Cincom

Cincom Smalltalk™

VisualWorks®

Release Notes

VisualWorks 7.8

P46-0106-17

SIMPLIFICATION THROUGH INNOVATION

Part Number: P46-0106-17

Software Release 7.8

This document is subject to change without notice.

Cincom Systems, Inc.

55 Merchant Street

Cincinnati, Ohio 45246

Phone: (513) 612-2300

Fax: (513) 612-2000

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

Contents

1

Introduction to VisualWorks 7.8

These release notes outline the changes made in the version 7.8 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

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 is marked as readonly by default, so that property must be changed to save changes you make to it.

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

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

Xtreams

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

Xtreams is now included in `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.

1

New and Enhanced Features

This section describes the major changes in this release.

Virtual Machine

Windows VM threads change

With the release of VisualWorks 7.8, the Windows VM has the VM running on a different thread of control than the thread on which VisualWorks windows are created. The principal motivation for this change was to prevent modal windows from keeping timer events from being handled in a timely manner. A secondary consequence of this change is that the runtime now handles window damage when dragging one Smalltalk window over another.

Beyond more responsive graphics updates and better timer behavior, for the most part, these changes should be transparent to the developer. The one case in which the developer should take care, is when making library calls from the VM that cause windows to be created by means other than the VM create window primitive.

When a window is created directly or indirectly on the VM thread, it is necessary to ensure that any messages sent to the VM thread get translated and dispatched. Note that if you do this, if one of the newly created non-Smalltalk windows goes system modal, the VM will not schedule until the window leaves the modal state.

VM startup size command line switches

The virtual machines now recognize seven command line switches ranging from -m1 to -m7. These switches can be used to override the image sizesAtStartup. The -m*n* switch will override the value provided by the *n*th element of the sizesAtStartup array. See the Application Development Guide for a description of these new switches.

The command line switches -h and -z are now obsolete, and will be removed in the next major VisualWorks release. Applications should use the switches -m6 and -m5 instead, respectively.

Scripting support command-line option

We added a new (image level) command line option -cli which will start a command line input loop in the image (the ScriptingSupport parcel must be on parcel path or pre-loaded in the image). This allows interacting with the image through the command line terminal. The image can run headful or headless, this option does not have any effect on that. Note however that the image communicates with the terminal through standard input/output. This has some potentially surprising consequences on Unix where reading from a pipe currently blocks the image and therefore the UI is unresponsive if it runs headful. The Windows pipe accessors use threaded DLL calls and therefore do not exhibit this side-effect.

Improvements to the incremental garbage collectors

In this release, the IGC has been improved to handle certain object changes which were previously ignored.

Under certain circumstances, it is possible for objects to change such that they are no longer strong, weak or ephemeral. Two such cases were fixed in the previous release, such that using isWeakContainer: and isActiveEphemeron: would not invalidate the IGC's working assumptions. Starting with this release, the IGC will also handle object changes due to become:, changeClassTo: and adoptInstance: (it is not necessary for the IGC to be aware of oneWayBecome:).

Finally, note that the finalization queue will be affected by become:, oneWayBecome:, changeClassTo: and adoptInstance:. Consequently, users must use care if their application might send these messages to weak objects or ephemerons in the finalization queue.

Performance improvements to the incremental garbage collectors.

In this release, the incremental garbage collectors have been improved so that they are less likely to encounter a mark stack overflow. This change may result in measurably better IGC performance when there are large collections of regular (non-immediate) objects in the image.

When an incremental GC mark stack overflow occurs, the IGC aborts its marking phase at significant computational expense. Aborts are time consuming because objects have to be unmarked and because, should the IGC abort in subsequent runs, the only remaining option to reclaim memory is to invoke the non-incremental garbage collectors. Typically, the non-incremental garbage collectors end up running when the image runs out of space. In other words, since the IGC fails to curb image growth, the GCs run with large images that are likely to contain a lot of garbage which will require a lot of work to collect. Consequently, such emergency GCs take significant time to run.

The incremental GC stores its mark stack in old space. Hence, any regular collection larger than the free space region of old space and containing unique objects not previously encountered by the IGC is guaranteed to cause an IGC mark stack overflow. Prior to this change, you could trigger an IGC mark stack overflow on purpose by evaluating an expression such as

```
collection := Array new: 10000000.
1 to: collection size do: [:index | collection at: index put: 2 @ 3]
```

and then requesting that the system performs an incremental garbage collect. Using the MemoryMonitor tool would show IGC mark stack overflows were ocuring.

The root cause of the IGC mark stack overflow is that, previously, the incremental GC would try to push all objects found by examining a large object into the mark stack. The new implementation can scan large objects in an incremental fashion, thus avoiding most mark stack overflow conditions. In addition, since large objects will be scanned incrementally, the IGC will better obey object and byte quotas, and will be more responsive to interrupt requests when running in interruptible mode. The approach is credited to Boehm, Demers and Weiser.

Performance improvements to the non-incremental garbage collectors

In this release, the non-incremental garbage collectors have been improved so that they are less likely to encounter a mark stack overflow. The changes may result in measurably better GC performance when there are large collections of regular objects in the image.

When a non-incremental GC mark stack overflow occurs, the GC has to rescan all the object headers, perhaps multiple times, at significant computation expense. Since the non-incremental GC uses eden as its mark stack, any regular collection larger than eden which contains unique objects not previously encountered by the GC is guaranteed to cause a mark stack overflow. You can trigger a mark stack overflow on purpose by evaluating an expression such as

```
collection := Array new: 10000000.
1 to: collection size do: [:index | collection at: index put: 2 @ 3]
```

and then requesting that the system performs a non-incremental garbage collect. Using the MemoryMonitor tool will show that mark stack overflows are ocurring.

The root cause of the mark stack overflow is that, previously, the non-incremental GC would try to push all objects found by examining a large object into the mark stack. The new implementation can scan large objects in an incremental fashion, thus avoiding most mark stack overflow conditions. The approach is credited to Boehm, Demers and Weiser.

F10 key processing on Windows

The Windows VM will now continue running normally when the user presses the F10 key. In addition, the F10 keyboard events will be passed to the image correctly. Moreover, the Windows VM now supports all function keys up to F24.

Mouse button event processing on Windows

The Windows VM will no longer swap the middle and right mouse buttons when using the CTRL key.

Windows mouse wheel event processing

In the past, Windows VMs filtered mouse wheel events when the control and shift keys are pressed. Starting with this release, the Windows VMs will pass these events to the image together with the key state information provided by Windows' WM_MOUSEWHEEL events.

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.

Changes to Memory Policies

In this release, we have addressed a number of problems that could lead to a false indication that fixed space allocations are no longer possible. In the course of addressing these issues, we have made the following changes to memory management.

- Memory policies no longer have to implement the message makeSpaceFor:ofType:. This message is now deprecated. Instead, memory policies have to implement the message handleAllocationError:withArgument:. This new message will be sent directly to the current memory policy by ObjectMemory when an allocation fails.
- We extracted a section of handleFailedBecome: and moved it to ObjectMemory. ObjectMemory now implements the message handleBecomeFailureBetween:and:. If your application copied the default implementation of handleFailedBecome: to, for example, custom subclasses of nil, you should consider examining these implementors and refactoring the code as appropriate.

In addition, the default memory policies have been amended to release unwanted free memory in the idle loop.

Upgraded compilation environment for AIX VMs

Starting with this release, the AIX virtual machines are compiled under AIX version 5.3. Please consult the DLLCC guide with regards to this and other compiler changes (e.g., 64-bit Windows).

Updated Virtual machine build instructions and environment

With this release, the VM sources are shipped (VW commercial licenses only) with a VM build directory structure suitable for building custom virtual machines. The necessary files can be found under src/utilities. Please read **`src/utilities/buildInstructions.txt`** for details on how to use the build directory structure to build virtual machines.

