Cincom

VisualWorks®

7.1 Release Notes

P46-0106-06

Part Number: P46-0106-06

Software Release 7.1

This document is subject to change without notice.

Cincom Systems, Inc.

55 Merchant Street

Cincinnati, Ohio 45246

Phone: (513) 612-2300

Fax: (513) 612-2000

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

Contents

1

Introduction to VisualWorks 7.1

These release notes outline the changes made in the version 7.1 release of VisualWorks. Both Commercial and Non-Commercial releases are covered. These notes are not intended to be a comprehensive explanation of new features and functionality nor are they intended to be used in lieu of the product documentation. Refer to the VisualWorks documentation set for more information.

For late-breaking information on VisualWorks, check the Cincom Smalltalk website at http://www.cincom.com/smalltalk. For a growing collection of recent, trouble-shooting tips, visit http://www.cincomsmalltalk.com:8080/CincomSmalltalkWiki/Trouble+Shooter.

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:8080/CincomSmalltalkWiki/Cincom+Smalltalk+Platform+Support+Guide

Product Patches

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

http://www.cincomsmalltalk.com/CincomSmalltalkWiki/VW+Patches

ARs Resolved in this Release

The Action Requests (ARs) resolved in this release are listed in: 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

Multi-process UI support

UI processes used to be on a single thread, and to fork an additional UI thread required loading the ForkedUI goodie. With this release, multiple UI process support is native to VisualWorks.

Using multiple UI process is described in the *Application Developer's Guide*.

Additional changes to windows have been required by this support. Conversion instructions are provided later in this document.

Professional Debugger

The Professional Debugger Package, from Crafted Smalltalk, is now the system debugging package. It provides many enhancements over the previous system debugger.

Notably, instead of inserting self halt into a method to cause a break, you can now insert "probes" to monitor a program's state. Breakpoints break program execution and display a debug window, as did self halt. In addition, watchpoints provide a mechanism for displaying status messages in a watch window.

The familiar Step commands are changed somewhat. The new commands are:

Step Into
Steps into message sends (things you don't see). This is the former "send" command.

Step
Steps through the visible source only, stepping into blocks. This is the former "Step into block" command.

Step Over

Steps through the visible source, stepping over block. This is the former “Step” command.

These three commands, in the above order, are bound to F5-F7 keys.

For additional information about the debugging facilities, refer to the *Application Developer’s Guide*, chapter 9, “Debugging Techniques.”

Keyboard Binding changes

A number of changes have been made to the shortcut key bindings, as described in “Keyboard bindings” later in this document. This change affects a number of old familiar shortcuts.

In particular note that the composition key combination has changed to Ctrl+K, from the former Ctrl+Q. Documentation has been updated to reflect this and other changes.

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.

Initializing Shared Variables

There are a number of inconsistencies known in how classes and shared variables are initialized when loading code from the several storage options. The following table summarizes cases where loading is correct (✓) and incorrect (✗).

	Parcel		**Package**			**Class**
	Save	**FileOut**	**Source**	**Binary**	**FileOut**	**FileOut**
New class with initialize method	✓	✓	✓	✓	✗	✓
Existing class with new initialize method	✗	✓	✓	✗	✗	✓
Overridden class initializer		✓	✓	✗	✗	✓
Shared variable in class with initializer	✓	✗	✓	✓	✓	✓
Shared variable in namespace with initializer	✓	✗	✓	✓	✓	✓

The columns describe a method of saving code, respectively (by System Browser menu command):

- Parcel ➞ Save
- Parcel ➞ File Out As
- Package ➞ Publish
- Package ➞ Publish As Parcel
- Package ➞ File Out ➞ Pacakge
- Class ➞ File Out As

The problem is that the results are not consisten for loading code saved using these various methods.

This problem is recognized, and will be corrected in the next release.

Limitations listed in other sections

- GUI: Known Limitations
- Application Server: Known Limitations

2

VW 7.1 New and Enhanced Features

This section describes the major changes in this release.

Base system

Deploying without sources

We have redefined removeFileAt: in SourceFileManager to use discardSourceAt:, and have added a removeAllSources. These methods provide two idioms for deploying without sources:

```
SourceFileManager default removeAllSources.
```

and

```
SourceFileManager default removeFileAt: 2
```

This change provides the solution to three ARs: 43549 (Need a method in SourceFileManager for deploying with no changes file), 43318 (Don't log parcels for runtime image), and 43372 (Can't create classes in plugin). The latter two were rejected in favor of the first, and all three resolved together.

Virtual Machine

Headless UNIX Engines

Most of the Unix platforms now provide a headless engine. These engines exclude the GUI and window management primitives, dynamically loading them as required from a shared library.

The all-in-one, “headful” engines are still provided.

The platforms supporting headless engines are:

- Compaq True64 Unix
- Linux x86
- Linux PowerPC
- Linux SPARC
- Silicon Gaphics IRIX
- Sun Solaris.

Each headless engine automatically searches for an associated GUI shared library when a GUI primitive is first invoked. Engines look for a shared library of the same name as the engine with "gui.so" appended. For example, the **`vwlinux86`** engine is headless, and will search for **`linux86gui.so`** if a GUI primitive is invoked. Headful engines have “gui” appended to their name, to the corresponding headful engine is **`vwlinux86gui`**.

These engines support two new command-line switches:

-gui:
Load default GUI subsystem shared library on startup.

-guilib guilib.so:
Load specified GUI subsystem shared library on startup.

The headless GUI engines currently do not support Input Management when used with their GUI library. This is a bug we will fix in a subsequent release. For now I18N users who want to use headful applications should use the all-in-one engines.

New Platforms

Three new platforms have been provided this release:

- the MacOS X platform, which previously was in beta, but is now fully supported.
- linuxSPARC for Linux on SPARC processors.
- linuxPPC for Linux on PowerPC/RS6000 processors.

The two new Linux platforms are unsupported. But they are, as far as we know, fully functional, and at least the LinuxPPC platform is in daily use by some members of VisualWorks engineering.

bin/ Directory Organization

The **`bin/`** directory has been reorganized in an attempt to make things a little less confusing. Extra engines for assert checking and debugging are in **`extras/`** subdirectories of each **`bin/`** directory. Further, to save space on the CD, their contents has been compressed. We apologies for this inconvenience and hope you'll still unpack these engines if you need to diagnose potential engine problems.

New Windows command switch

The Windows platform now provides an extra command-line switch, **`-walltime`**, which makes the engine derive its 64-bit microsecond clock from SystemTimeAsFileTime. This is more accurate over long periods of time, but has much lower resolution and is much slower to compute than the default. The default scheme, which is derived from the performance counter and synchronised with SystemTimeAsFileTime at startup, may drift from the system's time by a few seconds every day so. For many applications the higher resolution and performance is preferrable.

Large Cursor support

All platforms except MacOS 9 now support 32x32 cursors, although we do not as yet take advantage of this in the image, precisely because MacOS 9 only supports 16x16 cursors.

GUI Development

Multiproc UI

Prior to the introduction of the multiprocess UI, VisualWorks has had a number of longstanding problems, including:

- A dialog window for one window blocks all windows in the image. This restricted a developer's ability to use a single image to implement multiple applications.
- When an application waits on a semaphore all windows are blocked. This made it more difficult for a developer to use a single image to implement multiple applications that communicate externally.
- The designs of InputState, ControlManager, and the window sensor still had polling remnants that are actively used.
- Overly complex technique for background process to update a window (ForkedUI).

With the introduction of multiprocess UI, we have made the following Improvements:

- Dialog windows block only windows managed by same window manager.
- An application waiting on a semaphore does not block windows using a different window manager.
- Applications can run with different priorities.
- Design is more fully event driven. Elimination all calls to the polling loop in ControlManager.
- Improves non-user interface (background) process debugging.

Highlights

- The WindowManager class now does the job the ControlManager class used to do.
- Each WindowManager has its own event queue, which manages itself and the windows under its control. In the ControlManager, each window had its own event queue, but the global ControlManager (ScheduledControllers) centrally managed kicking each window into action.
- Each time you open a window, the window may be set to be managed by its parent window's WindowManager, or it can be set to be managed by its own new WindowManager.
- When a UI process is being debugged, no events are processed by the associated window manager. This also means that all the windows managed by that window manager are frozen. While this can be disconcerting for IDE tools like browsers this guarantees that an application being debugged does not inadvertently change state due to an event.
- Ctrl-\ opens a Process Manager, where all user and UI processes (and optionally all system processes), are listed. From this list of processes, you have a choice of many different actions, including debugging a process, inspecting a process, dumping the stack on a process, terminating a process, etc.
- All of the WindowManager code is tied into the new PDP Debugger.

Backward Compatibility

The ControlManager class and its associated global, ScheduledControllers, still exist in the system, although they are now obsolete and deprecated.

All common code that is written to talk to the ScheduledControllers global will still work, because ControlManager has been rewritten to forward these methods to a WindowManager.

While your old code will still work, there are some circumstances where the resulting behavior may be unexpected. This is because the forwarding code in ControlManager sometimes has to guess which WindowManager to send the message to.

Therefore, wherever you have code that reads:

```
ScheduledControllers checkForEvents
```

we suggest that you rewrite it to talk to the current WindowManager:

```
myWindow windowManager checkForEvents
```

Deferred and Background UI Actions

The old ForkedUI goodie code is now obsolete, and is totally incompatible with the new Multiprocess UI.

Multiprocess UI has added four new messages for blocks allowing you to perform the block deferred with or without waiting for the answer.

The uiEvent and uiEventFor: messages put the receiver block onto the event queue and do not wait for an answer. The difference between the two methods is that uiEvent assumes that the method should be performed by the currently active window, at which it can only do a best guess at finding, while uiEventFor: takes a specific window as the argumetn. uiEventFor: is the preferred method.

The uiEventNow and uiEventNowFor: messages put the receiver block onto the event queue and then, without blocking the window, separately waits for the result of the block being evaluated. Again, uiEventNow will make a best guess at which window's manager is to execute the block, while uiEventNowFor:, the preferred version, allows you to specify the window.

This new behavior is all written around a new event: DeferrableAction. DeferrableAction is designed to encapsulate a message send, that can be put on a window manager's event queue to be executed. DeferrableAction has a rich API that allows you to create complex actions that can be executed in the UI process.

The class side has two creation methods that allow you to easily create new instances of DeferrableAction. These methods have the same message signatures as the Trigger Event system: send:to: and send:to:with:.

The send: parameter is a symbol for the selector that you wish send, and the to: parameter is the receiver for that selector. The with: version allows you to specify a collection of parameters for keyword selectors.

An instance of DeferrableAction allows you to specify the window (by sending it a window: message, inherited from Event) for which you want to have the action taken.

