Cincom

VisualWorks®

7.2 Release Notes

P46-0106-07

Part Number: P46-0106-07

Software Release 7.2

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 © 1999–2003 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 7.2

These release notes outline the changes made in the version 7.2 release of VisualWorks. Both Commercial and Non-Commercial releases are covered. These notes are not intended to be a comprehensive explanation of new features and functionality nor are they intended to be used in lieu of the product documentation. Refer to the VisualWorks documentation set for more information.

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

Product Support

Support Status

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

http://www.cincomsmalltalk.com:8080/CincomSmalltalkWiki/Cincom+Smalltalk+Platform+Support+Guide

Product Patches

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

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

ARs Resolved in this Release

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

Additional ARs may be discussed in individual sections of these release notes.

Outstanding ARs and limitations are noted throughout these release notes, as appropriate.

Items Of Special Note

SGI VM delayed

Due to licensing issues, delivery of the SGI virtual machine has been delayed for a few weeks from the initial release. It will be made available as soon as possible.

Internationalization

A major effort has been made to complete the internationalization of the code. The base and all components now use UserMessages where they had Strings. Refer to "Internationalization" in chapter 2 for more information.

Class, Name Space, and Shared Variable Creation Dialogs

Definition dialogs have been added for creating class, name space, and shared variables. These provide a number of conveniences, including the selection of superclasses and containing name spaces.

The original definition templates are still available for use, and are still required for modifying definitions.

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.

Limitations listed in other sections

- GUI: Known Limitations

2

VW 7.2 New and Enhanced Features

This section describes the major changes in this release.

Virtual Machine

bin/ Directory Organization

In 7.1, debug and assert engines were moved into **`extra/`** subdirectories under each platform. For this release, they are in **`debug/`** and **`assert/`** subdirectories, respectively.

Compressed images

Loading compressed image files is now supported on all VisualWorks 7.2 virtual machines.

To create a compressed image, load the Image Compression parcel (**`packaging/ImageCompression.pcl`**), then select **File → Compress Image File** in the Launcher.

For the API to access the compression library, see "ZLib Compression Streams" below.

Dynamically loadable user primitives

The engines support plug-ins by providing for dynamically loadable user primitive code. This is done using a pragma modeled after the Squeak implementation. Additional information will be made available on a wiki page, accessible from the Cincom Smalltalk Developers page.

The Windows engines are organized differently as a result. A very small **`vwnt.exe`** executable (57 KB) now loads the bulk of the virtual machine code from a DLL, **`vwntoe.dll`**.

Creating DLLs that use the virtual machine API has now changed slightly on Windows and MacOS 8/9. Refer to the wiki page for details.

Close VW on Windows shutdown

Added in 7.1 but not noted was a change that allows VW to exit when Windows shuts down. This was previously prevented.

Windows shutdown events are delivered to the VisualWorks image as a QuitSystem event. By default this simply results in ObjectMemory being sent quit. This can have a bad effect in some cases, such as if an external resource (e.g., a database) is open and abruptly cut off. There are a couple of ways to handle this.

- You can put a dependent on ObjectMemory and have an update method that watches for #aboutToQuit (which would also be triggered every time the image is quit).
- You can modify InputState>>#send:eventQuitSystem: to provide some special hook only invoked on an exit event.
- Instead of modifying #send:eventQuitSystem:, you can add your own quit method to InputState, such as #send:myEventQuitSystem:, and then at system startup, you can go to InputState.EventDispatchTable and put your own message at position 19.

Alternatively, you can revert to the old behavior and block VisualWorks from exiting. By default, the InputState class method setDispatchTableForPlatform registers true with the acceptQuitEvents object. To prevent VisualWorks from being prematurely shut down, set this to false instead.

Itanium on HP-UP

The HP-UX engine appears to function on Itaniums running HP-UX-11i, running in dynamic translation mode, but does so very slowly. Consequently, we *do not* support VisualWorks on this platform at this time.

If you do want to run on the HP-UX Itanium, then you should use the maximum available code cache, either by using the **`-z 16m`** flag, or by doing:

```
ObjectMemory sizesAtStartup: #(1.0 1.0 1.0 1.0 14.0 1.0 1.0)
```

and snapshotting.

Base system

Internationalization

In 7.2, many literal strings in the system code have been converted to UserMessages, in order to make it possible to translate the system's UI into other languages (such as German or Japanese) without modifying the source code.

