Cincom

VisualWorks®

Release Notes 7.3

P46-0106-09

Part Number: P46-0106-09

Software Release 7.3

This document is subject to change without notice.

RESTRICTED RIGHTS LEGEND:

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Trademark acknowledgments:

CINCOM, CINCOM SYSTEMS, and the Cincom logo are registered trademarks of Cincom Systems, Inc. ParcPlace and VisualWorks are trademarks of Cincom Systems, Inc., its subsidiaries, or successors and are registered in the United States and other countries. ObjectLens, ObjectSupport, ParcPlace Smalltalk, Database Connect, DLL & C Connect, COM Connect, and StORE are trademarks of Cincom Systems, Inc., its subsidiaries, or successors. ENVY is a registered trademark of Object Technology International, Inc. All other products or services mentioned herein are trademarks of their respective companies. Specifications subject to change without notice.

The following copyright notices apply to software that accompanies this documentation:

VisualWorks is furnished under a license and may not be used, copied, disclosed, and/or distributed except in accordance with the terms of said license. No class names, hierarchies, or protocols may be copied for implementation in other systems.

This manual set and online system documentation © 1999–2004 by Cincom Systems, Inc. All rights reserved. No part of it may be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior written consent from Cincom.

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 7.3

These release notes outline the changes made in the version 7.3 release of VisualWorks. Both Commercial and Non-Commercial releases are covered.

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

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

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

http://www.cincomsmalltalk.com/

Product Support

Support Status

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

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

Product Patches

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

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

ARs Resolved in this Release

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

Store Database Update

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

```
DbRegistry update73
```

in a workspace.

After updating, you can still load from and publish to the database from older images. To take advantage of the added table structure, however, you must access the database from a 7.3 image.

Moving Packages into the Base

In 7.3 we have begun simplifying the views into the system by reducing parcel and package views into a single view. This was done by moving package/bundle functionality into the base. Parcels now exist as simply an external file representation of packagses, and are represented as packages in the system browser once loaded. There is no longer a parcel view.

However, the change is not complete, and there are still a number of operations on packages and parcels that are different. (Details details).

New Platforms

WinCE

Devices running two Win CE operating systems (ARM and x86) are now supported as deployment platforms. In general, these devices are not suitable for development purposes.

64-bit Linux

This release introduces a beta of native 64-bit VisualWorks on the first platform, linuxx86_64. Refer to “64-bit VM” for more information.

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.

Store

(AR 46654) If the ChangeList contains a change to a bundle structure that moves a package from a bundle, replaying the change causes the package to be unloaded.

Initializing Shared Variables

(AR 44594) In 7.1, a number of inconsistencies were reported in how classes and shared variables are initialized when loading code from the several storage options. Most of these are now corrected, reducing the matrix to the following table, which 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	✓	✓	✓	✓	✓	✓

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

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.

Limitations listed in other sections

- WinCE devices: Known limitations

2

VW 7.3 New and Enhanced Features

This section describes the major changes in this release.

Virtual Machine

New Virtual Machines

LinuxPPC

This engine was in preview, but is now released.

MacX X11

This is a version of the MacX engine adapted for using X11 graphics.

Windows CE

VMs are now supported for CE devices running x86 and ARM processors. More information is provided below.

64-bit Linux

This release introduces a beta of native 64-bit VisualWorks on the first platform, linuxx86_64. Refer to “64-bit VM” for more information.

VM Sources

Viritual machine source code is provided under license with the commercial release. Sources are installed in subdirectories of **`bin/`**. Once installed, refer to the **`README`** files in various directories for instructions on how to use the sources to compile a VM.

Microsoft Windows CE

Virutal machines for Microsoft Windows CE are intended for use on CE devices as an application deployment environment. Typically, an application is developed in a standard development environment, and prepared for deployment on a CE device. The image, VM, and any supporting files, are then copied to the CE device and executed.

VisualWorks has been successfully tested on the following hardware:

- Simpad SLC with StrongARM-SA-1110, Windows CE .NET Version 4.0
- skeye.pad with StrongARM-SA-1110, Windows CE .NET Version 4.1
- HP iPAQ H2210 with Intel PXA255 XScale, Windows Pocket PC 2003 (Windows Mobile 2003)
- Tatung WebPAD with Geode GXm, Windows CE .NET Version 4.10

There are, however, limitations. Refer to “Known limitations” below for details.

Distribution contents

There are two directories with virtual machines for the different processors:

- **`bin\cearm`** – for StrongARM and XScale processors,
- **`bin\cex86`** – for Pentium-compatible processors like the Geode.

Each directory contains three executables and a DLL:

- **`vwntoe.dll`** – the DLL containing the virtual machine.
- **`vwnt.exe`** – the GUI stub exe which is normally used to run GUI applications. It uses **`vwntoe.dll`**.
- **`vwntconsole.exe`** – the console stub executable which is normally used to run console applications. It uses **`vwntoe.dll`**.
- **`visual.exe`** – the single virtual machine, which is used for single-file executable packaged applications.

The assert and debug subdirectories contain versions of these executables with asserts turned on for debugging. The debug engines are not optimized and so can be used with the Microsoft eMbedded Visual C++ debugger. Refer to the engine type descriptions in the *Application Developer's Guide*, Appendix C, for further information.

Prerequisites

Windows CE VMs require a few additions to the standard image. These are provided in the parcel **`ce.pcl`**. On the PC, prior to the deployment to your CE machine, load this parcel into your image.

This parcel contains two major changes:

- A new SystemSupport subclass for CE – This is necessary because the name of the DLLs differs from other Windows versions and they contain different versions of the called functions. For example, only Unicode versions of most functions are provided and some convenience functions are missing.
- A new filename subclass, CEFilename – CE does not have a "current working directory" concept, so only absolute paths are supported. Therefore CEFilename stores the current directory and expands relative paths into absolute paths.

Developing an Application for CE

In general, developing an application for deployment on a CE device is the same as for any other application. The notable differences have to do with screen size, especially on small PDA-type devices, and filename handling, because CE does not use file volumes or disk drive letters.

Before beginning development, load the CE parcel (ce.pcl) into the development image. The changes it makes only take effect when the image is installed on the CE device, so you can develop as usual on your standard development system.

Filenames

WinCE does not use relative file paths or volume (disk) letters. This is transparent during development, because the CEFilename class handles converting all paths to absolute paths when the application is deployed on a CE device. No special development restrictions need to be observed.

DLL names

Similar, DLL names are modified appropriately when installed on a CE device.

Window sizes and options

CE devices come in a variety of screen sizes. For the larger devices, with a screen size of 640x400, the limitations are not extreme. However, on the smaller devices, such as a Pocket PC with a screen size or 240x320, the size greatly affects your GUI and application design.

As a deployment environment, you generally should have all development tools, such as browsers closed, and possibly removed from the system, though this is not required.

However, when testing and debugging it is convenient to have all of these development resources available, and this can present serious difficulties.

Also, especially for smaller devices, select an appropriate opening position for the GUI, in the canvas settings. Opening screen center is generally a safe choice.

Input devices

The input side limitations are also worth mentioning. Typically you only have a touch sensitive screen and a pen for it. There is no keyboard, hence no modifier keys. You have no mouse buttons where VisualWorks prefers to have three. So moving the pen somewhere always implies a pressed button. You can open the 'soft input panel', i.e. a small window with a keyboard in it. But it is not really comfortable to enter longer texts this way and this window needs some of your valuable screen space. So whenever you expect textual input, you should leave some free room for the keyboard. (At 240x320, a full screen work space contains 10 lines of text plus title bar, menu bar, tool bar buttons and the status bar at the bottom. The Keyboard window covers the lines 8 to 10 and the status bar.)

The CE parcel adds code which interpretes holding the pen for approx 1.3 seconds as a right button press to open the operate context menu. This behaviour can be turned on and off in the look and feel section of the settings window. On pocket PC, but not on the CE web pads, users are trained to expect this behaviour.

.NET access

While WinCE .NET uses the features of the Microsoft .NET platform, the DotNETConnect preview does not support their use.

Deploying on a CE Device

Load the CE parcel (**`$(VISUALWORKS)\bin\winCE\CE.pcl`**) into your development image. This provides the features described above (see “Prerequisites”).

Deployment preparation is, otherwise, the same as usual, though there may be practical considerations. On many devices

Starting VisualWorks on CE

There are several ways to start VisualWorks on Windows CE:

- In the command shell, execute:

  ```
  visual [options] visual.im
  ```

  (Not all CE environments have a command shell interface.)

- Double-click on **visual.exe**. This starts VisualWorks with the default image, **visual.im**..

  By default, the vm attempts to open an image with the same name as the vm and in the same directory. So, you can rename the the vm to match your image name and execute it in this way.

- Double-click on an image file. This works only if the **.im** extension is associated with VisualWorks in the registry of the CE device.

  If you are developing on the CE device, you can evaluate this expression in a workspace:

  ```
  WinCESystemSupport registerVisualworksExtension
  ```

- If you have packaged the vm and image as a single executable file (e.g. using ResHacker provided in the **packaging/win** directory), you can simply run the executable.

- Create a short-cut to read e.g.

  ```
  "\My Documents\vwnt" "\My Documents\visual.im"
  ```

  The default CE Windows explorer can be used to create associations by copying an existng short-cut (e.g., Control panel), renaming it, and editing its properties. On CE machines that lack the standard explorer, you can find free tools to edit associations.

Known limitations

Sockets

- Non-blocking calls are not yet supported.
- Conversion of hostnames to IP addresses, service names to ports, etc., is not implemented. Use addresses instead, e.g., 192.109.54.11 instead of www.cincom.com.

File I/O

- File locking does not exist on CE (prim 667)
- Delete, rename, etc., do not work on open files (prim 1601,1602,..)
- “ '\' asFilename fileSize “ fails with FILE_NOT_FOUND_ERROR.

Windows and Graphics

- Animation primitives not working properly (prims 935-937)
- Only full circles are supported by the OS; arcs and wedges are converted to polylines
- No pixmap <-> clipboard primitives

User primitive

- As yet there is no support for user primitives or primitive plugins.