Base Image

Zero-extent Pixmaps and Masks

Primitives 900 and 901 no longer accept zero extent arguments on any platform.

Limitations of Mask class>>forShape:

Mask implements a forShape: method which purports to build a mask out of any VisualComponent subclass. In general this method can be used for simple VisualComponent subclasses. However, the rich family of subclasses rooted by VisualPart (a subclass of VisualComponent) are not necessarily the best fit for this method. This may may work in some cases, but in general, these types of objects are not intended as arguments for forShape:.

Correct Hex Code for Dotless J

In previous versions, evaluating

```
CharacterComposer new compose2: $. with: $j
```

returned the character U+FC10, as configured in #initComposite. However, U+FC10 is “arabic ligature teh with yeh isolated form.” What is intended is to produce “latin small letter dotless J” which is U+0237.

This has been corrected to produce the correct composed character. Documentation has been updated to reflect this.

Location of Undeclareds

Starting in 7.8, when any undeclared is created, it is created in the **(none)** package, and no longer in the Base VisualWorks sub-package System-Name Spaces.

Purging undeclareds will leave the extension of Undeclared in the **(none)** package. Looking closely you will see that the reference is not bold, meaning that the namespace Undeclared is not defined in the **(none)** package, and is indeed still in System-Name Spaces. Indeed, if you look in the System-Name Spaces package, you will see Undeclared there, and it will be displayed in bold.

Leaving this extension in the **(none)** package is benign, and has no effect on any other part of the system.

One of the advantages of having this feature is that no longer will the accidental creation of Undeclared objects mark your Base VisualWorks dirty.

Disabling the Interrupt Key for Deployment

In a runtime image it is desirable to disable the interrupt key functionality. This is done by sending the interruptKeyValue: message.

However, in previous release, that accepted only positive integers and it was difficult to specify a key value there which could not be produced by some combination of keys on some keyboard.

The interruptKeyValue: message now also accepts nil as a parameter, disabling the interrupt key functionality completely:

```
InputState interruptKeyValue: nil
```

ExternalInterface fixes

In 7.7 and 7.7.1, recompiling the definition of an ExternalInterface subclass without changing its superclass caused the associated ExternalInterfaceDictionary shared variable of the class to be reinitialized. Usually this happened in ways that then caused the methods to be recompiled, but if it did not, the user needed to invoke

```
MyExternalInterfaceClass recompileMethods
```

(or recompile them individually by hand) to repopulate the dictionary.

In 7.6 and earlier, ExternalInterfaceDictionary shared variables were always placed in the package of their namespace, which could differ from that of the class. In 7.7 and 7.7.1, they were usually placed in the package of their class, but if they were created by the ExternalInterfaceFinder tool, or programmatically, they could be placed in the null package.

Both issues are fixed in 7.8.

VisualLauncher titlebar

For easier reading of long image paths, the VisualLauncher titlebar shows the filename first and the path (in brackets) second.

In the released images, the VisualLauncher's WindowManager process is called "VisualLauncher." If a new VisualLauncher is created, its process will have the same name as its titlebar at the time of creation.

Default Printing Options

Host and document printing are now the default printing settings for platforms that support host as opposed to Postscript printers.

Object>>unreachableCode

There is a pattern that is sometimes followed in Smalltalk code that looks something like the following:

```
[true] whileTrue: [self someCondition ifTrue: [^someResult]].
self error: 'Should never get here'
```

Over the years, a variety of patterns have sprung up to cover the second expression. Some have done as the example shows. Some use halt. Some send shouldNotImplement.

There is now a specific message send for these types of programming patterns, unreachableCode.

Exception>>description no longer breaks if objects respond to #asString

The previous implementation of GenericException>>description allowed the message text to be a block, but implemented this by testing respondsTo: #asString. If something extended the system with an asString implementation in Object or elsewhere that would make blocks respond to it, this would cause problems.

The test is now somewhat improved, in that it checks explicitly for block closures. The code still needs to be changed to increase flexibility and to be better compliant with ANSI exceptions, but this provides an immediate improvement.

Sort Functions

In the last few releases, VisualWorks has made it easier to use simply unary symbols in many places where we might normally use simple blocks. For example:

```
(aCollection select: [:each | each each isMale]) collect:
   [:each | each name]
```

can be represented more succinctly with:

```
(aCollection select: #isMale) collect: #name
```

We can use this pattern with ifNotNil: patterns as well:

```
^self fetchParent ifNotNil: [:parent | parent bounds]
```

can be equivalently expressed with:

```
^self fetchParent ifNotNil: #bounds
```

One of the areas that has eluded this kind of simplification though, is sort blocks used for sorting (arguments to sorted:, sort:, asSortedCollection:, etc.).

For most sorting cases, you have a collection of objects that you want to sort on some particular attribute. Maybe a sequence of customers you'd like to sort by name.

VisualWorks 7.8 closes this gap by adding a new object type, SortFunction. A SortFunction responds to the same API that blocks for SortedCollections have traditionally used, value:value:. Beyond this bit of polymorphism, they reify the idea of fetching some attribute of an object and comparing it to the same transform for another object. It also captures the direction we'd like to compare them.

SortFunctions are usually created by sending either the message ascending or descending to a unary symbol or to a single argument block.. Sort functions may be chained together using the , (comma) method.

SortFunctions can be chained together in series, because internally, they use the binary <=> method. This binary method is new to VisualWorks, but is commonly referred to as "the rocketship operator" in other languages such as Perl, Python, and Ruby. It is implemented in the base VisualWorks library for CharacterArray and Magnitude types. It returns one of -1, 0, or 1, indicating the collation order between two objects. If your block or unary message returns an object that does not inherit from Magnitude or CharacterArray, you would need to extend those objects with a meaningful <=> implementation.

It is not always possible to have a universal interpretation of how to collate objects. For example, you may want to sort strings, in a locale specific manner. For those cases, one can also use a two argument block as genesis of a sort function. Unlike the classic two argument block used for sorting though which returns a Boolean, it is created by sending either ascending or descending to the block, and it should return one of the 3 collation values (-1, 0, 1).

The following series of examples illustrates SortFunctions in action:

SortedCollection of customers by last name in descending order

Classic:

```
customers asSortedCollection: [:a :b | a lastName > b lastName]
```

SortFunction:

```
customers asSortedCollection: [:each | each lastName] descending
```

or

```
customers asSortedCollection: #lastName descending
```

Sort customers by last name in ascending order, and then by first name in ascending order

Classic:

```
customers sorted: [:a :b | a lastName = b lastName
   ifTrue: [a firstName < b firstName]
   ifFalse: [a lastName = b lastName]]]
```

SortFunction:

```
customers sorted:
   [:each | each lastName] ascending
      , [:each | each firstName] ascending
```

or

```
customers sorted:
   [:each | each lastName] ascending , [:each | each firstName]
```

or

```
customers sorted: #lastName ascending , #firstName ascending
```

or

```
customers sorted: #lastName ascending , #firstName
```

Note that the second and fourth SortFunction alternatives simply demonstrate that the ascending send to the secondary and so on parameters is implicit.

Sort customerNames with the Unicode #shifted algorithm in descending order

Classic:

```
customerNames sort: [:a :b | (UnicodeCollationAlgorithm
   collate: a with: b mode: #shifted) > 0]
```

SortFunction:

```
customerNames sort: [:a :b | UnicodeCollationAlgorithm
   collate: a with: b mode: #shifted] descending
```

GUI

In addition to the changes described here, refer to Deprecated Features for a large list of deprecated GUI messages.

Sanitizing VisualPart bounds

An oft confused point is the difference between the messages preferredBounds and bounds. For an object such as Mask, Icon, or AlphaCompositedImage, there is really only one rectangle that is of any import, so the two methods return basically the same thing for these static graphics: what is the rectangle I draw in?