An instance of DeferrableAction can be invoked by sending it either an activate or waitForResult message. activate simply puts the action on the queue of the window, while waitForResult does the same but waits for the result of the message it encapsulates.

If a window is not specified when the action is invoked, it makes a best guess attempt to find the current window, like its uiEvent cousins.

DeferrableAction has four class side methods that allow you to create AND invoke the action. send:to:for: and send:to:with:for: allow you to create and specify the target window, and invoke the action without waiting for a result. sendNow:to:for: and sendNow:to:with:for: do the same, except they wait for the result of sending the message.

Additional GUI Behavior

Events now have an extra instance variable: window. This allows an event to dispatch itself (also note the new dispatch method in Event).

There is a new method on the class side of ApplicationModel, called raiseSingleInstance. This provides a simpler way to ensure that only one instance of an ApplicationModel is active at one time. An example is in UISettings class side open method, which you can use on your own application:

```
open
    self raiseSingleInstance isNil ifTrue: [^super open]
```

There is now an accessor on Process, called windowManager, which you can use to access the windowManager associated with a process. Of course, the value will be nil for non UI processes.

There is a new test method in Window and its subclasses, called isInvalid. This is used by the WindowManager to clear out dead/closed windows that otherwise might be zombies, and can be used by you to find out if a window has died.

Usage changes

- Any code that needs to maintain control but still process events should do the following:

  ```
  myWindow windowManager processNextEvent
  ```

- Non-UI processes should communicate to UI processes using `DeferrableAction`.
- Communicating applications managed by different window managers must use interprocess communication techniques, such as `DeferrableAction`. This includes update notifications between the various source code management tools in the IDE.
- Windows now need to schedule themselves differently. Where the system sent a `map` message, it now sends a `scheduleWindow` message before it sends `map`. This is important. Without the `scheduleWindow` in the new system, no manager will be managing it.
- Places that used to read

  ```
  ScheduledControllers scheduledControllers do:
  ```

  have been rewritten to read

  ```
  Screen default allScheduledControllersDo:
  ```

  or, if they really just wanted to work on the windows,

  ```
  Screen default allScheduledWindowsDo:
  ```

  Your code should do the same.
- Places that used to read

  ```
  ScheduledControllers activeController
  ```

  now call

  ```
  Window activeController
  ```

  Your code should do the same.
- Places that used to read

  ```
  ScheduledControllers checkForEvents
  ```

  should now read

  ```
  windowManager checkForEvents
  ```

- Places that used to read

  ```
  ScheduledControllers unschedule: self
  ```

  should now read

```
view close
view unscheduleWindow
```

- Places that used to read:

  ```
  controller := (ScheduledControllers activeController).
  window := c view.
  ```

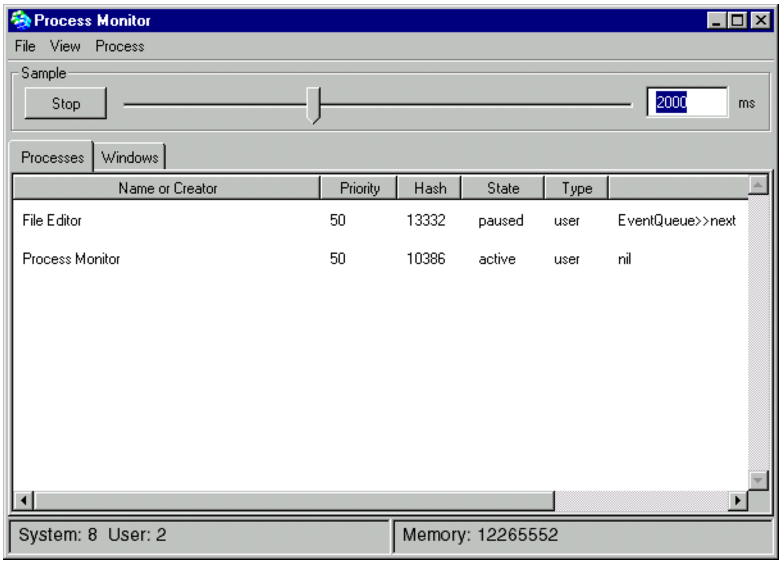
  should now read

  ```
  Window := Window currentWindow
  ```

Debugging in a the MultiprocUI environment

The debugging environment has been enhanced for multi-process UI.

Whenever you press Ctrl-\, all UI processes are paused. You can see that in the Process Monitor for the state column. You can select a process from the process monitor, the select an action from the Process menu.



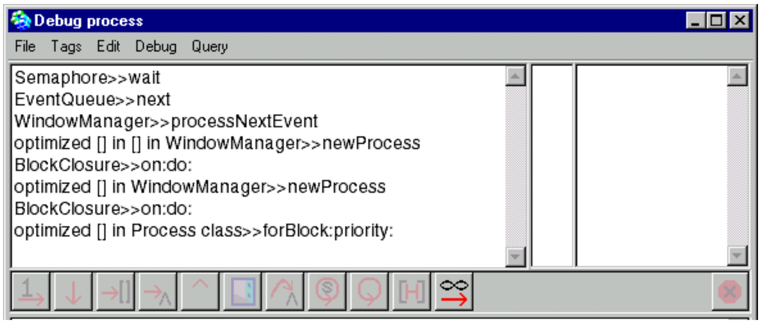
Name or Creator	Priority	Hash	State	Type	
File Editor	50	13332	paused	user	EventQueue>>next
Process Monitor	50	10386	active	user	nil

System: 8 User: 2

Memory: 12265552

The processes there are named by way of either the label or class name of one of the windows under control of that process (more on that in a moment). Thus, you might see one named Workspace, and another named "VisualWorks ...<yourpath>".

Now let’s do some exploring. Select the **File Editor** process and select the debug menu item from the **Process** menu. The stack you see in the debugger looks like:



```
Semaphore>>wait
EventQueue>>next
WindowManager>>processNextEvent
optimized [] in [] in WindowManager>>newProcess
BlockClosure>>on:do:
optimized [] in WindowManager>>newProcess
BlockClosure>>on:do:
optimized [] in Process class>>forBlock:priority:
```

Notice the WindowManager entry, rather than a window. This is due to replacing the class ControlManager with the class WindowManager. As explained above, all of the important messages you typically send to the ScheduledControllers global are forwarded to a WindowManager.

Before we go on, let's discuss how things have changed. The old ControlManager, of which there "should" have been only one instance, in the ScheduledControllers global, managed controllers. It was the single place, and had a single process for serving events to various controllers. It did this through its activeControllerLoop method.

Here is how the old way events were processed: A platform event would come into the image from the VM, and the VM would signal the image that there is an event waiting. The InputState run method would get that signal, and then send that raw event array to its process: method, which would then convert that into an Event subclass, and dispatch that event to the target window's sensor.

After that, each window, having it's own event queue in its sensor, would add that event to it's event queue, and then in effect kick the ControlManager (though the ScheduledControllers global) to process the event on its behest.

Finally, the window is told to process the event, and it either sends the event to itself or its dispatcher (which can send events to the sub-components and other things).

In the multiprocess UI world, instead of each window having its own local active event queue, each WindowManager has a single event queue for all windows it manages. A WindowManager can have just one window, or it can have a whole bunch. This is controlled by the process environment's new WindowManagerUsagePolicy. You can set an individual window to have one of these policies:

- MakeNewWindowManagerUsagePolicy
- UseParentWindowManagerUsagePolicy

If the former is in place for a window's process, then every new window opened will have its own WindowManager created for it. If the latter is in place, then every new window will share the window manager with the window's process that caused the new window to open.

By default the UseParentWindowMangerUsagePolicy is in place.

For example, the VisualLauncher has MakeNewWindowManagerUsagePolicy set. Thus, when you open the File Browser from the Launcher, and then do a **Ctrl-\** to open the process monitor, you'll notice that there is a new process for the File Browser.

Another example: Open up a Browser, which creates a new process. In that browser, open an implementors or senders of something (#yourself maybe). Now, if you press **Ctrl-\**, and debug on the Browser (it is named after its first open window, so it should be the title of the browser). Go down the stack to the **WindowManager>>processNextEvent**, and inspect the WindowManager instance. The windows instance variable will have two windows in there. One is the browser, and the other is the second window you opened.

To change the new process behavior for a window, do something like this. In your postOpenWith: method, add the following:

```
builder window windowManager activeControllerProcess
    environmentAt: #WindowManagerUsagePolicy
    put: MakeNewWindowManagerUsagePolicy new.
```

As before, a window has its own local sensor, an event queue, but it no longer collects events! Instead of putting an event into the local queue and telling the ControlManager to kick it into play, the event is sent to the WindowManager for that window. From there, the WindowManager passes along events to the window.

In the single-process world, InputState had a pollForActivity method that was called from activeControllerLoop in ControlManager. What this attempted to do was allow each window a chance to process events. Because it never worked quite right, we had to come up with the ForkedUI goodie. But, it had serious limitations. Additionally, because VW itself sends

Smalltalk events to itself from time to time, the whole thing had a heartbeat loop of 10hz (100hz if ForkedUI was loaded) to make sure that those events would get served.

In the new world, each WindowManager simply waits for something to show up in its queue. No more pollForActivity calls, and thus, no more activeControllerLoop horror. The result is a more responsive UI.

Tools development in the MultiProcUI environment

Multiprocess UI causes some complication for tools developers, particularly when it would be possible and problematic for two processes to be modifying the system at the same time. For example, connection problems occured in if trying to browse published items while loading from or publishing to a Store repository. Similar issues arise for parcel and file-in operations.

In a number of cases in the IDE, the solution has been to wrap some operations in Notice show:while: blocks, to prevent other windows from grabbing control while these operations are running. Notice, its signals, and its instance creation API, have been moved out of Store and into the UI namespace and Interface-Support category. The creation methods have been modified to grab mouse input. NoticeSystemController has been created to make sure mouse input isn’t ungrabbed when clicking on another window.

When developing tools that similarly affect the environment, you may need to wrap some operations in Notice show:while: blocks. Browse references to Notice for examples in the system tools.

By default, all user applications run in a single UI process, unless they set the policy of a window to be MakeNewWindowManagerUsagePolicy. Therefore, you will only have to worry about conflicts in two cases:

- You fork processes that update the UI.
- You use the MakeNewWindowManagerUsagePolicy in your windows.

New mouse events

Two new mouse events have been added to the system: MouseEnterEvent and MouseExitEvent.

MouseEnterEvent fires whenever you pass the mouse into the visible area of a VisualWorks window. Note, this only happens if you are NOT dragging the mouse with a mouse button pressed.

MouseExitEvent fires whenever you pass the mouse out of the visible area of a VisualWorks window. Again, this only happens if you are NOT dragging the mouse with a mouse button pressed.