Base system

Subsystems and Startup/Shutdown

VisualWorks 7.3 includes a number of changes in the way image save and startup processing is handled. In previous versions, these operations were a very tightly coupled series of operations. The new mechanism aims to make it simpler to extend these mechanisms, and to allow programs to add actions at various points in this processing. For most code, these changes will be backward-compatible, but there will be new facilities that can be taken advantage of.

In previous versions, the primary mechanism for getting notification of system events was to register as a dependent of ObjectMemory. This allowed the object to receive notification of all system events, but had no control over ordering. More recently, SystemEventInterest and CommandLineInterest have allowed more declarative registration for events, but again without ordering.

This version introduces additional pragma methods in SystemEventInterest and CommandLineInterest that allow control over the order in which events occur. In addition, it defines a new construct, a Subsystem, which can serve to group actions together, and which activates or deactivates as a unit. For example, WindowingSystem, or FinalizationSystem.

Subsystems define four possible actions: activate, deactivate, pause, and resume. On image startup, the system attempts to activate all the subsystems present in the image. Subsystems also activate after being loaded (this is triggered by the postLoad: mechanism), and can also be activated by sending the #activate message. Similarly, when the image saved, subsystems are paused while the save is in progress, and then resumed after the image returns from a snapshot but continues running. To respond to these events, subsystems can implement the methods

setUp, tearDown, pauseAction and resumeAction. The inherited versions of these methods do nothing, so a new subsystem need only define the ones it needs.

The order of activation is defined by the prerequisites of a subsystem. A subsystem must be deactivated or paused before its prerequisites, and activated or resumed after them. For example, the base image includes the subsystem WindowingSystem. That has prerequisites DelaySystem and BasicGraphicsSystem. If we attempt to activate the WindowingSystem, it will check first to make sure that the DelaySystem and BasicGraphicsSystem are running, and if either is not, it will attempt to activate them.

Prerequisites are defined either by the prerequisiteSystems method, or by a method that contains the <prerequisites> pragma. Both return a collection of Subsystem classes. If you're defining a subsystem, then you would normally use the prerequisiteSystems method. The pragma can be used to add prerequisites to an existing system. This is useful because it lets you add a new subsystem which runs at an arbitrary point during system setup. For example, if we needed to do something before windowing is active, we could add an extension method to WindowingSystem that would make our system be a prerequisite of windowing system.

Subsystems can indicate which system events they want to respond to by defining the methods activationEvent, deactivationEvent, pauseEvent and resumeEvent. For most purposes the inherited defaults will work, but it might, for example, be useful to define activationEvent to use earlySystemInstallation rather than returnFromSnapshot.

Subsystems also allow command-line options to be defined, using the <option:> and <option:sequence:> pragmas. For example

```
headless
    <option: '-headless'>
    HeadlessImage default isHeadless: true.

loadParcel: aStream
<option: '-pcl'>
    (CommandLineInterest argumentsFrom: parcelNameStream)
        do: [:each | Parcel loadParcelByName: each].
```

It's also possible to specify a sequence in command-line options, but note that this only determines the sequencing among options of the same subsystem. Other subsystems' command-line options will be processed when that system prepares to activate, and all command-line options within a subsystem are processed before the setUp method is called. Within the same priority level and subsystem, command-line options are processed left to right.

In the current system, startup and image save have been broken up into various small pieces. This included some very long and complex methods, such as ObjectMemory>>snapshotAs:thenQuit:withLoadPolicy:. In addition, much of the logic of making an image snapshot has been split out of ObjectMemory into a Snapshot class. Unless your applications override or call methods that have been moved, this should be transparent to existing applications.

The way system events are now signalled is as follows:

1. When the image starts up, there will be an active process at high priority, running postSnapshotBootstrap. This will mark all subsystems as inactive (without going through shutdown procedures). Then it will signal the system event #earlySystemInstallation. We call this an event, though it is not an event in the GUI event sense, since it does not have an arbitrary list of objects registered. Instead, the event is sent directly to the Subsystem class, which then notifies all of its subclasses, via the #reactToEvent: method. Each subsystem class knows the set of four events that it can be interested in, and checks for each of them. If the event matches, then it will invoke the appropriate class method, for example, #activate.
2. The #activate class method will tell the singleton instance of the subsystem to privateActivate. If the subsystem is already active, this does nothing. Similarly, if the #canActivate flag is false, then it will do nothing. This allows us to disable certain systems, and can be very useful. For example, in version 7.3, the greater part of the headless functionality is implemented by just setting canActivate to false for the WindowingSystem. This also means that we can make an image headfull later on by setting that to true and activating WindowingSystem. The subsystem will then check if all of its prerequisites are active. If any of them are not, then it will recursively tell them to activate. If they fail, then the subsystem will not activate.
3. Finally, the subsystem will run its activation actions. The activation actions are the set of command line options defined for the subsystem, followed by running the setUp method. At the end of this, if the canActivate flag is still true (a command-line option may have set it to false), then the subsystem marks itself active.

There are several subsystems worth noting. These include ImageConfigurationSystem, UserApplication, and InterestNotificationSystem. ImageConfigurationSystem defines activities for loading code or other

information into the system at startup. It includes a number of command-line options, some new and some old. These include **-pcl**, **-cnf**, -**filein**, **-settings**, **-doit**, and **-evaluate**. These act as follows:

-pcl	Load the named parcel(s), checking both as a filename and as a name to be searched in the parcel path
-cnf	Treat the argument(s) as configuration files, containing parcels to be loaded, one per line
-filein	Treat the argument(s) as Smalltalk files to be filed in
-settings	Treat the argument(s) as XML files containing Smalltalk settings, and load thm.
-doit	Treat the argument(s) as strings to be evaluated
-evaluate	Treat the argument (only one) as a string to be evaluated. After evaluation, put the displayString of the result onto the standard output and exit the image.

UserApplication is intended to serve as a stub for an application. It is a very simple Subsystem, which depends on WindowingSystem, and whose setUp action simply calls a main method. This allows you to define an application class with a main method which will be called automatically on image startup, and which can be used to define additional actions or command-line options.

InterestNotificationSystem is what handles notifying the dependents of ObjectMemory and the SystemEventInterest/CommandLineInterest methods. When it receives a system event, it calls AbstractSystemEventInterest to notify all the interests on that event, and then does an ObjectMemory changed:. However, note that that only handles four of the events, since that's all that any subsystem knows about. In order to notify of the earlySystemInstallation event, the CacheFlushingSystem, which is otherwise responsible for getting rid of any stale handles that the image has from before it was saved, also notifies AbstractSystemEventInterest and ObjectMemory dependents.

If we want to define a more complex subsystem, which needs to activate part way through the system startup procedures, then we need to determine where it fits into the sequence of startup events. The first question is whether it needs the windowing system to be running or not. Systems that run before windowing is active are much harder to debug, because if there is a problem, we can't get a debugger. For debugging these the command-line option **-o10s**, available in the debug virtual machine, dumps a trace of (almost) every message send. This can be very helpful. So if we do need the windowing system to run, we can make

WindowingSystem a prerequisite in the prerequisiteSystems method, or even DevelopmentSystem, which ensure that the development environment is fully set up. If we don't, then we must determine what we need to have active, and what we want to run before. One way to do this is to trace through the set of subsystems following the prerequiste relationships. Alternatively, the following script will list to the transcript all the subsystems activating in response to a particular event, in the order that they will activate. For each subsystem it will also list the command-line options that apply. To list all the subsystems that activate on system startup you would need to list this twice, once for the event earlySystemInstallation, and once for returnFromSnapshot.

```
| tiers event systems systemsWithPrerequisites |
event := #returnFromSnapshot.
tiers := OrderedCollection new.
systemsWithPrerequisites := Dictionary new.
systems := (Subsystem allSubclasses
    select: [:each | each activationEvent == event]) asOrderedCollection.
systems do: [:eachSystem |
    systemsWithPrerequisites
        at: eachSystem
        put: eachSystem current allPrerequisiteSystems].
[systems isEmpty]
    whileFalse:
        [| roots |
        roots := systems
            reject: [:sub | (systemsWithPrerequisites at: sub)
                anySatisfy: [:pre | systems includes: pre]].
        tiers add: roots.
        systems removeAll: roots].
Transcript cr; cr.
tiers do: [:eachSet |
    eachSet do: [:eachSystem |
        Transcript cr; show: eachSystem name , ': '.
        eachSystem current commandLineOptions
            do: [:each | Transcript show: each option , ', ']].
        Transcript show: 'setUp']].
```

Refer to chapter 8, “Application Framework,” of the *Application Developer’s Guide* for application level explanations and examples.

Tools

Packages Replacing Categories

Bundles and packages have replaced categories as the primary way of structuring code. In practical terms it means that the "Package" view of the browser, previously available only after loading Store, is now the standard.

Loading Store only adds the ability to version packages and store them in a database. Category-related API is still supported for backward compatibility, but categories no longer appear in the developement tools.

In images without Store loaded, packages and bundles do not curently keep track of their modifications.

Context Menu Changes

Context menus in the browser code views have been changed to allow easier access to more frequently used commands, such as breakpoint addition/removal, and especially context-sensitive cross-referencing.

The top-level context menu includes commands to browse the item containing the insertion point, which changes depending on the location of the insertion point. For example, when the insertion point is inside a selector, the context menu includes "Browse Senders of Selector" and "Browse Implementors of Selector" items. When it is inside a class name, the menu includes the items "Go to Class" and "Browse Class in New Window".

Context-sensitive cross-referencing support has also been added to the debugger code views.

Probe Dialogs

Probe modification dialogs in the debugger previously required the user to separately "Accept" the changes in probe condition and expression views before closing the dialog. This exlicit "Accept" is no longer necessary.

Miscellaneous

Functionality from the following packages from the public repository has been included into the base VisualWorks:

AddInstVarAtCompile

An option is added to the "undeclared variable" compilation warning dialog to declare a variable as an instance variable of the current class, or as a new class.

WorkspaceFormatting

A "Format Selection" command is added to the standard context menu for code views. Note that this command is different from the "Format" command in browsers, in that it formats only the selected text rather than the entire content of the view (since text in a workspace typically consists of several independent code snippets).

ExtraIcons

(Partially incorporated) Package and bundle icons in the browser now show whether the package or the bundle comes from the current repository (tinted icons) or elsewhere (gray icons). In images with Store loaded, the shape of the icons changes to indicate the modified/unmodified state. In images without Store loaded, packages and bundles do not curently keep track of their modifications.

