Cincom

Cincom Smalltalk™
VisualWorks®

Release Notes 7.9

P46-0106-19

Simplification Through Innovation™

Part Number: P46-0106-19

Software Release 7.9

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 7.9

These release notes outline the changes made in the version 7.9 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/

Product Patches

Fixes to known problems may become available for this release, and will be posted at this web site:

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>/vw7.9 - full commercial distribution
- <user-files>/vw7.9pul - personal use license (non-commercial)
- <user-files>/vw7.9ne - 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 VisualWorks.ini file used by VisualWorks.exe is now installed as writable: the permission does not need to be set post-installation. During installation, the paths in the .ini file are written to point at the installed directory.

Another way of launching images 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.

The files **`VisualWorks-1.15.exe`** and **`VisualWorks-1.15.ini`** are in preview. These match more of the bits in an image's signature, allowing fine-grained matching of VMs to images. The difference is not a concern for users who only want to match a given VM with all the images of a given release.

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 **`contributed/xtreams`**. Refer to its documentation at http://code.google.com/p/xtreams/ for usage instructions.

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 loaded parcel MyParcel contains 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 in the next release.

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.

2

New and Enhanced Features

This section describes the major changes in this release.

Virtual Machine

64-Bit Windows Now Supported

The 64-bit Windows VM and image are now fully supported.

Windows OE DLL Requirement

Due to a compiler upgrade, the Windows VMs require that **`msvcr100.dll`** be installed in its proper location (**`System32`** or **`SysWOW64`**).If this DLL is not already installed, the installer installs and runs a program (VisualWorks32bitDLLPrereqs, and VisualWorks64bitDLLPrereqs on 64-bit systems) that installs the DLL(s) and registers VisualWorks' interest in them. After installation, an information screen tells you this has been done. If you uninstall this program, VisualWorks is deregistered for it, and the DLL is removed if no other program has an interest.

When deploying an application with the VM, you are responsible for ensuring that this file is present on the target system.

Unix acceptQuitEvents SIGTERM handler removed

The VisualWorks 7.3.1 release notes describe procedure to register a Semaphore with the VM to be signalled when the VM receives a SIGTERM signal. With 7.9, that procedure is no longer needed.

The ObjectMemory class>>registerObject:withEngineFor: mechanism is the canonical method for adding custom handlers for signals like SIGTERM, and acceptQuitEvents no longer needs to be disabled for the method to work.

Linux/SPARC platform discontinued

Starting with this release, we have discontinued distribution of the unsupported Linux/SPARC platform engines. This decision is due to lack of interest from the Linux community on a Linux SPARC port.

Upgraded VM compilers

Starting with this release, AIX and Windows VMs are compiled with updated compilers:

- AIX is now compiled using IBM XLC for AIX version 10.1.0.11
- Windows is now compiled using Visual Studio 2010 (version 16).

Consult page 8-2 of the DLL & C Connect User's Guide for more details.

Additional small integer primitives

With this release, the virtual machine implements primitives 55 through 58 to speed up transition of small integer arithmetic to large integers. For addition, subtraction and multiplication, the new implementation is on the order of 4 times faster.

Large integer performance on big endian platforms

With this release, large integer performance has been improved on most big endian platforms. Although your mileage may vary depending on the actual calculations performed, the speedup ranges from 10% to 25% for Linux/PPC, AIX, HPUX, and Mac/PPC.

Improvements to non-interruptible GCs

This release provides a much improved non-interruptible GC. Besides a number of fixes, performance has been significantly improved. Generally, the baseline new GC implementation is a few percent faster than that in VisualWorks 7.8. Moreover, in the common case where images have perm space and the requested GC is not global, additional improvements allow the new implementation present in this release to execute up to 40% faster in reasonably configured images. Your results will vary depending on the shape of your image's object relationship graph.

64-Bit VM Limitations

The current 64-bit VMs do not support perm space objects. Moreover, 64-bit VMs will refuse to create images with perm space objects (the snapshot primitive, index 405, will fail with a bad arguments error). Attempting to load a 64-bit image with perm space objects, or to promote old space objects into perm space, is not currently supported.

Incremental GC performance improvements

With this release, the VM implementation of the incremental GC has been significantly improved with respect to performance. A summary of the changes is below.

- The marking, sweeping, nilling and unmarking phases of the IGC have been optimized. Although the behavior is essentially the same, the VM implementation was rearranged for speed. In some cases, the difference may be significant, e.g., the sweeping phase can now execute up to 3 times faster.
- The become: primitive could cause weak objects to be added multiple times to the IGC's weak list, and ephemerons to be added multiple times to the IGC's ephemeron stack. Depending on the size of the objects involved, the effect would manifest in terms of a performance impact during the IGC's nilling phase. This issue has been corrected.
- In very rare cases, the IGC's incremental marking of large objects could cause weak objects to be added multiple times to the IGC's weak list, and ephemerons to be added multiple times to the IGC's ephemeron stack. Although this effect is not possible in normal or even contrived circumstances (weak objects and ephemerons with more than 4096 strong slots would be required), the mere possibility has been eliminated with this release.
- In previous releases, the IGC gained the ability to react properly in response to become:, isWeakContainer:, isActiveEphemeron:, and class change operations. With this release, the IGC's performance has been improved significantly. Performing huge batches of the above operations will no longer cause the IGC to grow its lists, potentially causing the IGC to abort. Moreover, in some cases the IGC is now far more efficient. In particular, when weak objects are involved, mass become: operations can be

several times faster, and mass isWeakContainer: operations can be an order of magnitude faster.

In addition, the VI's memory policies now have a much improved IGC managing mechanism.

- Primitive 324 was used to invoke the IGC. A new primitive, 449, is now available which answers the number of microseconds the IGC operation took to execute. This new primitive allows the granular memory policies to stop relying on enumerationCallsPerMillisecond to determine the relevant marking, sweeping and unmarking quotas. Using enumerationCallsPerMillisecond led to mispredictions and other problems because the measurements it provided did not reflect actual, current performance. Starting with this release, enumerationCallsPerMillisecond is obsolete and is provided only so that the legacy MemoryPolicy class continues to work.
- Granularity-based memory policies now implement a self-tuning, interruption-time based mechanism to drive the IGC in non-interruptible mode. Using the new 449 primitive allows the memory policies to measure the actual, current performance of the IGC. In turn, the memory policies adjust the maximum marking, sweeping and unmarking quotas on the fly so that IGC interruptions do not exceed a preset time limit. The time limit can be adjusted by changing the value of incMaxPauseMicroseconds.
- The above changes also resolve a situation in which the legacy memory policy class could react to an IGC abort in a very large image by unmarking objects a few hundred objects at a time.
- The incremental allocation threshold in granularity-based memory policies has been tuned to provide a better response to image growth.
- The granularity-based memory policies' initialize method has been factored into smaller parts to facilitate application tuning and maintenance.

Using External Interfaces with the Windows Object Engine

The object engine was made multithreaded in 7.8 so that the engine remains responsive to timers, even when windows are being resized or moved. When an external call is made from the image using DLLCC, the object engine will be suspended until that call returns. This means that expiring timers will not be handled, and windows

messages arriving on the object engine thread will not be dispatched until that call returns. Because of this, the following two rules should be considered by the developer.

1 External calls that are expected to take a significant amount of time should be declared _threaded calls.

2 External calls that open a modal dialog must be declared _threaded.

Base Image

New weak segmented collection

With this release, the base image also provides the LargeWeakArray class. This type of segmented collection can improve garbage collection performance when replacing very large instances of WeakArray. Although specific measurements depend on the image, here are a couple data points for the sake of reference:

```
Time millisecondsToRun:
   [50 timesRepeat: [
      | w |
      w := WeakArray new: 1000000.
      1 to: w size do: [:x | w at: x put: 2 @ 3]
   ]] 4059

Time millisecondsToRun:
   [50 timesRepeat: [
      | w |
      w := LargeWeakArray new: 1000000.
      1 to: w size do: [:x | w at: x put: 2 @ 3]
   ]] 2777
```

and

```
Time millisecondsToRun:
   [
      | w |
      w := WeakArray new: 10000000.
      ObjectMemory flushNewSpace.
      5 timesRepeat:
         [
            w at: 1 put: 2 @ 3.
            ObjectMemory garbageCollect
```

```
        ]
    ] 1189

Time millisecondsToRun:
    [
        | w |
        w := LargeWeakArray new: 10000000.
        ObjectMemory flushNewSpace.
        5 timesRepeat:
            [
                w at: 1 put: 2 @ 3.
                ObjectMemory garbageCollect
            ]
    ] 146
```

Duration Class Shape Changed

The implementation of Duration has changed, so it now has two instance variables instead of one: period and scale. Otherwise, the protocol is unchanged.

This has been seen cause errors in Opentalk STST communications between images built in different versions of VisualWorks. Similarly, the change would affect communications that pass aDuration by value.

New DLLCC-Tools and BOSS-Lens Packages

DLLCC-Tools and BOSS-Lens are two new packages introduced to disentangle DLLCC and Lens dependencies.

If you use BOSS to marshal out Lens objects, you must now load BOSS-Lens as this behavior was removed from BOSS

If you use the DLLCC External Interface Finder / Builder, you must now load DLLCC-Tools

ScriptingSupport Now Supported

Scripting support (ScriptingSupport parcel) is now fully supported. Refer to the Application Developer's Guide for information.

WriteBarriers

WriteBarriers, which was in preview in previous releases, has been integrated into the VisualWorks base. The class Tracker has been renamed to ModificationTracker. Documentation has been added to the Application Developer's Guide.

Internationalization

ISO-8601 Status

ISO-8601 is an unambiguous datestamp format. It exists primarily for data exchange so that the exchanging parties have a format to use that is unaffected by regional or other locale specializations - as different cultures order months and days (among other things) differently in writing.

As ISO-8601 output option when printing timestamps with TimestampPrintPolicy will mean that ISO 8601 timestamps can become a standard for when we need a representation that is completely unambiguous and allows Timestamp objects to be accurately re-instantiated, down to the millisecond (and with full time zone awareness).

We've implemented it, but for it to work, a Timestamp has to know its offset from GMT (Universal Time). That work is delayed until 7.10. Consequently, ISO-8601 support has been been added in advance of supporting code, and will be fully functional in VW 7.10.

CLDR updated to 1.9.1

Up to Visualworks 7.8.1, CLDR 1.6.1 was used. In VW 7.9, we have updated to CLDR 1.9.1.

One significant change between these version is that in 1.6.1 the month March (März) in standaloneShortMonths is abbreviated as "Mrz". In 1.7 and newer this has been changed to "Mär". VisualWorks 7.9 cannot interpret files with the old representation any more.

There may be other changes of similar sort as well.

GUI

Skins

One of the larger changes for release 7.9 in the GUI portion of VisualWorks is the ongoing work on the Skins project. This is a project to bring the widgets found in VisualWorks into much better accordance with the native platforms they run on. By default you should notice nothing. At this time, it is far from complete. It might be

best to think of it as “in preview” but integrated into the base product. If you want to enable skinned widgets, you can execute the line of code:

```
UIBuilder.UseSkins := true
```

(This is what you might put in a startup script.)

Alternately, you can use the check box preference **Use Skins** found on the **UI Looks** page of the System Settings.

Skins currently change the rendering (and instantiation) of three basic button types: push buttons, radio buttons, and check boxes.

We are pursuing a couple of concurrent goals with the Skins work. The first, is that we are trying to reduce the LookAndFeel fan out. For every widget we currently have, we end up with at least one subclass per look and feel. This makes a number of maintenance issues difficult. It creates couplings such as the one between the Win95 and WinXP class hierarchies. If you want to derive a new kind of widget from an existing one, you can't because the existing subclass fan out gets in the way. It also means the behavior of the widget is intermingled with the drawing of it. So, if you want to make a new L&F, you have to create a whole new raft of subclasses which not only implement new displayOn: methods, but fit in with the rest of the framework.

The Skin stuff tries to separate the rendering of the widget from the model of the widget itself. It places all of the drawing code, and drawing metrics in a Skin object. On purpose, a Skin is a stateless object. It is a predefined set of functions for working with various objects. The hope is that this makes it easier to implement a new Skin.

The two skins we've done to date are an UxTheme skin, and an OS X skin. The UxTheme dll is how Microsoft does their own skinning. It is a family of functions that can draw the different standard Windows controls. So it will only work on Windows. You cannot use it on OS X to emulate the Windows look and feel. It is also does not support Microsoft’s “Classic” Look and Feel. This is a shortcoming of the UxTheme technology.

The OSXSkin will only work correctly on an OSX platform. It directly uses the art database that Apple widgets themselves draw on to render themselves.