For VisualPart types though, which participate as members of a view tree, they do have distinct and separate meanings. In that domain, bounds becomes the rectangle that defines the framing box of the object, relative to its own origin. Only nominally, does it have anything to do with the size the object might like to be allocated to draw in. The ultimate decision is up to whatever container object manages its bounds.

For these same objects, the message preferredBounds is meant to return the preferred area the VisualPart would like to occupy. It is not necessarily what the object occupies, but what it might like to occupy.

These two message paths should remain separate as much as possible. A container object may query the preferredBounds of a child, before assigning a bounds to it. But that preferredBounds shouldn’t turn around and send bounds messages, otherwise the possibility of creating infinite loops arises.

Accordingly the default implementation of bounds found in VisualPart has been changed from

```
^container == nil
   ifTrue: [self preferredBounds]
   ifFalse: [container compositionBoundsFor: self]
```

to

```
^container == nil
   ifTrue: [Rectangle zero]
   ifFalse: [container compositionBoundsFor: self]
```

The important distinction is found in the ifTrue: branch, encountered when a view tree is not yet connect to a root node that defines given width and height (such as a Window or PrintMedium).

This causes bounds to be more consistent regardless of whether it has yet been snapped into a view tree with a root, or if the root is not yet realized (such as an umapped window).

A large number of methods in the base were audited to insure that implementors of bounds send bounds to other objects when computing things, and that implementors of preferredBounds restricted themselves to sending preferredBounds of other objects, except where it was clear that the two were verifiably the same values.

If this causes problems for legacy VisualPart subclasses, you may want to copy the original implementation into your subtype.

One use case that is no longer possible with this change is to use CompositeParts as aggregators for static graphics. With the old behavior it was possible to create a CompositePart without parent, add some GeometricWrappers, and then sometimes get them drawn as anticipated. It would work, because when the CompositePart had no parent, it would decide to use its preferredBounds as its bounds. Depending on what you added as children, and how, this may or may not have been what one would expect, and would suddenly behave differently when the CompositePart was parented. It is preferred to use objects such as VisualRow or VisualStack, which are VisualComponents in their own right, to aggregate other VisualComponent like objects. Use of VisualBlock provides the ultimate flexibility in such layout, and does not rely on side effects to work correctly.

Event Reactions

7.8 introduces a framework that brings us one step closer to no longer requiring explicitly separate Controller objects to define user interactions for VisualPart objects. View-Controller pairs still work fine, but it is now possible to write a VisualPart object that doesn’t need a Controller. Going forward, this will be the preferred way to write widgets, unless there is a high need for pluggability in a widget's interaction model.

To be interactive, a VisualPart must implement the following three methods at least:

```
getEventHandler
    "Should return self, unless a controller is being used, then it can return
the controller. Default implementation is to return nil."

handlerForMouseEvent: aMouseEvent
    "This method actually determines which event handler should be used
for a given event. It usually tests using a 'self containsMouseEvent:
aMouseEvent' send at least, and may do other filtering. Receivers with
children, need to allow children to be the handler as appropriate."

handleEvent: anEvent
    self eventReactions reactTo: anEvent
```

Note that this last method is boilerplate. No customization is necessary. The actual code to respond to events, goes in arbitrarily named methods that have a <event:> tags in them. These arbitrary methods make take zero or one argument. The <event:> tag includes a parameter which indicates for what kind of event the method is to be performed.

The following example is a unary method that causes the receiver to takeKeyboardFocus if it doesn't already have it, when it receives a #Button1Down event.

```
myButtonOneDownMethod
    <event: #Button1Down>
    self hasFocus ifFalse: [self takeKeyboardFocus]
```

The next example is a reaction to a mouse scroll wheel event, where the event argument is used.

```
myMouseWheelReaction: aMouseWheelEvent
    <event: #MouseWheel>
    self scrollBy: (aMouseWheelEvent up ifTrue: [-1] ifFalse: [1])
```

It is quite common when putting together an interaction model, to have state that effects how an object interacts. For example, the ClickableGraphic widget has three states: idle, mouse over, and pressed. An alternative to the <event:> tag is to use the <event:filter:>

tag. The filter: parameter is a unary message sent to the receiver which should return a Boolean, indicating if that reaction method should be considered applicable.

The following example is an example of a method that wants to be sent when a #Button1Down event is received IF the receiver currently responds true to #isIdle.

```
idleButton1Down
    <event: #Button1Down filter: #isIdle>
    self beginConsumingMouse.
    self state: #pressed.
    self isInBounds: true
```

Using this technique makes it possible to easily build state machine like interaction models for widgets.

When an Event is dispatched, the filtered methods are searched first. If any of them are found to merit evaluation, then no unfiltered methods are evaluated. But if none of the filtered entries are found to be active, then an unfiltered variant will run, if there is one. This allows one to implement “default” responses to be used when no filter is active.

The parameters for the <event:> tag are similar to the various UI.Event subclass names, but not exactly the same. They are derived by sending the message eventType to the various classes. The current list is:

```
#Button1Down

#Button1Up

#Button2Down

#Button2Up

#Button3Down

#Button3Up

#DoubleClick

#MouseEnter

#MouseExit

#MouseMoved

#MouseWheel

#KeyDown
```

```
#PrefocusKeyDown

#KeyUp
```

Note that if <event: > tagged methods are added or removed to a class, existing instances will not see the changes, unless the following expression is evaluated for the instance:

```
theView propertyAt: #eventReactions put: nil
```

Numerous 7.8 tools widgets have been upgraded to use this model. You can browse senders of event: and event:filter: to get a number of examples of these methods.

Reduction in KeyboardProcessor state duplication

In 7.8, the KeyboardProcessor object underwent a significant overhaul in implementation. The original design involved having an ordered collection of keyboardConsumers (VisualPart objects). These were registered with a KeyboardProcessor when the UIBuilder is emitting widgets from specs.

This leads to a number of problems though. It means that any time a widget is removed from the view tree, it needs to be removed from the KeyboardProcessor's list of keyboardConsumers. This is complicated by the fact that the removal may be indirect (e.g., a parent container may be where the removal from the view tree occurs at). Conversely, if a widget is added in dynamically, it needs to be added to the list. The order they are added and removed, is directly input to the order of the consumers. So it became a case of double book keeping, with lots of opportunities for them to be out of sync and create bugs. It also made it harder to programmatically add keyboard savvy widgets to a view tree.

The KeyboardProcessor was changed to remove this list, and instead point to a view tree object (nominally a ScheduledWindow) and traverse the view tree directly. To help it identify widgets in the view tree, it makes use of the method isTabStop. As the KeyboardProcessor moves focus forward or backwards, or just looks look for an initial focus target, it sends this method to the members of the view tree.

The prior model caused some confusion by maintaining said list of view tree objects, but it then kept track of the currentConsumer, which was an associated controller (rather than one of the members from its keyboardConsumers list). The new method for setting the currently focusedView is focusedView:. The KeyboardProcessor will dispatch to the object returned by sending getEvenHandler to these view tree elements. That will likely be a Controller object if following the MVC pattern, the view tree element itself if the VisualPart is handling its own interaction, or nil if no interaction is desired.

currentConsumer remains in the system for the time being, but is redundant against focusedView. It is scheduled to be removed in the next release.

These changes allowed a number of bugs to just disappear, as well as many call sites that were managing the addition and removal of consumers to the KeyboardProcessor. And it is now much easier to programmatically snap in keyboard savvy widgets to the view tree.