Advanced Tools

Profiler

Summaries of Recursive Functions

The profiler's **Spawn** menu selection generates a summary of all the resources consumed by a selected, target method. The very simplest example of a summary is shown below:

```
10.4 SomeClass>>parent1:
    10.4 SomeClass>>nonRecursive:
        10.4 SomeClass>>child1:
```

The target method, nonRecursive:, consumes 10.4% of the resources. All of its resources are spent in child1:. It has only one caller in the entire activation tree, parent1:, and calls from that parent account for 10.4%, or all, of the resources consumed by it.

The summaries provide:

1. a list of all immediate parents or callers of the target method, with the percentage of the resources consumed by the target as a result of all direct calls from the parent,

2. the target method with the percentage of resource consumed by all calls to it, and
3. a list of all the children or methods immediately called by the target with the percentage of the resources spent in the child as the result of all immediate target calls.

All of these three sets of percentages should have the same, or very nearly the same, total. In the past, these summaries were rarely if ever correct when the selected method was recursive. Line percentages in excess of 100% were commonly produced. Now, the summaries are correct for all classes of directly, mutually, or indirectly recursive methods.

A very simple direct recursion, where the recursive method has distinct termination and continuation branches, is now summarized in the following form:

```
54.5 SomeClass>>directlyRecursive:
12.0 SomeClass>>parent1:
        66.5 SomeClass>>directlyRecursive:
            58.2 SomeClass>>continuationBranch:
            8.3 SomeClass>>terminationBranch:
            0.0 SomeClass>>directlyRecursive:
```

The method directlyRecursive: consumes 66.5% of the resources. It consumes 58.2% of them in continuationBranch: and 8.3 % of them in terminationBranch:. Of the total, 12.0% is attributable to the initial call from parent1: and the rest, 54.5%, is consumed as the result of directly recursive calls. Note that in the case of direct recursion, the child entry for the target method always has a percentage of zero. The parent entry has the percentage of the initial recursive call.

The implementation of the algorithm now used to generate the summaries has known inefficiencies, and summaries continue to be incorrect for low-level target methods that are removed from the MethodDictionary after the profiling run, because they are wrappers representing a primitive. Both issues will be addressed in a future release.

Publishing in Store

If publishing the parser/compiler in Store, it must be published binary; otherwise it cannot be loaded from the databes.

Store

External File Support

Store has added the capability of including arbitrary files in a bundle, allowing non-code to be included in a bundled project. This is useful, for example, if a release of a project includes documentation, HTML, or graphics files.

The publish dialog for bundles includes a **Files** page on which you select the files in the bundle to publish with the new version.

Adding files to a bundle is currently done by evaluating expressions such as:

```
bundle := Store.Registry bundleNamed: 'Foo'.
bundle addFile: 'foo.txt'
```

When you load a bundle with a file attached, you are prompted whether to download the file.

Store for Oracle

The Store for Oracle parcel has been renamed to StoreForOracle (changed from StoreForOracle8), because the same parcel contains support for Oracle 8, 9, and 10.

Support for Oracle 7 is deprecated, but still available in the **`obsolete/`** directory.

WebService

WSDL Wizard

A new tool, the WSDL Wizard, has been added to assist in generating a WSDL document from application code, and generating client or server support classes and stub methods from a WSDL Document. Refer to the *Web Service Developer's Guide* for information about using the tool.

Net Clients

Cookie Support

HTTPClient now automatically handles cookies, caching incoming cookies and using cached cookies for outgoing requests. By default, cookie processing is disabled, but can be enabled on the HTTP Cookie page of the Settings Tool. Refer to the *Internet Client Developer's Guide* for additional information.

HttpURL

(AR 48015) HttpURL has several enhancements.

- Added decompressContents and decodeContents instance variables and corresponding accessors. These allow setting options for the response stream decompression and decoding.
- Added new readByteStreamDo: aBlock method that sends two parameters to the block: a raw socket stream and a Dictionary in which the key is the response field name and the value is the response header field.
- Changed behavior readStreamDo:. The old implementation set the parameter dictionary values as the response header field values. This implementation was good for simple headers but headers with complicated values returned the ValueWithParams object. Getting specific header field information was too difficult from the ValueWithParams object. The new implementation returns the header fields themselves. Existing applications will need to send a value message to a header field to get the field value.

Merging HTTP and HTTPS

(AR 47101) HttpsClient functionality has been merged with HttpClient. The HttpsClient class is obsolete now.

HttpClient now supports both types of connection: regular (HttpStreamHandler) and secure (HttpsStreamHandler). The decision about which connection to use is based on the URL protocol. If there is no protocol provided, the default protocol is HTTP, as set by HttpClient.Protocol.

This change allows creating subclasses of HttpClient for either type of connection.

Autoloading Prerequisites

HttpClient can now auto-load parcels that are necessary for secure connection. The auto-load feature is optional, with the default option set to ON (VW Setting Tool, URI page). The auto-load feature works the Http and Ftp parcels as well.

ASN.1

ASN.1 is an abstract syntax notation often used to specify transfer encodings. It is used in LDAP, SNMP, and X509. Recent VW releases have contained three different implementation of ASN.1: one in Net Clients, one in the X509 package, and another in the Opentalk SNMP preview. All of these implementations had defects, largely a product of the fact that they were developed to satisfy very focused needs. Each was the first attempt of a different developer to come up to speed on ASN.1, which is not a light subject. As a result of such factors, hardly blameworthy, none of the three implementations is both extensible and fast. All are incomplete. None support more than one of the several ASN.1 encoding rules. All of them address many of the following responsibilities in the same class hierarchy:

- representing the structure and characteristics of a foreign type, for example, the fact that a given ASN.1 SEQUENCE type has two elements, with the second optional,
- representing the special semantics of a type, for example, the fact that SNMPCounters are ASN.1 positive INTEGERs, with an upper limit, that cannot be decremented but only reset,
- implementing the logic for encoding or decoding a type, which arguably should be implemented in a marshaler specific to one of the ASN.1 encoding rules, and not in the class that represents the structure or semantics of the type,
- recording the bytes from which an instance of the type was decoded, which is important in some security applications, where critical byte objects should not be programmatically regenerated for transmission,
- representing the decoded instance of a type, which is possible using a class that also implements decoding and encoding logic and type-specific semantics, and represents type structure, if its instances are also value holders, and
- representing the intended output of an ASN.1 compiler, for the Smalltalk compilation of an ASN.1 file should generate Smalltalk representations of the types the file defines, and the previous

implementations implied, to varying degrees, that an ASN.1 compiler would extend their hierarchy of ASN.1 type classes.

In addition, none of the former implementations employed the approach found in the Opentalk marshaling framework. The latter was originally developed for Smalltalk-to-Smalltalk communication using only native Smalltalk types, but the architectural and design task of extending it for use with a foreign type system was and is a necessary one, perhaps especially with ASN.1, because ASN.1 adds the particular challenge of a type system with multiple encodings.

The new NetClients ASN.1 parcel contains the ASN.1 implementation we intend to go forward with. It improves on its predecessors in several ways.

The only ASN.1 type classes present are those used to represent ASN.1's UNIVERSAL types. This 'Asn1Type' hierarchy is used to represent the type-specific information that marshalers need to encode and decode. It does not implement low-level marshaling logic and its instances are not value holders for decoded objects. Instead, its instances represent concrete, constrained types. The instances are stored in a type registry. A separate Asn1Constraint hierarchy represents the various ASN.1 constraints that are used in constructing new derived ASN.1 types from its UNIVERSAL base types. The **Asn1Type** hierarchy defines the protocol for such construction, and class Asn1Element is used to represent the elements of ASN.1 compound types. Together these classes allows one to represent derived type defined in ASN.1. For example, the code below creates a Smalltalk representation one of the Extension type of the X509 specification:

```
"Extension  ::=  SEQUENCE  {
        extnID        OBJECT IDENTIFIER,
        critical        BOOLEAN DEFAULT FALSE,
        extnValue  OCTET STRING  }"
Asn1TypeSEQUENCE
        register: #Extension
        constraint: Asn1ConstraintNull default
        elements: ( OrderedCollection
                        with: ( Asn1Element
                            symbol: #extnID
                            type: ( Asn1Type findTypeNamed: #'OBJECT IDENTIFIER' ) )
                        with: ( Asn1Element
                            default: false
                            symbol: #critical
                            type: ( Asn1Type findTypeNamed: #BOOLEAN ) )
                        with: ( Asn1Element
                            symbol: #extnValue
                            type: ( Asn1Type findTypeNamed: #'OCTET STRING' ) ) ).
```

A separate Asn1Stream hierarchy implements the streams used to marshal ASN.1 types according to one or another of the several ASN.1 encoding schemes. An Asn1Stream is intended to be used with Opentalk adaptors and request brokers in the same manner as STSTStream. BER and DER ASN.1 marshaling streams are provided in the new implementation. The streams take care of the particularities of the encoding rules they implement. They also extend the protocol Opentalk users have seen before in STSTStream by supporting “type-in-hand” encoding and decoding with the top-level API:

```
marshalObject: anObject withType: anAsn1Type
unmarshalObjectType: anAsn1Type
```

The marshalers also support “raw” or “default” encoding and decoding, using the familiar selectors:

```
marshalObject: anObject
unmarshalObject
```

The former protocol makes use of the type supplied as an argument. The latter makes use of default mappings.

When encoding or decoding “type-in-hand,” the marshalers use double-dispatch to allow the type to appropriately direct marshaling. For example, “type-in-hand” encoding methods send:

```
anAsn1Type encode: anObject with: aMarshaler.
```

And if, for example, the type is an ASN.1 GeneralizedTime, the type dispatches control back to the marshaler with:

```
aMarshaler encodeGeneralizedTime: anObject.
```

The “type-in-hand” decoding methods send:

```
anAsn1Type decode: aByteCount with: aMarshaler.
```

Again, if the type is and ASN.1 GeneralizedTime, the type dispatched back to the marshaler with:

```
aMarshaler decodeGeneralizedTime: aByteCount
```

We know that though a byte count is suitable for BER, DER, and some other ASN.1 encodings, it is unlikely to be suitable for PER, where the only a bit count would be of potential use. Also, please note that the exchange between marshalers and types can be more complex than depicted in the samples above. The marshaling of ASN.1 SEQUENCES, in particular, must account for optional elements, elements with untransmitted default values, type extensions know to only one party in

the communication, the presence of multiple tagging schemes, and encoding-specific differences in element transmission order. Such factors may elaborate the interaction.