There are other goals we're pursuing with this project. When a UIBuilder emits a PushButton, it does it in a different way than if it were to emit an ActionButtonView subclass. An ActionButtonView would be emitted inside at least two layers of wrappers. One would be a layout wrapper, the other would be a widget state wrapper. We are critical of both of these things. Where a view is at, relative to its parent, is part of its inherent makeup. It is not the decomposition of behavior that is bad, but that the relationship with which the core widget and something that is part of it are placed to one another. Even worse, is the WidgetStateWrapper which acts as a placeholder of common widget behavior, such as #isEnabled and #isVisible. When a PushButton is emitted, it will still include a LayoutWrapper around it (one problem at a time), but *not* the WidgetStateWrapper. A *PushButton* keeps track of its own "state."

If one has code that hardwires things like wrapper wrapper, or component component, this may lead to problems with a UI using Skinned widgets. Using API methods such as bottomComponent, widget, and widgetState, hide this difference.

Skinned widgets are never drawn when using the UIPainter, even if skins are turned on. The work to smooth out/fix up the Painter to support skinned widgets hasn't been able to happen yet.

The other thing we're revisiting is our assumptions about the breadth to which the MVC pattern should be used. What MVC, with its separation of model from presentation demos really well, is the classic "a slider and an input field pointed at the same value." There is no reason that support for this type of thing cannot exist, while at the same time, not requiring every UI designer to think first about what the model is for a particular widget. What for example, is the model of a button? A PluggableAdaptor around a loose boolean value model? We don't believe that is intuitive or lends itself to the "Smalltalk Enlightenment Experience." Push buttons don't have models. They get clicked. Something happens when they get clicked. To wit, PushButton doesn't require a model. But it *can* be hooked up to one. So that things like the Builder can do what they want. We put these methods in a protocol called "MVC/UIBuilder linkage", and they typically read like #linkToBooleanModel:.

Skinned widgets do not have parallel Controller classes. We use the eventReactions stuff that was introduced in the last release. The nice thing about eventReactions is that while it allows us to colocate the

interaction methods with the widget, it is actually still possible to place a custom reaction table on a view, with instance-customized behavior to it.

Making new widgets, means we can adds some enhancements easily as well. RadioButtons and CheckBoxes can do right-to-left layout in addition to the classic left-to-right. They can do top-to-bottom and bottom-to-top as well. There's a #block100 example on the class side of both which demonstrates this. CheckBoxes can be set with #beMixed to get the "partially selected" look.

One of the primary challenges with this work is how to make it work (at least mostly) in light of all the code people have written that make expectations and assumptions about the original framework. One particularly challenging area has been that of the WidgetWrapper/ SpecWrapper. Most built widgets involve an MVC triad, where the V is wrapped with some sort of BoundedWrapper which does layout for it, and then further wrapped with a WidgetWrapper of some sort which provides common stuff such as #isEnabled. When we build these skinned widgets, we omit the WidgetWrapers. But a *lot* of code has been written that expects 'aBuilder componentAt: #aName' to return a WidgetWrapper. Simply returning the BoundedWrapper that was now the top level wrapper in the namedComponents dictionary caused all kinds of problems. Since the new skinned widgets support the same APIs directly that the wrappers did (e.g. isEnabled:) a bridge object was introduced (WidgetWrapperFacade) that is not a real wrapper, but can be transiently used to provide the same sort of API. So, this code:

```
builder componentAt: #blah
```

now will return the result of sending #wrapperInterface to the object there. Which for a WidgetWrapper will remain itself, but for other visuals, will return a facade, that can present the same API without having to inject a wrapper into the view tree.

Finally, we've added another "standard section header" to the Comment Tool. We added the properties instance variable to VisualPart two releases ago. We've used it more and more for "optional instance variables" or ones that we're prototyping and not willing to commit all objects to use yet. When doing class comments for VisualParts with properties now, we're now including a **Common Properties** section similar to the **Instance Variables** section.

A presentation about the Skin project, its motivation, goals, and and status was given at STIC 2012. Video and slides for that presentation can be found here:

http://www.stic.st/conferences/stic12/stic12-abstracts/gimme-some-skin/

OS X Look and Feel enhancements

Parallel to the Skins work, some minor tweaks were done to the emulated OS X look and feel. The pin-striped backgrounds, reminiscent of OSX 10.2, are now gone. The class that managed that (MacOSXBorderedWrapper) has been removed. The menu selection color has also been updated to match the gradient found in 10.7 more closely.

Class GeometricWrapper moved

Class GeometricWrapper has been moved out of the Wrapper hierarchy. Though it continues to bear the “Wrapper” label in its name, it was never a wrapper in the same sense of the reset of the Wrapper classes, which by definition are view tree elements that have one child view, and are usually used to provide decoration and adjustment to a primary view object.

UIBuilder>>namedComponents returns a collection

Formerly, namedComponents returned the actual Dictionary that the UIBuilder uses. It now returns a copy. For most cases this is transparent. However, the following case would no longer work the same:

```
aBuilder namedComponents at: #key put: anObject
```

The correct way to programmatically add to a Builder's set of named components is to send it a componentAt:put: message.

Hover overlay for lists

(ARs 53222, 62097, 64343, 64492, 64493, 64565) Lists using SequenceView (and subclasses) will now automatically display an expanded hover overlay window showing the individual item under the mouse cursor when that item is curtailed by the width of the view's display window. For the standard development environment the most visible change is in the navigator lists of the System Browser where you will now be able to view the entire text of a curtailed item in the overlay.

SequenceView now has an additional instance variable, mouseOverIndex, that indicates which item is currently under the mouse position. This is updated in response to a new Announcement, MouseOverIndexChanged, fired when the mouse moves over a different item or when the mouse exits the view. In addition, when displaying

either a normal or selected list item, SequenceView will install a lineGrid hint into the clientData dictionary of the current GraphicsContext. This hint can then be used by the item's display Label or LabelAndIcon to correctly position itself in one of three supported vertical alignment options, #alignTop, #alignBaseline or #alignVerticalCenter.

The primary change in the HoverHelp system is the addition of two sibling classes to TooltipAssistant, SequenceHoverAssistant and RolloverAssistant, which are specialized to respond for one item in the sequence list. The common assistant behavior has been moved up to HoverHelpAssistant (the abstract superclass).

SequenceHoverAssistant provides behavior to target the tooltip to a specific element in a SequenceView list. The default is to display an overlay window for a list item whose content is otherwise clipped by the bounds of the view. SequenceView uses this kind of tooltip assistant by default, and implements two helper methods, #exposeElementIfCurtailed and #exposeElementAll, which can be used to install a SequenceHoverAssistant that will display either the full width text for a clipped list item or the text for every item in an overlay hover window.

RolloverAssistant provides behavior to rollover highlight a specific element in a SequenceView list. The default behavior also extends any item that is otherwise clipped by the view bounds.

In addition, HoverHelpWindow now expresses an interest in aboutToSnapshot. When that event occurs, it will clear any tooltip activation still in place. This keeps us from saving an image with tooltip or overlay windows about to open. We also fixed issues with HoverHelpWindow to ensure we really do recycle a single instance for reuse throughout the HoverHelp system and prevent a walkback when the window is not mapped.

To make the overlays match the text of the original item we need to obtain the text style from the list items themselves. In order to do this, changes were necessary in TreeView's private helper classes, TreeNodeWrapper and TreeNodeVisualWrapper, to defer the caching of the lastLabel visual until we actually display it, instead of every time we want to construct it (even if we are just interested in its properties). Therefore TreeNodeVisualWrapper now holds its companion TreeNodeWrapper so that TreeView can access it and cache the lastLabel there only when the element is actually displayed.

There is also a new package of examples which demonstrates configuring hover overlays and offset text for SequenceView, and its subclasses HierarchicalSequenceView and TreeView which have all been updated to support SequenceHoverAssistant. The package contains two examples of the RolloverAssistant used with a List widget to implement a rollover similar to the menu rollover effect, plus a custom assistant to display an offset text box containing various item properties. Following the examples in this package you should be able to develop custom tooltips to meet the needs of most widgets if they differ from what is provided by the various HoverHelpAssistant subclasses.

In the VisualPart hierarchy your application can use the following methods:

tooltip

Installs a single tooltip (TooltipAssistant) to be used for an entire widget.

elementTooltip

Installs an overlay (SequenceHoverAssistant) to be be used on an individual item in a list.

addTooltipAssistant

Installs your own assistant with its internal state machine, plus provides a tooltip block, to meet the specific requirements of your widget.

MouseWheelEvent improvements

(AR 61734) Class MouseWheelEvent has been updated to provide consistent scrolling across all platforms. Platform-specific methods decode the event data array, and we build a new event data array to be decoded into a MouseWheelEvent on X11 platforms where the event is emulated by Button4 (up) and Button5 (down). For all platforms we store a raw delta value from the event data array in the MouseWheelEvent as follows:

- On OS X, in units of 1.
- On MS-Windows, in units of 120.
- On X11, an emulated value in units of 1.

Icons in ComboBox lists

(AR 62838) Icons may now be used in ComboBox lists. These will be displayed in the drop-down selection list as a visual guide to selection, but will not display in the resulting selection text box. Use #iconBlock: to provide a block with which your ComboBox can display the icon for an item in its selection list, and if any item might supply a nil icon be sure to use #defaultIconExtent: to provide an appropriately-sized empty placeholder.

New ComboBox examples demonstrate read-only and editable widgets, with or without icons, including the case in which some domain objects have associated icons, and others do not. There is also an example that displays the icon associated with the list selection text box in a separate field updated by notification when the selection changes.

Consolidation/Simplification of LabelAndIcon

(ARs 62838, 63128, 64179, 64319, 64320, 64491) Class LabelAndIcon has been simplified and refactored to enable using LabelAndIcon in lists where an icon is generally expected but for which there might not be an icon for a particular item. It is no longer appropriate to mix Label with LabelAndIcon to represent list items in such cases. Icons or PixelSpace placeholders should occupy their appropriate space and offset is reserved for indentation of any label text, horizontal and/or vertical, independent of the icon space. A Label or LabelAndIcon can now be verticaly aligned using the helper methods, #alignTop, #alignBaseline or #alignVerticalCenter, within the lineGrid as designated by the view.

For consistency in its role representing a 'static graphic', class PixelSpace is now a subclass of VisualComponent rather than a VisualPart or 'widget'.

“Sensor” instance variable has been retired

The Controller hierarchy no longer retains an instance variable for a TranslatingSensor. We are trying to remove away from the Sensor objects at the view/widget level. They were an artifact from many years ago when the VisualWorks event flow was transitioned from a polling architecture to an event-driven one.

The message #sensor is still understood and will retain an appropriately configured TranslatingSensor when sent.

One fallout of this, is that the #flushCoordinateCaches method is no longer needed. It has been removed. That information is now dynamic and performed at run time. This solved a number of caching bugs that cropped up in edge scenarios and simplifies code.

One side effect of the #flushCoordinateCaches method was that it was the single message a View object would receive whenever its parent container, or any of the parent containers above it in the view tree changed. To facilitate View objects that still need to know this information, the newer and more general API methods #changeParent and #changedParentage have been added. #changedParent is a replacement for the traditional #changeContainer method (which will still be called by default from #changedParent) and #changedParentage will be sent to all children a visual object when its parent changes.

KeyboardProcessor widget registration is now dynamic

Historically, KeyboardProcessor held a list of widgets (in the keyboardConsumers instance variable) that were to be enumerated in the tab chain as the keyboard focus shifted from widget to widget.

ToolSafe recursion

The #toolSafeIn:else: method in class BlockClosure gives the tools a mechanism to execute blocks of code with *some* degree of robust protection against failure. A portion of that mechanism is to protect against code that appeared to be an infinite recursion. This public protocol (Process>>isRecursive) could become quite expensive as the stack depth grew to non-trivial values. The result was that the test for recursiveness could take a very long time, thus countering the very robustness that the #toolSafeIn:else: API was attempting to provide. The API method #isRecursive has been removed.

Truly determining recursion is a contextual sort of heuristic guessing, and an API that purports to determine it in all cases, was misleading for some application developers who ended up misusing it with unfulfillable expectations. A new method, #mayBeRecursive, is now provided in its place. It is not as thorough as the original, but it also is guaranteed to only consume a fixed amount of time in its computation.

Tools

Published Items Tool now shows Loaded Items

The Published Items Tool now indicates a package or bundle that has any version loaded, as bold in the list. This gives quick feedback for a bundle or package that is loaded in some form or another in the image already.

Create 32-bit or 64-bit project images

(AR 63113) The LaunchPad has been enhanced to enable creating either a 32-bit or 64-bit image where appropriate. If you install both the 32-bit and 64-bit versions of the VM for your platform, the LaunchPad will display **Add** buttons for both 32- and 64-bit projects. If only one version is installed, either 32-bit or 64-bit, a single unbadged button is displayed. Note that if you have installed VisualWorks 64-bit support on a platform for which we provide 64-bit support, but your current OS is not 64-bit, trying to create and start a 64-bit image will result in a VM failure. The VisualWorks VM will only run in its corresponding version of the platform operating system.