Things To Look For when Upgrading Old Code

There are a couple of issues to be aware of when considering the new KeyboardProcessor implementation, if you have written custom widgets and are having interaction issues.

childrenDo:

Traversal of the view tree depends on this method being able to enumerate the whole active view tree. Any object that would currently answer true to isOpen in a view tree, should be reachable with this method. If you have widgets that manage their own child widgets, you need to make sure that you have a childrenDo: method in your container that enumerates them with this method. If you are inheriting from system objects, this should be taken care of already. You can see system implementations of childrenDo: for examples.

getEventHandler must correlate to a view tree object

If possible, you should use focusedView: instead of currentConsumer: to set the view counterpart of the eventHandler. It is the preferred API.

When using the currentConsumer: method, you are setting the focused event handler. That must correlate with a reachable view tree object though. The way they are correlated, is that the view tree is enumerated using childrenDo: (that's why it is important that you have correct childrenDo: methods) and for each child enumerated, it is sent getEventHandler. One of the children enumerated must return an event handler object which is the same as you are setting. If none is found to do so, the event handler you are trying to set will not be set as the currentConsumer.

Methods keyboardProcessor:, resendToKeyboard, and others no longer sent

It was common for code to need to send messages such as resendToKeyboard, which managed the double bookkeeping. None of these messages are sent anymore to event handler objects. This means that if you can remove your implementations of them, and if you were expecting things to happen in resendToKeyboard or removeFromKeyboard, they will not happen anymore.

The more probable issue is that keyboardProcessor: is never sent anymore to view tree objects. If a view followed the pattern of using this method to store a local reference to the keyboardProcessor, and then expected to send messages to its local keyboardProcessor, this will no longer work. Widgets should use the inherited method keyboardProcessor to get at the keyboardProcessor, rather than referring to a local variable.

Focus Navigation API

There are two sets of APIs that must be implemented by objects to participate in focus management. Historically, one API is filled by a VisualPart object, and the other by its counterpart Controller. The more general roles, are the methods that will be sent to a VisualPart and those sent to the object returned from the VisualPart's getEventHandler message. This has traditionally been a Controller object, but may be the VisualPart itself. Many of the newer custom widgets in the IDE follow this pattern. We refer here to the two roles: view and event handler.

View Messages:

isTabStop

Return true if this view is currently a focus candidate.

isDefaultAction

Return true if this view should be the target of the KeyboardProcessor's doDefaultAction. Usually used for widgets that take a default action, such as activating the "Accept" button when the user presses the Enter key.

Event Handler Messages:

desiresFocus

This should return a Boolean. The KeyboardProcessor sends this method to a keyboard consuming object when it's trying to determine if it is a candidate to be tabbed into. You can simply return true, or do something more complex, such as filter based on whether the receiver is "enabled."

requestFocusIn

This should return a Boolean. Before the KeyboardProcessor will activate your widget as the currently focused widget, it must answer true to this message.

requestFocusOut

This should return a Boolean. Before the KeyboardProcessor will deactivate your widget (moving focus to another widget) it must return true to this method, if it returns false, your widget retains focus. An example of this is to complete user edits or validation.

activate

This is sent to your widget, after you've successfully negotiated the requestFocusIn/requestFocusOut/desiresFocus APIs. It's where you make things happen as a result of focusing your widget. If your widget draws differently when it's focused, you may want to do things such as invalidate here.

deactivate

This is sent to your widget, when focus is being moved to another widget (after successful negotiation of the requestFocusIn/requestFocusOut). It's where you make things happen as a result of defocusing your widget. If your widget draws differently when it's focused, you may want to do things such as invalidate here.

hasControl

This is similar to the activate API. It is sent when the window containing your widget gets focus, and your widget is the focused widget in the window. An example use is for toggling indication of focus that only shows up whether the window is focused or not.

The following two messages would need to be implemented ONLY IF a view had returned true to isDefaultAction:

desiresToTakeAction

This handler is being asked to act as the “default” widget. Return true if it is willing to do so. An example would be the default **Accept** button on a dialog, which would answer based on whether it was currently enabled or not.

simulateMousePress

Do the same thing as clicking the mouse on this widget would result in.

Tools

Enable UI for Internationalization mode now standard

The option to put tools in modes for doing international development (e.g., entering message catalog information when painting UI specs) has been integrated and made standard.

Compare Bundle Contents

A **Compare Bundle Contents** menu item has been added to the versions list of the Published Items Tool.Right click menus exist, when the mouse is over any of the entries which show up. The tool shows changes not only in versions of children components, but also when load order changes.

Blueprint objects moved to Tools-Blueprints

In 7.7.1, a hierarchy of objects known as Blueprints (rooted in AbstractBlueprint) were introduced in the Store-Code Comparison package. These have been moved from Store into the Tools-IDE bundle, in their own Tools-Blueprints package.

Blueprint objects describe code objects (e.g., methods, classes, namespaces, etc.) in their bare minimum form, with literal objects, so that no external framework/system/references need to exist for them, and no linkage is required them. It is a declaration of what it is.

Debugger Short Class Names menu option removed

The toolListDisplayString API does a better job of printing classes with short names whenever they can be, without requiring the user to toggle between modes.

Override Code Tool gets a facelift

The Code Browser Override Code Tool (active when an overridden method, class, shared variable, or namespace is selected) received a substantial facelift. The tool now shows the current code on the right side, and overridden (i.e., original inactive variant) on the left side. In the rare event that there is more than one inactive original, the label above the left side, changes to a menu button and allows the user to pick which overridden variant is being compared. The view below is a compare tool. Buttons with "-" graphics can be used to remove either the overriding entity, or the overridden activity. Also, there is an **Adopt** button in the middle. This button has the effect of "integrating" the override into the original package. The original inactive variant is removed, leaving the override as the sole entity, but it is moved into the original's package.

The menu options associated with overrides in various browser lists were changed from remove and revert to Discard Original... and Revert to Original... respectively.

ToolSafeSection

The toolSafeIn:else: method was refactored to create a first class object around the block of code to run safely and within a time limit. Using a real object makes it possible to serialize these with a single watch dog timer, rather than forking a new process for each invocation. This speeds up toolSafeIn:else: sends by about a factor of about 7.

SideBySideTextComparisonView

This class, which premiered in Store-Code Comparison in 7.7.1, found its way to Tools-Differences so that it could be of wider general use. It received some speed improvements as well as some visual refinement of colors for highlighting.

LaunchPad Improvements

For users who install multiple versions of VisualWorks, the VisualWorks Projects (LaunchPad) application has been enhanced to understand which VisualWorks version created your project. Each project in the project list now displays a tooltip containing its version string, a human readable image signature, and the installation location. This provides the most descriptive information available about the version of VisualWorks which was used to create your project.

The LaunchPad application also has the ability to launch a project image with its corresponding VM, provided that version of VisualWorks is still installed. The application manages a central **`VisualWorks.ini`** file of image signatures, created or appended when you first build a project from a new VisualWorks installation. This file, saved in a fixed platform dependent location, can be edited in the LaunchPad UI. We use the image signature and installed location pair to find and use the image's own VM to start it up. If we do not know the installed location for a project (for example if there is no corresponding signature entry in the file) or the registered install location is no longer accessible, you will be offered the opportunity to run your project with the VM currently in use by the LaunchPad application.

We have updated the LaunchPad UI to create a menu of management items accessed via an **Advanced Options** triangle at the upper right of the UI. Currently you can select an existing project root directory or create a new directory, or you can open and edit the image signature file from here. The Folders button has been removed since that functionality is contained in the **Advanced Options** menu.

When you create a new project with the LaunchPad, if either the installed location of the running LaunchPad image or that image's version information do not match the entry in the signature file, the old entry will be commented out and a new, correct entry will be appended to the file. For situations where you uninstall VisualWorks and re-install it in a different location, this will automatically update the signature file. You always have the option to manually edit the signature file, to provide entries for prior installed versions, or to change the install location if you rename a directory. The signature file contains comments with example syntax in the event you need to make manual changes.