Under “raw” encoding and decoding, the marshaler does the best it can using a default mapping between Smalltalk and ASN.1 types when encoding.

For example, when encoding a Timestamp, the marshaler will send:

```
anObject encodeASN1With: aMarshaler
```

The Timestamp will dispatch back with:

```
aMarshaler encodeGeneralizedTime: self.
```

Note that the method encodeASN1With: is implemented by native Smalltalk classes, not classes in the Asn1Type hierarchy. Under “raw” decoding, a marshaler does the best it can with the type tags it detects. At present, this decoding mode may fail in the present of a compound type with implicit or explicit, rather than universal, tagging of elements. It is, however, of use in debugging, when attempting to find out what is coming over the wire, when type information is unavailable, or in doubt.

Distinct from the Asn1Type, Asn1Constaint, and Asn11Stream hierarchies, the Imported hierarchy is a hierarchy of value holders, used to implement, as Smalltalk extensions, the ASN.1 types that have special semantics. There are four concrete classes in this hierarchy:

Asn1BitString
: provides support for the ASN.1 notion of 'unused bits',

Asn1OID
: enforces the uniqueness of its instances and mandatory retention of encodings,

Asn1Choice
: sometimes has a use in Smalltalk application code, and is not, technically, and ASN.1 type, and

ImportedEnumeration
: represents ASN.1 ENUMERATIONs.

These classes appear only as output from decoding or as input to encoding. They, like other Smalltalk classes, implement the “raw” mode double-dispatch method encodeASN1With: aMarshaler. They do not implement the double-dispatch protocol particular to the Asn1Type hierarchy.

Marshalers may be configured to retain encodings. In that case, they wrap decoded objects in an Asn1TypeWrapper which includes the Asn1Type used in decoding and an instance of Asn1Encoding. The Asn1Encoding has a pointer to the source bytes and the offsets for the start of the header (which includes one or more tags and the size of the encoded object, at least under BER and DER), the start of the body, and the end of the body. For an example, see class Asn1StreamBERDefiniteEncodings, which implements a BER marshaler that retains encodings.

Under “type-in-hand” decoding, class Asn1Struct is commonly used to represent the decoded value of an ASN.1 SEQUENCE or SET. It retains the names associated with the elements in the type definition. This class may be reworked in the future, and placed in a separate package, because it is generally useful and already has analogues in other VW add-ons.

Class SMINode provides a simple mechanism for obtaining the symbolic names of Asn1OIDs. It will be significantly extended in the future.

Following is a simple example for experimenting with this code:

```
| derMarshaler testType testObject resultObject |

"SSN ::= NumericString ( SIZE 9 )"
Asn1TypeNumericString
	register:#SSN
	constraint:( Asn1ConstraintSize size: 9 ).

"TimeOfBirth ::= GeneralizedTime"
Asn1TypeGeneralizedTime
	register:#TimeOfBirth.

"Sex ::= ENUMERATED { male(1), female(2) }"
Asn1TypeENUMERATED
	register:#Sex
	elements:( ( OrderedCollection new )
			add: #male -> 1;
			add: #female -> 2;
			yourself ).

"ST001 ::=SET {
	a   ::= SSN,
	b   ::= TimeOfBirth,
	c   ::= [0] EXPLICIT Sex
	}"
```

```
Asn1TypeSET
	register: #ST001
	constraint: ( Asn1ConstraintSize size: 3 )
	elements: ( OrderedCollection new )
		add: ( Asn1Element symbol: #a type:
		    ( Asn1Type findTypeNamed: #SSN ) );
		add: ( Asn1Element symbol: #b type:
		    ( Asn1Type findTypeNamed: #TimeOfBirth ) );
		add: ( Asn1Element
			symbol: #c
			tag: 0
			taggingMode: #explicit
			type: ( Asn1Type findTypeNamed: #Sex ) );
		yourself ).

derMarshaler := ( Asn1StreamDER on: ( ByteArray new: 4096 ) )
		maxReadLimit.
testType := Asn1Type findTypeNamed: #ST001.
testObject := ( OrderedCollection new )
		add: #a -> '193759845';
		add: #b -> (Timestamp new fromDate: ( Date newDay: 15
			monthNumber: 6 year: 1963 ) );
		add: #c -> ( ImportedEnumeration
			type: ( Asn1Type findTypeNamed: #Sex )
			value: 1 );
		yourself.
derMarshaler
	marshalObject: testObject
	withType: testType.
derMarshaler setReadLimit: derMarshaler position.
derMarshaler position: 0.
resultObject := derMarshaler unmarshalObjectType: testType.
```

We hope you observed that this new implementation does its best to avoid the conflations of its predecessors, by segregating type information, encoding-specific marshaling logic, the objects used to retain encodings or element names, and imported types. It is also extremely fast at decoding and encoding, at least in comparison with its progenitors. On a Dell OptiPlex GX110 (598 MHz Pentium III), it decoded certificates at a rate over 1000 bytes/millisecond. This is a somewhat deceptive statistic because certificates have parts that are bytes, decoded as bytes, but it represents a ten-fold improvement.

This said, though this implementation is more complete than its predecessors, it does not yet support the ASN.1 exception marker or implement every one of the ASN.1 constraint types. It does not yet support use of the more arcane ASN.1 UNIVERSAL types, even though it

does support more types than necessary to implement SNMP or X509. (ASN.1 has several types that have fallen out of general use.) There are other lacunae. The X509 parcel now uses this new ASN.1 implementation, though the SNMP preview does not as yet. So, we have gone from three ASN.1 implementations to two. Though that is an improvement, and perhaps a large one given the other factors mentioned above, there is still work to do.

Security

Main task of this release cycle was adoption of the new ASN.1 framework by our X.509 certificate implementation. While this change is confined to the internals of the implementation and isn't exposed at the user API level it enables us to further enhance our X.509 support in the future (certificate generation, CRL support, etc).

Other notable achievements are HMAC implementation and addition of password based cryptography support as defined by PKCS#5. Both are described in the significantly updated Security document, which features two new chapters, “Hashes and MACs” and “Random Number Generators.”

We have also implemented a long overdue optimization of RSA signing and decryption based on "Chinese remainder theorem" (CRT), which makes these rather expensive operations about 3 times faster. RSA key generator now allows to control selection of the public parameter e.

We've made some API changes in the DH algorithm. We renamed #computePublicValue and #computeSharedSecretUsing: to #publicValue and #sharedSecretUsing:. The deprecated selectors are still available for backward compatibility. Also, it is no longer necessary to explicitly invoke #generateP on DHParamaterGenerator. The #p, #q accessors will trigger generation automatically if necessary.

We have also improved management of secure random generators in simplified API calls that do not require the user to maintain a generator instance. This concerns all algorithms that employ random number generation (DH, DSA, key generators). Previously these calls would create a new generator each time. This practice has some serious security drawbacks because it effectively reduces security of these generators to quality of the auto-generated seeding. We are now caching these generators for all the algorithms that use them (see implementors of #defaultRandom) to make better use of unique qualities of these generators.

As was mentioned earlier, there is a new chapter in the Security documentation dedicated to this topic. We highly recommend that anybody, who is serious about security of an application, gets at least basic understanding of the issues involved in this important aspect of cryptography.

Opentalk

Opentalk Namespace

The Opentalk namespace is no longer imported by the Smalltalk namespace. This is proper for add-on packages, but applications using Opentalk will need to either employ imports or refer to Opentalk classes using full dotted names, e.g., Opentalk.RequestBroker. The impact on most applications should be minimal; however, custom Opentalk extensions may need to import the whole Opentalk namespace.

This change also entails that Transcript expressions, including the name of an Opentalk class without a preceding Opentalk, will raise the usual dialog for an unknown identifier. Please use a workspace rather than the Transcript for *ad hoc* experiments with Opentalk code. The workspace is more flexibly namespace-aware.

System Events

Brokers now receive system events through OpentalkSystem, a subclass of Subsystem, the new framework for system events. Brokers may now be explicitly configured to either:

- always restart after an image snapshot or shutdown,
- restart only if they had been running prior to the snapshot or shutdown, or
- never restart.

The new restart machinery is implemented in class RestartProtocol, a new, abstract subclass of GenericProtocol. All of the “outer” Opentalk components, such as request brokers and adaptors, which regulate the startup and shutdown of their subcomponents, and usually elect to subscribe to system events, now inherit from RestartProtocol.

Request Dispatch

Request dispatch is now configurable. An instance of class RequestDispatcher regulates the number and the priority level of the worker processes that are forked to handle incoming requests. Three basic request dispatchers are provided:

standard
: A new worker process is forked, at the worker process priority level, for each incoming request. This is the default dispatch strategy, and it is the strategy used in all previous releases of Opentalk.

pool
: A shared queue of incoming messages and a process pool of configurable size are used to constrain the number of forked worker processes. This method of dispatch is critical for the users of ObjectStudio Opentalk, which spawns native threads on Windows to handle incoming requests. There, any significant increase in the number of threads rapidly decreases the duration of the time-slice allocated to any thread, which very nearly freezes the system.

high-low
: The dispatcher forks worker process at one of two configurable priority levels, using a configurable discrimination block that takes the incoming remote request as a parameter.

Users may extend the RequestDispatcher hierarchy with dispatchers of their own design. We would like to hear of particularly useful ones.

As a part of adding support for configurable request dispatch, the dispatch API was refined, the instance variable requestDispatcher was added to class BasicObjectAdaptor, and the instance variable workerPriority was moved from class Transport to class RequestDispatcher. Parallel changes were made to the configuration classes associated with these components.

Scheduling

Imagine a burst of incoming remote requests, that is bi-modal with respect to execution time. On the server, it will fill the process queue at the Opentalk worker priority level with a randomly ordered, but nearly equal, mix of processes that are either long-running (i.e., measured in minutes) or short-running (i.e., measured in milliseconds). In this case, under the default VW scheduling policy (and assuming that none of these processes employ yield, are regulated by semaphores, or the like), short-running requests will not run until after any long-running requests that precede them on the queue terminate. The short-running requests will have a server-side waiting time far in excess of their execution time. Some feel that this is defective and sub-optimal behavior, even though it has the positive, and often sought, property of preserving the order of the incoming requests in the order of the outgoing replies. They feel that Opentalk should explicitly support time-slicing.