Manage multiple VisualWorks installations or projects directories

(AR 63114) The LaunchPad application has been enhanced to provide support for experienced developers who wish to manage multiple concurrent VisualWorks installations and/or multiple VisualWorks Projects directories. The new LaunchPad application can be configured to create new projects with any currently installed version of VisualWorks on your machine using these facilities.

The root VisualWorks Projects directory shared variable, which is normally read from its persisted location in the VWPROJECTS environment variable, can now also be temporarily set in the LaunchPad image via a command line doIt or fileIn. To set the current image's projects root directory use:

```
LaunchPad.Project useProjectRoot: myRootDirectoryFilename
```

To accommodate creating projects from any installed VisualWorks release, a new shared variable can also be set from a doIt or fileIn. You can force new projects to be created from the desired installation directory by providing that directory's Filename with:

```
LaunchPad.Project useInstalledLocation: myVWInstalledLocation
```

To use multiple LaunchPad configurations, start by creating a directory that will hold your various configuration files. On Windows this might be:

```
C:\Smalltalk\Settings
```

On OS X or Linux/UNIX you could use:

```
$HOME/Smalltalk/Settings
```

To configure the LaunchPad to use a specific VisualWorks Projects directory and create new projects using a specific installation of VisualWorks, create a file in your new Settings directory containing something like the following example configurations.

On Windows:

```
"Set a specific project root directory"
LaunchPad.Project useProjectRoot: (Filename fromComponents:
    #('$(USERPROFILE)' 'Documents' 'VisualWorks Projects' 'vw790' )) !"
... and the corresponding VW7.9 install location"LaunchPad.Project
useInstallLocation: (Filename fromComponents:#('C:\' 'Smalltalk'
'VW_Releases' 'vw7.9' )) !
```

On OS X or Linux/UNIX:

```
"Set a specific project root directory"
LaunchPad.Project useProjectRoot: (Filename fromComponents:
    #('$(HOME)' 'VisualWorks Projects' 'vw790' )) !
" ... and the corresponding VW7.9 install location"
LaunchPad.Project useInstallLocation: (Filename fromComponents:
    #('$(HOME)' 'Smalltalk' 'VW_Releases' 'vw7.9' )) !
```

Now update your VisualWorks desktop shortcut, applet or startup script to reference your new settings file with a fileIn command line parameter. Note that you should always be running the most recent LaunchPad image, from the current VisualWorks release, with the current release's VM. This will take advantage of any new features of the LaunchPad. Setting the install location only affects the creation of new projects, not the execution of the LaunchPad application itself.

On Windows, or Linux/UNIX simply copy the VisualWorks desktop shortcut or startup script, and update the command line to reference the individual fileIn with something like:

```
"C:\Smalltalk\VW_Releases\vw7.9\bin\win\vwnt.exe"
"C:\Smalltalk\VW_Releases\vw7.9\launchpad\launchpad.im" -filein
"C:\Smalltalk\Settings\VW790_config.st"
```

On OS X, updating the desktop applet is slightly more complicated since you will need to recompile the AppleScript that starts the LaunchPad image. Copy the entire VisualWorks Projects.app to your desktop. Open it in the Finder and use **Show Package Contents** to navigate to:

VisualWorks Projects.app/Contents/Resources/Scripts/main.scpt

Set write permissions by unlocking the properties of the file on its **Info** page. Now you can open the main.scpt file in the Script Editor and replace the contents with something like the following. For each tailored applet you will only need to reference the individual fileIn. The rest of the script (below) will remain the same for all. Be sure to compile and save the script once you have finished.

```
tell application "System Events"
    set vwHome to (value of property list item "VISUALWORKS" of
contents of property list file "~/Library/Preferences/
com.cincom.vw7.9.plist")

    set myHome to do shell script ("echo $HOME")
    set oePath to vwHome & "/bin/macx/visual.app"
    set imPath to vwHome & "/launchpad/launchpad.im"

    --  Reference the specific configuration file here
    set cfgPath to myHome & "/Smalltalk/Settings/VW790_config.st"

    set imParam to "  -filein " & cfgPath
    set myScript to (oePath) & "/Contents/MacOS/visual " & (imPath) &
(imParam) & " > /dev/null 2> /dev/null & "
    do shell script myScript
end tell
```

Compare Tool enhancements

When the new Compare Tool is working with package properties, it naively uses the #printString of those objects to compare them. Package properties can be any arbitrary object type though, and in some cases, using the printString result as a string representation for comparison can be less than helpful.

This is has been made extensible now to allow given properties to be represented differently. When printing a property, the tool will look amongst the methods found in AbstractPropertyComparisonRollupView for those that have a method tag of the form:

<printProperty: #aPropertyKeyName>

If such a single argument method is found, this method will be invoked to fetch the appropriate string representation for the property for comparison/presentation purposes, rather than #printString. For examples, see the “printing properties” method category in class AbstractPropertyComparisonRollupView.

Customizing the Code Critic

A new topic has been added to the tools documentation that discusses how to add or modify the categories in this tool, and how to write new rules for the Critic.

Database

Added Supported Database Platforms

This release adds full support for the following platforms:

- 64-bit Oracle on all platforms
- 64-bit ODBC on MS-Windows
- SQLite 3

In addition to MS-Windows, 64-bit ODBC support should work on other platforms as well. However, at the time of the VisualWorks 7.9 release, Cincom has not yet had the chance to test this using a variety of different vendor libraries.

If you wish to try 64-bit ODBC support on other platforms, you can use class ODBCCLINT64Interface as an example to add the interface class for the platform, and change the method ODBCConnection class>>initializeInterfaceMapping to add the map between the platform symbol and the interface class. Once that has been done, the ODBC EXDI should work with 64-bit on the desired platform.

Working with SQLite

Note that a SQLite prepared session should not be reused for different SQL. With most VisualWorks EXDI clients, you can reuse the same session object for different prepared statements. SQLite differs; it dedicates each session to a single prepared statement. That statement can be run repeatedly, and the bound values changed, but the session should be sent prepare: only once. Doing otherwise can appear to work, but tends to leak memory. Accordingly, send a prepare: statement to a SQLite session object only once.

Also, note that a very large SQLite query string may not fit in FixedSpace, which can generate an exception. This appears to be an extremely rare issue. If a long query string is too big for FixedSpace, it yields a “bad argument” exception. One suggested workaround for this is to remove the changeClassToThatOf: message send from SQLiteInterface>>prepare:connectionHandle:statementHandle:. But typically, large objects can be bound, rather than embedded in the SQL, which avoids the problem.

Array binding support for ODBC

(AR 56690) The ODBC EXDI now supports Array binding. When adding large numbers of objects using INSERT, a substantial performance increase can be achieved by inserting an arry of objects in a single operation.

When binding arrays of values, the size of the array must match the size specified by the INSERT statement. Arrays may be bound either by position or by name.

For details, see: "Array Binding" in the chapter "Using the ODBC Connect" of the Database Application Developer’s Guide.

Client password encryption supported in CTLib

(AR 61741) Class CTLibConnection supports Sybase's default password encryption. By default, it is disabled. To use this feature, you must set the password encryption attribute *before* establishing a connection to the database server.

The following code examples illustrate how to use password encryption:

```
"Create a new connection."
conn := CTLibConnection new.
"Before enabling encryption, status is 0. By default, it is disabled."
conn getPasswordEncryptionStatus.
"After enabling, you should get 1 back."
conn enablePasswordEncryption.
conn getPasswordEncryptionStatus.
conn username: 'username'.
conn password: 'pwd'.
conn environment: 'sybaseDB'.
conn connect.
```

Enhanced Unicode support for ODBC

(AR 60464) In the 7.9 release, the ODBC Connect (ODBCEXDI and ODBCThapiEXDI) includes support (1) for full-Unicode applications, (2) for "mixed" applications that combine Unicode and non-Unicode columns, (3) and legacy non-Unicode applications. To use Unicode, you can either (a) set the default encoding of the database connection to #UTF16 or #'UCS-2', or (b) use binding templates to specify particular columns to be encoded as Unicode.

In release 7.9, the recommended API for setting encoding is ODBCConnection>>encoding:. However, for backward compatibility, you can also specify Unicode encoding using a session object (i.e., ODBCSession>>unicodeEncoding: and #unicode:).

For details, see the discussion of "Storing and Retrieving Unicode" in the chapter "Using the ODBC Connect" of the Database Application Developer's Guide.

Oracle Unicode Support

(ARs 64507, 64185) In VisualWorks 7.9, we have added full Unicode Support to Oracle connects. Users are now able to use Unicode table and column names, include Unicode strings in their SQL statements, bind Unicode strings to host variables, and even input and output Unicode strings to and from their Oracle functions and procedures.

In the new implementation, we start to use Oracle function OCIEnvNlsCreate in which users can set your initial client-side character set and national character set for their current environment handle, but to use which function will be automatically decided based upon what version of Oracle client is loaded, if the loaded Oracle library supports OCIEnvNlsCreate, it will be used, otherwise, OCIEnvInit will be used.

Here's how the new implementation works for you:

1 If you don't set the main encoding to Unicode, it will work as before, all existing applications will continue to work.

2 If you don't set the main encoding to Unicode but want to use Unicode columns, you can do so as well, but when binding String values, you have to set binding templates for the binding values to indicate which Strings should be encoded as Unicode. This will provide the flexibility for users to develop "mixed" applications.

3 If you set the main encoding to Unicode, no binding template is necessary, all connect strings, SQL statements, error messages, insert and retrieved Strings will be treated as Unicode.

To illustrate, here is code for selecting various encodings:

```
"Connect to the Oracle database."
conn := OracleConnection new.

"Set encoding to WE8MSWIN1252"
conn oracleEncodingId: 178.

"Set UTF-8 as unicode encoding."
conn oracleUnicodeEncodingId: 871.
conn
    username: 'username';
    password: 'password';
    environment: 'oracleDB'.
conn connect.
```

From the above example, we can see that oracleEncodingId and oracleUnicodeEncodingId are used to indicate what encodings you want for encoding and unicode encoding. In this particular example, we want the “mixed” usage, and so use WE8MSWIN1252 as the main encoding for metadata and normal string columns, and UTF8 as encoding for Unicode string columns.

Also, users can set both encoding and unicode encoding to Unicode, as in this example:

```
"Connect to the Oracle database."
conn := OracleConnection new.
"Set UTF16 as encoding."
conn oracleEncodingId: 1000.
conn encoding: #utf_16.

"Set UTF16 as unicodeEncoding."
conn oracleUnicodeEncodingId: 1000.
conn
    username: 'username';
    password: 'password';
    environment: 'oracleDB'.
conn connect.
```

While setting the main encoding ID, users have to set the value of encoding (#utf_16 in the example above) initially as well. The reason is that after the environment handle is created using the Unicode encoding ID, everything (e.g., table and column names, error messages, and so on) exchanged between the client and server will

be in Unicode. We have to use that initial encoding to get the right encoding and Unicode encoding names. For the single byte character set IDs like 178, users don't have to do it.

Note: do not set encoding to AL16UTF16 (2000), because AL16UTF16 is the national character set for the server, use OCI_UTF16ID (1000) instead. Also, when sending data to Oracle server, you have to take server-side character set into consideration, otherwise, you may experience data loss. For example, if you send two-byte characters to a server whose database character set only allows single byte character, the data will be lost.