Database

Expanded ODBC support

Before 7.8, the ODBC EXDI only supported Windows. In this release, we have added ODBC support on Mac OS X, Linux, AIX, Solaris and HP Unix.

There are many ODBC drivers on these platforms (3 most popular ones: iODBC, unixODBC, and Merant). We have chosen iODBC from OpenLink Software as our default ODBC driver on the added platforms. We have tested this driver on MacOS X, Ubuntu Linux and Solaris Unix.

If you want to use another ODBC driver, you can simply change the library name in the appropriate ODBC interface classes. For instructions for the driver, refer to the documentation from the vendor.

Code Management and Store

Store update for 7.8

The Store schema has been updated, requiring an update to the database. To update the database, the administrator must evaluate

```
DbRegistry update78
```

in a workspace.

Note that, if applying all updates from 7.3 or earlier, an update for 7.4 (update74) was obsoleted and no longer used.

Loader restriction

The Analysis loader can load definitions in packages that are out of order:

```
Bundle
   Package1 defines Subclass of ClassX
   Package2 defines ClassX
```

However, because an early install is executed when a postLoadBlock is detected, Package1 can not have a postLoadBlock of any kind.

The solution is to either move the definition of ClassX to Package1, or move the postLoadBlock to Package2.

Override restriction

It is no longer permitted to create an override in either:

- a package containing the original definition, or
- a package already containing an override of the definition.

For example:

1. Create method #target in Package A
2. Override it in Package B (no problem)
3. Override it in Package B (rejected)
4. Override it in Package A (rejected)
5. Override it in Package C (no problem)
6. Override it in Package B (rejected)
7. Override it in Package A (rejected)
8. Override it in Package C (rejected)

ScheduledWindow and Overrides

In a previous release, ApplicationWindow was collapsed into ScheduledWindow, and ApplicationWindow is now a Shared variable.

Because of this, if you had previously had a Parcel with code for both ScheduledWindow and ApplicationWindow, the methods would now all show up on ScheduledWindow.

Additionally, if you had in this parcel the a method with the same signature for ScheduledWindow and one for ApplicationWindow, then when loading the parcel, you would end up with a method ScheduledWindow, and it ALSO being overridden.

Unfortunately, which version of the method would be the "original" and which the "override" is undeterminable by the Parcel loader, and so you may end up with a different method than than what you may expect as the active definition.

If you see this, the solution is to go to the overridden method and choose which definition you want to be the true definition, and remove the override or overridden definition. Then republishing your Parcel will produce a correct result.

Glorp SQLServerPlatform

Glorp SQLServerPlatform defines all blob types as varX(max).

However, this definition is not supported on SQLServer 2000. Thus, defining a new Store database on a SQLServer 2000 installation requires the following change :

```
SQLServerPlatform>>blob
  ^self typeNamed: #blob
    ifAbsentPut: [GlorpBlobType new typeString: 'varbinary(8000)';
      queryType: (self varbinary)].
```

For other blob types in Glorp, changes to the following are needed:

```
SQLServerPlatform>>clob
  ^self typeNamed: #clob
    ifAbsentPut: [GlorpClobType new typeString: 'varchar(8000)'].


SQLServerPlatform>>nclob^self typeNamed: #nclob
    ifAbsentPut: [GlorpClobType new typeString: 'nvarchar(8000)'].
```

Opentalk

Short listener backlog on the server causes TCP RSTs on Windows

A while ago our customers alerted us to the fact that very short socket listening queues can easily elicit TCP connection resets on Windows platforms. The solution is increasing the listener backlog size. Consequently we have changed the default backlog size for VisualWave and Seaside brokers to 128 in previous releases. In this release cycle we've evaluated the impact of such increase on other platforms and decided to adopt this default value across all platforms and all types of Opentalk brokers. The value can always be changed in specific broker configurations via the #listenerBacklog parameter

WebServices

Building WSDL specifications

- WsdlBuilder class is now deprecated. For backward compatibility the class is present but all its functionality is implemented by WsdlConfigurationDescriptor.
- To build WSDL 1.1 or 2.0 specifications, use the corresponding subclass of WsdlConfigurationDescriptor:
  - Wsdl11ConfigurationDescriptor creates WSDL 1.1.
  - Wsdl20ConfigurationDescriptor creates WSDL 2.0.

- Sending an asWsdl message to an instance of WsdlConfigurationDescriptor creates a WSDL specification element. For help on how to build a WsdlConfigurationDescriptor, see the examples in the WebServices Demo TestCreateWSDLSchema class
- For backward compatibility, WsdlConfigurationDescriptor can create WSDL specifications from existing operation pragmas

Creating classes from a WSDL specification

- WsdlClassBuilder creates domain, service, client and server classes for WSDL 1.1 and 2.0 specifications.
- WsdlClassBuilder *does not* create operation and server port description pragmas anymore.
- The client and server classes are created using the serviceMap class method that links WSDL interface operations and service class methods.

Note: If you start an existing Opentalk server there is a utility that automatically creates the serviceMap method from existing operations pragmas.

- WsdlClassBuilder *does not* create header processor classes anymore. Instead the user is expected to create the Soap header interceptors for the headers processing. For more information see the WebServices Demo and documentation
- Old applications that use header processors should run in 7.8 without problem.

Web Services GUI Tool

Creating X2O binding

XMLSchemaBuilder class is deprecated. To create an X2O binding from old style pragmas use:

```
BindingBuilder>>
   buildFromClasses:classNamespace:targetNamespace:
```

The X2O Tool on the **Tools** menu helps to create a new instance of XMLObjectBinding or update an existing binding registered in the XMLBindingRegistry. The tool creates complex type marshalers to describe selected classes. The XMLObjectBinding class can create X2O and XML Schema specifications:

asX2OSpecification

creates XML element with X2O specification

asXMLSchemaSpecification

creates XML element with XML Schema specification.

WebServices Wizard

When creating an application from a WSDL schema

- the tool *does not* generate operation pragmas
- the tool *does not* generate classes for Soap header processors
- the client script includes comments when an operation needs Soap headers. For how to create header interceptors see the WebServices Demo and documentation
- if generated client and server classes need to use Soap headers in messages, the Request Brokers must be created with SOAPProcessingPolicy and header interceptors.

See WebServices Demo ClientForAuthSearch11>> newBroker and AuthSearchServer11>> newBrokerFor: for examples.

Expose an application as a web service

Before you start using this Web Services Wizard option you need to have registered X2O binding that includes all domain types, faults and Soap headers description that you may need to describe WSDL interface operations. The Wizard will import the XML Schema <types> created from this X2O in to WSDL schema. If you create WSDL with targetNamespace as, for example "urn:wsdl20", the wizard creates the X2O binding at "urn:wsdl20_WSDLOperations" and includes Struct marshalers for describing operation parameters and return types.

Build new Web Services

This option helps to create a new instance of WsdlBinding. There are two options:

- Create WSDL 2.0
- Create WSDL 1.1

WSDL 2.0 option creates the schema with the following default operation description:

```
pattern="http://www.w3.org/ns/wsdl/in-out"
style="http://www.w3.org/ns/wsdl/style/iri"
type="http://www.w3.org/ns/wsdl/soap"
wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/
   request-response/"
wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
wsoap:version="1.2"
```

In the next step you need to select X2O binding from the list of XMLObjectBinding registered instances and describe the interface operations from a selected service class. If your services include interface faults or Soap headers you need to go to Faults or Soap Headers tabs first and describe them. Then these Faults and headers descriptions will be used as references when you describe operations.

Note: If the service class you selected to create interface operations has operation pragmas from old versions of the wizard, the operation pragmas will be used to describe the operation but new pragmas won't be created.