To address this concern, Opentalk now provides a small set of optional, extensible, priority-level schedulers. A priority-level scheduler is one that takes over the scheduling of the processes at a single, specified priority level, e.g., the one at which Opentalk request brokers spawn worker processes. These schedulers should be used with extreme caution, and a clear understanding of their dangers and limitations.

This change discharges the limitation about highly bi-modal request streams noted in Known Limitations in the Load Balancing chapter of the *Opentalk Communication Layber Developer's Guide*.

The schedulers are implemented using a high-priority process that, at a set interval, rearranges the quiescent processes at a single, target priority level. They add scheduling overhead to the server-side load. All schedulers are configured, created, started, or stopped separately from other Opentalk components. A priority-level scheduler will affect every request broker, and any other system component, that forks processes at the priority level it targets.

Two schedulers are provided

LotterySchedulingPolicy

This lottery scheduler assigns each process at the target priority level some variable number of lottery tickets, then randomly selects a winning ticket and moves the process with the winning ticket to the head of the queue. The number of tickets assigned to a process—and this is what affects its odds of "winning"—is a function of the

number of times that process has been at the head of the queue in the past. Users may devise lottery schedulers that assign tickets on the basis of other properties. It is a flexible scheduling model.

CyclicSchedulingPolicy

This round-robin scheduler always moves the first process on the designated queue to the back of that queue.

The round-robin scheduler best demonstrates one of the many pitfalls involved in employing an alternative scheduling policy. In the case of the bi-modal request stream described above, suppose that the interval at which the high-priority round-robin scheduling policy is executed is between the mean execution time of the long-running and the short-running processes created to service the request burst (e.g., the interval is measured in seconds rather than either minutes or milliseconds). In that case, the short-running processes (of millisecond duration) are nearly guaranteed to complete execution before any of the long-running ones (of minute duration), but all the long-running processes will tend to complete after a time that is the sum of all of their individual execution times. This is far beneath ideal.

Users are free to create priority-level schedulers of their own, and we are interested in hearing about particularly useful ones. However, please use or deploy such schedulers with grave caution, and only after careful experiment with several, sample request streams that typify the actual service loads experienced by your application.

Design Changes in Configurations

Originally, Opentalk configurations were designed to avoid a configuration hierarchy that paralleled that of the configured components. That was achieved by linking configurations to components at the instance level through the 'componentClass' instance variable. We found that this approach did not much reduce the number of configuration classes because of the extent to which configured components had unique configuration parameters. Moreover, the design was less transparent and, at times, inconvenient. Therefore, we have switched to the straightforward approach. The Configuration class hierarchy now parallels the component hierarchy under GenericProtocol.

The instance creation methods of the root configuration classes, such as BrokerConfiguration and AdaptorConfiguration, which used to be of the form

```
TransportConfiguration>>http

    ^HTTPTransportConfiguration new: HTTPTransport
```

now simply return the appropriate configuration instance, as in

```
TransportConfiguration>>http

    ^HTTPTransportConfiguration new
```

All the configuration classes must answer the corresponding component class in response to the message componentClass, implemented in this fashion:

```
HTTPTransportConfiguration>>componentClass

    ^HTTPTransport
```

As a result of these changes, the standard configuration creation pattern

```
(BrokerConfiguration basic
    adaptor: (AdaptorConfiguration asymmetricConnectionOriented
        transport: (TransportConfiguration http
            marshaler: (MarshalerConfiguration soap))))
```

changes to

```
(BasicBrokerConfiguration new
    adaptor: (AsymmetricAdaptorConfiguration new
        transport: (HttpTransportConfiguration new
            marshaler: (SoapMarshalerConfiguration new))))
```

The older configuration style should continue to work without modification, because we have retained the componentClass instance variable for backward compatibility. That said, most configuration classes should no longer need it. Similar observations apply to the configuration class methods new:, componentTypes, and rootConfigurationMetaclass. Note, however, that we are moving away from the old design and may stop supporting it in future releases.

Along with these changes, we are abandoning support for the ability to restore a configuration instance from a literal array (using decodeFromLiteralArray), because many new configuration parameters, like blocks, cannot be effectively captured in a literal array.

New server side error event

We have added a new failure event #sendingReply:in:failedBecause:. This will cause reply sending errors to show up in this new event rather than in #evaluatingMessage:in:failedBecause:, allowing us to distinguish true request evaluation errors and reply sending errors and react appropriately. This has following benefits:

First, a communication failure during remote request execution does not shut down the client connection anymore. Previously these errors would yield just a “Connection Closed!” error on the client side making it harder to debug these circumstances.

Second, when there is an error (not a communication failure) during reply sending, we will generate a different error reply with this error instead. That way a reply marshaling error can be propagated to the client instead of having the client just time out.

Miscellaneous

Several minor improvements were made to the Opentalk code.

- The triggering of transport message events has been refactored.
- The names of the marshaling and unmarshaling selectors of STSTStream have been coordinated more closely.
- Datagram transports no longer shut down in response to protocol errors.
- RemoteObject now implements ifNil: and its congeners.

Also, several minor changes were made to the code to facilitate the port of Opentalk to ObjectStudio and potentially other Smalltalk dialects. A full list of completed tasks can be found in **`doc/fixed_ars.txt`**, under “Opentalk.”

Browser Plugin

The VisualWorks Plugin for VW 7.3 is an ActiveX Control only. Therefore, the VisualWorks Plugin can only run in Internet Explorer. It has not been tested with any other browsers, but it is likely to work with another browser that fully supports ActiveX.

A future release will also include a version of the Plugin that implements the new Netscape NPAPI and will run in browsers such as Netscape, Mozilla and Firefox.

In addition, the Plugin parcel for VW 7.3 is not backward compatible with the old Plugin and PluginBase parcels. The PluginBase parcel is now obsolete. If you have an existing custom Plugin image you must rebuild the image with the new parcel. You must also compile and test your application parcel, even if you are using the generic image. The API internal to the AppletModel class has changed very little, so it is likely your

application will not need modification except if you make significant use of GET/POST, especially if you POST directly from a file, which is no longer supported.

Note also that the Plugin requires Unicode support. The Plugin will not run in a browser on those platforms which do not provide Unicode support (Windows 95/Me/98). If your Plugin Applet must support a browser client on one of these platforms please consult the Microsoft documentation concerning the Microsoft Layer for Unicode, and the instructions for including this translation layer in an ActiveX Control.

Instructions for the source code changes required to build a custom version of the plugin are in **`PluginActiveXDev.pdf`**, provided with the source code, not in **`VWPluginReadme.txt`** as stated in the manual.

Application Server

Better handling of encodings

The most significant change in this release is much more robust handling of non-ascii characters, encoded in various character sets. This has several aspects. One is that the Locale (which controls things like formatting of dates, numbers, and so on) is now treated separately from the character set to be used to encode the request and the response. Another is that we now attempt to detect the encoding used by an incoming HTTP request. There is no way to reliably detect this, because the relevant specs only specify that characters not in the allowable subset of the ISO-8859-1 character set will be "URL-encoded" as one or more strings of the form "%HH". Unfortunately it does not specify the encoding with respect to which these strings are interpreted, and different browsers have different behaviour. The emerging standard is to interpret these sequences as UTF-8. However, many browsers will use the encoding of the page that contained a link, if there is one, and may use an arbitrary encoding if the link is not part of a page.

In version 7.3, we first check if the string is valid in UTF-8. If it is, then we check if it is possible to resolve the resulting string. If it is, we proceed. If not, then we check whether the request has a session cookie, and if it corresponds to an active session that we control. If so, and that session has an encoding, we try resolving the string using that encoding. If it fails, or if there is no session, then we fall back to a default encoding. These behaviours can be controlled, and may be turned off, via the "Character Sets and Locales" setting page in the Web settings.

Form Data and Non-ASCII Characters in Web Toolkit

Form data submitted from web pages has issues with different character encodings. For ASCII data there is no problem, but if data is outside that range, it has to be encoded. This uses the URL-encoding scheme, and out of range characters are represented using %HH notation. Unfortunately, the particular encoding is not specified, so it is up to the server to know how to decode these characters. It is highly recommended, when creating forms, to force the browser to use a known character set by including an accept-charset parameter in the form definition.

The server also needs to know which character set is to be used. This is controlled by the setting **Charset for Form Data** on the **Character Sets and Locales** settings page. See also WebToolkitSettings>>formEncoding. This defaults to utf-8, which has the advantage of being a universal character set, so all characters can be represented. However, it's unlikely to be the default character set of a browser for form data, so it's important that the form specify a character set. If you know what character set your client browsers are likely to use, you can switch this setting and avoid needing to set the accept-charset on the form. Be aware, however, that even within a particular Locale, different clients may be using different character sets, and they may differ only in a few characters. For example, in the US-English locale, the ISO-8859-1 character set (common on Unix) mostly overlaps with the Microsoft Code Page 1252, but a few characters are different. The most common cause of problems are the Microsoft quote characters.

Remove Old Servers

In version 7.2 we introduced Opentalk-based servers to replace the old TinyHTTP and IPServer. The older servers were, however, left in the image for backward-compatibility, and in case of problems with the new ones. The older servers have now been removed.

FastCGI Removed

The FastCGI gateway has been removed. It was not well-maintained, and for Apache connectivity it was much more difficult to install and configure than either the CGI or Perl gateways, and did not offer better performance. In particular, the Perl gateway has proven to perform very well, making FastCGI unnecessary. Many Apache users have also found it unnecessary to use a gateway at all, and simply to configure a reverse proxy to the VisualWorks HTTP server.

Bug Fixes

A number of small bug fixes have been made, notably in JSP handling, making server restart with the Opentalk servers consistent with the older behaviour, and in VisualWave rendering.

Headless Changes

The introduction of Subsystems has made significant changes to the Headless functionality, mostly simplifications. For the most part, these should be fully backward-compatible. One non-compatible change is that the old, short, filenames for headless functionality, which have been deprecated for some time, have been removed. So, for example, you must now use **`headless-startup.st`** as the startup file name, rather than the old **`hlstrc.st`**. Also note the presence of various command-line options like -filein, which allows loading arbitrary files. In addition, the becomeHeadfull functionality should be noticeably more robust, although users are still not encouraged to rely on it.