Some users may have difficulties finding the correct character set IDs or names. Here are some code samples that may be useful for finding character set names from IDs, and vice versa:

```
"Find character set name from an ID"
......
sess := conn getSession.
sess prepare: 'SELECT NLS_CHARSET_NAME(178) FROM dual' ;
   execute.
ans := sess answer.
ans upToEnd.
......

"Find character set ID from a name"
......
sess := conn getSession.
sess prepare: 'select NLS_CHARSET_ID(''UTF8'') from dual' ;
   execute.
ans := sess answer.
ans upToEnd.
......
```

In the following Workspace script, we demonstrate different ways of using Unicode strings and bind templates (please note, in the code examples, we use some Chinese characters):

```
"Connect to the Oracle database and set initial encoding and unicode
encoding IDs."
conn := OracleConnection new.
"Set UTF16 as main encoding."
conn oracleEncodingId: 1000.
conn encoding: #utf_16.

"Set UTF16 as unicode encoding."
```

```
conn oracleUnicodeEncodingId: 1000.
conn
      username: 'username';
      password: 'password';
      environment: 'oracleDB'.
conn connect.

"Drop the test table if existed."
sess := conn getSession.
sess prepare: 'drop table test_unicode';
   execute;
   answer.

"Create a test table."
sess prepare:  'create table test_unicode (cid number, cc char(100), cuc
nchar(100), cname varchar2(100), cname1 nvarchar2(100), cl clob, ncl
nclob)';
   execute;
    answer.

"Insert test data."
sess := conn getSession.
sess prepare:   'insert into test_unicode values(10, ''?'', ''??'', ''?1'',
''????'', ''?2'', ''????'')';
   execute;
    answer.

"Insert test data without using template, all strings will be encoded using
the main encoding."
sess := conn getSession.
sess prepare:   'insert into test_unicode values(?, ?, ?, ?, ?, ?, ?)';
bindInput: #(1 'a' '?' 'b' '?123456' 'cc' '?1234567');
   execute;
    answer.

"Insert test data using template to indicate which strings are unicode
strings. For this example, it is not necessary because the main encoding
is UTF16."
sess := conn getSession.
sess prepare:   'insert into test_unicode values(?, ?, ?, ?, ?, ?, ?)';
bindInput: #(1  '?' '??' '?1' '????' '?2' '????') template: #(nil nil
#UnicodeString  nil  #UnicodeString  nil  #UnicodeString);
   execute;
    answer.

"Retrieve the test data."
sess := conn getSession.
```

```
sess prepare: 'select * from test_unicode' ;
   execute.
ans := sess answer.
ans upToEnd.

"Test LOB retrieval and updates."
conn begin.
"Retrieve the test data."
sess := conn getSession.
sess answerLobAsProxy.
sess prepare: 'select * from test_unicode where cid = 10 for update' ;
   execute.
ans := sess answer.
res := ans upToEnd.
nclob := (res at: 1) at: 7.
"See the original value."
nclob readAll.
"Update the LOB value."
nclob writeFrom: 1 with: ('?1234567' asByteArrayEncoding: conn
unicodeEncoding).
conn commit.


"Verify the change."
sess := conn getSession.
sess prepare: 'select * from test_unicode' ;
   execute.
ans := sess answer.
ans upToEnd.

"The following examples demonstrate the usage of Unicode strings in
procedures and functions."

"Connect to the Oracle database without setting encodings."
conn := OracleConnection new.
conn
      username: 'username';
      password: 'password';
      environment: 'oracleDB'.
conn connect.

"Drop the test table if existed."
sess := conn getSession.
sess prepare: 'drop table test_unicode';
   execute;
   answer.
```

```
"Create a test table."
sess prepare:  'create table test_unicode (cid number, cc char(100), cuc
nchar(100), cname varchar2(100), cname1 nvarchar2(100), cl clob, ncl
nclob)';
    execute;
     answer.

"Create a test procedure for inserting a record."
sess := conn getSession.
sess prepare:   'create or replace procedure test_insert_unicode (
        cid in integer,
        cc char,
        cuc nchar,
        cname varchar2,
        cname1 nvarchar2,
        cl clob,
        ncl nclob
    ) is
    begin
        insert into test_unicode values (cid, cc, cuc, cname, cname1, cl,
ncl);
    end;
';
    execute;
     answer.

"Calling the procedure to insert test data."
sess := conn getSession.
sess preparePLSQL: '
    BEGIN
        test_insert_unicode(:cid, :cc, :cuc, :cname, :cname1, :cl, :ncl);
    END;
'.

"We have to define a binding template since we want the strings to be
encoded differently."
template := Dictionary new.
template at: #cid put: nil;
        at: #cc put: nil;
        at: #cuc put: #UnicodeString;
        at: #cname put: nil;
        at: #cname1 put: #UnicodeString;
        at: #cl put: #nil;
        at: #ncl put: #UnicodeString.

sess bindTemplate: template.
sess bindVariable: #cid value: 1.
```

```
sess bindVariable: #cc value: 'a'.
sess bindVariable: #cuc value: '??'.
sess bindVariable: #cname value: 'ab'.
sess bindVariable: #cname1 value: '????'.
sess bindVariable: #cl value: 'abc'.
sess bindVariable: #ncl value: '????'.
sess execute.
answer := sess answer.
answer := sess answer.

"Retrieve the test data."
sess := conn getSession.
sess prepare: 'select * from test_unicode' ;
   execute.
ans := sess answer.
ans upToEnd.

"Create a test procedure to retrieve part of the data."
sess := conn getSession.
sess prepare:   'create or replace procedure test_select_unicode (
       vCid in integer,
       vCname out varchar2,
       vCname1 out nvarchar2
   ) is
   begin
      select cname, cname1 into vCname, vCname1 from test_unicode
where cid=vCid;
   end;
';
   execute;
    answer.

"Calling the retrieval procedure."
sess := conn getSession.
sess preparePLSQL: '
   BEGIN
      test_select_unicode(:cid, :cname, :cname1);
   END;
'.

"We have to define a binding template since we want the strings to be
encoded differently."
template := Dictionary new.
template at: #cid put: nil;
       at: #cname put: nil;
       at: #cname1 put: #UnicodeString.
sess bindTemplate: template.
```

```
sess bindVariable: #cid value: 1.
sess bindVariable: #cname value: '0000000000000000'.
sess bindVariable: #cname1 value: '0000000000000000'.
sess execute.
answer := sess answer.
answer := sess answer.

"Verify the retrieved strings."
normalString := sess bindVariable: #cname.
unicodeString := sess bindVariable: #cname1.

"Function example."

"Create a function to test."
sess := conn getSession.
sess prepare:  'create or replace function testFunction (inID number)
return nvarchar2
is
begin
   declare
   ret_name nvarchar2(30);
   begin
      select cname1 into ret_name from test_unicode where cid=inID;
      return ret_name;
   end;
end;
';
   execute;
    answer.

"Calling the function."
sess := conn getSession.
sess preparePLSQL: '
   BEGIN
      :res := testFunction(:cid);
   END;
'.

"We have to define a binding template since we want the return value to
be encoded correctly."
template := Dictionary new.
template at: #cid put: nil;
       at: #res put: #UnicodeString.
sess bindTemplate: template.
sess bindVariable: #cid value: 1.
sess bindVariable: #res value: '0000000000000000'.
sess execute.
```

```
answer := sess answer.
answer := sess answer.

"Verify the returned string."
returnedString := sess bindVariable: #res.
```

ODBC Array Binding and Fetching

(ARs 63833, 56690, and 62883) Some databases and their ODBC drivers allow clients to control the number of rows that will be physically transferred between the server and the client in one logical bind or fetch. These features are called array binding and array fetching. Using array binding and array fetching can greatly improve the performance of many applications by trading buffer space for time (network traffic).

Since not all of the ODBC drivers from the database vendors are implemented in the same way, when using array binding and array fetching you may see different performance gains against different databases.

While doing the development, we ve found that there are bugs in numerous ODBC drivers. SQL Server's early ODBC drivers have problems in processing LOBs that are more than 400KB in array binding. Only the SQL Native Client coming from SQL Server 2008 but not the normal ODBC driver works OK with LOBs that are bigger than 400K. DB2's early ODBC drivers also have problems when binding arrays of LOBs at execution time, but the ODBC driver coming with DB2 9.7 works fine. Sybase's ODBC driver doesn't work with binding array LOBs, but it works when binding non-array LOBs. Therefore, if you plan to use these features, we recommend that you always get the latest ODBC drivers from the database vendors.

There are behavioral differences among the ODBC drivers too, especially when binding arrays of Large Objects (LOBs) at execution time. For example, if binding multiple arrays of LOBs, SQL Server insists that you send the data row by row. Also, if you bind two arrays of LOBs, first is a CLOB array and second is a BLOB array. SQL Server wants you to send the first row, a CLOB and a BLOB, and then the second row, and so on. However, Oracle wants you to send data column by column. So, in the previous example, to feed the LOB data to Oracle, you have to send all of the CLOB data first, and then the BLOB data. DB2 seems to work in the same way as SQL Server. There are also differences when dealing with NULL data. SQL Server and DB2 get the information at binding time, but Oracle wants you send the NULL data at execution time.

In general, ODBC has some limitations on fetching arrays of LOB data.

LOB data is large, so we normally use SQLGetData to get it instead of binding buffers. However, the function SQLGetData we use has some special requirements. First, it can only be used after all of the bound columns are done. Secondly, it cannot be called if the row set size is greater than 1. This means that if the select list includes LOB columns, we have to always set the blockFactor to 1, and the LOB data columns have to appear after the normal data columns in the select list.

Based on what we have learned from the array binding feature in our OracleEXDI, we've included several features to improve the reliability, flexibility and performance of our ODBC connects.

In the following examples, we demonstrate the usage of array binding and array fetching in VisualWorks and ObjectStudio.

The following are the VisualWorks code examples for using array binding and array fetching in ODBCEXDI (the server is SQL Server 2008):

```
"Tests against SQL Server 2008."
 "Array Binding example."
conn := ODBCConnection new.
conn username: 'username';
password: 'password';
environment: 'SQLServerNativeClientDSN'.
conn connect.

"Drop the test table if existed."
sess := conn getSession.
sess prepare:  'drop table test_array_binding'.
sess execute.
sess answer.
 "Create a test table with multiple columns of different data types."
sess := conn getSession.
sess prepare:  'create table test_array_binding (cid int, cf float, cbigInt
bigint, cvc varchar(100), cc char(100), cbit bit, cm money, cdatetime
datetime, cbin binary(100), cvb varbinary(100), cb varbinary(max), ct
varchar(max))'.
sess execute.
sess answer.
 "Create lists of binding values.  Please note, they don t always have the
same size."
listInt := Array with: 1 with: 2 with: 3.
listFloat := Array with: 1.2 with: 2.34 with: 3.456.
```

```
listBigInt := Array with: -9223372036854775808 with:
9223372036854775807 with: 1234567890123456789.
listVarchar := Array with: 'test1234567' with: 'test12345678'.
listChar := Array with: 'test1234567' with: 'test12345678'.
listBit := Array with: true with: false.
listMoney := Array with: 2.56 with: 3.42.
ts := Timestamp now.
listDateTime := Array with: ts with: ts.
listBibary := Array with: (ByteArray new: 99 withAll: 1) with: (ByteArray
new: 95 withAll: 0).
listVarBibary := Array with: (ByteArray new: 99 withAll: 1) with:
(ByteArray new: 95 withAll: 0).
RS := ByteArray new: 1444096 withAll: 1.
RS1 := ByteArray new: 444096 withAll: 0.
listBlob := Array with: RS with:nil with: RS1.
TX := String new: 444096 withAll: $a.
TX1 := String new: 1444096 withAll: $b.
listClob := Array with: TX with: TX1 with: nil.

"Add the lists of binding values to a single binding list."
bindList := Array new: 12.
bindList at: 1 put: listInt.
bindList at: 2 put: listFloat.
bindList at: 3 put: listBigInt.
bindList at: 4 put: listVarchar.
bindList at: 5 put: listChar.
bindList at: 6 put: listBit.
bindList at: 7 put: listMoney.
bindList at: 8 put: listDateTime.
bindList at: 9 put: listBibary.
bindList at: 10 put: listVarBibary.
bindList at: 11 put: listBlob.
bindList at: 12 put: listClob.

"Insert the binding list into the SQL Server database."
sess := conn getSession.
sess prepare:  'INSERT INTO test_array_binding VALUES (?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?)'.
sess bindInput: bindList.
sess execute.
sess answer.
 "Array Fetching examples."

"Example1: Array fetching works since the select list does not contain
LOB columns."
sess := conn getSession.
"The default blockFactor is 1."
```

```
sess blockFactor: 3.
sess prepare: 'select cid, cf, cbigInt, cvc varchar, cc char, cbit, cm,
cdatetime, cbin, cvb from test_array_binding'.
sess execute.
ansStrm := sess answer upToEnd.

"Example 2: blockFactor will be set back to 1 since the select list contains
LOB columns."
sess := conn getSession.
sess blockFactor: 3.
sess prepare: 'select * from test_array_binding'.
sess execute.
ansStrm := sess answer upToEnd.
```

For details, see the discussion of "Array Binding and Fetching" in the chapter "Using the ODBC Connect" of the Database Application Developer's Guide.

Security

Security Framework enhancements

Security libraries underwent significant overhaul, especially at the lower levels of the cryptographic primitives. The main goal was to open up and allow using external cryptographic toolkits with the higher-level security frameworks. There is a number of reasons why that may be desirable that are relevant in various application contexts:

- Industrial or legislative requirements preventing the use of the toolkits provided out of the box
- Corporate standards requiring use of specific approved cryptographic toolkits
- A need to employ toolkits providing specific capabilities, e.g. hardware tokens or smart-cards
- A need to use toolkits providing better performance on given deployment platform

This was achieved by adopting the approach provided by the cryptographic streams from the Xtreams library, specifically its Xtreams-Crypto package. This package provides cryptographic capabilities via cryptographic stream layers allowing to combine cryptographic operations with other data processing functions in an

efficient and scalable manner, enabled by the stream processing paradigm. Refer to the package comment of Xtreams-Crypto for more details and examples of how to use these transforms.