The messages that your controller or window can override to trap these events are mouseEnter: and mouseExit: respectively.

New Look Policies

- Windows XP look is provided in the WinXPLookPolicy class.
- MacOSX Aqua look is provided in the MacOSXLookPolicy class. Keyboard support is also improved.

Widget enhancements

Dataset

The **Basics** page of the Dataset properties now allows you to set a size for the row, in pixels. Accessor methods in DataSetSpec, rowSize and rowSize:, allow programmatic access to the row size.

TreeView

ARs: 43698, 44893

You can now optimize building a tree of items by defining a hasChildren block, and can use the many new methods to access collections of tree node parents or children.

Input field

AR 45238.

Input field widgets now vertically center their entries.

New widget layout option

In addition to bounded and unbounded layout options, a new fixed-size widget layout, **Origin + Width and Height**, was added in VW7, but not noted or documented.

This new layout allows you to specify the origin (top left) of a widget in the usual proportion and offset. The size of the widget is then set by specifying its width and height in pixels.

The new option is available on the **Position** page of the GUI Painter Tool.

Menu enhancements

ARs: 44957, 45000, 45011, 45014, 45175, 45178.

- Tool bar buttons will now appear depressed if their indication is on.
- Menu and tool bars now update their enablement, visibility, or indication based on a change to their defining MenuItem.
- Menu and tool bars support exclusion or one-of-n selection groups.
- New pragmas add shortcut keys, enablement or indication accessors, and help text to menus.
- New menu and tool bar examples demonstrate the features above.

UIPainter

- A widget interior to a composite or group of widgets may now be selected in the canvas by holding down the **Alt** key. Additional interior widgets may be selected by holding down the **Shift** key as well (AR 42974).
- Widget layout in the GUI Painter Tool for composites and multiple selections has been corrected (AR 45544, 45186, 45180, 45026).

Retract ValueModel support for TriggerEvents

AR: 45370

The modification to Object>>changed:with: in VW7 to participate with the TriggerEvent #when:send: mechanism has been retracted due to performance issues.

Known Limitations

Sawfish and MultiProcUI

We have seen a problem with a window regaining focus, when running under the Sawfish window manager on Linux. There is no known work-around, other than selecting the window.

Tools development

Tools developers need to be aware that it is a bad idea for two tools to be modifying the system at the same time, but is possible with MultiProc UI. We have modified our tools so that, for example, other windows cannot grab control while code is loading. You may need to do similarly. Refer to “Tools development in the MultiProcUI environment” below, under Tools, for more information.

System unresponsive after closing a window

If an application opens a dialog, and the application's window is closed before the process using the dialog is completed, the system freezes to mouse and keyboard input. This occurs, for example, if you are unloading a parcel using the Parcel Manager, and close the Parcel Manager window before the parcel is fully unloaded. Or, as simple an application as a windows with an action button with the action:

```
Dialog warn: 'Hello'.
(Delay forSeconds: 5) wait.
Dialog warn: 'Hello'.
```

In this last case, closing the application window before the second dialog opens will freeze the system.

Opening the Debugger with **Ctrl+Y** and then closing the Debugger unfreezes the system.

Headless support

There were 3 obscure file names being used in HeadlessImage: **`hlst_dbg`** (image), **`hlstrc.st`** (startup), and **`hlst.tr`** (transcript).

The text string for these names has been extracted to instance methods, and the file names have been changed as follows:

```
defaultDebugImageName
    ^ 'headless-debug'.

defaultStartupFilename
    ^ 'headless-startup.st'

defaultTranscriptFilename
    ^ 'headless-transcript.log'
```

The change in the default name for the startup file may impact users who have existing startup files and were simply using the default file name for that file.

Tools

Keyboard bindings

We have redefined the default keyboard bindings to make them more consistent and, in some casses, “standard.” The following is a summary of the changes.

Key sequence	Current meaning	Old meaning
Ctrl+A	Select All in all text views	FindAgain
Ctrl+B	Debug It in all code views.	
Ctrl+D	Do It in code views; insert Date in non-code views	Insert date
Ctrl+Shift+D	Insert Date	
Ctrl+E	Explain in code views.	
Ctrl+F	Insert ifFalse: in code views; Find in non-code views	
Ctrl+G	Insert := in code views; Find Again in non-code views	
Ctrl+J	Select text just typed	F1
Ctrl+K	Compose characters	
Ctrl+L	Find in all views	
Ctrl+Shift+L	Find Again	
Ctrl+P	Print It	
Ctrl+Q	Inspect It	Compose characters
Ctrl+R	Replace	
Ctrl+Shift+R	Replace Again	Browse references, in System Bowser
Ctrl+S	Accept in all text views	Find
Ctrl+T	Insert ifTrue: in code views	
Ctrl+Shift+V	Paste from list	Inspect variable, in Visual Launcher
Ctrl+X	Cut	

Key sequence	Current meaning	Old meaning
Ctrl+Z	Undo	
F3	Find Again	
Ctrl+F5	Browse references to instance variable, in Browser and Launcher	
Alt+F5	Inspect variable, in Browser and Launcher	
Ctrl+F6	Browse senders, in Browser and Launcher	
Ctrl+F7	Browse implementors, in Browser and Launcher	

The traditional binding for Ctrl+J was F1, but F1 is now over-shadowed by Help menu shortcut in most windows.

Note that Ctrl+Shift+R in the System Browser (Refactoring Browser), to browse references to a variable, has been changed to Ctrl+F5, and Ctrl+Shift+V in the Launcher, to inspect a variable, has been changed to Alt+F5, to allow those key combinations to be made consistent with the new scheme.

Please Wait…

The introduction of MultiProc UI brought along some complications in the tools. One that will be evident the first time you load a parcel or package, or file in some code is a progress dialog labeled “Please Wait.” Unfortunately, there is no progress bar or other indication of degree of progress. This will be addressed in the next release.

Tools development in the MultiProcUI environment

Multiprocess UI causes some complication for tools developers, particularly when it would be possible and problematic for two processes to be modifying the system at the same time. For example, connection problems occured if trying to browse published items while loading from or publishing to a Store repository. Similar issues arise for parcel and file-in operations.

In a number of cases in the IDE, the solution has been to wrap some operations in `Notice show:while:` blocks, to prevent other windows from grabbing control while these operations are running. `Notice`, its signals, and its instance creation API, have been moved out of Store and into the

UI namespace and Interface-Support category. The creation methods have been modified to grab mouse input. NoticeSystemController has been created to make sure mouse input isn't ungrabbed when clicking on another window.

When developing tools that similarly affect the environment, you may need to wrap some operations in Notice show:while: blocks. Browse references to Notice for examples in the system tools.

By default, all user applications run in a single UI process, unless they set the policy of a window to be MakeNewWindowManagerUsagePolicy. Therefore, you will only have to worry about conflicts in two cases:

- You fork processes that update the UI.
- You use the MakeNewWindowManagerUsagePolicy in your windows.

Professional Debug Package (PDP)

The Professional Debug Package, by Crafted Smalltalk, is now the default system debugger tool. It provides several much needed enhancements, notably the ability to set break points and watch points, and a view of the viriable stack.

The PDP is unloadable, by unloading the parcels that define it. The resulting system returns to the old debugger.

Refer to chapter 9, "Debugging Techniques," in the *Application Developer's Guide* for a full description and guidelines for using these new facilities.

New Settings framework

The Settings Tool has been replaced with a new tool and framework.

The Settings Manager window consists of the three main parts: a tree of settings pages on the left, with the currently selected page displayed on the right.

The framework simplifies and expands the capabilities of adding new pages to the Settings tool. This is useful for add-ons and applications that need special settings.

The *Application Developer's Guide* has not yet been updated for this new framework, so that part of chapter 14 is out of date. This will be corrected in a future release.

For a description of the framework and how to add pages, refer to this page on the Cincom Smalltalk Wiki:

> http://www.cincomsmalltalk.com:8080/CincomSmalltalkWiki/DOWNLOAD/vb/visualworks_settings_framework.htm

Advanced Tools

Profiling

The ATProfiling parcel in **`advanced/`** has been replaced by two new parcels, ATProfilingCore and ATProfilingUI. These new parcels include the functionality previously shipped as ATProfilingEnhancements in the **`goodies/`** directory. They also segregate profiling functionality into “core” and “user interface” components. This is a prelude to shipping “attach-and-profile” and “distributed process” profiling in some later release. Only the “core” components needs to be loaded into a remotely profiled, and possibly headless, image.

The new parcels support single-process as well as multi-process profiling. Users are urged to remember that all the profilers rely upon a statistical sampling heuristic to estimate, rather than on instrumentation to directly measure, the resources consumed by a process. Multiprocess profilers distribute the probes used to estimate resource consumption over several processes, rather than one, and the distribution may be uneven. Also, running multiprocess profilers does cause garbage collection and other maintenance processes to run more frequently than otherwise. These facts should be kept firmly in view when setting up multiprocess profiling runs and when estimating the reliability of their results. Within these limitations, multiprocess profilers have proven useful in tuning web applications involving many hundreds of processes.

In these new parcels, the pre-existing public profiling API has been preserved. The primitive lists have been revised. The Profile Outline Browser is no longer limited to a maximum of three reports. The profiling user interfaces, by default, now open showing new advisory text and multiple examples. To make these user interfaces show code templates instead, evaluate

```
Profiler showTemplates: true
```

The old profiling parcel, ATProfiling is still shipped in the **`obsolete/`** directory, because other obsolete and preview parcels list it as a prerequisite. If appropriate, users should explicitly update the prerequisites of their parcels to list the new profiling parcels rather than the old one. Note that the old ATProfiling parcel is incompatible with the new set of parcels, and vice versa. They should not both be loaded into the same image.

Note also that the *Advanced Tools User's Guide* has not been updated for this change in time for release.

WebService enhancements

Separated from NetClients

Web Service support is now in its own parcel, and its own name space, and its own document (*Web Service Developer's Guide*).

WebService Tool

A WSDL tool has been added to generate client, service, and server classes from a schema, and for generating a schema from classes. Refer to the *Web Service Developer's Guide* for instructions.

The schema generating tool is subject to substantial enhancement in future releases. In particular, an editor to provide pragma definitions is intended.

Net Clients

Renamed #beCurrentDirectory:

FTPClient method beCurrentDirectory: has been renamed to setCurrentDirectory:. The old message has been deprecated.

Moved #defaultPortNumber

This message, implemented in several client classes, has been moved to the class side.

User Agent support

Accessor methods for the HTTP user agent field. Some servers refuse access without this information. The accessor methods are:

userAgent
Returns the user agent field value.

userAgent: *aString*
Sets the user agent field to *aString*.