As a result of this, occasionally you may ask system code for some piece of information which you expect to be a String, such as the label of a window, and get back a UserMessage instead. UserMessages do not currently understand all of the messages that Strings do, and so when you use the returned value, you may get an error. This can be solved by sending the return value asString before using it, if you know that it will always be either a String or a UserMessage.

If you want to start converting your own literal strings to UserMessages, so that your code can be translated into other languages, see especially chapter 2 of the *Internationalization Guide*. Remember that not all system code (and probably not all of your own code) will be expecting to be passed UserMessages in place of strings, so sometimes you may need to send your UserMessage an asString message before passing it to another method.

Line end detection

(AR 46063) In 7.1, we introduced the provisonal #adoptContentsLineEndConvention so that users can tell streams to adopt the line end convention of the stream that they are reading. In 7.2, this has been replaced with #lineEndAuto. So, if you had code that relied on the provisonal code you should convert over to use the #lineEndAuto now. For example if you had code like:

```
someStream lineEndTransparent.
someStream adoptContentsLineEndConvention
```

this should now be replaced with:

```
someStream lineEndAuto.
```

PostScript support configuration

To simplify PostScript printing configuration, support has been added to enable color printing and to set the landscape orientation. A page has been added to the Settings Manager, **Printing → PostScript Settings**, to set these options:

Color printing - Enable color printing to PostScript printers that implement PostScript Level 2. If you attempt to print PostScript generated for such a printer on a Level 1 printer, you will not get any output.

Invert landscape position - Rotate landscape orientation 180 degrees.

In addition, a few changes have been made to font and character support:

- The unicode for the Euro has been added to the supported character set for PostScript printing.
- The following fonts are not generally supported by PS 3 printers, and so have been removed:
    - Helvetica-Light
    - Helvetica-Black
    - Helvetica-BlackOblique
    - Helvetica-LightOblique

There are also a number of font families that are supported by PS 3 printers but are not included in the VisualWorks PostScript font set. These can be purchased by the user and installed using the PSFontPrinting goodie.

Implemented #= and #hash in Method

(AR 45300) There were cases where two instances of Message, though equivalent, appeared unequal. The solution was to implement #= and #hash in Message.

This change may conceivably break customer's subclasses of Message, since instances of those subclasses, which were formerly not =, may now be = without paying attention to instance variables that the subclass added to Message. For example, if MessageSend had not already implemented #=, and inherited the new implementation from Message, then two MessageSends that have the same selector and arguments would be = even if they had different receivers.

ZLib Compression Streams

The zlib library is included in the object engine to enable image decompression on loading. ZLibInterface provides the interface to the library. GZipReadStream and GZipWriteStream use ZLibInterface to provide the ability to decompress or compress a stream.

Load the Compression-ZLib parcel, in the Application Development group in the Parcel Manager, to access this functionality. The API classes are DeflateStream and GZipWriteStream for writing, and InflateStream and GZipReadStream for reading. Using these uses the usual internal stream protocol.

Browse the code comments for examples.

GUI Development

GUI Painter Tab ordering

Tab-ordering and Z-ordering (back to front) in the UI Painter have been confusing, but have now been made consistent. Moving a widget “back” puts it earlier in the tab order; moving it forward puts it later in the tab order. Tab order is still top to bottom, as the widgets are shown in the GUI Painter Tool widget list.

New cursor

(AR 46100) A new pointing finger cursor has been added and may be accessed as follows:

```
Cursor fingerPointer
```

Trigger event lookup optimization

(AR 46007) When looking up events, the event triggering and accessing methods of Object always use at:ifAbsent:. However, it is known that this lookup can be very slow when there are no events registered for the receiver. To improve speed, simple tests have been added to skip lookup when the receiver event table is nil.

New selection widget contents setters

(AR 45331) SelectionInList and MultiSelectionInList have two new methods that may be used to set their contents: setList:selecting: and refreshList:. The list, dataset, tab control, and notebook widgets derive their models from either SelectionInList and MultiSelectionInList. Any of these widgets may have their contents and selection both set at once by sending setList:selecting: to its model. When used, setList:selecting: attempts to keep a selection visible and seemingly immobile during a change in list. Similarly, the refreshList: message will update a list and attempt to keep the first selection held over of items in the new list visible and seemingly immobile. The new example parcels RefreshSelectionList and SetSelectionList demonstrate these methods.