Xtreams-Crypto comes with a pluggable architecture that accommodates multiple implementations of cryptographic primitives. Out of the box it relies on libraries that are normally provided with the underlying operating system. On MS-Windows it uses the BCrypt library (available on Vista and later) and on other platforms it looks for libcrypto from the OpenSSL suite which is usually bundled with the operating system.

The pre-existing native Smalltalk implementations of cryptographic algorithms are somewhat de-emphasised by this move, but efforts are under way to allow using those as an alternative implementation back-end as well. Currently, the public-key algorithms are already integrated in the form of the Security-Xtreams package. When this package is loaded, the public-key API from Xtreams-Crypto will automatically fall back onto the Smalltalk implementation under following circumstances:

1 On MS Windows it supercedes the BCrypt implementation of DH and DSA algorithms, as those have significant usability restrictions in BCrypt.

2 On non-MS Windows platforms it supercedes the libcrypto implementations if the library cannot be loaded or its version is less than 1.0.0 (the public-key algorithms are not available via the required EVP layer in older versions of the library).

The default implementation selection can be further tuned on a per-algorithm basis with additional extension methods annotated with the #priority: pragma (see the existing uses). Other cryptographic toolkits can be integrated the same way the default ones are. There is an abstract superclass for each family of algorithms (Hash, Cipher, PrivateKey, PublicKey, SecureRandom) which defines the required API and implementation specific subclasses that can be instantiated directly or selected by the implementation selection mechanism controlled by the #priority: pragma (for details, see the class-side 'implementation selection' protocol of these classes).

PKCS8 support

Since there aren't generic, algorithm-specific key classes anymore, private keys are now marshaled to/from PKCS8-specific key classes (PKCS8RSAPrivateKey, PKCS8DSAPrivateKey). The PKCS8 classes are

just data-holders that cannot be used for cryptographic operations directly. Use the message #getKey to obtain a real key from those. Use the message #asPKCS8Key to convert a real private key to a PKCS8 form for export.

Note that the PKCS8 key form also contains values that should be kept secret, therefore PKCS8 keys should also be explicitly released when no longer needed.

X50 support

Since there aren't generic, algorithm specific key classes anymore, all the API dealing with the public key returns and expects the X509 specific key classes (X509RSAPublicKey, X509DSAPublicKey). The X509 classes are just data-holders that cannot be used for cryptographic operations directly. Use the message #getKey to obtain a real key from those. Use the message #asX509Key to convert a real public key to its corresponding X509 form.

There are no algorithm instances anymore either, so the certificate signing API (#signUsing:) expects a functional private key (not the X509 equivalent). Consequently there is also an extended version #signUsing:hash:padding: that allows specifying the hash algorithm and padding (needed for RSA signatures).

Implementation Note: The cryptographic operations are controlled by class AlgorithmIdentifier. A full list of supported algorithms is managed in the variable AlgorithmIdentifier.AlgorithmMap. Another place where specific algorithms have to be reflected are signature/bytes conversions which are handled by #getSignatureBytesFor: and #signatureFromBytes:for:.

SSL/TLS implementation

The 7.9 release includes a new implementation of the SSL/TLS protocol adding support for TLS 1.0, 1.1 and 1.2 (as well as the previously supported SSL 3.0). Consequently it also adds cipher suites with more modern algorithms e.g. AES or the SHA-2 family of hash functions (e.g. SHA256). The new implementation continues the trend of opening up to alternative or external supporting infrastructure for certificate and session management. The new TLSCertificateStore class completely encapsulates handling of certificates, the rest of the TLS stack treats certificates as opaque entities. Similarly TLSSessionCache completely encapsulates creation, lookup and

access of TLS sessions enabling implementation of alternative strategies or interfacing with external (e.g. distributed) TLS session caching solutions.

The new TLS implementation leverages scalable composition of streams from the Xtreams suite. Consequently the TLS package provides Xtreams-style data streams from an established TLSConnection. The TLS-Classic package can be used to obtain classic read-append stream facade for backward compatibility. The usage patterns are quite similar to the old implementation, but all the classes use TLS prefix now.

```
    socket := SocketAccessor
                newTCPclientToHost: 'www.google.com' port: 443.
context := TLSContext newClientWithDefaults.connection := context
newConnection: socket.connection
    when: TLSAnnouncement
    do: [:message | Transcript cr; print: message; flush].connection
connect: [:certificate | certificate inspect. true].stream := connection
readAppendStream.(HttpRequest get: 'https://www.google.com')
writeOn: stream.stream flush.response := HttpResponse readFrom:
stream.stream close. connection close. socket close.context release
```

A notable change is the subject verification block, which now gets the entire certificate as the argument rather than just the subject name. To get the name send the message #subject to the certificate argument. The block can use an optional second argument which is the TLSConnection. Another notable difference is that TLSConnection makes Announcements rather than triggering object events for any messages sent or received.

The TLSContext structure is somewhat refined compared to the old SSLContext. The certificate and session related aspects have been reified into separate TLSCertificateStore and TLSSessionCache components, that can be specialized or replaced by different implementations. The context retained and added properties pertaining to the protocol itself. See the documentation or the package comments for more details.

Currently there are two notable limitations compared to the previous implementation. First, client authentication is not supported yet. Second, session renegotiation is supported but turned off by default because it is not secure. This will be remedied by adding support for the renegotiation_info extension (see RFC#5746 "TLS Renegotiation Indication Extension" for more details). Note that the same renegotiation vulnerability also applies to the previous SSL implementation.

Code Management and Store

Migrating code to use new Glorp database objects

Store has changed from using its original database objects (ODBO) to using Glorp based objects.

The biggest change are the names of the main database accessing objects. Code that used to talk to the classes Store.Bundle or Store.Package, should now be made to talk to Store.Glorp.StoreBundle and Store.Glorp.StorePackage respectively.

We have taken care to duplicate the APIs that were on the ODBO classes and move them to the new Glorp based classes.

When working with image packages and bundles (PackageModel and BundleModel), sending

```
imageModel class storeDatabaseClass
```

will answer the Store.Glorp class. Starting in 7.9, sending

```
imageModel databaseClass
```

will answer the Store.Glorp class. However, if you have loaded the OldStoreDatabse parcel, that will then answer the ODBO Store.Package/Store.Bundle classes.

The biggest difference between the ODBO objects and the new Glorp objects is that you no longer have to explicitly go out to the database to get package and bundle contents. A Store.Glorp object has proxies within it for its contents, and for instance, simply sending methods to a StorePackage will retrieve the StoreMethod objects in that package.

All Glorp objects come in with immediate instantiated values and deferred uninstantiated values. For instance, the version value for a StorePackage is immediate and instantiated as is the name and the current blessing level. However, the blessings themselves are not, nor is the package comment (which is a blob object in the Store schema) nor package classes. Once you send a message to an uninstantiated object, it will automatically retrieve and instantiate its real value from the database.

The ODBO “One To Many” table objects (such as Methods that represented the TW_Methods table and associated Method (TW_Method) objects to Package (TW_Package) objects) are not directly represented in the new Glorp based classes. Instead the

relationships use Glorp mechanisms to automagically attach the final objects to their related owners via the above proxy/instantiation behavior.

Therefore, when you used to get back a Store.Package object, with which you would then use the primary key of the package to get the related Store.PkgClasses and then used the collection of class keys to get the Store.ClassRecord objects, you can now simply send classes or classDefinitions (the former is an alias for the latter) to get the Store.Glorp.StoreDefinitionInPackage objects.

The new Glorp based Store objects do not use any of the database views that were defined as part of the store schema. Thus there are no Glorp objects that represent these views classes. These views are now obsolete. When creating a new store repository, these views are not created. If you are creating a new store repository that will still be accessed by pre 7.8 code, you will want to load the OldStoreDatabase parcel, and then execute:

```
Store.DbRegistry update73
```

This will create all of the required database views required for use with ODBO objects.

Merge Tool Blessing Behavior Change

Starting with 7.9, the way the merge tool blesses source packages and target pundles has changed.

The pre 7.9 behavior is still available, by sending:

```
MergeTool useOldMergeBlessingBehavior: aBoolean
```

The default is false, and changing it to true will make the merge tool create blessings on changed packages and bundles in the same way as it used to do. This is a sticky setting, and once set for an image, it does not need to be reset on the next image or merge tool start.

The new default behavior is to never change the blessing of a package or bundle if the new blessing would be a *lower value* than it currently has. Instead, it creates an “Informative” blessing with all the new information and attach that to the package.

This works for both the new merged version, as well as the from package and bundle versions.

For example, suppose you start with a version with blessing level Release (99), you then want to merge a nw version blessed Development (20).

The old way would mark the new version Merged (60), the source branch version Integrated (50) and the source base version Integrated (50). Thus, the original would be downgraded from 99 to 50, the new version would be set to 60 and the branch version upgraded to 50.

The new way makes the new version is not downgraded from its prior version (99), so a Informative blessing is added named Merged (60) is added to the new package, but the actual blessing level is carried forward to 99.

Also, the original version is not downgraded from 99 to 50, but rather a new informative blessing marked Integrated is added, without changing the actual blessing level of the original version.

Finally, since the branch is upgraded, it will have a new blessing added marked Integrated (50) and the package itself will have its blessing level set to 50.

A new feature is added for those who never want to ever change the blessing levels of packages and only informative blessings added for a merge:

```
MergeTool informativeBlessingsOnly: aBoolean
```

The default is false, and changing it is sticky. This option is also shown in the main menu of the merge tool under the Package menu item, as a menu toggled item.

Resolution Manager

The ResolutionManager is now a fully stand alone engine, StoreBase-Tests-Merge contain 25 + new tests plus updates for all the rest to cover the new behavior(s).

This is step 1 of 3 of making a "One Step Merge" behavior.

Here is the beginning of a potential script for using the ResolutionManager:

```
resolutionManager := ResolutionManager onPundle:
   (StorePackage pundleWithName: 'Toy' version: '8').
```

or:

```
resolutionManager := ResolutionManager onPundle:
   (StoreBundle pundleWithName: 'ToolsBase' version: '0.5')
```

or:

```
collection := Array
    with: (StorePackage pundleWithName: 'Toy' version: '8')
    with: (StoreBundle pundleWithName: 'ToolsBase' version: '0.5').
resolutionManager := ResolutionManager onPundles: collection.
```

This is the standard method for creating a Resolution Manager.

There are four exceptions that could be raised during this :

MismatchedDatabaseError

NoCorrespondingImageItemError

NoCommonAncestor

MergingMayCausePublish

These are all subclasses of a new common parent exception: Store.MergeException.

MismatchedDatabaseError is raised if the image item is not reconciled to the current database. If this exception is handled, returning or resuming will cancel the merge. If this exception is NOT handled, then a dialog will be raised asking if the user wants to reconcile the target package or bundle, if the user answers **yes**, then a reconcile dialog will be opened which the user may choose a version to reconcile against or cancel.

NoCorrespondingImageItemError is raised if there is no loaded image component for the merging package or bundle. If this exception is handled, returing will cancel the merge. If this exception is NOT handled, then a dialog will open warning the user of the problem, and when the user presses **OK**, the merge will be cancelled.

NoCommonAncestor is raised if the image component has no ancestor in common with the package or bundle being merged. If this exception is handled, a resuming will cancel the merge. If this exception is NOT handled, then a error dialog will be opened.

MergingMayCausePublish is raised if a sub component of a bundle does not exist in the image, but does exist in the bundle being merged. If this exception is handled, resuming will continue by creating an empty image pundle for the missing sub component(s). If this exception is NOT handled, then a dialog will open warning the user of the problem, and when the user presses **OK**, the merge will be continue to merge, creating an empty image pundle for the missing sub components(s).

Once a Resolution Manager has been successfully created, you can test the manager and send messages to it:

resolutionManager canMerge

This answers true if the setup has been successfully executed, even if there are no potential resolutions to apply.

resolutionManger doMerge

This creates resolutions for all changes. The result is false if there were no resolutions created, if for instance, the merge only detects differences that are already in the image, this will answer false. It will be true if there are resolutions created.

resolutionManager hasResolutions

This always answers false before doMerge is executed. Afterward, it answers the same value as the result of the doMerge behavior.

resolutionManager successfullyCompared

This answers false before a doMerge is executed. Afterward, it answers true as long as the engine has had something to compare, even if the result is no resolutions.

resolutionManager canApplyAll

This answers true if after a merge, there are resolutions, and all of them can be applied, and there are no Unresolved items. An Unresolved item is one that has more than two alternative resolutions and thus not able to determine a default resolution.

resolutionManager canApplySome

This answers true if after a merge, there are resolutions, and at least one of them can be applied,

resolutionManger applyAll

This behavior will apply all resolutions only if ALL resolutions are applyable (ie #canApplyAll answers true).

resolutionManager applyAllResolved

This behavior will apply all applyable resolutions only if at least one resolution is applyable (ie #canApplyAll or #canApplySome answers true).