Security

Public key algorithms were the focus of this release cycle. This allows us to further extend the suite of supported algorithms in SSL.

Public key algorithm APIs

The public key ciphers were refactored to make their APIs simpler and more uniform. Generally one is expected to create an instance of the algorithm, set any required parameters, set the key and then invoke a signing or encryption operation. An instance of the algorithm can be reused to perform multiple operations.

Encrypting a ByteArray with RSA goes like this:

```
rsa := RSA new.
rsa publicKey: anRSAPublicKey.
anEncryptedByteArray := rsa encrypt: aByteArray
```

Note that the size of the ByteArray cannot exceed the byte size of the modulus 'n' of the private key. If you need to encrypt more data, use symmetric cipher, which is a lot faster. Here's how to decrypt the encrypted ByteArray:

```
rsa privateKey: anRSAPrivateKey.
aByteArray := rsa decrypt: anEncryptedByteArray
```

Signing with RSA algorithm is similar but requires an additional step specifying the hash function to generate the digest of the data to be signed (#useMD5, #useSHA or #hashAlgorithm:). If digest is not specified before invoking a signing operation, the data is expected to be already digested. This is used in SSL for example.

```
rsa := RSA new.
rsa useMD5.
rsa privateKey: anRSAPrivateKey.
aSignatureByteArray := rsa sign: aByteArray
```

Here's how to verify a signature.

```
rsa publicKey: anRSAPublicKey.
rsa verify: aSignatureByteArray of: aByteArray
```

The result of the #verify:of: message is a Boolean indicating whether the signature is valid.

Signing with DSA algorithm is almost the same, except that there is no hash function setting, because DSA must use SHA. To sign, do:

```
dsa := DSA new.
dsa privateKey: aDSAPrivateKey.
aDSASignature := dsa sign: aByteArray.
```

and to verify, do:

```
dsa publicKey: aDSAPublicKey
dsa verify: aDSASignature of: aByteArray
```

Key generators, random generators & primality tests

In contrast with the symmetric ciphers, for which the keys can be fairly arbitrary byte arrays, the keys used by public key ciphers are more complex, generally involving large integers with special properties. Generation of such key pairs is a computationally heavy process employing random generation of large integers, and testing large integers for primality.

The selection and seeding of high quality random generators and robust primality tests has a significant and direct impact on the security of generated keys. For this reason, these algorithms are reified into objects, to enhance pluggability and to allow for easier customization. The previously released CrpRandom generator has been replaced by a refactored version of it, now called DSSRandom to reflect the fact that it is derived from the DSS standard. The new version is more in line with the Random hierarchy, by responding to the standard #seed: and #next messages, and the API allows a bit more flexibility (refer to the class and method comments for the details). The primality tests have been reified into a PrimalityTest hierarchy, to facilitate instance reuse and customization.

Default initialization and seeding of most of the algorithms should suffice for an average, personal use. However, the security of a solution can be increased significantly with careful management, reuse, and sharing of the random generator instances, exploiting the quality and extent of the generators themselves rather than just the quality of the seeding algorithm.

Note well that truly random seeding of the random generators is a critical requirement for any serious application. Relying on any kind of "computed" seeding (including the ones that our framework uses as a default) is generally considered to be a serious security risk.

RSAKeyGenerator has one required parameter, bitLength, which determines the size of the keys to be generated. In accordance with the motivations outlined above, there are variants of the instance creation methods allowing initializing the generator with pre-existing instances of random generator and primality test. Here's a code sample creating a generator for 1024 bit keys:

```
kg := RSAKeyGenerator size: 1024.
```

After this the key generation can be triggered using the key accessors #privateKey and #publicKey. To generate a new set of keys with the same generator instance, send #flush to the generator to flush the generated keys and parameters.

As was mentioned earlier, generation is a computationally demanding process and the time it takes is proportional to the size of the keys being generated. To facilitate user feedback during generation, the generator signals various object events through the various stages of the process. As usual, the class method #eventsTriggered lists the kinds of events that are signaled.

DSAKeyGenerator is used in very much the same way and most of what was said above applies. The key size parameter is referred to as “l”, to follow the terminology of the standard:

```
kg := DSAKeyGenerator l: 512.
private := kg privateKey.
public := kg publicKey.
kg flush.
anotherPrivate := kg privateKey.
anotherPublic := kg publicKey.
```

Diffie-Hellman key exchange

Our implementation follows the description in RFC2631, though only the part about generation of the shared secret, and not the rest of the key agreement specified there. DH is a different kind of public key algorithm, in that it does not encrypt or sign, but rather allows remote parties to establish a shared secret value over an unprotected channel.

Before invoking the algorithm the parties need to agree on two large primes, *p* and *g*. These do not need to be secret, and often are precomputed. In this case an instance of the algorithm can be created as follows:

```
DH p: p g: g
```

These parameters can also be generated on the fly, with one party generating them and the other party accepting them as such. In this case, the first party creates the algorithm as

```
dh1 := DH new
```

and the second party will use the parameters generated by the first one

```
dh2 := DH p: dh1 p g: dh1 g
```

The secret value is established in two phases. First both parties generate their own private/public key pairs, usually referred to as *x* and *y*:

```
y1 := dh1 computePublicKey
y2 := dh2 computePublicKey
```

The private key *x* can usually stay hidden inside the algorithm instance.

After this the parties exchange the public keys (usually over an unprotected communication channel), and continue with second phase in which they compute the shared secret value using the other party's public key:

```
s1 := dh1 computeSharedSecretUsing: y2.
s2 := dh2 computeSharedSecretUsing: y1.
```

The values s1 and s2 are the same and it is computationally infeasible to produce them without knowing one of the private keys.

Note that the size of of the public key *y* and the size of the shared secret is the size of *p*, which should be at least 512 bits, and the size of the private key *x* is up to the size of *q*, which should be at least 160 bits. So, the key lengths can be tuned by using appropriately sized *p* and *q* parameters.

Parameters *p* and *q* can be generated by an instance of `DHParameterGenerator`. Similar to the key generators mentioned earlier, it can be configured with pre-existing instances of random generator and primality test. Required parameters are bit-lenghts of *q* and *p* referred to as *m* and *l*, respectively (following the RFC 2631 conventions).

```
pg := DHParameterGenerator m: m l: l.
p := pg generateP.
q := pg generateQ.
```

To generate a new set of parameters with the same generator, it has to be flushed using the message `#flush`.

Again, the events specified by the `#eventsTriggered` method allow to monitor the progress of generation.

Diffie-Hellman cipher suites in SSL

Adding support for DH algorithm almost doubles the number of SSL cipher suites that we can handle. The newly added cipher suites are:

```
SSL_DHE_RSA_WITH_DES_CBC_SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA
SSL_DH_anon_WITH_RC4_128_MD5
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
```

Because of the nature of Diffie-Hellman key exchange, there are several options for use of temporary keys for DHE cipher suites (Refer to the comment of the DH class for the details on the meaning of the various parameters mentioned below):

A. Both keys can be precomputed and registered with the context using #dhKeyPair: message. In this case the first element of the pair is a 3-element array containing the group parameters *p* and *g*, and the public key *y*, in that order. The second element of the pair is a standalone private value *x*.

B. It is also possible to register with the context just the group parameters and a fresh pair of the private/public values *x*/*y* will be generated for each handshake. In this case the parameters are registered using #dhParameters: message as an (Array with: p with: g).

C. If nothing is registered with the context an SSLNoDHParameters exception is signaled. However if the exception is resumed, SSL will generate a whole DH parameter suite on the fly and proceed the handshake with those. Note however that generation of the parameter p is time consuming and the delay caused by that may not be acceptable. If it is not, then we recommend pregenerating the *p* and *g* parameters and use variant B above.

AES

AES is a new NIST standard (FIPS 197) that is aimed at replacing the aging DES standard (FIPS 46). More information can be found at http://csrc.nist.gov/encryption/aes.

It is a block cipher and therefore it complies with the BlockCipher API, i.e. an instance of AES can be created with the #key: instance creation method (or with #new followed by #setKey:). The argument is the secret key which is expected to be a ByteArray of size 16, 24 or 32. An AES instance can be used to both encrypt and decrypt 16 byte blocks arbitrarily.

Database

Exception handling updated

The EXDI exception handling architecture has changed, by replacing the Signal mechanism with class-based Exception subclasses. Existing code is expected to function as before, so no conversion is expected to be required. We added resumable and non-resumable exceptions descending from the common Exception and Error classes. This architecture is consistent with the previous scheme, but more up to date.

CLOB/BLOB support

LOB (Large Object) support has been added to the OracleEXDI. Both CLOB (Character LOB) and BLOB (Binary LOB) data is supported.

We do not differentiate the Lob columns from longs and others when doing binding. Accordingly, any limitations of different Oracle versions on binding LOBs will apply.

When retrieving LOBs, you can choose whether to get values or LOB proxies. Getting values limits the size of the values to 4000 bytes. Getting proxies returns a LOB proxy, which contains the LOB locator and necessary methods to do LOB writes and reads. Using LOB proxies is the recommended way to deal with large LOBs.

The following sample demonstrates binding:

```
| aConnection aSession clob blob clobLength blobLength |
    aConnection := OracleConnection new.
    aConnection username: 'name';
        password: 'passw';
        environment: 'env'.
    aConnection connect.
    aConnection begin.
    aSession := aConnection getSession.
    aSession prepare:  'INSERT INTO TestLob (a, b, c) VALUES ( ?, ?, ?)'.
    clobLength := 1048576.        "1M"
    blobLength := 1048576.        "1M"
    clob := String new: clobLength withAll: $a.
    blob := ByteArray new: blobLength withAll: 1.
    aSession bindInput: (Array with: clob with: blob with: 1).
    aSession execute.
    aSession answer.
    aConnection commit.
```

The following sample demonstrates Lob writing:

```
| aConnection aSession clobProxy blobProxy clob blob clobLength
    blobLength ansStrm res |

    aConnection := OracleConnection new.
    aConnection username: 'name';
        password: 'passw';
        environment: 'env'.
    aConnection connect.
    aConnection begin.
    aSession := aConnection getSession.
    aSession prepare:  'SELECT a, b FROM TestLob WHERE c = 1
        FOR UPDATE'.
    aSession answerLobAsProxy.
    aSession execute.
    ansStrm := aSession answer.
    res := ansStrm upToEnd.
    clobLength := 1048576.
    blobLength := 1048576.
    clob := String new: clobLength withAll: $e.
    blob := ByteArray new: blobLength withAll: 0.
    clobProxy := (res at: 1) at: 1.
    clobProxy writeFrom: 1 with: clob asByteArray.
    blobProxy := (res at: 1) at: 2.
    blobProxy writeFrom: 1 with: blob.
    aConnection commit.
```

The following sample extends the above examples specifically for Oracle_8 users, showing how to avoid restrictions against multiple LONGs on a single INSERT, insert empty Lobs, and update values later.

