (WTK).

Web Server Configuration Guide

Removed from the basic documentation set, but still available via an install option. Henceforth, the VWAS and WTK are considered to be legacy subsystems. When optionally installed, the old documentation is available as *WebServerConfig-Legacy.pdf*.

Web Service Developer's Guide

Update links to support and documentation Wiki.

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.

Glorp

Reading DescriptorSystems from Existing Databases

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

GlorpAtlasSystemsInVW

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 ActiveRecord, 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 ActiveRecord 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 8.0, Diggy won't be updated for 8.0. 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:

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

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.