During either of the above apply behaviors, if a shared variable is created or modified, a ReInitializeSharedVariableNotification exception will be raised.

ReInitializeSharedVariableNotification is raised when a shared variable is created or modified as part of applying merge resolutions. If handled resuming with true ("exception resume: true") or just resuming with no value ("exception resume"), the shared variable will be (re)initialized. Resuming with false ("exception resume: false") will not (re)initialize the shared variable. If NOT handled, a dialog will open asking the if they wish to (re)initialize the shared varaible.

Because of this, an additional API is available for the Resolution Manager:

resolutionManager quitelyApplyAll

This behavior will do an apply all, and automatically do a resume for any Shared Variable that is modified or created, (re)initializing that Shared Variable.

resolutionManager successfullyAppliedAll

This answers true if after an applyAll, applyAllResolved or quietlyApplyAll there are no unapplied or unresolved resolutions.

SQLite Supported

The StoreForSQLite3 package, which has been available as contributed code, is now fully supported.

Miscellaneous

If you added a reconciled package to a bundle, then re-reconciled the bundle to a prior version that lacked the package, the package became unlinked, so publishing the bundle would create a spurious copy of the package unless you re-reconciled it as well. This is now fixed.

Parcel StoreSqlServer has been renamed StoreForSqlServer to match the general pattern.

Opentalk

WebServices

XMLBindingRegistry change

(AR 63224) The BindingBuilder.Registry had been changed. The registry values are no longer the prototype marshalers but marshaler classes. The change is not backward compatible.

The BindingBuilder creates marshaler instances using newProxyFor: *aString*, where *aString* is a tag name.

For example, the object marshaler entry in the BindingBuilder.Registry is described as:

```
at: 'object' put: ComplexObjectMarshaler
```

BindingBuilder creates a new instance of ComplexObjectMarshaler:

```
^(self registry at: object ifAbsent: [^nil]) newProxyFor: object
```

The XPath expression is set in the ComplexObjectMarshaler>>initialize method.

If you add a new marshaler you need:

1 Register the new marshaler class in BindingBuilder.Registry

```
BindingBuilder.Registry at: foo put: FooMarshaler.
```

2 Use the marshaler #initialize method to set the marshaler attributes:

```
FooMarshaler>>initialize
    super initialize.
    self xpathPrefix: self:: .
```

Customizing WSDL 1.1 part names in messages

(AR 63639) For document style in WSDL 1.1, you can now customize the part name attribute in input and output messages. By default Web services tool generates WSDL 1.1 messages like:

```
<message name="foobarSoapIn">
   <part name="parameter" element="tns:foobar"/></message>
<message name="foobarSoapOut">
   <part name="return" element="tns:foobarResponse"/></message>
```

where input part name is "parameter" and output part name is "return". The new option allows changing the part names.

Refer to the Web Service Developer's Guide for more information.

Support for <sequence> and <all> elements in XML schema

(AR 60766) In the 7.9 release, the <sequence> and <all> elements can be used in an XML schema. Two new marshaler classes have been added: SequentialMarshaler for <sequence>, and UnorderedMarshaler for <all>. By default, building a XMLObjectBinding from an old mapping without <sequence> or <all> will create a SequentialMarshaler. To reset this behavior, use ComplexObjectMarshaler class>>defaultCompositor:, e.g.:

```
ComplexObjectMarshaler defaultCompositor: UnorderedMarshaler.
```

When unmarshaling XML, the new SequentialMarshaler and UnorderedMarshaler classes process XML nodes differently. If a ComplexObjectMarchaler is created with an instance pf SequentialMarshaler as a compositor, the collection of XML nodes is passed to the sequential compositor for each element marshaler. Each element marshaler accepts its nodes, unmarshals and removes them from the collection. If a ComplexObjectMarchaler is created with an UnorderedMarshaler as a compositor, each element marshaler from the compositor uses XPath to collect the nodes to unmarshal. UnorderedMarshaler doesn't support wildcards.

Wildcard processing using <any> and <anyAttribute>

(AR 60766) Wildcard processing is now fully supported. The wildcard bindings <any.../> and <anyAttribute.../> may be used to specify mappings between arbitrary XML elements and attributes. These bindings are especially useful when you need to accommodate loosely-structured data inside elements of your content model.

For details, see the discussion of "Marshaling XML Wildcard Elements" in the Web Service Developer's Guide.

Support for derived types

(AR 61965) Support for derived types follows the model defined by XML Schema. It defines a mechanism to force substitution for a particular type. Just as it is possible to derive a complex from a simple type, so too you can derive a new complex type from an existing one.

In general, the following invariants apply when encoding derived types:

- An instance of ComplexObjectMarshaler specified as abstract expects to marshal an instance of a class inherited from its Smalltalk class, or raises a WrongObjectException.

- If an instance of ComplexObjectMarshaler is not specified as abstract, it can marshal instances derived from its Smalltalk class
- All derived types are encoded with an inline type attribute

In general, when decoding derived types:

- An instance of ComplexObjectMarshaler specified as abstract expects an inline type attribute to decode the XML element or raises a MissingTypeAttributeSignal exceptionIf an instance of ComplexObjectMarshaler specified as abstract cannot find a marshaler for an element or the marshaler found is not derived from the object marshaler, an NoMarshalerSignal exception is raised.

> **Note:** In VisualWorks 7.9, the #useDerivedTypes option has been deprecated and is no used for encoding or decoding. This option has likewise been removed from the Web Services wizard. In addition, the default value for #useInlineType is now set to false. This option was created for the first version of the VisualWorks Web Services implementation, when the principle style was RPC/encoded. The #literal style now causes a xsi:type attribute to be added, as required by the XML Schema specification. RPC/encoded still uses an inline type to encode messages.

For details, see the discussion of "Using Derived Types" in the Web Service Developer's Guide.

Updating a binding and its schema

(AR 51590) The Web Services wizard can now update a WsdlBinding and its schema. The wizard can validate the binding's service map, checking for a service class and proper mappings between operations and methods in the service class. If either of these are missing (e.g., because the binding is an old style that uses pragmas), you can easily set them using two specialized dialogs.

Additionally, the wizard now includes buttons to add and remove WsdlBinding operations (in the **Operations** tab of the **Describe Interface** page).

For details, see the discussion of "Updating a Binding and its Schema" in the Web Service Developer's Guide.

Using XML schema datatypes to specify simple types

(AR 64380) To describe binding and WSDL operation types in the X2O binding wizard, the **Simple Types** menu now uses XML Schema datatypes instead of Smalltalk classes (e.g., xsd:string, xsd:boolean, xsd:time; for details on these datatypes, see: http://www.w3.org/TR/xmlschema-2/). The content of this menu, its order and type mapping is defined by a X2O specification in XMLObjectBinding class>>defaultXsdBindingMap.

For details, see the discussion of "Creating an XML to Object Binding" in the Web Service Developer's Guide.

Using OpenTalk broker events

(AR 50674) With this enhancement, your application can register event handlers that are triggered at significant points in the process of sending and receiving XML documents via HTTP. The Web Services framework includes two events that can be used to activate blocks of user-provided code:

#receivingDOM:

This event is sent after receiving an HttpEntity and creating an XML.Document from the entity contents, just before the document is going to be unmarshaled.

#sendingDOM:

This event is sent after an operation marshals a request/reply as an XML.Document, just before the document is going to be written to the connection stream.

For details and example code, see the discussion of "Using OpenTalk broker events" in the Web Service Developer's Guide.

Marshaling a <group> with unbounded cardinality

(AR 64111) A <group .../> element in a binding specification may now have unbounded cardinality. Unbounded groups are mapped to a collection of Struct objects.

For details and example code, see: "Marshaling a <group> with unbounded cardinality" in the Web Service Developer's Guide.

Support for unbounded and nested <sequence> elements

(AR 63919) A <sequence> element in a XML schema may now include a maxOccurs attribute set to unbounded. It is also possible to nest <sequence> elements in a complex type.

In the VisualWorks implementation, XML complex types with unbounded sequences can only be mapped to <object> elements. Due to their ambiguous semantics (e.g., attribute names can collide), there is no support for the default <struct> mapping. If the X2O Binding Tool is being used, it detects unbounded sequences and converts them to <object> elements.

For details and example code, see: "Using Unbounded Sequences" and "Using Nested <sequence> Elements" in the Web Service Developer's Guide.

XMLBindingRegistry uses marshaler classes not instances

(AR 63224) In the 7.9 release, the process of parsing a binding specification document and creating marshalers has been changed. Henceforth, the implementation creates a new instance from the appropriate marshaler class, instead of copying an existing instance. More specifically, class BindingBuilder creates an instance of XMLObjectBinding with marshalers from the X2O specification.

For details, see: "Binding Specifications" in the Web Service Developer's Guide.

Enhancements to the Web Services wizard

(AR 62095) The WS wizard now includes the following options:

- Create binding faults with SOAP headers (WSDL 2.0).
- Set and update binding fault attributes (WSDL 2.0).
- Set and update SOAP header block attributes (WSDL 2.0).
- Add input, output headers and faults to selected operations.

For details and step-by-step usage guides, see the chapters "Web Services Wizard" and "Building Web Services" in the Web Service Developer's Guide.

Attribute name spaces and white space

(AR 50643) Attributes may now be scoped using XML name spaces, including special attributes such as xml:lang and xml:space (the latter may be used to control white space processing). For this, an

<implicitAttribute ...> definition is necessary because ref="xml:space" is used instead of type="xml:space". For details on the xml: name space, see the class-side method #xmlBinding1998Namespace in XMLObjectBinding, and: http://www.w3.org/XML/1998/namespace.

For details and example code, see the discussion of "Attribute Name Spaces and White Space" in the Web Service Developer's Guide.

Specifying XML schema name spaces

(AR 51833) Class WsdlClassBuilder provides methods that allow you to set the Smalltalk name spaces and packages for each XML schema target name space. The mappings between them are specified using Dictionary objects.

To set these mappings use the #bindingPackageMap: and #bindingNamespaceMap: methods, as follows:

```
(builder := WsdlClassBuilder new)
    package: 'CustomerServices';
    classNamespace: 'CustomerServices';
    bindingPackageMap: (Dictionary new
                        at: 'urn:customer' put: 'CustomerPackage';
                        at: 'urn:address' put: 'AddressPackage';
                        yourself );
    bindingNamespaceMap: (Dictionary new
                        at: 'urn:customer' put: 'CustomerNS';
                        at:  'urn:address' put: 'AddressNS';
                        yourself);
    readFrom: wsdl readStream;
    createClientClasses.
```

Here, the domain classes for the target name space 'urn:customer' will be created in the package named CustomerPackage, and in the CustomerNS name space. The domain classes for 'urn:address' target name space will be created in the package named AddressPackage, and in the AddressNS name space.

Support for arbitrary SOAP headers

(AR 63534) The VisualWorks implementation uses interceptor classes as a mechanism for adding headers to client requests, and also for servers to validate incoming requests.Your application can include interceptor classes to perform this function. An working example is provided in the WebServicesTimeDemo package, using the LoginTimeClient and LoginTimeServer to illustrate.

For details, see: "Using SOAP Header Interceptors" in the Web Service Developer's Guide.

Security Demo

(AR 63424) A new Web Services Security Demo package shows how to create a X2O binding for security headers, and to add headers to a message.

At present, the VisualWorks Web Services framework doesn't support WS-Security specifications, but using by interceptor classes it is possible to add, marshal and unmarshal security headers. The demo illustrates how to accomplish this with a UsernameToken header entry. To run the demo, use class TestSecurityDemo.

For additional details, see: "Security Demo" in the Web Service Developer's Guide.

Preserve Object indentity when marshaling

(AR 62527) It is now possible to marshal Smalltalk objects to XML such that their identity is preserved. The WSDL spec supports XML schema encoding (#literal) and SOAP encoding (#encoded). If a WSDL schema uses the latter #encoded style, identical objects can be encoded using id and href attributes. For this, your application can send #useReference: true to an instance of XMLObjectBinding.

In release 7.9 of VisualWorks, the default value of this option has been changed to false. Note that with this default setting, objects that contain cyclical structures can cause an infinite loop during the process of marshaling to XML.

For details, see the discussion of "Preserving Object Identity" in the Web Service Developer's Guide.

Net Clients

SMTPResponseError

The proceedable exception SMTPResponseError is raised when SMTPClient receives a server reply with the code higher than 400. The exception provides information about the error code and status.The exception allows proceeding with the message if a mail server fails to verify one of recipients.