If you create another new Web Service with the same target namespace as the first one but for a different service class, the Wizard creates one WSDL specification with two <interface> or <portType> elements for each service class. For example:

```
Created fisrt interface from ServiceClass1 at target namespace
"urn:test20"
Created second interface from ServiceClass2 at target namespace
"urn:test20"
```

The server class #wsdlSchema method will be created as:

```
<description targetNamespace="urn:test20"
<interface name="ServiceClass1"..>..</>
<interface name="ServiceClass2"..>..</>
<binding interface="tns: ServiceClass1" name=" ServiceClass1"..>
<binding interface="tns: ServiceClass2" name=" ServiceClass2"..>
<service interface="tns: ServiceClass1"..>
<service interface="tns: ServiceClass2"..>
```

Update Web Services

This option allows updating WsdlBinding instances registered in the WsdlBinding.WsdlBindings global registry. The list of WsdlBinding will be displayed as targetNamespace@ServiceClassName. The Wizard updates one of the WSDL schema interfaces, then collects all interfaces at this target namespace and updates the whole WSDL specification for client and server classes.

The Build new Web Services and Update Web Services options do not create Soap header processors. To run an application that has Soap header in WSDL schema you need to create Soap header interceptors and start server and client Request Brokers with a Soap policy that includes these interceptors.

Mapping XML choice element

XMLTypesParser creates the same mapping for choice element in both the default and object X2O mapping. ChoiceRelation class is deprecated.

For the XML Schema choice element:

```
<complexType name="MyData">
<choice maxOccurs="unbounded">
   <element maxOccurs="3" name="aString" type="xsd:string"></element>
   <element name="anInt" type="xsd:int"></element>
   <element name="aDate" type="xsd:date"></element>
</choice>
</complexType>
```

XMLTypesParser creates the object mapping as:

```
<object  name="MyData" smalltalkClass= MyData >
<choice name= choiceValue  maxOccurs="unbounded">
   <element maxOccurs="3" name="aString" type="xsd:string"></element>
   <element name="anInt" type="xsd:int"></element>
   <element name="aDate" type="xsd:date"></element>
</choice>
</object>
```

and creates the default mapping as:

```
<struct  name="MyData >
<choice name= choiceValue  maxOccurs="unbounded">
   <element maxOccurs="3" name="aString" type="xsd:string"></element>
   <element name="anInt" type="xsd:int"></element>
   <element name="aDate" type="xsd:date"></element>
</choice>
</struct>
```

Marshaling choice values

The new implementation expects a choice value as an instance of Association or a Collection of Associations. For backward compatibility there is an option in Setting->XMLObject Marshaling ->"Use Struct as Choice value".

To switch using a Struct as the choice values, set the option on or execute:

```
ChoiceMarshaler useStructAsChoiceValue: true
```

Examples

1 General case:

```
manager :=  XMLObjectMarshalingManager
    on: (XMLObjectBinding loadFrom: '<?xml version ="1.0"?>
<xmlToSmalltalkBinding name="testChoiceObject"
    targetNamespace="urn: testChoiceObject"
    useInlineType="false"
    xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
    xmlns:tns="urn: testChoiceObject">
<object name="Parameter" smalltalkClass="Association">
    <element name="key" ref="xsd:string" aspect="key"  />
    <choice name="value" aspect="value" >
        <element name="string" ref="xsd:string"/>
        <element name="int" ref="xsd:short"/>
    </choice>
</object>
</xmlToSmalltalkBinding>' readStream).

anAssociation := '1'->( #string -> 'string1' ).
xml := manager marshal: anAssociation atNamespace:
    'urn:testChoiceObject'.
obj := manager unmarshal: xml printString readStream
atNamespace:
    'urn:testChoiceObject'.
obj value is an instance of Association.
```

Marshaling with useStructAsChoiceValue turned on:

```
ChoiceMarshaler useStructAsChoiceValue: true.
anAssociation1 := '1'->( Struct new at: #string put: 'string1';
    yourself ).
xml := manager marshal: anAssociation1 atNamespace: ns.
obj := manager unmarshal: xml printString readStream
    atNamespace: ns.
obj value is an instance of Struct.
```

2 Choice with maxOccurs="unbounded" attribute.

```
manager :=  XMLObjectMarshalingManager
    on: (XMLObjectBinding loadFrom: '<?xml version ="1.0"?>
<xmlToSmalltalkBinding
name="testChoiceObjectWithMaxOccurs"
    targetNamespace="urn: testChoiceObjectWithMaxOccurs"
    useInlineType="false"
    xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
    xmlns:tns="urn:testChoiceObjectWithMaxOccurs">

<object name="Parameter" smalltalkClass="Association">
    <element name="key" ref="xsd:string" aspect="key"  />
    <choice name="value" minOccurs="1" maxOccurs="unbounded">
       <element name="string" ref="xsd:string"/>
       <element name="int" ref="xsd:short" maxOccurs="2"/>
    </choice>
</object>
</xmlToSmalltalkBinding>' readStream).

ns := 'urn: testChoiceObjectWithMaxOccurs'.
anAssociation := '1'->(OrderedCollection
    with: ( #string -> 'string1' )
    with: ( #int -> (OrderedCollection with: 1 with: 2 ))
    with: ( #string ->  'string2' )).

xml := manager marshal: anAssociation atNamespace: ns.
obj := manager unmarshal: xml printString readStream
    atNamespace: ns.
obj value is OrderedCollection of associations
```

Marshaling with # useStructAsChoiceValue turned on:

```
ChoiceMarshaler useStructAsChoiceValue: true.
anAssociation1:= '1'->(OrderedCollection
       with: (Struct new at: #string put: 'string1'; yourself)
       with: (Struct new at: #int
          put: (OrderedCollection with: 1 with: 2);
          yourself)
       with: (Struct new at: #string put: 'string2';  yourself)).

xml := manager marshal: anAssociation1 atNamespace: ns.
obj := manager unmarshal: xml printString readStream
    atNamespace: ns.
obj value is OrderedCollection of associations.
```

In the case of a choice with the maxOccurs="unbounded" attribute, the new implementation no longer accepts a choice values as:

```
value:= '1'-> (Struct new
    at: #string put: (OrderedCollection with: 'string1' with: 'string2');
        yourself);
    at: #int put: (OrderedCollection with: 1 with: 2); yourself);
        yourself).
```

Net Clients

MimeEntity contents: change

The new behavior of MimeEntity>>contents: added in previous release makes an attempt to automatically adjust the content-type to make sure it is valid in case a pre-existing entity is being morphed into a different one. However, mime-type validation in the general case is rather difficult, and we've received a number of customer complaints that it is actually interfering with their intent, when trying to use it for creation of new entities. Given that the later use case is more common than the former, we've decided to drop the validation and leave content-type of the entity alone if it is already set.

This means that, if the user decides to set content-type manually, it's his/her responsibility to set it correctly. Failure to do so can result in creation of invalid MIME entities that can fail in unexpected ways.

HttpClient provides authentication info for a proxy remote server

In order to support separate and independent authentication for a proxy server and a remote server on the same connection we had to split the existing AuthenticationPolicy into two:

- AuthenticationPolicy for a remote server
- ProxyAuthenticationPolicy for a proxy server

To set policies for HttpProtocolInterpreter, use authPolicy and proxyAuthPolicy methods.

HttpProtocolInterpreter methods username:password: and username:password:realm: set authentication information for AuthenticationPolicy.

HttpProtocolInterpreter method proxyUser: sets proxy user for both ProxyAuthenticationPolicy and proxy host.