The comments explain the intentions:

```
"CREATE TABLE TestLob (A CLOB, B BLOB, C INTEGER)"

    | aConnection aSession |
    aConnection := OracleConnection new.
    aConnection username: 'name';
    password: 'passw';
    environment: 'env'.
    aConnection connect.
    aConnection begin.
    aSession := aConnection getSession.
    aSession prepare: 'INSERT INTO TestLob (a, b, c)
        VALUES ( EMPTY_CLOB(), EMPTY_BLOB(), ?)'.
    aSession bindInput: (Array with: 1).
    aSession execute.
    aSession answer.
    aConnection commit.
```

Oracle element size corrected

In the old OCI 7 interface, the elementSize parameter when binding a parameter need to be -1 for buffers size > 65k, because the elementSize parameter was a short. In the new OCI 8 this is no longer needed and, in fact, Oracle will complain that it doesn't accept -1 anymore. The OracleBuffer>>boundedElementSize method has been updated for this change.

Application Server

New ISAPI Gateway

This release provides a new ISAPI Gateway, including both a release and a debug version of the DLL. In addition, we are going back to the hostmap form of configuration, which means that we have eliminated the **`isapi.ini`** file.

The DLLs are in:

```
$(VISUALWORKS)\waveserver\waverelays\isapi\nt
    isapi2vw.dll
    isapi2vw-debug.dll
```

We are also releasing the source code for this module, in the form of a Visual C++ project, in

```
$(VISUALWORKS)\waveserver\waverelays\isapi\source
```

See the readme.txt file in this directory for additional information.

IIS Virtual Directories

This release of the VisualWorks Application Server enhances support for IIS virtual directories. It is now possible to use a virtual directory and hide the **`/scripts/host@port.dll/`** that formerly appeared in the URL.

Named virtual directories may now be specified in the Server Console. For details, refer to chapter 5 of the *Web Server Configuration Guide*.

Opentalk

Opentalk STST Marshaling

In the Opentalk 7.1 release, the version number of the Opentalk STST protocol changes from 1.0 to 2.0. This change, in the major version integer, indicates that Opentalk STST backward compatibility is broken in 7.1. That break is a consequence of a revision of the Opentalk STST type tagging scheme.

The number of class-specific type tags used in STST marshaling has been increased to allow optimized marshaling of several commonly-used classes. Several blocks of type tags are also reserved for customer use, so that customers may implement and test application- and class-specific marshaling optimizations of their own. We do not assume any responsibility for conflicts in the use of the tag values within those reserved blocks.

The new type tagging scheme also supports two new pass modes, described below.

Pass-By-Name

Classes, NameSpaces, NameSpacesOfClasses, BindingReferences, LiteralBindingReferences, and Signals may be passed by name using, for example,

```
#{Object.DependentsFields} asPassedByName.
```

On receipt, a passed-by-name object will resolve to either (a) the local object that bears the passed name, or (b) an exception if there is no such object. The default pass mode for the six classes mentioned remains `#reference`. This is because use of pass-by-name should be explicit; pass-by-name is reliable and straightforward to debug only when identity (or scrupulously calculated divergence) or implementation is ensured.

Pass-By-OID

To improve efficiency, some distributed applications pre-replicate selected objects to all involved locales. In such cases, if a replicated object is an argument to a remotely invoked operation, it is a waste of resources to pass the replicate by either reference or value. Pass-by-reference entails remote message sends; pass-by-value entails the marshaling of a complete copy. In contrast, pass-by-OID allows pre-replicated objects to be passed by no more than the object identifier (OID) under which they were pre-registered in the object tables of both the sending and the receiving object adaptors. A passed-by-OID object, on receipt, resolves to either (a) its local replicate, or (b) an exception if the passed OID has not been pre-registered at both sending and receiving locales. You may think of pass-by-OID as a species of 'pass-by-name' for domain class instances.

Opentalk Pass Mode Control

In the past, when there were only two available pass modes—by value and by reference—the method isPassedByValue was overridden to affect the default pass mode of an object. This is no longer true. Current Opentalk users should pay special attention to this fact. Unmodified user implementations of isPassedByValue may not have their intended effect in the new version of the STST protocol. The methods

```
isPassedByValue
isPassedByReference
isPassedByName
isPassedByOID
```

are now used only to test the pass mode of an object or a PassModeWrapper. An object's default pass mode is now changed by overriding the method passMode.

A particular instance's pass default mode may be superseded by folding the instance in a PassModeWrapper. This is accomplished by sending the instance any one of the following messages:

```
asPassedByValue
asPassedByReference
asPassedByName
asPassedByOID
```

If an instance already sent by value is sent asPassedByValue, the instance rather than a PassModeWrapper is returned. The same holds for the other three methods. If an instance's default pass mode cannot legally be superseded by the desired pass mode, a pass mode exception is raised.

Opentalk Tools

Two lightweight tools have been added to the Opentalk release.

- The OpentalkConsole supports the configuration, creation, and registration of all release-quality request brokers.
- The OpentalkMonitor supports inspection of and registration for all the events generated by release-quality brokers and object adaptors.

DST

Documentation consolidation

Rather than four documents dealing with DST, there is now only one, the *Distributed Smalltalk Application Developer's Guide*. This allowed us to eliminate a great deal of redundancy, but also simplifies the problem of finding relevant information about DST, and will facilitate further updates. In addition, we have recovered and included important information that had been lost when the old *DST User's Guide* was discontinued.

Immutability supported

It is no longer necessary to alter the default immutability settings to use DST.

Other changes

The range of <type_name>OrNil declarations in the IDL SmalltalkTypes module has been extended, and other minor bugs were addressed.

Application Server

The VisualWorks Application Server (formerly referred to as the VisualWave Server) has been enhanced.

New ISAPI Gateway

We have a new ISAPI Gateway, which is being provided as both a release and a debug version of the DLL. In addition, we are going back to the **`hostmap`** form of configuration, so the **`isapi.ini`** file has been eliminated.

Sources for this module are also provided in the form of a Visual C++ project (Microsoft-only).

Refer to the *Web Server Configuration Guide* for more information.

Significant Changes

Changes noted in this section include things that might affect the way existing applications function.

Repartitioning of VisualWave Components and Infrastructure

There is no longer a distinction between VisualWave Developer and VisualWave Server, only a single VisualWave parcel. The appropriate parcels to load are now VisualWave and/or Web Toolkit. The web infrastructure portion is shared between these two, but neither requires the presence of the other. For details, see chapter 3 of the *Web Server Configuration Guide*."

VisualWave Namespace No Longer Imported

The VisualWave namespace is no longer imported automatically into the Smalltalk namespace. It may be necessary to change some of your code to import this namespace, individual classes, or (more rarely) use qualified names to refer to classes in these namespaces.

Renaming of Headless Files

The files associated with headless images have been renamed to be more indicative of their function, and no longer required to conform to 8.3 file naming conventions. In particular:

- **`hlst.tr`** is now **`headless-transcript.log`**
- **`hlstrc.st`** is now **`headless-startup.st`**
- **`hlst-dbg.im`** is now **`headless-debug.im`**

If no **`headless-startup.st`** file is found, the image will look under the old name (**`hlstrc.st`**).

For details, see chapter 13 of the *Application Developer's Guide*.

Logging Changes

There have been a number of changes to Web Toolkit's file logging. Most visibly, there is now a toggle on the launcher's Web menu to enable/disable logging. Note that logging is **off** by default in production mode. This is because we do not at present have a mechanism to roll over logs automatically, and so logs can grow indefinitely. This could be dangerous in a production system. If you are confident you have enough disk space, or a mechanism in place to prevent indefinite log growth, you can enable logging by editing the **`webtools.ini`** file and setting logging=on. For details, see chapter 3 of the *Web Application Developer's Guide*.

Virtual Directories

You may now specify "virtual directories" in the server console. These are useful when you have an external web server as a front end, and need to specify portions of the path that are used by the front-end web server and to be ignored by Smalltalk. For example, if you had an IIS virtual directory or an Apache location directive set up to forward requests of the form http://host/smalltalk/ to the Smalltalk server, the "smalltalk" portion of the path would still be passed along, and the application server would attempt to treat this as part of a file or servlet path. By specifying this in Smalltalk, we tell Smalltalk to ignore that portion of the path. For details, see chapter 5 of the *Web Server Configuration Guide*."

NSAPI Gateway Removed from Distribution

The NSAPI gateway has been removed from the distribution, and is likely to become unsupported in a future release. The code is still available from Cincom by individual request. See the **`readme.txt`** in the **`$(VISUALWORKS)/waveserver/waverelays/nsapi`** directory.

Changed Memory and Network Parameters

Version 7.1 includes changes to some of the default network and memory parameters. The server sockets will by default use the option to re-use addresses, so when a server crashes it can immediately be restarted. In addition, the #listenFor: parameter has been increased to 128. Finally, the default maximum memory has been increased to 150MB, and the growthRegimeUpperBound changed to default to 2/3 of the memory upper bound.

Changes to Command Line Handling

The startup command line options options **`-pcl`** and **`-cnf`** (which are enabled by running Runtime Packager on an image) no longer cause errors when used with the VisualWorks Application Server. Notifiers and dialogs encountered when loading parcels no longer cause the image to exit due to headlessness errors, but are logged to the transcript. This also applies when doing a server reset on the Web Toolkit.

Enhancements and Bug Fixes

Load Balancer on Opentalk

The load balancing functionality now uses the Opentalk Smalltalk-to-Smalltalk communication facilities as its underlying mechanism, rather than DST and CORBA. For details, see chapter 6 of the *Web Server Configuration Guide*."

Production/Debug Toggle

There is also a toggle for switching between production and debug mode from the Launcher window. This has the same effect as evaluating ProcessEnvironment isDevelopingOverride: true/false.

Startup Events

There is an additional startup event available: #finishedServerConfiguration. This can be used to perform initialization activities after the server configuration has been completely read.

Buffering

Servlets and Server Pages now support response buffering, and buffering is enabled by default. This offers some performance benefits, and also makes it possible to set header values anywhere in a server page, rather than only at the very beginning. For details, see chapter 5 of the *Web Application Developer's Guide*.

Multipart forms and File Upload

Multipart forms and uploaded files are now recognized and automatically handled. Since there is no clear API spec for handling these, we materialize them using the Net Connectivity mechanism — effectively as a dictionary of parts.

Web Toolkit Session Extensions

There are a number of changes to Web Toolkit session handling. Each web site now has its own independent session registry. Session which have timed out will be automatically recreated (empty) if a request comes in which uses its session key. Web Toolkit and VisualWave also use separate cookie names for the session key, preventing conflicts when running both on the same host.