For example if a mail server retuns a reply as:

```
C: RCPT TO:<bogusRecipient>S: 550 5.1.1 <bogusRecipient>... User
unknown:
```

You can catch the exception and proceed sending the message to the rest of the recipients

```
[self client send: message]
   on: SMTPResponseError
   do: [:exc |
       (exc code = 550 and: [exc status = '5.1.1'])
           ifTrue: [exc proceed]]
```

Net Clients Settings

Most global Net Clients settings are defined in Net.Settings class. The Settings class API allows setting default global options for mail and FTP protocols. The HTTP protocol options are moved from Settings class to the HttpSettings class. The HTTP global options are held in class instance variables and local values can be set in an HttpSettings instance. For backward compatibility, the HTTP settings API is still supported by the Settings class, but all methods are moved to the deprecation protocol.

The HTTP settings can be accessed from the HttpClient using settings method.

For example, to reset the redirectRequest option use:

```
client := HttpClient new.
client settings redirectRequest: false.
client get: 'http://server.com/xxx'.
```

The keepAlive option is deprecated. By default the HttpClient creates a HTTP 1.1 message and opens a persistent connection. To close the connection right after a request, use the following code:

```
client := HttpClient new.
client dontUsePersistentConnection.
client get: 'http://server.com/xxx'.
```

For backward compatibility the keepAlive/keepAlive: API will set the global connectionPersists value in the HttpSettings class.

Timestamp Approximation on a Unix FTP server

An FTP Unix server can send a file's modification time in two formats. Recently modified files can be sent without a year. Some servers set the cutoff at six month, some (e.g., NetWare) at one year, making it

impossible for the client to reliably distinguish dates near the cutoff from recent dates. The protocol does not provide any indication of the server timezone to calculate correct timestamp on the client.

FTPClient unixServerTimestampBlock attempts to return a reasonable approximation of the current time on the server. This timestamp is needed to correctly interpret some of the time related file attributes received from the server. The default block simply assumes that it is the same as the client's. This block allows overriding this default for specific cases where the correct time of the server can be deduced via other means, e.g., if you know that the server clock is 5 hours behind UTC, the block body can be: 'Timestamp nowUTC - 5 hours'."

To set the global value, implement code like the following on the class side:

```
FTPClient unixServerTimestampBlock:
    [ :cl | Timestamp now subtractSeconds: (-5*60*60) ].
```

or on the instance side:

```
client := FTPClient new.client unixServerTimestampBlock:
    [ :cl | Timestamp now subtractSeconds: (-5*60*60) ].
client setUnixServerTimestamp.
```

DLL & C Connect

COM Connect

Support on 64-bit Windows VM

With release 7.9 of VisualWorks, COM Connect henceforth supports 64-bit VMs. This applies to client as well as to server functionality. This was realized by converting functionality previously located in primitives to Smalltalk code.

Automatic Memory Management in COM Connect

Until and including VisualWorks 7.8, the developer was responsible for taking care of COM resources such as interfaces, buffers and structures. This required detailed knowledge of COM memory management rules and distracted the developer from the actual task of developing applications.

In release 7.9, COM Connect introduces automatic memory management; COM resources are now managed through their lifetime by the VisualWorks garbage collector. It will take care of allocated parameters, interfaces and buffers, and release them once they are no longer referenced. This means, however, that since resources are managed by the VisualWorks garbage collector, their actual release may be deferred.

So, excluding C structure access, it is no longer necessary to deal with reference counting, manual releasing or invalidation of resources. For a discussion of how to access COM resources in C structures, see the COM Connect User's Guide.

Compatibility

COM Connect Memory Management was designed to be backwards compatible with previous versions of COM Connect. So in most cases no changes will be required. Code parts using COM running with VisualWorks 7.8 should also be able to run in 7.9. This includes implementation of Interfaces, COM objects, usage of COM resources.

This was achieved by keeping methods, which where previously used to manage memory but disabling or changing their functionality where it was required or suitable. The following sections give more detail.

Changes

Interface pointer and interface reference changes

Interfaces carry only one reference. AddRef and Release calls will be ignored. An interface exists as long as it is strongly referenced by any (also referenced) object. Except when accessing structures, it is no longer required to copy interface references using separateReference.

Interface implementation changes

Interface implementations of a COM object do not carry any references (in the COM sense) themselves any more. As long as they are not passed across image boundaries the reference count of the implementing object will not increase (unless methods like enforceAddRed or enforceDecRef are called). Nevertheless, it is still enough the keep a reference to an interface implementation of a COM object to keep it alive. Additional AddRef/Release calls are not required and will be ignored.

To get a reference counted interface reference to the COM object, it is required to convert the interface implementation into an interface pointer using the asInterfacePointer method.

COMObject adaptations

COM objects are now managed by the VisualWorks garbage collection. They will not be released when the reference count falls to zero but when the last reference to it has been given up. If there are no in-image references to the object this will still happen when the reference count falls to zero. For aggregated objects it is required to set all interface references to nil when they are not used any more. Otherwise, it might protect them from garbage collection.

(Event)Sink changes

In previous versions of COM Connect, calling release on an EventSink disconnected it from the source. Now that COM server objects are released by finalization it would not be sensible to suggest calling release to disconnect them. We suggest calling releaseConnection instead to disconnect Sink objects and set any in-image references to these objects to nil in order to allow garbage collection of these COM Server objects.

This also refers to GenericComEventSink (a VTable Sink) and AdviseSink, which have been introduced by the ActiveX component.

Changes to resource management methods

General resource methods

Methods like release, invalidate and destroyReference still exist and calling them will not cause an exception but will be ignored by the system, as they are not required any more. In development images a transcript message will inform the developer that these calls are not required any more.

To enforce an invalidation or release of a specific resource, please use enforceInvalidation or enforceRelease respectively. But please be careful, because incorrect usage may cause access violations or memory leaks.

Interface methods

As Interface reference counting is managed by COM Connect, addRef and decrementReferenceCount are now without any function. Again, in development images a transcript output will inform developers, that it is not required to call these methods any more.

If you nevertheless need to modify the reference counting of an interface, use enforceAddRef and enforceDecRef instead.

separateReference almost works as before; it creates a separately reference counted copy of an interface. However, it is assumed that calling it is not required any more in most cases. Please note that the created copy will also be managed by COM Connect and it is not required to release it explicitly.

COMInterfacePointer>>newTemporaryInterfacePointerAtAddress: now creates interface pointers which are reference counted, and which are managed and released by COM Connect. If you want to access an interface pointer without modifying it reference count, please use COMInterfacePointer>>newUncountedInterfacePointerAtAddress:.

Deprecation of COM Primitives

(AR 64970) In release 7.9, functionality that was previously located in primitives has been moved to Smalltalk code. Every COM call is now a normal DLLCC call, and every COM callback now is a normal CCallback.

Deprecation of Threaded COM DLL

(AR 64970) The removal of COM primitives also allowed moving multithreading functionality to Smalltalk. Therefore, the MultithreadedCom.DLL which was required in previous versions to perform threaded calls is now also obsolete. Apart from this, these changes will not require changes in your existing Smalltalk applications.

Web Application Server

ISAPI Enhancements

(AR 63322) With the move from Windows Server 2003 to Windows Server 2008 and 2008 R2, web applications need to be upgraded from IIS 7.0 to IIS 7.5. For this, the steps to configure the VisualWorks Application Server ISAPI extension have been revised, to accommodate the changes Microsoft has made to tighten up security and improve manageability.

Here are the steps to install and run the ISAPI extension DLL on Windows 7 with IIS 7.5.

Before Installing IIS

Remember to first confirm access from your installation computer to your Intranet or Internet. The steps to install and configure the ISAPI DLL assume the existence of **`C:\Windows\VisualWave`**, which contains a valid hostmap file and which has **Read**, **Read & execute**, and **List Folder Contents** permissions granted to the local machine IIS_IUSRS group.

If it doesn't already exist, create **`C:\Windows\VisualWave`**, and modify its permissions as specified above. Then, copy the **`hostmap`** file from:

$(VISUALWORKS)\waveserver\waverelays\VisualWave\

to:

C:\Windows\VisualWave

and edit it to have only the appropriate entries for your application.

For the initial configuration and installation, we suggest that you use **`isapi2vw-debug.dll`** instead of **`isapi2vw.dll`**.

Install IIS

1 From the Windows **Start** menu, select **Control Panel > Programs > Programs and Features > Turn Windows features on or off**.

2 Expand **Internet Information Services**.

3 Select the following required items:

- **Web Management Tools > IIS Managment Console**
- **World Wide Web Services > Application Development Features > ISAPI Extensions**

4 The following items may be useful for development:

- **`DebugView.exe`** (See downloads from Microsoft's "Windows Sysinternals" Web page).

5 Press **OK**.

Create a physical directory to hold the ISAPI DLL

Microsoft suggests that the ISAPI DLL not be located inside **`Program Files`** or **`Program Files (x86)`**. For the purpose of these instructions, we'll name it **`C:\vwisapi`**.

1 Create **`C:\vwisapi`**.

2 In the Windows Explorer, right click on **C:\vwisapi** and select **Properties** from the menu.

3 Choose the **Security** tab and then add the local machine's IIS_IUSERS group (Specify **Object Types** to include **Computers**, and select the computer found under **Location**). Give it **Read**, **Read & Execute**, and **List Directory Contents** permissions.

(Select **Add > Advanced > Find Now**.)

4 Press **OK**.

5 Copy **isapi2vw.dll** (or **isapi2vw-debug.dll**) from **$(VISUALWORKS)\waveserver\waverelays\isapi\nt** to **C:\vwisapi**.

Add the application

1 From the Windows Start menu, navigate to **Control Panel > System and Security > Administrative Tools > Internet Information Services Manager** and right click on **IIS Manager**. Choose **Open in Administrative mode** (or **Run as Administrator**).

2 Expand the entry in the left pane for the local machine and Sites (at the second level).

3 Right click on **Default Web Site** and choose **Add Application**. In the **Alias** field, enter **isapi**; in the **Application Pool** field, choose **DefaultAppPool** and in the **Physical Path** field, enter **c:\vwisapi**. Leave **Pass-through authentication** alone (this allows the VisualWorks application to handle authentication).

4 Press **OK**.

Configure IIS

First, configure the Application Pool:

1 In the left pane, choose **Application Pools**.

2 In the **Application Pool** list, choose **DefaultAppPool**. Right-click on **DefaultAppPool** and choose **Advanced Settings**. In the **General** section, select **Enable 32-Bit Applications** and set it to **True**. Ensure **Start Automatically** is set to **True**.

3 Press **OK**.

Next, configure ISAPI and CGI Restrictions:

1 In the left pane, choose the local machine (top level).

2 In the middle pane, ensure that the **Group by** entry is selected as **Category**.

3 Under **Security**, in the middle pane, double click **ISAPI and CGI Restrictions**. Right click in the middle pane and choose **Add...**

4 In the **ISAPI or CGI path** field, enter **c:\vwisapi\isapi2vw.dll**, and in the **Description** field, enter **VW-ISAPI** and make sure that **Allow extension path to execute** is checked.

5 Press **OK**.

Configure the Application

The following steps must be done after configuring the ISAPI and CGI Restrictions:

1 Select your Application (or **Default Web Site**) in the left pane.

2 Double click on **Handler Mappings** in the middle pane. Select **ISAPI-dll** in the middle pane. Right-click and choose **Edit Feature Permissions**. Check the **Execute** check box.

3 Press **OK**.

Restart the Server

1 In the left pane, choose the local machine (top level).

2 Select **Stop**.

3 When it becomes available, select **Start**.

Test your Application

To test the configuration and application, load the following URI in your Web browser:

```
localhost//isapi/isapi2vw-debug.dll?EchoIsapi2vw
```

The browser should display a response similar to the following:

```
VisualWorks ISAPI Extension Test

Echo
VW ISAPI Extension [isapi2vw-debug]
lpszMethod [GET]
lpszPathInfo []
cbTotalBytes [0]
cbAvailable [0]

HexDump of lpbData[0 bytes]
End HexDump
```

```
Start Request
HTTP_CONNECTION=Keep-Alive
HTTP_ACCEPT=*/*
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_ACCEPT_LANGUAGE=en-US
HTTP_HOST=localhost
HTTP_USER_AGENT=Mozilla/4.0 (compatible; MSIE 8.0; Windows NT
6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR
3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
CONTENT_LENGTH=0
HTTPS=off
PATH_TRANSLATED=C:\vwisapi
QUERY_STRING=EchoIsapi2vw
REMOTE_ADDR=::1
REMOTE_HOST=::1
REQUEST_METHOD=GET
SCRIPT_NAME=/isapi/isapi2vw-debug.dll
SERVER_NAME=localhost
SERVER_PORT=80
SERVER_PORT_SECURE=0
SERVER_PROTOCOL=HTTP/1.1
SERVER_SOFTWARE=Microsoft-IIS/7.5
URL=/isapi/isapi2vw-debug.dll
%
End Request
```

Documentation

This section provides a summary of the main documentation changes.