The following AuthenticationPolicy methods were deprecated:

- username:password:realm:useProxy:

- username:password:useProxy:
- useProxy:

Use the corresponding policy instead. Here is an example of Http request with separate proxy and server authentication. The proxy requires proxyUserid/proxyPassword to authenticate the client and the remote server requires userid/password.

```
proxy := HostSpec new
    name: 'proxy';
    port: 3129;
    type: 'http';
    user: (NetUser username: 'proxyUserid' password: 'proxyPassword');
    yourself.

(client := HttpClient new)
    proxyHost: proxy;
    useProxy: true;
    username: 'userid' password: 'password'.

reply := client get: url
```

STARTTLS support to SMTP

The SMTPS package supported only one configuration where the server uses a dedicated port (usually 465) for SMTPS. A common server configuration is to share the common SMTP port (25) for both plain SMTP and SMTPS connection. In this configuration a special STARTTLS extension of the SMTP protocol has to be used. We've added support for this extension in this release.

To employ this extension the client has to be configured to useSecureConnectionIfAvailable (as opposed to the useSecureConnection configuration that still requires a dedicated port). Note that configuring the client in this way does not guarantee a secure connection. If the server is not configured to support a secure connection the client will proceed with unsecured connection.

Distributed Smalltalk

We reviewed the DST tools and removed most of them, because they were either based on obsolete frameworks (OldDebugger, OldBrowser) or not particularly useful (in many cases they weren’t functioning properly anymore). Notably:

- DST settings were ported to the new settings framework.

- The standalone DST launcher has been replaced with a dedicated menu on the main Launcher.
- The presentation-semantic split framework (DST_PS) was also removed, because nothing depends on it anymore. The core functionality of DST is not affected.

These changes are in line with our long term intent to sunset DST and replace it with Opentalk-IIOP.

DLLCC

Win32SystemSupport>>GetLastError should not be invoked

In this release, we reimplemented Win32SystemSupport>>GetLastError as:

```
self shouldNotImplement
```

If you need to capture an error associated with a Windows API call, then use the _wincall keyword in the DLLCC function definition pragma. If a called Windows API function fails by returning zero, DLLCC will call GetLastError() and return the error in the pseudo variable _errorCode. This pseudo variable is available to the primitive failure code. Thus, you can write

```
<C: DWORD _wincall SomeFunction(void)>
   ^self externalAccessFailedWith: _errorCode
```

Calling GetLastError() directly is not guaranteed to return the error associated with the Windows API function last called from DLLCC. This is because a Smalltalk process switch could have occurred between the time SomeFunction() failed when called and the time GetLastError() is to be called to find out what happened. Moreover, calling GetLastError() may not return the error code desired because the VM itself may call other Windows API functions before continuing execution up to the point that a Smalltalk process calls GetLastError() to determine why SomeFunction() failed. Consequently, the image should never call GetLastError() directly, and instead should always rely on the _wincall keyword to capture error conditions arising from calling Windows APIs.

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

COM Connect Guide

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

Database Application Developer's Guide

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

No changes

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

Various updates for 7.8.

3

Deprecated Features

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

Virtual Machine

Command line switches

The command line switches -h and -z are now deprecated, and will be removed in the next major VisualWorks release. Applications should use the switches -m6 and -m5 instead, respectively.

Memory policy

The message makeSpaceFor:ofType: is now deprecated. Instead, memory policies have to implement the message handleAllocationError:withArgument:.

GUI

ReadOnlyTextEditorController Obsolete

The ReadOnlyTextEditorController class is now obsolete, and has been removed from the system.

Any text editor component can now be designated “Read Only” (whether it it accept character input) or “Selectable” (can contained text be selected for copy operations, or cut/paste operations if it is not read-only). For the most common case where a component is both read-only and selectable, the convenience API beReadOnly is implemented on TextEditorView.

If you need the old ReadOnlyTextEditorController class prior to converting your application, please load the Obsolete-ReadOnlyTextEditorController parcel from the **/obsolete** directory.

Deprecated Methods

The following table lists methods that have been deprecated in 7.8 for use in the system, as well as some hint about the nature of their deprecation where applicable.

Method	**Type(s)**	**Replacement**
newDropDownWindowInBounds: eventDriven:builder:model:	ComboBoxButtonController	newDropDownWindowInBounds: builder:model:
openDropDownListWithEvent:	ComboBoxButtonController	newDropDownWindowInBounds: builder:model:
removeFromKeyboard	Controller,VisualPart	*no need to send this message*
resendToKeyboard	Controller,VisualPart	*no need to send this message*
addKeyboardReceiver:	KeyboardProcessor	*no need to send this message*
default:	KeyboardProcessor	*no need to send this message*
keyboardConsumers:	KeyboardProcessor	*no need to send this message*
removeKeyboardReceiver:	KeyboardProcessor	*no need to send this message*
sendKeyboardTo:	KeyboardProcessor,UIBuilder	*no need to send this message*
setActive:	KeyboardProcessor	focusedView:
augmentFrom:to:menuName:	Menu	augmentFrom:to:menuName:for:
menuWindowRecycle:	MenuView	closeAndUnschedule
displayPart:on:	Panel	displayChild:on:
partFramingBlock:	Panel	childFramingBlock:
subparts	Panel	children
subpartsDo:	Panel	childrenDo:
updatePartFrames:	Panel	updateChildFrames:
updatePartFrames:animate:for:	Panel	updateChildFrames:animate:for:
mostSpecificObject	AbstractNavigatorState	narrowestSelectionType

Method	Type(s)	Replacement
mostSpecificSelection	CodeTool	narrowestSelectionType
named:buildWith:match:enabled:	TabPartDescription	*cascade of single argument setters*
findSuperpart:	VisualPart	findParent:
superpart	VisualPart	parent
superpartsDo:	VisualPart, Window	parentsDo:
isEventDriven	Window	*no need to send this message*

WebServices

WsdlBuilder class is now deprecated. For backward compatibility the class is present but all its functionality is implemented by WsdlConfigurationDescriptor.

XMLSchemaBuilder class is deprecated. To create an X2O binding from old style pragmas use:

```
BindingBuilder
   buildFromClasses: classNamespace: targetNamespace:
```

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.

Win64 VirtualMachine

A new 64-bit Windows VM is included in this distribution as preview. The usual executable files are included.

Known Limitations

- The debug VM might print assertion failure warnings for DLLCC callouts that return structures that cannot be returned in a register.
- The assert VM may crash at the end of a callback due to a failure in saving a non-volatile register.
- Debug and Assert VMs will only print the low 32-bits of addresses when printing assertions to the console.
- The traditional COM mechanism is not yet supported.
- The "Microsoft Visual C++ 2010 Redistributable Package (x64)" must be installed. Download and install this package (**`vcredist_x64.exe`**) from Microsoft.

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.

DB2 Support

Updated support for Glorp for DB2 is available in the /preview/glorp directory. EXDI support for DB2 is now supported product.

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. It has not been finished yet, but may be loaded. 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 7.7 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 of the Smalltalk 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:

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

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

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

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

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

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

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

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

The Context Stack

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

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

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

Grouping Operations

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

Deploying VisualWorks with Cairo Support

MS-Windows

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

Mac OS X

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

Ongoing Work

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

Smalltalk Archives

Even though Smalltalk Archives are a supported feature of ObjectStudio 8, they are supported only in preview for VisualWorks at this time.

A Smalltalk Archive is a file containing a collection of parcels, compressed (using tar). The archive specifies a load order for the parcels, and supports override behavior.

Unlike publishing a bundle as a parcel, a Smalltalk Archive preserves package (and parcel) override behavior and package load order. Accordingly, Smalltalk Archives are a good alternative to publishing a bundle, in some circumstances.

Smalltalk archive files have a .store filename extension.

To load an archive, use the File Browser to locate the file, then right-click on it and select **Load**.

The archive loads with the parcel and bundle structure it had when it was saved. Database links might or might not be preserved, depending on the settings at the time it was saved.

To publish a Smalltalk Archive, select the bundle (or package) in the System Brower, the select **Package** > **Publish as Parcel…** The options in the publish dialog are the standard publish options, except for the Store options section. Because you want to save as a Smalltalk Archive, leave that checkbox selected. If you are using the Smalltalk Archive for deployment, and so do not need the database links, uncheck the **With database links checkbox**; otherwise, leave it checked.