ISAPI Gateway Improvements

Backward-Compatibility

The current Web Application Server parcels *are not* compatible with previous VisualWorks versions, because they rely on some of the new features in Opentalk, which in turn rely on new base system features.

ISAPI Gateway Improvements

There have been various improvements to the ISAPI gateway. Most notably it now statically links to the appropriate versions of the Microsoft DLL's it requires, which makes it much more reliable in different configurations and operating system versions.

The previous version had problems both on some installations of Windows XP and on Windows Server 2003. The current version has passed basic testing on Windows 2000, Windows XP (with and without Service Pack 2) and Windows Server 2003.

Authentication

We have observed some issues with configuring authentication, where the steps vary between different versions, and even different locales of VisualWorks. For example, to have IIS pass Basic Authentication requests on to VisualWorks, the Basic Authentication setting must be off in North American Windows, but the same setting must be on in Japanese Windows.

Installer Framework

The Installer Framework has moved from goodies to the packaging directory. This is the framework used to create the VisualWorks installer. Use instructions for the installer are provided in *Install.pdf*.

The VWInstallerFramework parcel provides the basic functionality for the installer, while the VWInstaller parcel serves as an example of customizing this framework for an individual company and product. The installer application is a wizard with a set of pages that are displayed in sequence. Creating a custom installer is largely a matter of changing the **`install.map`** file for that installation. See the **`install.map`** files on either the Commercial or Non-commercial CDs for examples. These can be hand-edited to suit your particular installation needs.

Customizing the install.map File

Dynamic Attributes

The first item in this file is a dictionary containing version information about the particular distribution to be installed. Edit this section as appropriate for your needs. Many attriburtes are self explanatory, but others may require some explanation.

#defaultTargetTail
: The default name of the installation subdirectory, which the user can change at install time.

#imageSignature
: Used for updating VisualWorks.ini file at install time (auto update of this file is currently a no-op).

#installDirectoryVariableName
: The name of the system variable (or registry key) representing the installed location of the product. For VisualWorks, this is $VISUALWORKS. This can be changed as necessary.

#mapVersion
: This can be used by the installer to identify older or newer install.map formats.

#requiresKey
: Setting this value to true will display the KeyVerifierPage, and will only proceed with the installation once a proper product key has been supplied by the user. VisualWorks installations no longer require this, but the feature remains for those who want it.

#sourcePathVariableName

The name of the system variable (or registry key) representing the location from which the product was installed. For VisualWorks, this is $SOURCE_PATH. This can be changed as necessary.

#variablePath

The path in the Windows registry to use for setting variables on that platform (see Win95SystemSupport.CurrentVersion).

There is also a section of dictionary entries with integer keys and string values of the form “VM *”. The integers represent bytes from the engine thumbprint of the running installer, and are used to identify to the installer the name of the default VM component for the platform on which the installer is run.

Components

Each component is listed in **`install.map`** with various attributes. Many of these are self explanatory, but others require some explanation.

#target: #tgtDir

Although the VisualWorks components are all installed to the main installation directory, the framework anticipates that a need might arise for some components to be installed to a different location. The symbol #tgtDir resolves to the installation directory chosen by the user. However, one could add other symbols, along with supporting code, to allow multiple target directories. For example, if the same installer were to install ObjectStudio and VisualWorks, the symbols #osTgtDir and #vwTgtDir could be used if methods by these names were implemented to answer the appropriate directories.

#environmentItems:

These represent system variables (or Windows registry entries) to be set when the containing component is installed. In the VisualWorks installation, only the Base VisualWorks component contains these.

#startItems:

These describe the attributes necessary to create a Windows shortcut, such as in the start menu or on the desktop. On Unix these attributes are used to create a small script to launch the newly installed image and VM.

#sizes:

A collection of the uncompressed sizes of all the files in the archive, for determining disk space requirements at install time.

License

The presence of the optional license string in **`install.map`** determines whether the LicenseVerifierPage will be displayed. This string is present in the Non-Commercial installer application, and so the page is displayed, but not in the Commercial installer.

Customizing the Code

The wizard application is called InstallerMainApplication, and the wizard pages are subclasses of AbstractWizardPage. These pages are only displayed when listed in InstallerMainApplication>>subapplicationsForInstall.

Some pages are conditionally displayed, as determined by implementors of #okToBuild. For example, CheckServerPage is only displayed if the server has not yet been checked, or if available updates have not yet been applied. Also, as mentioned earlier LicenseVerifierPage is only displayed if the **`install.map`** to be installed contains a license string.

To change the GUI of either the wizard or its pages, simply subclass and tailor the window or subcanvas spec to suit your needs. Then reference your subclass in #subapplicationsForInstall and it will become part of your installer.

The graphic at the top of the wizard window can be changed by implementing #defaultBanner in a class method of your subclass of InstallerMainApplication.

Once your customizations are done, you can strip your install image from the launcher by selecting **Tools → Strip Install Image**.

Creating Component Archives

The packaging tool (**`goodies/parc/PackingList.pcl`**) that automatically packages our product. However, it is very tailored to our particular build processes, and is not recommended for general use. It runs on a linux box, and creates component archives by first staging all the files in a directory structure and then invoking the following code:

```
UnixProcess
    cshOne: ('tar --create --directory="<1s>" --file="<2s>" --owner=0 --totals
        --verify --same-order <3s>'
            expandMacrosWith: directoryString
            with: fileString
            with: contentString)
```

Note that any Mac files with resource forks must be added to the archive in MacBinaryIII format (*.bin) to be installed properly later.

Local Installations

The scripts and structure of our CDs serve as examples of a working packaged CD. Any archive could be installed from another part of the CD if its #path: attribute is adjusted in the install.map file.

Cincom uses and recommends CDEveryWhere (www.cdeverywhere.com) to create hybrid CDs for distribution that run on Win, Mac, and Unix/linux.

Remote installations

Your wizard subclass should implement #configFileLocation, which answers anFtpURL. This XML file should reside on your server and list the current installer image version, available patches, and available products to install. An example from our NC download site follows:

```
<?xml version="1.0"?>

<configuration>
    <installerImageVersion>'1.1'</installerImageVersion>
    <installerParcelVersions>
        '#()'
    </installerParcelVersions>
    <applicationsToInstall>
        '#(#(''VisualWorks 7.1 Non-Commercial'' ''vwnc7.1'')
        #(''VisualWorks 7.2 Non-Commercial'' ''vwnc7.2'')
        #(''VisualWorks 7.2.1 Non-Commercial'' ''vwnc7.2.1''))'
    </applicationsToInstall>
</configuration>
```

In the above example, the last application listed is 'VisualWorks 7.2.1 Non-Commercial', which is the string that will appear in the drop down list of available versions. The string following that, 'vwnc7.2.1', is the subdirectory on the ftp server which contains the application. This subdirectory is flat, unlike the CST CD directory structure, and contains the **`install.map`** and archive files. The same **`install.map`** file can work unchanged for CD and remote installations. For remote installations, only the tail of the component archive file is used, since it is assumed that the FTP server does not need the deeper directory structure of the CST CDs.

In addition to the default configuration file location hard coded into your wizard class, users can also keep a local config file, named **`installerConfiguration.xml`**, which can list alternate local install sources or remote servers. For example, the following local config file lists two additional servers from which one could install any products available there:

```
<?xml version="1.0"?>

<configuration>
    <additionalConfigFiles>
        '#(''ftp://anonymous:foo@myServer//remoteInstall/
            installerConfiguration.xml''
        ''ftp://anonymous:foo@theirServer//remoteInstall/
            installerConfiguration.xml'')'
    </additionalConfigFiles>
</configuration>
```

This may be useful anywhere frequent installations might be performed, such as a QA or Tech Support computer lab.

Documentation

This section provides a summary of the main documentation changes.

SmalltalkDoc

SmalltalkDoc is a system for generating reference documents from package comments and properties. It is introduced in 7.3 as a preview. See “SmalltalkDoc” for more information.

PDF Documents

Advanced Tools Guide

No changes

Application Developer’s Guide

- Updated for packages/bundles in the base
- Chapter organization was changed, eliminating redundant coverage of trigger-events in the “Application Framework” and “Trigger-Event System” chapters.
- New system event section, covering the new Subsystem class and subclasses, was added to the “Application Framework” chapter.
- Various smaller updates.

COM Connect Guide

No changes

Database Application Developer’s Guide

No changes

DLL and C Connect Guide

No changes

DotNETConnect User's Guide

New document

DST Application Developer's Guide

No changes

GUI Developer's Guide

Various small changes.

Internationalization Guide

No changes

Internet Client Developer's Guide

- Added HTTP Cookie support
- Various small corrections/updates

Opentalk Communication Layer Developer's Guide

No changes

Plugin Developer's Guide

Major revision to cover replaced functionality for IE.

Security Guide

- Added a chapter on Hashes and Messages Digests, including coverage of HMAC.
- Added a chapter on Random Numbers, covering DSSRandom in more detail, with better instructions for generating and maintaining quality seeding.

Source Code Management Guide

No changes

Walk Through

No changes

Web Application Developer's Guide

Minor changes, rearranged chapters

Web GUI Developer's Guide

No changes.

Web Server Configuration Guide

- Updates to CGI relay documentation
- Various updates/changes.

Web Service Developer's Guide

- Added a description of the WSDL Wizard tool
- Various updates/changes.

TechNotes

The following documents in the TechNotes directory have been updated:

ImplementationLimits7x
This document has been added.

Goodies

Dual Monitor Support on Windows

A goodie parcel distributed with 7.3 helps alleviate the menu and window placement on Windows with dual monitors. Eventually an OE fix for dual monitor support is planned. Read its use and limitations in the parcel comment.

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.

Plugin Parcels

The Plugin parcel for VisualWorks 7.3 is not backward compatible with the Plugin and PluginBase parcels from earlier releases. The PluginBase parcel is obsolete.

Net Clients

HttpsClient is now deprecated. Its functionality has been merged with HttpClient.