New and Enhanced Gateways

The CGI relay is more robust when the configuration is missing or incorrect. In addition, the ISAPI gateway has been thoroughly redone, addressing a number of issues, and more tightly integrating with IIS facilities. Finally, there is a Perl gateway which can be used either as a CGI or with Apache's mod_perl. This can be easier to configure and debug than the other gateways, and can perform quite competitively, especially if used with mod_perl. Note that there is still some inconsistency in configuration. The CGI and FastCGI gateways use the .INI file located in **`%SystemRoot%/VisualWave`**. The ISAPI relay currently does not support the same level of file logging, and reads the hostmap file from the same place. Our intent is to make it interface directly to the IIS logging mechanisms rather than writing its own files.

The perl relay does not read any configuration file, but rather has the configuration information directly in the script. For details, see chapter 5 of the *Web Server Configuration Guide*."

Cache Control headers For Proxy Servers

VisualWave now includes cache control headers for use with proxy servers.

Widget Enhanced for HTML 4.0 Compliance in VisualWave

Table cells and action buttons now include attributes so that their size can be specified. These may be set either using the UI Painter, or programmatically. Note that these changes should not affect existing client code since they are additions.

Better Handling of Encodings

Handling of internationalization has been improved in number of areas. Form data is now handled correctly, regardless of encoding. The output stream for servlets and server pages now defaults according to the locale for that session (which now defaults to a special "web" locale). Characters which cannot be represented in the output encoding will be printed as escaped unicode values.

Robustness

There have been a number of improvements to robustness. Previous versions could crash the server when running out of file descriptors, or encountering the upper limit of 1024 simultaneous open sockets in a process in some operating systems (most notably Linux). These errors are now handled. Server responsiveness under very heavy loads should also be improved, and under heavy load shorter session expiration times will be used, overriding the values specified by the sessions themselves.

Various

- Web hit time is now logged in UTC, in accordance with the W3C specs.
- Treatment of host names is improved. When constructing links, the TinyHttpServer will now respect the host to which the request was sent, rather than always using the hostname set up in the server configuration GUI.
- URLs generated with #linkNamed: can use other protocols than http: (https, ftp, etc.).
- Class RequestDispatcher now correctly forwards/includes URLs that include query parameters

- Request>>queryString now actually returns a string rather than a parsed dictionary of query data. To get the parsed form, use Request>>query
- The authorization policy was changed to allow more flexibility, particularly authorization policies that vary by site
- Setting a cookie in a redirect response is problematic -- many browsers will not pass back the cookie in the next request. This now issues a transcript warning when in development mode.
- Some bugs in parsing JSP tags were corrected (incorrect case in our implementation of standard bean tags, setProperty checking wrong parameter)
- Fixed intermittent errors in #purgeExpires, showing up as an attempt to subtract from nil.
- The issue of VisualWave not automatically loading the Wave UI Painter Tools has been resolved as part of the repartitioning of the VisualWave packages.
- Various issues with VisualWave preview functionality were fixed.
- Turning off visual error display in the VisualWave Settings now stops visual error display.
- Javascript in VisualWave subcanvases now runs.

Known Limitations

“Out of Memory” Error When Packaging

When packaging a Web Toolkit runtime, it is possible to get an intermittent error in which the VM crashes with an “out of memory” error message, during the first save of the three-step save process. Sometimes retrying the operation will succeed, but if the error recurs, you can work around this by removing the call to rehash the symbol table, which is not necessary for producing a runtime image. To do this, load the Runtime Packager, go to the method RuntimeManagerStripper class>>createFinalImage, and comment out the line near the end with "Symbol rehash".

Documentation

This section provides a summary of the main documentation changes.

Advanced Tools Guide

No changes.

Application Developer's Guide

- Restored Signal hierarchy.
- Replaced old debugger with PDP .
- Added brief comments about MultiProc UI (see GUI Developer's Guide for more).
- Improved/corrected graphics for composed characters.

COM Connect Guide

No changes.

Database Application Developer's Guide

No changes.

DLL and C Connect Guide

No changes.

DST Application Developer's Guide

- Merged in documentation from the *DST Programmer's Reference*, *DST Configuration Guide*, and *IDL Programmer's Reference* (those documents are now discontinued).
- Restored information from the old DST User's Guide.
- Updated installation instructions.
- Updated tutorial.
- Added new data types.
- Miscellaneous updates and corrections.

GUI Developer's Guide

- Added MultiProc UI information
- Updated Menu Editor doc
- Updated TreeView doc

Internationalization Guide

No changes.

Internet Client Developer’s Guide

- Spun out SOAP, WSDL, and UDDI doc into *Web Service Developer’s Guide*.

Opentalk Communication Layer Developer's Guide

- Moved Opentalk-SOAP chapter to *Web Service Developer’s Guide*

Plugin Developer’s Guide

No changes.

Source Code Management Guide

- Added information on PostgreSQL configuration
- Other minor additions

Walk Through

- Revised throughout for 7.1

Web Application Developer’s Guide

- New chapter on support for SSP tag libraries
- Discussion of cookie support improved
- Numerous minor enhancements

Web GUI Developer’s Guide

No changes.

Web Server Configuration Guide

- Documentation for ISAPI relays completely revised
- Numerous minor enhancements

Web Service Developer’s Guide

- New document.
- Added WSDL Tool documentation.
- Updates to Web Service Servers section.

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.

Advanced Tools

Profiler

The ATProfiling parcel in **advanced/** has been replaced by two new parcels, ATProfilingCore and ATProfilingUI. See Chapter 2, “Advanced Tools” of the release notes for more information.

Base System

TimeZone reference time

ReferenceTimeZone was introduced in VW 3.0 specifically to cache the proper time zone for those platforms (NT and Mac) whereon DefaultTimeZone was nulled out. It allowed the image to automatically revert to the correct value when restarted on a non-null-TimeZone platform (Win95, Unix). Now that there are no longer any such null-TimeZone platforms, ReferenceTimeZone is no longer needed and has been removed from the system. This compatability method is temporarily provided to allow time to change user code to send #default instead of #reference when querying the currently installed TimeZone.

Documentation does not yet reflect this change.

GUI

ScheduledControllers

The ControlManager class and its associated global, ScheduledControllers, still exist in the system, although they are now obsolete and deprecated. UI process control is now managed by WindowManager instances. Refer to "Multiproc UI" for further details.

ForkedUI

Because of the introduction of the multiprocess UI, the old ForkedUI goodie code is now obsolete, and is totally incompatible with the new Multi-Process UI.

Net Clients

Deprecated messages

Several messages introduced in the first release of NetClients have since been deprecated. These messages are now fully deprecated:

IMAPCommand>>completedSuccessfully

SMTPClient>>sendMailMessage:

Net.MimeEntity>>scanFieldsFrom:do:

Net.MimeEntity>>scanFieldFrom: rfc822Scanner do: aBlock

NetworkAddressDescriptor>>printAsWords:on:separatedBy:

FTPClient>>beCurrentDirectory:

The Net.FTPClient>>beCurrentDirectory: method was poorly named, and has been renamed to setCurrentDirectory:. The old method remains in the system, but in commented as "obsolete."

The Net Clients document has not been updated, and refers to the old method on pages 66 and 69.

Advanced Tools

The ATProfiling parcel is now obsolete, and is placed in the **`obsolete/`** directory. It is retained because it is required by the obsolete DST parcels.

VisualWave

Effective in release 7.1 of the VisualWorks Application Server, support for NSAPI relays has been removed from the shipped product. The existing NSAPI code for HP, MS-Windows, and Sun platforms is still supported through version 7.0, and is henceforth available from Cincom by individual request.

4

Preview Components

Several features are included in a `preview/` and available on a “beta test” basis. This is a renaming of the directory from prior releases, and reflects looser criteria for inclusion, allowing us to provide pre-beta quality, early access to forthcoming features. Several are described in the following sections. Browse the directory contents for last minute inclusions.

New GUI Framework (Pollock), Beta 2

Pollock remains in preview in 7.1.

Over the last several years, we have become increasingly dissatisfied with both the speed and structure of our GUI Frameworks. In that time, it has become obvious that the current GUI Frameworks have reached a plateau in terms of flexibility. Our list of GUI enhancements is long, supplemented as it has been by comments from the larger VisualWorks communities on comp.lang.smalltalk and the VWNC list. There is nothing we would like more than to be able to provide every enhancement on that list, and more.

But, the current GUI Frameworks aren't up to the job of providing the enhancements we all want and need, and still remain maintainable. In fact, we are actually beyond the point of our current GUI frameworks being reasonably maintainable.

This is not in any way meant to denigrate the outstanding work of those who created and maintained the current GUI system in the past. Quite the opposite, we admire the fact that the existing frameworks, now over a decade old, have been able to show the flexibility and capability that have allowed us to reach as far as we have.

However, the time has come to move on. As time has passed, and new capabilities have been added to VisualWorks, the decisions of the past no longer hold up as well as they once did.

Over the past several decades, our GUI Project Leader, Samuel S. Shuster, has studied the work of other GUI Framework tools including, VisualWorks, VisualAge Smalltalk, Smalltalk/X, Dolphin, VisualSmalltalk, Smalltalk MT, PARTS, WindowBuilder, Delphi, OS/2, CUI, Windows, MFC, X11, MacOS. He has also been lucky enough to have been privy to the “private” code bases and been able to have discussions with developers of such projects as WindowBuilder, Jigsaw, Van Gogh and PARTS.

Even with that background, we have realized that we have nothing new to say on the subject of GUI Frameworks. We have no new ideas. What we do have is the tremendous body of information that comes from the successes and failures of those who came before us.

With that background, we intend to build a new GUI Framework.

High Level Goals

The goals of the new Framework are really quite simple. Make a GUI Framework that maintains all of the goals of the current VisualWorks GUI and is flexible and capable enough to see us forward for at least the next decade.

We add must add additional, less concrete goals too:

- The new GUI Framework must be more accessible to both novice and expert developers.
- The new GUI Framework must be more modular.
- The new GUI Framework must be more adaptable to new looks and feels.
- The new GUI Framework must have comprehensive unit tests.

Finally, and most importantly:

- The new GUI Framework must be developed out in the open.

Pollock

The name for this new Framework has been “code named” Pollock after the painter Jackson Pollock. It's not a secret code name. We came up with the name during our review of other VisualWorks GUI frameworks, most directly, Van Gogh. It's just our way of saying we need a new, modern abstraction.