Known Limitations

Sawfish and MultiProcUI

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

Tools

Class, Name Space, Shared Variable creation dialogs

When you select **Class → New → Class**, **Class Extension**, **Name Space** or **Shared Variable**, you are now presented with a dialog for creating the item, rather than a blank template. These are described in the *Application Developer's Guide*, but are also pretty self-explanatory.

The templates are still available and can be used. For the class template, select a category, parcel or package, then edit the displayed template as before. For a shared variable, select a class or name space and a protocol for the variable, and click the **Shared Variable** tab. For a name space, copy and paste a name space definition into the code pane and edit it.

The **Class → New → Class Extension…** item (available when Store is loaded) prompts for a class and then creates an extension of that class in the current package. Note that in the previous versions of VisualWorks, this item used to create an extension of the current class in the package selected by the user. The new behavior is consistent with that of the other items of that menu. The old behavior is now available as the menu item **Class → Extend in Package…** .

The **Class → New → Class…** command is also available as a toolbar button.

Find dialogs

Find dialogs have been enhanced to provide a filtered list of options (e.g., classes or methods) based on initial entries in the input field. The filtering is not case-sensitive.

Advanced Tools

Profiler updates

Some updates were made in 7.1 to prepare for future enhancements, such as headless, distributed profiling. These changes are now reflected in the *Advanced Tools User's Guide*.

Benchmarks

The benchmarks browser has been modernized, with the transcript integrated into the test selector.

WebService

SOAP Header support

Headers are now supported for SOAP messages, and is described in the *Web Service Developer's Guide*. No automatic handling for headers is provided yet.

Net Clients

HttpURL change

The behavior of HttpURL>>readStream has being changed to return a ByteString (AR46533). To return a ByteArray, use HttpURL>>binaryReadStream.

Security

New Document

This release includes a new document covering VisualWorks security features, the *VisualWorks Security Guide*.

New namespace and directory

The library is becoming large and mature enough to be granted status of a stand-alone component. The algorithms it provides are applicable far beyond networking protocols, so we have moved all the code into a new namespace called Security. All the parcels have also been moved to a new directory, **`security/`**.

Hashes

There have been significant changes in the implementation of hash algorithms, including their public APIs. The effort was motivated by a desire to support “running” hash computations, in which data is handed in piece by piece, rather than all at once. (These are usually referred to as “Update” and “Final” operations in other libraries.)

This required an extensive refactoring of the hash algorithms, which also allowed us to add implementation of SHA256 with very little effort and should make implementation of additional SHA versions (384 and 512) much simpler as well. In that process MD5 was sped up almost three times, but the speed increase on SHA-1 algorithm was insignificant. All the algorithms were cleaned up to generate no garbage, with the exception of register backups for final digest value computation. That could be eliminated as well, but since that is fully under control of the developer it seems unnecessary. This should make the implementation scale better when processing large amounts of data. The code of the algorithms was updated to match the specification descriptions as much as possible (instance variable names, selectors, etc.).

The API is now split into two layers. The higher level (protocol “services”) provides methods that allow to obtain the digest with a single message. It is very similar to the old API, except that we have deprecated the former “integer” bias and opened up the byte-oriented API. Because the algorithms are strictly byte oriented, creating large integers first and then converting them back to byte arrays is usually unnecessary. Since most callers seem to prefer a byte-oriented API as well, we decided to use the unprefixed #hash* selectors for that. The original integer oriented API is now prefixed with #integer, that is, the selectors are of the form #integerHash* and will most likely be phased out (protocol “deprecated”). The old #byteHash* methods are also preserved in the “deprecated” protocol to provide backward compatibility.

The lower level API (protocol “services-basic”) is derived from the popular Update/Final scheme used by other crypto libraries, except that in our case the “Final” phase is embedded into the call for obtaining the digest value, #digest. This makes the API simpler. Moreover, it allows us to

support intermediate calls for a digest value before all the data is processed. It eliminates the common limitation that once you call Final, you can't call Update anymore. With our implementation you can mix #updates and #digests freely. We used #digest for the result message because #hash would conflict with the standard #hash message. That is also why we added #digestSize as a preferred form to #hashSize, although the latter is preserved for backward compatibility. We also added another constant accessor #blockSize answering the size of basic processing unit of the algorithm, and enabled the accessors on the class side as well, so that one does not have to create a hash instance to get at the constants. The API also includes #messageLength to obtain current bit size of the data that was already provided to the hash algorithm, and #reset which allows you to reinitialize and reuse the algorithm for new hash computation.