Security

The selectors #computePublicValue and #computeSharedSecretUsing: are deprecated, but still available for backward compatibility. These are replaced with to #publicValue and #sharedSecretUsing:, respectively.

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.

Base Image for Packaging

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

MQInterface

The **`preview/mqinterface/`** directory contains an interface for IBM WebSphere-MQ (formerly MQ Series). The implementation makes use of the IBM shared libraries. The subdirectory also includes a developer's guide, which describes the architecture and the API and includes a class reference.

The parcel named MQ-XIF takes some time to load, because it must recompile the external interfaces in the correct order.

Unicode Support for Windows

Extended support for Unicode character sets is provided as a preview, on *Windows 2000 and later* platforms. Support is restricted to the character sets that Windows supports.

The parcels provide support for copying via clipboard (the whole character set), and for displaying more than 33.000 different characters, without any special locales.

The workspace included in **`preview/unicode/unicode.ws`** is provided for testing character display, and displays the entire character set found in Arial Unicode MS.

First, open the workspace; you'll see a lot of black rectangles. Then load **`preview/unicode/AllEncodings.pcl`** and instantly the workspace will update to display all the unicode characters that you have loaded. You can copy and paste text, for example from MS Word to VW, without problems.

If there are still black rectangles, you need to load Windows support for the character sets. In the Windows control panel, open **Regional and Language Options**. (Instructions are for Windows XP; other versions may differ slightly.) Check the **Supplemental language support** options you want to install, and click **OK**. The additional characters will then be installed.

To write these characters using a Input Method Editor (IME) pad, load the **`UnicodeCharacterInput.pcl`**.

Menu UI Compatibility

(AR47543)

MenuUICompatibility is a preview parcel that addresses incompatibilities of VW menu bar menus with host OS menu feel requirements. Issues addressed include:

- On Windows, a mouse drag (i.e., mouse button hold and move) to a menu item with a submenu should not close the menu or submenu upon button release.
- On Windows only, menus highlight disabled menu items; other platforms skip past disabled menu items entirely.
- MacOSX menus do not wrap highlight of menu items for key up/down navigation.

- Mac OS8/9 closes all menus upon any key press.
- Unix platforms do not highlight menu items while moving the mouse cursor over an item; the mouse button must be down to do this.
- Except on Unix, submenu item menus open after about a 0.5 second delay when the mouse cursor is over them.
- For the Motif and Windows look, a menu and submenu should remain open after a mouse click and release on a menu item with a submenu.
- On Mac platforms, moving the mouse outside a menu to the right should not close all submenus open.
- On Mac OSX, <Tab> and <Shift><Tab> navigate and open menus right and left in the menu bar.
- On Mac OSX, a menu and all its submenus should not close prematurely if the up or down arrow keys are used to navigate to a menu item with an unopened submenu.
- For MS Windows and Motif menus, a menu item that opens a submenu cannot become a selection itself and then close; only a menu item without submenu may be the menu selection.

Regarding the last item, that a menu item cannot become a selection itself, there are VW users who wish to preserve the ability to assume a menu item with a submenu as selection. By default this behavior has been preserved. To assume the MS Windows and Motif standard behavior, the class variable SubmenuAssumesSelection has been added to Win95MenuController and MotifMenuController. To change to the standard behavior, send an submenuAssumesSelection: message to the class with false as the argument value:

```
Win95MenuController submenuAssumesSelection: false
```

New GUI Framework (Pollock), Feature Set 1

Pollock remains in preview in 7.3.

Featureset one includes a full complement of widgets, and so is a major milestone one the way to a full production release. There is still a lot of work left to be done, however.

Background

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, which we call Pollock.

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.

To this general goal, we add the following more specific goals:

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

Finally, and most importantly:

- The new 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. 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 SpecWrappers, and BorderedWrappers, and WidgetWrappers, and many 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 role. 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 (currently projected for 7.3), 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 an 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. The tests are in the PollockTesting parcel. To load this parcel, you must have both the Pollock and SUnit parcels loaded.

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.

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. Widgets are kinds of panes. There is an AbstractPane class. A Window is also a kind of pane, but it will remain in it's own hierarchy so we don't have to reinvent every wheel. Also, the Screen becomes in effect the outermost pane. Other than those, all panes, and notably all widgets, will be subclassed in one way or another from the AbstractPane.

A *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, which is like our existing layout classes, and defines where it sits in the enclosing pane. It may also 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 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 represents the 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 of pane contents. 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 TextPane could have a separate artist for drawing word-wrapped and non-word-wrapped text. A ComposedTextPane could have one artist for viewing the text composed, and another for XML format. Additionally, the plug-and-play ability of the artist allows for automatically updating 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, together with agents, can be plugged directly into the pane as needed.

Agents interact with the artists and the panes 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 distributors 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 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 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. Beta 2 was included in VisualWorks 7.1, and had several of the basic widgets done: InputField, TextEdit, CheckBox, RadioButton and ListBox.

VisualWorks 7.2 includes Beta 3, which adds DropDownList, Menu, Grid (Table/Dataset combination), DialogWindow, Toolbar, TreeView and TabControl.

The first production release will have all of the remaining widgets done and complete. All of the tools, including a GUI Painter, will be 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. This is hoped to be included in VisualWorks 7.3.

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

The Opentalk preview provides extensions to 7.2 and the Opentalk Communication Layer. It includes a preview implementation of SNMP, a remote debugger and a distributed profiler. The load balancing components formerly shipped as preview components in 7.0 is now part of the Opentalk release.

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

SNMP

SNMP is described in the previous section.

Distributed Profiler

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

Installing the Opentalk Profiler in a Target Image

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

- Opentalk-Profiler-Core

Installing the Opentalk Profiler in a Client Image

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

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

Opentalk Remote Debugger

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

The remote debugger is contained in two parcels.

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

```
RemoteDebuggerClient startMonitor
```

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

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

```
RemoteDebuggerClient stopMonitor
```

The Opentalk-Debugger-Remote-Target parcel loads support for the image that is expected to be debugged remotely. To enable remote debugging this image has to be “attached” to a monitor, i.e., to the image that runs the remote debugger UI. Attaching is performed with one of the

"attach*' messages defined on the class side of RemoteDebuggerService. Use detachMonitor to stop forwarding of unhandled exceptions to the remote monitor image.

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

Opentalk SOAP Header Support

The Opentalk-SOAP-Headers parcel includes classes that support invoking web services with SOAP headers. Opentalk-SOAP-HeadersDemo contains example code.

Server SOAP Header Support

WsdlServiceProvider is the RequestBroker wrapper and holds the request header repository for the specific Wsdl binding port.

Opentalk-SOAP-HeadersDemo includes the the CustomerServer class, which implements an Opentalk server with services that are a collection of instances of WsdlServiceProvider.

WsdlServiceProvider creates an instance of ServerOperationHeaderProcessor for each request. This object holds request header entry state and will call the request or response header entry processors.

ProcessingPolicy is responsible for the soap header verification and unmarshaling. The header entry verification is done as follows:

1. If there are any header entry nodes, check if they are described in the Wsdl schema for this operation. If the soap message is missing header entries from the Wsdl schema, a SoapClientFault message is sent back as the reply.
2. If the header entry targets this soap node and mustUnderstand='1', but there is no a header processor on this server, the MustUnderstandFault is sent back as the reply.
3. If mustUnderstand='0' and there is no a header processor, the header entry is just ignored
4. Header entries that do not target this soap node are ignored

ServerHeaderEntryProcessor is superclass for the Opentalk server specific header entry processors. Subclasses must implement actions on header entries when a request is received or reply is sent.

A specific header entry processor must implement the headerName class method that returns the header node tag described by namespace and type.

Subclasses must implement the following messages:

- headerEntryFor:transport:
- processHeaderFrom: transport:

Client SOAP Header Support

WsdlServiceConsumer is the abstract superclass for all user specific client classes. The class holds a request header repository for the Wsdl binding.

You should create a client class as subclass of WsdlServiceConsumer and implement the following methods:

- binding
- port
- serverUrl

Also, the “public api” category must include all methods necessary to access the web services.

SOAP Header Demo

The header demo can be loaded from the Opentalk-SOAP-HeadersDemo parcel. In this demo, CustomerClient implements the user specific client.

ClientHeaderEntryProcessor is superclass for the user specific header entry processors. The class implements the default action on header entry when a request is sent or response is received.

A specific header entry processor must implement the headerName class method that returns the header node tag described by namespace and type. The header entry processors are optional. The default header entry processor adds necessary header entries from the WsdlServiceConsumer repository to the request. If there is no header entry but the Wsdl schema describes a header entries for the operation, the MissingRequestHeader exception will be raised. When the response is received the default

header entry processor does not perform any action on this header entry. If the response header is missing some header entries the MissingResponseHeader exception will be raised.

While sending the request, a specific header entry processor can override the default behavior by reimplementing the method processHeaderFrom:transport: in ClientHeaderEntryProcessor.

When the response is received, the specific header entry processor can add some processing to the header entry in the method process:reply:transport: in ClientHeaderEntryProcessor.

The header entry can be added to the request in two ways:

- In the client repository:

```
client := Smalltalk.CustomerClient new.
client start.
(client headerFor: #AuthenticationToken)
      value: ( AuthenticationToken new
           userID: 'UserID';
           password: 'password';
           yourself).
```

- In the header entry processor:

```
processHeaderFrom: aSOAPRequest transport: aTransport
      self headerEntry value isNil
          ifTrue:
              [(aSOAPRequest headerFor: #AuthenticationID)
                  value: (WebServices.Struct new
                      id: 'ID#1234';
                  yourself)]
          ifFalse: [ super processHeaderFrom: aSOAPRequest
              transport: aTransport ]
```

The result of the client service invocation can be set up to return either:

- A value, which is the body value. This is the default.

  ```
  SOAPMarshaler defaultReplyReturnValue: #value.
  ```

  Access to response header will available only in client header entry processors.

- An envelope, which is instance of WebServices.SoapEnvelope:

  ```
  SOAPMarshaler defaultReplyReturnValue: #envelope
  ```

  Having the envelope as the result allows to get access to response header and body.

WsdlServiceConsumer creates an instance of ClientOperationHeaderProcessor for each request. This object holds request header entry state and will call response header entry processors when the response arrives.