Pollock Requirements

The high level goals lead to a number of design decisions and requirements. These include:

No Wrappers

The whole structure of the current GUI is complicated by the wrappers. We have Spec wrappers, and Border wrappers, and Widget wrappers, and Bounded wrappers and more. There is no doubt that they all work, but learning and understanding how they work has always been difficult. Over the years, the wrappers have had to take on more and more ugly code in order to support needed enhancements such as mouse wheel support. Pollock will instead build the knowledge of how to deal with all of these right into the widgets.

No UIBuilder at runtime

The UIBuilder has taken on a huge rule. Not only does it build your user interface from the specification you give it, it then hangs around and acts as a widget inventory. Pollock will break these behaviors in two, with two separate mechanisms; a UI Builder for building and a Widget Inventory for runtime access to widgets and other important information in your user interface.

New Drag/Drop Framework

The current Drag/Drop is limited and hard to work with. It also doesn't respect platform mouse feel aspects, nor does it cleanly support multiple window drag drop. Pollock will redo the Drag/Drop framework as a state machine. It will also use the trigger event system instead of the change / update system of the current framework. Finally, it will be more configurable to follow platform feels, as well as developer extensions.

The Default/Smalltalk look is dead

We will have at the minimum the following looks and feels: Win95/NT, Win98/2K, MacOSX and Motif. We will provide a Win2K look soon after the first production version of Pollock.

Better hotkey mapping

Roel Wuyts has been kind enough to give permission allowing us to use his MagicKeys hot key mapping tool and adapt it for inclusion in the base product. Thank you Roel.

XML Specs

We will be providing both traditional, array-based, and XML based spec support, but our main format for the specifications will be XML. We will provide a DTD and tools to translate old array specifications to and from the new XML format. Additionally, in Pollock, the specs will be able to be saved to disk, as well as loaded from disk at runtime.

Conversion Tools

With the release of the first production version of the Pollock UI Framework, we will also produce tools that will allow you to convert existing applications to the new framework. These tools will be in the form of refactorings that can be used in conjunction with the Refactoring tools that are now a integral part of VisualWorks, as well as other tools and documentation to ease the developer in transitioning to the new framework.

Unit Tests

Pollock will and already does, have a large suite of unit tests. These will help maintain the quality of the Pollock framework as it evolves.

New Metaphor

The Pollock framework is based on a guiding metaphor; “Panes with Frames, with Agents and Artists.” More on that below.

Automatic look and feel adaptation

In the current UI framework, when you change the look and/or feel, not all of your windows will update themselves to the new look or feel. In Pollock, all widgets will know how to automatically adapt themselves to new looks and feels without special code having to be supplied by the developer. This comes “free” with the new “Panes with Frames, with Agents and Artists” metaphor.

Loading tests

Pollock includes an SUnit test suite, in the PollockTesting parcel. To load this parcel, you must have the Pollock and SUnit parcels loaded.

The New Metaphor: Panes with frames, agents, and artists

In Pollock, a Pane at its simplest is akin to the existing VisualComponent. A Pane may have subpanes. There will be an AbstractPane class. The Window is also a kind of Pane, but because we don't plan to re-write the whole world, it will remain in it's own hierarchy. Also, the Screen becomes in effect the outermost Pane. Other than that, all panes (widgets) will be subclassed in one way or another from the AbstractPane.

The Frame has a couple of pieces, but in general can be thought of as that which surrounds a pane. One part of a Frame is its layout. That is like our existing layout classes, that which defines where it sits in the pane that encloses it. It optionally may have information about where it resides in relation to sibling panes (and their Frames).

A border or scroll bar in the pane may “clip” the view inside the Pane. In this case, the Frame also works as the view port into the pane. As such, a pane may be actually larger than its Frame, and the Frame then could provide the scrolling offsets into the view of the Pane. The old bounds and preferred bounds terminology is gone, and replaced by two new, more consistent terms: visible bounds and displayable bounds. The visible bounds represents is the whole outer bounds of the pane. The displayable bounds represents that area inside the pane that is allowed to be displayed on by any subpane. For example, a button typically has a border. The visible bounds is the whole outer bounds of the pane, while the displayable bounds will represent that area that is not “clipped” by the border.

Another example is a text editor pane. The pane itself has a border, and typically has scroll bars. The visible bounds are the outer bounds of the pane, and the displayable bounds are the inner area of the text editor pane that the text inside it can be displayed in. The text that is displayed in a text editor, may have its own calculated visible bounds that is larger than the displayable bounds of the text editor pane. In this case, the Frame of the text editor pane will interact with the scroll bars and the position of the text inside the pane to show a view of the text.

Artists are objects that do the drawing Pane contents. Note: No longer does the “view” handle all of the drawing. All of the `displayOn:` messages simply get re-routed to the Artist for the Pane. This allows plugging different Artists into the same Pane. For instance, a Text Pane could have a separate Artist for drawing word-wrapped and non-word-wrapped text. A “Composed Text Pane” could have a separate artist for viewing the text composed, as well as maybe in XML format. Additionally, the plug and play ability of the Artist allows for the automatic updating of panes when the underlying look changes. No longer will there be multiple versions of views or controllers, one for each look or feel. Instead, the Artists and Agents will, when needed, be able to be plugged directly into the pane.

Agents are that which interact with the Artist and the Pane on behalf of the user. Now, if this sounds like a replacement of the Controller, you're partially correct. In the Pollock framework, the Controllers will have much less “view” related behavior. Instead, they will simply be the distributor of events to the Agent via the Pane. This means that our Controllers, while they'll still be there, will be much more stupid, and thus, able to be much less complex and less coupled to the Pane. Like the Artist, the Agent is pluggable. Thus, a TextPane may have a read-only Agent, which doesn't allow modifying the model.

Other notes of interest

The Change/Update mechanism will be taking a back seat to the TriggerEvent mechanism. The ValueModel will still remain, and Pollock will be adding a set of TriggerEvent based subclasses that will have changed, value: and value events. Internal to the Pollock GUI, there simply will not be a single place where components will communicate with each other via the change/update mechanism as they do today. While they will continue to talk to the Model in the usual way, there will be much less chatty change/update noise going on.

The ApplicationModel in name is gone. It was never really a model, nor did it typically represent an application. Instead, a new class named UserInteface replaces it. This new class will know how to do all things Pollock. Conversion tools will take existing ApplicationModel subclasses and make UserInterface subclasses.

A new ScheduledWindow class (in the Pollock namespace) with two subclasses: ApplicationWindow and DialogWindow. The ScheduledWindow will be a full-fledged handler of all events, not just mouse events like the current ScheduledWindow. The ApplicationWindow will be allowed to have menus and toolbars, the ScheduledWindow and DialogWindow will not. The ApplicationWindow and DialogWindow will know how to build and open UserInterface specifications, the ScheduledWindow will not. Conversely the UserInterface will only create instances of ApplicationWindow and DialogWindow.

So, What Now?

The work on Pollock has already started. In the VisualWorks 7 distribution, we provided a very basic beta framework. The goal of the first beta was very simple: a window that has a label and an icon, and a button that has a label and an icon.

The next milestone is Beta 2 in VisualWorks 7.1. For that, we have several of the basic widgets done: InputField, TextEdit, CheckBox, RadioButton and ListBox.

Following that will be Beta 3 in VisualWorks 7.2. For that, the following widgets are planned: DropDownList, Menu, Grid (Table/Dataset combination), DialogWindow, Toolbar, TreeView and TabControl. Additionally, an initial GUI Painter tool and all of the basic supporting builders to create windows with these widgets.

The first Production release is scheduled for VisualWorks 7.3. For that, all of the remaining widgets will be done and complete. All of the tools completed. Additionally, tools and utilities will be provided for converting existing GUIs to run on Pollock. Pollock will co-reside in the image along side the existing GUI framework.

After that, it's on to migrating our own tools and browsers to Pollock. Followed in time by the obsoleting of the old GUI framework to a compatibility parcel.

Opentalk SNMP

SNMP is a widely deployed protocol that is commonly used to monitor, configure, and manage network devices such as routers and hosts. SNMP uses ASN.1 BER as its wire encoding and it is specified in several IETF RFCs.

The Opentalk SNMP preview partially implements two of the three versions of the SNMP protocol: SNMPv1 and SNMPv2. It does so in the context of a framework that both derives from the Opentalk Communication Layer and maintains large-scale fidelity to the recommended SNMPv3 implementation architecture specified in IETF RFC 2571.

Usage

Initial Configuration

Opentalk SNMP cares about the location of one DTD file and several MIB XML files. So, before you start to experiment, be sure to modify 'SNMPContext>>mibDirectories' if you have relocated the Opentalk SNMP directories.

Broker or Engine Creation and Configuration

In SNMPv3 parlance a broker is called an “engine”. An engine has more components that a typical Opentalk broker. In addition to a single transport mapping, a single marshaler, and so on, it must have or be able to have

- several transport mappings,
- a PDU dispatcher,
- several possible security systems,
- several possible access control subsystems,
- a logically distinct marshaler for each SNMP dialect, plus

- an attached MIB module for recording data about its own performance.

So, under the hood, SNMP engine configuration is more complex than the usual Opentalk broker configuration. You can create a simple SNMP engine with

```
SNMPEngine newUDPAtPort: 161.
```

But, this is implemented in terms of the more complex method below. Note that, for the moment, within the code SNMP protocol versions are distinguished by the integer used to identify them on the wire.

```
newUdpAtPorts: aSet
| oacs |

oacs := aSet collect: [ :pn |
      AdaptorConfiguration snmpUDP
          accessPointPort: pn;
          transport: ( TransportConfiguration snmpUDP
               marshaler: ( SNMPMarshalerConfiguration snmp ) )].

^(( SNMPEngineConfiguration snmp )
      accessControl: ( SNMPAccessControlSystemConfiguration snmp
          accessControlModels: ( Set
              with: SNMPAccessControlModelConfiguration snmpv0
              with: SNMPAccessControlModelConfiguration snmpv1 ) );
          instrumentation: ( SNMPInstrumentationConfiguration snmp
              contexts: ( Set with: (
                  SNMPContextConfiguration snmp
                      name: SNMP.DefaultContextName;
                      values: ( Set with: 'SNMPv2-MIB' ) ) ) );
          securitySystem: ( SNMPSecuritySystemConfiguration snmp
              securityModels: ( Set
                      with: SNMPSecurityModelConfiguration snmpv0
                      with: SNMPSecurityModelConfiguration snmpv1 ) );
              adaptors: oacs;
              yourself
          ) new
```

As you can see, it is a bit more complex, and the creation method makes several assumptions about just how you want your engine configured, which, of course, you may change.

Engine Use

Engines are useful in themselves only as lightweight SNMP clients. You can use an engine to send a message and get a response in two ways. The Opentalk SNMP Preview now supports an object-reference based usage style, as well as a lower-level API.