Note that we changed the documentation source format in 7.7. While we have attempted not to lose any content or formatting, some errors have nearly certainly occurred. Please notify us of serious lapses.

Basic Libraries Guide

Restored missing text and character formatting sections in chapter 7.

Tool Guide

Added a link to Code Critic conditions.

Application Developer's Guide

Updates

COM Connect Guide

Add chapters on multithreaded COM and user-defined types support.

Database Application Developer's Guide

Added documentation for new Unicode support, 64-bit support, CTLib client password, SQLite, and enhanced functionality for Array binding and fetching in the ODBC EXDI.

DLL and C Connect Guide

Updates, removing obsolete platforms, and a few clean-ups.

DotNETConnect User's Guide

No changes

DST Application Developer's Guide

No changes

GUI Developer's Guide

No changes

Internationalization Guide

No changes

Internet Client Developer's Guide

No changes

Opentalk Communication Layer Developer's Guide

No changes

Plugin Developer's Guide

No changes

Security Guide

Updated for use of external cryptographic libraries via Xtreams.

Source Code Management Guide

Update atomic load notes.

Walk Through

Updated for 7.8

Web Application Developer's Guide

No changes.

Web GUI Developer's Guide

No changes.

Web Server Configuration Guide

Minor updates.

Web Service Developer's Guide

Extensive revision for 7.9, especially the final chapter on "XML to Smalltalk Mapping". Additional usage scenarios have been added to cover working with SOAP headers (AR 63483), and the terminology used to describe X2O bindings has been clarified throughout the entire document (AR 51150).

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.

GUI

Screen Drawing APIs

Screen drawing APIs zoom:* and displayShape:* have been deprecated. They are not well supported on all platforms any more.

The drag:* variants are still available to simulate drag and drop.

These API will be sunset in 8.0. Until then, the primitive has been replaced with a method version that draws only in the currently focused window. This covers the majority of use-cases found to be calling displayShape:* and zoom:*.

PlugIn

VisualWorks ActiveX PlugIn will be deprecated

The VisualWorks ActiveX Plugin will be deprecated after this release. Microsoft has been moving away from supporting ActiveX controls for some time and have recently announced a plugin-free technology in their Metro-Style browser for Windows 8.

The VisualWorks Plugin code, both the C++ source for the ActiveX control and the Smalltalk parcels, will be moved to obsolete status with the following release. Users who still require this technology will need to build the DLL and runtime images themselves.

COM Connect

Deprecation of COM Primitives

(AR 64970) In release 7.9, functionality that was previously located in primitives has been moved to Smalltalk code. Every COM call is now a normal DLLCC call, and every COM callback now is a normal CCallback.

Deprecation of Threaded COM DLL

(AR 64970) The removal of COM primitives also allowed moving multithreading functionality to Smalltalk. Therefore, the MultithreadedCom.DLL which was required in previous versions to perform threaded calls is now also obsolete. Apart from this, these changes will not require changes in your existing Smalltalk applications.

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.

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`**.

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

VisualWorks 7.8 includes ready-to-use Cairo libraries for use on MS-Windows (Windows 2000 and newer) and Apple Mac 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 Mac 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:

XX XY Xo
YX YY Yo

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

Xn = Xi•XX + Yi•XY + Xo
Yn = Xi•YX + Yi•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.

WriteBarriers

The immutability mechanism in VW can be used to detect any attempts to modify an object. All it takes is marking the object as immutable and hooking into the code raising the NoModificationError. The ability to track changes to objects can be useful for number of different purposes, e.g., transparent database updates for persistent objects, change logging, debugging, etc. While the mechanism itself is relatively simple, it is difficult to share it as is between multiple independently developed frameworks.

WriteBarriers (loadable from preview/parcels/WriteBarriers.pcl) allow multiple frameworks to monitor immutability exceptions at the same time. This framework makes object change tracking pluggable through subclasses of Tracker. A Tracker must implement a couple of methods:

isTracking: *anObject*

Answer true if the tracker is tracking *anObject*

privateTrack: *anObject*

Register and remember *anObject*

privateUntrack: *anObject*

Forget about the object you were tracking

applyModificationTo: *anObject* **selector:** *selector* **index:** *index* **value:** *value*

We've accepted that we are tracking the object and a change has been made to it, what do we want to do? The default behavior is to apply the change to the object.

Note that the framework does not provide a mechanism for keeping track of which objects each tracker is managing. Instead, it leaves the options open. Frameworks may already have a registry of objects that they want to track (e.g., a persistency framework will likely cache all persitent objects to maintain their identity) in which case a separate registry for the corresponding tracker would waste memory unnecessarily.

A deliberate limitation of WriteBarriers is that they will refuse to track any previously immutable objects. Trackers can decide to not apply the modification and emulate the original immutability that way, and refusing to track immutable objects reduces complexity of the solution.

Here's a sketch of a tracker that will announce a Modified announcement for any modification that occurs. Let's assume that this AnnouncingTracker will have its own registry for tracked objects in the form of an IdentitySet (inst var. objects). The first three required methods are fairly obvious:

```
privateTrack: anObject
    objects add: anObject

privateUntrack: anObject
    objects remove: anObject ifAbsent: []

isTracking: anObject
^objects includes: anObject
```

The modification callback needs to call super so that the modification is actually applied, but in addition makes the announcement as well. Note that Tracker subclasses Announcer to make Announcement use easy.

```
applyModificationTo: anObject selector: selector index: index value: value

    super applyModificationTo: anObject selector: selector

    index: index value: value.
    self announce: (Modified subject: anObject)
```

To make use of the tracker, it has to be instantiated, which automatically registers it in the global registry, Tracker.Trackers. Any objects to be tracked by it have to be explicitly registered with it using the #track: message.

```
tracker := AnnouncingTracker new.
tracker when: Modified do: [ :ann | Transcript space; print: ann subject ]
```

```
string := 'Hello' copy.
tracker track: string.
string at: 1 put: $Y.
```

The last statement will trigger the Transcript logging block. To stop tracking an object use the #untrack: message.

```
tracker untrack: string
```

And to deactivate the tracker altogether use the #release message.

```
tracker release
```

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 will search for the translations in the most specific catalogs first and then look 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 preview implementation of SNMP, a remote debugger and a distributed profiler.

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

Opentalk HTTPS

This release includes a preview of HTTPS support for Opentalk. HTTPS is normal HTTP protocol tunneled through an SSL protected socket connection. Similar to Opentalk-HTTP, the package Opentalk-HTTPS only provides the transport level infrastructure and needs to be combined with application level protocol like Opentalk-XML or Opentalk-SOAP.

An HTTPS broker must be configured with a SSLContext for the role that it will be playing in the SSL connections, i.e., #serverContext: for server roles and #clientContext: for client roles. Also, the authenticating side (which is almost always the client) needs to have a

corresponding validator block set as well. The client broker will usually need to have the #serverValidator: block set to validate server certificates. The server broker will only have its #clientValidator: block set if it wishes to authenticate the clients. Note that the presence or absence of the #clientValidator: block is interpreted as a trigger for client authentication.

Here's the full list of all HTTPSTransportConfiguration parameters:

clientContext

The context used for connections where we act as a client.

serverContext

The context used for connections where we act as a server.

clientValidator

The subject validation block used by the server to validate client certificates.

serverValidator

The subject validation block used by the client to validate server certificates.

Note that the same broker instance can be set up to play both client and server roles, so all 4 parameters can be present in a broker configuration. For more information on setting up SSLContext for clients or servers please refer to the relevant chapters of the *Security Guide*.

This example shows how to set up a secure Web Services broker as a client:

```
context := Security.SSLContext newWithSecureCipherSuites.
broker :=
    (BasicBrokerConfiguration new
        adaptor:(
            ConnectionAdaptorConfiguration new
                isBiDirectional: false;
                transport: (
                    HTTPSTransportConfiguration new
                        clientContext: context;
                        serverValidator:
                            [ :name | name commonName = 'Test Server' ];
                        marshaler: (
                            SOAPMarshalerConfiguration new
```

```
                        binding: aWsdlBinding)))
    ) newAtPort: 4242.
```

This example shows how to set up a secure Web Services broker as a server:

```
context := Security.SSLContext newWithSecureCipherSuites.
"Servers almost always need a certificate and private key, clients only
when
client authetication is required."
"Assume the server certificate is stored in a binary (DER) format."
file := 'certificate.cer' asFilename readStream binary.
[ certificate := Security.X509.Certificate readFrom: file ] ensure: [ file
close ].
"Assume the private key is stored in a standard, password encrypted
PKCS8
format"
file := 'key.pk8' asFilename readStream binary.
[ key := Security.PKCS8 readKeyFrom: file password: 'password' ]
ensure:
    [ file close ].
context certificate: certificate key: key.
broker :=
    (BasicBrokerConfiguration new
        adaptor: (
            ConnectionAdaptorConfiguration new
                isBiDirectional: false;
                transport: (
                    HTTPSTransportConfiguration new
                        serverContext: server;
                        marshaler: (
                            SOAPMarshalerConfiguration new
                                binding: aWsdlBinding)))
    ) newAtPort: 4242.
```

This release also includes a toy web server built on top of Opentalk as contributed code, and is not supported by Cincom. It is, however, quite handy for testing the HTTP/HTTPS transports without having other complex infrastructure involved. So here is another example how to set up a simple secure web server as well:

```
| resource ctx |
resource := Security.X509.RegistryTestResource new setUp.
ctx := Security.SSLContext
    suites: (Array with: Security.SSLCipherSuite
            SSL_RSA_WITH_RC4_128_MD5)
    registry: resource asRegistry.
ctx rsaCertificatePair: resource fullChainKeyPair.
(Opentalk.AdaptorConfiguration webServer
```

```
        addExecutor: (Opentalk.WebFileServer prefix: #('picture')
            directory: '$(HOME)\photo\web' asFilename);
        addExecutor: (Opentalk.WebFileServer prefix: #('ws')
            directory: '..\ws' asFilename);
        addExecutor: Opentalk.WebHello new;
        addExecutor: Opentalk.WebEcho new;
        transport: (Opentalk.TransportConfiguration https
            serverContext: ctx;
            marshaler: Opentalk.MarshalerConfiguration web )
    ) newAtPort: 4433
```

Once the server is started, it should be accessible using a web browser, for example https://localhost:4433/hello.

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 I wasn't able to find an IP address that would successfully resolve to an IDN, so I wasn't able to test it. Even our example gives me only the numeric result:

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

Polycephaly

This package provides simple mechanisms for spawning multiple running copies of the 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.

The spawned images (drones) are represented by instances of VirtualMachine. The primary API are the variations of the #do: message which take an action expressed either as a String containing valid Smalltalk code

```
[ :vm | [ vm do: '3 + 4' ] ensure: [ vm release ] ]
   value: VirtualMachine new
```

or an instance of "clean" BlockClosure (one that does not reference any variables, including globals, outside of its scope)

```
| vm |
   vm := VirtualMachine new.
   [ vm do: [3 + 4]
   ] ensure: [ vm release ].
```

Optional arguments can be attached as well.

```
| vm |
vm := VirtualMachine new.
[   vm do: [ :a :b | a + b ] with: 3 with: 4
] ensure: [ vm release ].
```

Note that the arguments will be "BOSS-ed out" for transport, so anything in the scope of objects they reference (transitively) will be included as well. Avoid including things like classes or external resources. For more complex cases where the clean block with arguments is hard to achieve, or when the action needs to hook into the objects in the drone image, the String based action is more suitable. Its advantage is that the code will be compiled in the Drone and can therefore reference classes and globals available in that image. In this case any arguments are referenced in the code as named variables and has to be passed in as a dictionary mapping the variable names to values. They will be included in the compilation scope as other shared variable bindings.

```
| vm |vm := VirtualMachine new.[ vm do: 'a + b' environment: (Dictionary
new at: #a put: 3;
```

```
        at: #b put: 4;
        yourself)
    ] ensure: [ vm release ].
```

Note that a VM instance can be reused multiple times.

```
| vm |
    vm := VirtualMachine new.
    [ (1 to: 5) collect: [ :i | vm do: '3 + 4' ]
    ] ensure: [ vm release ].
```

Consequently it needs to be explicitly shut down with the #release message when no longer needed.For cases when multiple VMs are needed to execute the same action in parallel, VirtualMachines class allows to maintain the whole set of machines as one.

```
| vm |
    vm := VirtualMachines new: 2.
    [ vm do: '3 + 4'
    ] ensure: [ vm release ].
```