The archive is published, by default, in the current directory, typically the image directory. The file name is the bundle name with a .store filename extension.

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
```

Sparing Scrollbars

Sparing Scrollbars extends VisualWorks widgets to optionally set scrollbars to be dynamic and only appear when necessary. When loaded on top of the UIPainter, the dataset, list, table, tree, text editor, hierarchical view, or view holder widgets, or the window itself may be specified to use this feature in the **Details** page of the UIPainter tool. Sparing Scrollbars includes some test and sample classes which demonstrate the usage of this new behavior.

The Sparing Scrollbars component can be loaded from the parcel preview/SparingScrollbars.pcl.

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

Store for Access

The StoreForAccess parcel, formerly in “goodies,” has been enhanced by Cincom and moved into preview. It is now called StoreForMSAccess, to distinguish it from the former parcel.

The enhancements include:

- A schema identical to that for the supported Store databases.
- Ability to upgrade the schema with new Cincom releases (e.g., running `DbRegistry update74`).
- Ability to create the database and install the tables all from within Smalltalk, as described in the documentation.
- No need to use the Windows Control Panel to create the Data Source Name.

The original parcel is no longer compatible with VW 7.4, because it does not have the same schema and ignores the newer Store features.

While MS Access is very useful for personal repositories, for multi-user environments we recommend using a more powerful database.

Store for Supra

In order to allow Store to use Supra SQL as the repository, the `StoreForSupra` package provides a slightly modified version of the Supra EXDI, and implements circumventions for the limitations and restrictions of Supra SQL which are exposed by Store. The Store version of Supra EXDI does not modify/override anything in the base `SupraEXDI` package. Instead, modifications to the Supra EXDI are achieved by subclassify the Supra EXDI classes.

Circumventions are implemented by catching error codes produced when attempting SQL constructs that are unsupported by Supra SQL and inserting one or more specifically modified SQL requests. The Supra SQL limitations that are circumvented are:

- Blob data (i.e. LONGVARCHAR column) is returned as null when accessed through a view.
- INSERT statement may not be combined with a SELECT on the same table (CSWK7025)
- UPDATE statement may not update any portion of the primary KEY (CSWK7042)
- DELETE statement may not have a WHERE... IN (...) clause with lots of values (CSWK1101, CSWK1103)
- When blob data (i.e. LONGVARCHAR column) is retrieved from the data base, the maximum length is returned rather than the actual length
- Supra SQL does not have SEQUENCE

StoreForSupra requires Supra SQL 2.9 or newer, with the following tuning parameters:

```
SQLLONGVARCHAR = Y
SQLMAXRESULTSETS = 256
```

StoreForSupra installation instructions

1. Install Supra SQL
2. Create a Supra database
3. Use XPARAM under Windows to set the following
   - set Password Case Conversion = Mixed
   - set Supra tuning variable SQLLONGVARCHAR = Y
   - set Supra tuning variable SQLMAXRESULTSETS = 256
4. Start the Supra database
5. From the SUPERDBA user, create the Store administration user with DBA privileges.
   - User ID BERN is recommended, password is your own choice.
   - Sample SQL for creating the Store administration use:

   ```
   create user BERN password BERN DBA not exclusive
   ```
6. Load the `StoreForSupra` parcel.
7. To create the Store tables in the Supra database, run the following Smalltalk code from a workspace (You will be prompted for the Supra database name, the Supra administration user id and password.)

   ```
   Store.DbRegistry goOffLine installDatabaseTables.
   ```
8. To remove the Store tables from the Supra database, run the following Smalltalk code from a workspace

   ```
   Store.DbRegistry goOffLine deinstallDatabaseTables.
   ```

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.

Security

OpenSSL cryptographic function wrapper

The OpenSSL-EVP package provides access to most of the cryptographic functions of the popular OpenSSL library (http://www.openssl.org). The functions currently available include

- symmetric ciphers: ARC4, AES, DES and Blowfish
- hash functions: MD5, SHA, SHA256, SHA512
- public ciphers: RSA, DSA

The API of this wrapper is modelled after the native Smalltalk cryptography classes so that they can be polymorphically substituted where necessary. Since these classes use the same name they have to live in their own namespace, Security.OpenSSL. The intent is that each set of classes can be used interchangably with minimal modification of existing user code.

Along these lines, you can instantiate an instance of an OpenSSL algorithm in the same way as the native ones. For example:

```
| des ciphertext plaintext |
des := Security.OpenSSL.DES newBP_CBC
      setKey: '12345678' asByteArray;
      setIV: '87654321' asByteArray;
      yourself.
ciphertext := des encrypt: ('Hello World!' asByteArrayEncoding: #utf_8).
plaintext := (des decrypt: ciphertext) asStringEncoding: #utf_8
```

An alternative way to configure an algorithm instance is using cipher wrappers. The equivalent of the #newBP_CBC method shown above would be the following.

```
des := Security.OpenSSL.BlockPadding on: (
   Security.OpenSSL.CipherBlockChaining on:
      Security.OpenSSL.DES new ).
```

Note that while the APIs look the same the two implementations have different underlying architectures, so generally their components should not be mixed. That is, OpenSSL wrappers merely call the OpenSSL library with some additional "flags", whereas the Smalltalk versions augment the calculations. In general, it won't work properly to use a Smalltalk cipher mode wrapper class around an OpenSSL algorithm and vice versa.

The only thing that is different with hash functions is that the OpenSSL version does not support cloning, so #copy will raise an error. Consequently, it is currently hard to use them with HMAC, which uses cloning internally. We have yet to modify the HMAC implementation to avoid that. However the wrapper already provides SHA512 which is not yet available with the Smalltalk library.

The wrapper also supports 2 public key algorithms, RSA and DSA. The keys for these algorithms are more complex than the simple byte sequences used with symmetric ciphers. However, the wrapper is written so that the API uses the exact same kind of objects for both the Smalltalk version and the OpenSSL version. Similarly for DSA signatures, both versions use DSASignature instances. Here's an example.

```
| message keys dsa signature |
message := 'This is the end of the world as we know it ...' asByteArray.
keys := Security.DSAKeyGenerator keySize: 512.
dsa := Security.OpenSSL.DSA new.
dsa privateKey: keys privateKey.
signature := dsa sign: message.
dsa publicKey: keys publicKey.
dsa verify: signature of: message
```

The current version of the wrapper should support typical OpenSSL installations on Windows and Linux and various Unixes out of the box. There is only one interface class, with platform specific library file and directory specifications in it. If you get a LibraryNotFoundError when trying to use this package, you may need to change or add these entries for your specific platform. You need to find out what is the correct name of the OpenSSL cryptographic library on your platform and where is it located, and update the #libraryFiles: and #libraryDirectories: attributes of the OpenSSLInterface class accordingly. More information can be found in the *DLL and C Connect User's Guide* (p.51). To obtain the shared library for your platform, see http://www.openssl.org/source. Note that the library is usually included with many of the popular Linux distributions, in many cases this package should just work.

A note about HP platforms. If your version of the openssl library doesn't contain (export) the requested function, the image can hang. On Windows, an exception is thrown instead (object not found). The workaround is to verify that your version of the library has the functions you need. For example, the "CFB" encryption facility wasn't available until version 0.9.7.e. And the sha256 and sha512 are only available after 0.9.8 and higher.

Also, on all platforms, remember that the openssl library uses pointers to memory areas which are valid only while the image is still running. After an image shutdown, all pointers are invalid. Your code should therefore discard OpenSSL objects, and generate new ones with each image restart. Even though your old objects will be alive at startup, (a return from snapshot), the pointers are invalid, and the openssl library no longer remembers any of its own state information from the previous session.

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 ].
```