OR-Style Usage

If you play the object reference game, you get back an Association or a Dictionary of ASN.1 OIDs and the objects associated with them. For example, the port 3161 broker sets up its request using an object reference:

```
| broker3161 broker3162  oid ref return |

broker3161 := SNMPEngine newUdpAtPort: 3161.
broker3162 := self snmpv0CommandResponderAt: 3162.
broker3161 start.
broker3162 start.
oid := CanonicalAsn1OID symbol: #'sysDescr.0'.
ref := RemoteObject
      newOnOID: oid
      hostName: <aHostname>
      port: 3162
      requestBroker: broker3161.
^return := ref get.
```

This expression returns:

```
Asn1OBJECTIDENTIFIER(CanonicalAsn1OID(#'1.3.6.1.2.1.1.1.0'))->
      Asn1OCTETSTRING('VisualWorks®, Pre-Release 7 godot
      mar02.3 of March 20, 2002')
```

Object references with ASN.1 OIDs respond to get, set:, and so forth. These are translated into the corresponding SNMP PDU type, for example, a GetRequest and a SetRequest PDU in the two cases mentioned.

Explicit Style Usage

You can do the same thing more explicitly the following way, in which case you will get back a whole message:

```
| oid broker1 entity2 msg returnMsg |

oid := CanonicalAsn1OID symbol: #'1.3.6.1.2.1.1.1.0'.
broker1 := SNMPEngine newUdpAtPort: 161.
entity2 := self snmpv1CommandResponderAt: 162.
broker1 start.
entity2 start.
msg := SNMPAbstractMessage getRequest.
msg version: 1.
msg destTransportAddress: ( IPSocketAddress hostName: self
      localHostName port: 162 ).
msg pdu addPduBindingKey: ( Asn1OBJECTIDENTIFIER value: oid ).
returnMsg := broker1 send: msg.
```

which returns:

```
SNMPAbstractMessage:GetResponse[1]
```

Note that in this example, you must explicitly create a request with the appropriate PDU and explicitly add bindings to the message's binding list.

Entity Configuration

In the SNMPv3 architecture, an engine does not amount to much. It must be connected to several SNMP 'applications' in order to do useful work. And 'entity' is an engine conjoined with a set of applications. Applications are things like command generators, command responders, notification originators, and so on. There are several methods that create the usually useful kinds of SNMP entities, like

```
SNMP snmpv0CommandResponderAt: anInteger
```

Again, this invokes a method of greater complexity, but with a standard and easily modifiable pattern. There as several examples in the code.

MIBs

Opentalk SNMP comes with a small selection MIBS that define a subtree for Cincom-specific managed objects. So far, we only provide MIBs for reading or writing a few ObjectMemory and MemoryPolicy parameters. A set of standard MIBS is also provided. Note that MIBs are provided in both text and XML format. The Opentalk SNMP MIB parser required MIBS in XML format.

If you need to create an XML version of a MIB that is not provided, use the 'snmpdump' utility. It is a part of the 'libsmi' package produced by the Institute of Operating Systems and Computer Networks, TU

Braunschweig. The package is available for download through http://www.ibr.cs.tu-bs.de/projects/libsmi/index.html, and at http://rpmfind.net.

Limitations

The Opentalk SNMP Preview is raw and has several limitations. Despite them, the current code allows a user, using the SNMPv2 protocol, to modify and examine a running VW image with a standard SNMP tool like ucd-snmp. However, one constraint should be especially noted.

Port 161 and the AGENTX MIB

SNMP is a protocol used for talking to devices, not applications, and by default SNMP uses a UDP socket at port 161. This means that in the absence of coordination between co-located SNMP agents, they will conflict over ownership of port 161. This problem is partially addressed by the AGENTX MIB, which specifies an SNMP inter-agent protocol. Opentalk SNMP does not yet support the AGENTX MIB. This means that an Opentalk SNMP agent for a VisualWorks application (only a virtual device) must either displace the host level SNMP agent on port 161, or run on some other port. Opentalk SNMP can run on any port, however many commercial SNMP management applications are hard-wired to communicate only on port 161. This places limitations on the extent to which existing SNMP management applications can now be used to manage VisualWorks images.

Opentalk

The Opentalk preview is extension to VisualWorks 7.1 and the Opentalk Communication Layer, and provides tools for remote profiling. Communications using the SNMP protocol are described in the previous section.

The **`preview/opentalk/`** directory has been emptied of some of its former contents. Nearly all of the preview components in that directory were affected by the new additions of this release. In a future release, a new Opentalk remote debugger will be implemented under PDP and other former Opentalk preview components will be revised.

For installation and usage information, see the readme.txt file in the Opentalk preview directory.

SocratesEXDI and SocratesThapiEXDI

SocratesXML support at the EXDI level is included with this release in the **`preview/database/`** directory, in the SocratesEXDI and SocratesThapiEXDI parcels. The code is still under study and development for full release at a later time.

Currently this code supports:

- Supports MindSpeed 5.1 and SocratesXML 1.2.0 across Windows, Solaris and HPUX platforms.
- The SocratesXML API allows threaded calls, through thread safe drivers.
- All SocratesXML types (except MONETARY), collections and object references (OID) supported.
- Both placed and named input parameter binding is supported though SocratesXML only supports placed input binding.

Installation

SocratesXML 1.2.0

To install under Solaris and HPUX, simply load the SocratesEXDI parcel.

For Windows you must manually install the **`1880.016.map`** file. Do this by executing the external interface initialization code below and selecting the **`1880.016.map`** file:

```
SocratesInterface userInitialize
SocratesThapiInterface userInitialize
```

The class instance variable build defines the current build of the external interface classes on Windows platforms and can be ignored for the other platforms. The default value is set to 1881.016 on parcel loading.

MindSpeed 5.1

To install under Solaris and HPUX, simply load the SocratesEXDI parcel.

For Windows you must manually install the 1690.014.map file. Do this by executing the external interface initialization code below and selecting the 1690.014.map file when prompted:

```
SocratesInterface userInitialize
SocratesThapiInterface userInitialize
```

The class instance variable build defines the current build of the external interface classes on Windows platforms and can be ignored for the other platforms. The default value is set to '1881.016' on parcel loading.

Data Interchange

The Socrates database type to Smalltalk class mapping is given in table 1 below. Table 2 defines the mapping for database collection types.

The Socrates EXDI automatically converts Socrates database types to/from instances of concrete Smalltalk classes. Database bit types (BIT, VARBIT) are mapped to a new Smalltalk class BitArray. This class provides efficient uni-dimensional access to a collection of bits.

Table 1 - Socrates scalar type to Smalltalk class mappings

Socrates Data type	**Smalltalk Class**
BIT, VARBIT	BitArray
CHAR, NCHAR, VARCHAR (STRING), VARNCHAR	String
DATE	Date
DOUBLE	Double
FLOAT	Float
INTEGER, SHORT, SMALLINT	Integer
NULL	UndefinedObject
NUMERIC	FixedPoint, LargeInteger
TIME	Time
TIMESTAMP	Timestamp

Table 2 - Socrates collection type to Smalltalk class mappings

Socrates Collection Data type	**Smalltalk Collection Class**
LIST, SEQUENCE	OrderedCollection
MULTISET	Array
SET	Set

Socrates support for heterogeneous collection maps naturally onto Smalltalk collections and is fully supported within in the limits defined by the SocratesXML C API.

For this release collections will be fetched and written in their entirety.

Reference Support

The Socrates EXDI provides transparent support for database object references, Socrates OIDs (similar to the SQL Ref data type).

A Socrates OID is represented by a lightweight Smalltalk object (class SocratesOID) that contains sufficient information to uniquely identify the database object across all accessible database servers. SocratesOID instances are not related to active database connections and so can exist outside the normal database server connection scope. SocratesOIDs can be instantiated back into live database objects (represented by instances of class SocratesObject) via an appropriate active connection i.e. one connected to the original database server.

Object Support

The Socrates EXDI provides access to raw Socrates database objects through instances of class SocratesObject. SocratesObject instances are intimately connected to the Socrates database server and so their scope is that of the underlying database connection.

A key feature of SocratesObject is high-level support for server side method (function) invocation. Simple server methods can be supported directly; methods with multiple or non-standard return values must be explicitly coded by the developer using in-build method invocation support methods. This typically involves defining a Smalltalk class (as a subclass of SocratesObject) to represent the target server class. This new class will be the place holder for both class and instance server method wrappers. All Smalltalk wrapper methods are defined as instance methods irrespective of whether they represent class or instance methods in the server. The Smalltalk wrapper methods are coded to extract the returned value(s) from the original method argument list, free any resources and returning the extracted value(s). The Smalltalk GLO hierarchy provides numerous examples of simple and complex wrapper methods.

GLOs

The Socrates EXDI supports LOB as a subset of the capabilities provided by SocratesXML GLOs. The Socrates EXDI implements the LOB interface through the SocratesGLO class hierarchy. SocratesGLOs provide a stream-like access to GLO data. All GLO subclasses have been modeled, i.e. audio, image and mm_root hierarchies. Each modeled subclass implements the majority of class and instance server side methods as Smalltalk methods. The user can easily add/extend this functionality by modeling any user-defined subclasses and server side methods.

The initial release of Socrates EXDI supports read-only support for Socrates GLOs.

Virtual Machine

IEEE floating point

The engine now supports IEEE floating-point primitives. The old system used IEEE floats, but would fail primitives that would have answered an IEEE **`Inf`** or **`NaN`** value. The new engine does likewise but can run in a mode where the primitives return **`Infs`** and **`NaNs`** rather than fail.

Again due to time constraints the system has not been changed to use this new scheme and we intend to move to it in the next release. In the interim, candidate code is provided as a goodie, and the engine can be put in the new mode by a **`-ieee`** command line option.

OE Profiler

The OEProfiler, an engine-level pc-sampling profiler now supports profiling native methods in the nmethod zone. The image-level code (**`goodies/parc/OEProfiler.pcl`**) is still only goodie quality but we hope to integrate properly these facilities with the Advanced Tools profilers soon.

Reader Comment Sheet

Name: ______________________________

Job title/function: ______________________________

Company name: ______________________________

Address: ______________________________

Telephone number: ( ) - ______________ Date: / / ______

How often do you use this product? ❑ Daily ❑ Weekly ❑ Monthly ❑ Less

How long have you been using this product? ❑ Months ❑ Years

Can you find the information you need? ❑ Yes ❑ No

Please comment. ______________________________

____

Is the information easy to understand? ❑ Yes ❑ No

Please comment. ______________________________

____

Is the information adequate to perform your task? ❑ Yes ❑ No

Please comment. ______________________________

____

General comment: ______________________________

____

____

____

To respond, please fax to Larry Fasse at (513) 612-2000.

CINCOM®
The Smart Choice®

P46-0106-06