Testing and Remote Testing

The **`preview/opentalk`** subdirectory contains two new parcels, included for those users who expressed an interest in the multi-image extension to the SUnit framework used to demonstrate the Opentalk Load Balancing Facility:

- Opentalk-Tests-Core contains basic extensions to the SUnit framework used to test Opentalk. Version number 73 6 is shipped with this release.
- Opentalk-Remote-Tests-Core contains the central classes of the remote testing framework and some simple examples. Version number 73 9 is shipped with this release.

The framework these packages implement is known to have defects and is evolving. Future versions will differ, substantially.

The central idea behind the framework is that since SUnit resources are classes, there is no reason why references to remote classes cannot be substituted for them in a test case.

The are two central classes in the framework.

OpentalkTestCaseWithRemoteResources
: This is the superclass of all concrete, multi-image test cases. It contains an instance variable named 'resources' that is populated with references to remote resource classes. The references are constructed from the data returned by the method resourceObjRefs, which any concrete test case must implement. The class has a shared variable named CaseBroker that contains the broker in which the resource references are registered. This request broker is the one used by all multi-image test cases to communicate with remote resources.

OpentalkTestRemoteResource
: This is the superclass of all concrete remote resources. It has a shared variable named ResourceBroker that holds the broker through which test resources communicate with test cases. Concrete resources register themselves with this broker, using their class name as an OID, so that test cases may programmatically generate references to them.

Since multi-image tests usually involve resources that start up brokers and exchange messages of their own, care must be taken in any test to determine that the communication exchange under test has completed before any 'assert:'s are evaluated. Also, since the exchange between resources may be complex, the assert: messages are usually phrased in terms of the contents of event logs. Much use is made of the Opentalk event and event logging facilities. Test may create event logs of their own, or analyze the remote event logs created by a remote resource.

The current scheme assumes that there will be only one resource per image, but you may construct a resources with arbitrary complexity.

The drill for configuring a multi-image test is now overly complex, because port numbers are derived from the suffix of the image name, expected to consist of two decimal digits. Port numbers are also hard-coded in the method resourceObjRefs. This is the wrong way to do things that we intend to move to a scheme where the test case image starts its broker on a well known port, and resource images register with the test case image on startup.

That said, the current drill goes as follows. The essentials are also discussed in the class comment of CaseRemoteClientServer.

1. Make sure that the machines you intend to use are not already listening on the default ports used by the multi-image testing framework. The CaseBroker, if you follow our recommendations, will come up on port 1800, and resource brokers will come up in the range 1900-1999. If your machines are already using these ports, alter the class-side method basePortNumber in OpentalkTestCaseWithRemoteResources or OpentalkTestCaseRemoteResource, as appropriate. The following directions will assume that you did not need to change an implementation of basePortNumber.
2. Write your resource class or classes. You may use any of the concrete classes under ResourceWithConfiguration as models.
3. Write your test case class. You may use any of the concrete classes under CaseRemoteClientServer as models.
4. Save your image.
5. Remind yourself of how many resources your test case employs. For example, class CaseRemoteClients1Servers1 requires three images. You can check this by examining its implementation of resourceObjRefs. Two references are set up, one for a client and one for a server. The third image will be the one that runs the test case. So, if your image is

named **otwrk.im**, clone copies of it now, named **otwrk00.im**, **otwrk01.im** and **otwrk02.im**. All the image names must end in two digits. The name ending in “00” is conveniently reserved for the test running image, making its broker come up on port 1800. All the images derive the port of their broker from their image name. In this case, the resource images will start their brokers on ports 1901 and 1902.

6 After saving the images, reopen them, and start the relevant brokers. Remember that in the test case image you only want to start the CaseBroker. In the resource images, you start their ResourceBroker. The class-side protocol of OpentalkTestCaseWithRemoteResources and OpentalkTestCaseRemoteResource both contain start up methods with useful executable comments, if you like doing things that way. (You will use only one image, and start both kinds of brokers in it, only when you intend that everything run in the same image. And that setup is very useful in debugging.)

7 Run your tests, from the test case image, and run them one at a time. The framework has known difficulties running a test suite.

If you ever find that your event logs show record of, say, 50 messages, when your test only sends 30, then the preceding test run—which you probably thought you had successfully terminated by, say, closing your debugger—was still going strong. Clean up as necessary and start again.

Miscellaneous

The listeners now bind to specific network interface if an explicit IP address is specified (as opposed to the default IPSocketAddress>>thisHost).

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.

OpentalkCORBA

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

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

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

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

- supports IIOP 1.0, 1.1, 1.2
- defaults to IIOP 1.2
- does not support value types
- does not support Bi-Directional IIOP
- doesn't support the NEEDS_ADDRESSING_MODE reply status
- system exceptions are currently raised as Opentalk.SystemExceptions

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

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

- there is a new IIOPTransport and CDRMarshaler with corresponding configuration classes
- these transport and marshaler configurations must be included in the configuration of an IIOP broker in the usual way
- the new broker creation API consists of the following methods
  - #newCdrIIOPAt:
  - #newCdrIIOPAt:minorVersion:
  - #newCdrIIOPAtPort:
  - #newCdrIIOPAtPort:minorVersion:
- IIOP proxies are created using Broker>>remoteObjectAt:oid:interfaceId:
- there is an extended object reference class named IIOPObjRef
- the LocateRequest capabilities are accessible via
  - Broker>>locate: anIIOPObjRef

    - RemoteObject>>_locate
- LocateRequests are handled transparently on the server side.
- A location forward is achieved by exporting a remote object on the server side (see the example below)

Examples

Remote Stream Access

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

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

“Locate” API

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

```
| broker |
broker := Opentalk.BasicRequestBroker newCdrIiopAtPort: 4242.
broker start.
[    | result stream oid proxy found |
     found := OrderedCollection new.

     "Try to locate a non-existent remote object"
     oid := 'stream' asByteArray.
     proxy := broker
         remoteObjectAt: (
             IPSocketAddress
                 hostName: 'localhost'
                 port: 4242)
         oid: oid
         interfaceId: 'IDL:SmalltalkTypes/Stream:1.0'.
     result := proxy _locate.
     found add: result.

     "Now try to locate an existing remote object"
     stream := 'Hello World' asByteArray readStream.
     broker objectAdaptor export: stream oid: oid.
     result := proxy _locate.
     found add: result.
     found
] ensure: [ broker stop ]
```

Transparent Request Forwarding

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

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

Listing contents of a Java Naming Service

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

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

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

List Initial DST Services

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

```
| broker dst |
broker := Opentalk.BasicRequestBroker
        newCdrIiopAtPort: 4242
        minorVersion: 0.
broker start.
[   dst := broker
        remoteObjectAt: (
            IPSocketAddress
                hostName: 'localhost'
                port: 3460)
        oid: #[0 0 0 0 0 1 0 0 2 0 0 0 0 0 0 0]
        interfaceId: 'IDL:CORBA/ORB:1.0'.
    dst listInitialServices
] ensure: [ broker stop ]
```

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, Image-level support for printing and creating NaNs and Infs has been kindly contributed by Mark Ballard and is in **`preview/parcels/IEEEMath.pcl`**. To use this facility load the IEEE Math parcel and start the engine with the **`-ieeefp`** 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.

64-bit VM

This release introduces a beta of native 64-bit VisualWorks on the first platform, linuxx86_64, which is 64-bit Linux running on the AMD x86-64 architecture. Note that AMD x86-64 is, for these purposes, compatible with Intel's EMT64 architecture. To use the 64-bit system, you must transform a 32-bt image into a 64-bit one using the ImageWriter and run it on a 64-bit engine. The ImageWriter can be found in **`$(VISUALWORKS)/preview/64-bit/ImageWriter.pcl`**. The x86-64 engines are in **`bin/preview/linuxx86_64`**. The ImageWriter parcel's comment includes instructions for transforming images. You can find more information in **`bin/preview/linuxx86_64/readme.txt`**.

The 64-bit system is changed little from the 32-bit system. The most significant change is the addition of an immediate double floatng-point type, SmallDouble, which should reduce the memory footprint and

increase the performance of double floating-point intensive applications (see the class comment). Also, in the 64-bit system SmallIntegers are in the range -2 raisedTo: 60 to (2 raisedTo: 60) - 1.

In general Smalltalk code should "just run" unchanged. For example, parcels can be read, written, and freely interchanged between 32-bit and 64-bit systems. But images can only be moved between the two widths using the ImageWriter.

We expect to move linuxx86_64 to a fully-supported state in 2005 and to add a number of additional 64-bit platforms. The highest priorities currently are:

- HP-UX PA-RISC 64-bit
- Solaris SPARC 64-bit
- AIX PowerOC 64-bit
- Windows x86-64/EMT64 64-bit when it emerges from beta

Please contact Cinsom support to voice preferences for any other potential 64-bit platforms.

GLORP

GLORP (Generic Lightweight Object-Relational Persistence) is an open-source project for mapping Smalltalk objects to and from relational databases. While it is still missing many useful properties for such a mapping, it can already do quite a few useful things.

> **Warning:** This is UNSUPPORTED PREVIEW CODE. While it should be harmless to use this code for reading, use of this code to write into a Store database MAY CAUSE LOSS OF DATA.

GLORP is licensed under the LGPL(S), which is the Lesser GNU Public License with some additional explanation of how the authors consider those conditions to apply to Smalltalk. Note that as part of this licensing the code is unsupported and comes with absolutely no warranty. See the licensing information accompanying the software for more information.

Cincom currently plans to do a signficant overhaul of the current database mapping facilities in Lens, using GLORP as one component of that overhaul. GLORP is included in preview as an illustration of what these future capabilities might include.

Included on the CD is the GLORP library, its test suite, some rudimentary user-provided documentation, and some supplementary parcels. For more information, see the $VISUALWORKS/preview/glorp directory. Note that one of these includes a preliminary mapping to the Store database schema.

SmalltalkDoc

SmalltalkDoc is a Smalltalk application for creating and presenting comprehensive XML documentation for VisualWorks Smalltalk. It consists of a new System Browser tool for creating and editing documentation, a content management facility (a database) for managing snapshots of system documentation, and a web application for presenting documentation. SmalltalkDoc will be used to document VisualWorks components, and it can be used to document customer applications.

For additional information, refer to the HTML documentation in **`preview/SmalltalkDoc/Docs/Overview.html`**.

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®