Finally, we've also added support for algorithm “cloning.” Hash>>copy now allows you to clone a hash computation in progress. This is useful when you have multiple chunks of data with the same initial part. You can hash the initial part just once, then clone the hash and continue with multiple instances from there.

Public Key Ciphers

We have also made a significant progress reconciling the APIs of our public key ciphers and making them more accessible. Ciphers are now separated into SymmetricCipher (secret key) and AsymmetricCipher (public key) hierarchies. RSA, DSA and DH algorithms are now subclasses of AsymmetricCipher with a common API for encryption and signing abstracted out. The key elements of the API are:

accessing

- publicKey/publicKey:
- privateKey/privateKey:

encryption

- encrypt:
- decrypt:

signing

- sign:
- verify:of:

Having DH in the AsymmetricCipher hierarchy, even though it marks most of the common API as #shouldNotImplement, is mostly a proclamation of the nature of the algorithm, reinforced by the #privateKey/#publicKey accessors. However the true API of the algorithm are messages #computePublicValue and #computeSharedSecretUsing: representing the two stages of DH negotiation. Message #computePublicKey was renamed to #computePublicValue to match #computeSharedSecretUsing: better and to reduce the conflict with the #publicKey/#privateKey messages.

Common aspects of DSAKeyGenerator and RSAKeyGenerator were abstracted out into the KeyGenerator superclass, providing unified API for instance creation protocol, key accessors, flushing, etc.:

instance creation

- keySize:
- keySize:random:
- keySize:random:primalityTest:

services

- publicKey
- privateKey
- flush

Finally, the RSA and DSA keys were reconciled in the EncryptionKey hierarchy.

Overall we hope that the AsymmetricCipher, KeyGenerator and EncryptionKey hierarchies, along with their comments, will make the use of public key ciphers clearer as the SymmetricCipher hierarchy hopefully already does. This is covered in the new *Security Guide* as well.

All the obsolete message selectors are retained in the “deprecated” protocol for backward compatibility. Any class renames concern abstract classes so they should not have much impact on user code.

Opentalk

Load Balancing

Opentalk Load Balancing, issued as a preview in previous releases and retracted in release 7.1, has been reissued as product in the 7.2 release. The load balancing facility provides components for constructing most synchronous unicast load balancing architectures. It implements client-

side reference wrappers, that support message retry and redirection, a generic load balancing client, load balancers and server-side load monitors of several types, a range of distribution, load data transfer, and server group update mechanisms, an extensible hierarchy of load definitions, a generic load balancing server, and supporting classes. The code of the Opentalk Load Balancing facility is commented, and extensively documented in chapter 7 of the *Opentalk Communication Layer Developer's Guide*.

Known Limitations

The Opentalk Load Balancing facility possesses three major limitations:

- It is now restricted to use with the Opentalk STST TCP protocol.
- Message redirection now entails remarshaling.
- Limited but adequate support for multiple balancers is provided for only a subset of the supported architectures and configurations.

A more complete list of limitations, including minor infelicities, is presented in the documentation.

> **Note:** The VisualWave load balancer has not yet been rewritten to fall under the framework provided by Opentalk Load Balancing.

Starting and stopping

The starting and stopping sequences of Opentalk components were refactored and extended. The #start and #stop operations were each broken into three phases: #preStart, #doStart, and #postStart, and #preStop, #doStop, and #postStop. These phases trigger events of the same name. The original #starting: and #stopping: events are no longer triggered, but can be trivially revived using #doStart and #doStop handlers. The messages #startRequest and #stopRequest are deprecated; use #start, #stop instead. The message #shutDown is deprecated as well; use #doStop instead.

Concurrent connection limits

To accommodate web server load patterns we've introduced configurable limits on the number of concurrent connections. This change concerns ConnectionOrientedAdaptors in a following way. There are two limit values, a soft “lowerConnectionLimit” and a hard “upperConnectionLimit”.

It is assumed that “lowerConnectionLimit” is less than “upperConnectionLimit”. When a new connection is added and current number of connections exceeds “lowerConnectionLimit”, the adaptor triggers a #reachingConnectionLimit:with: event. When the number reaches “upperConnectionLimit” a #reachedConnectionLimit:with: event is signaled and the listener process is suspended. When a closing connection is being removed and the current number of connections exceeds “lowerConnectionLimit” a #leavingConnectionLimit:with: event is triggered. When the number reaches “lowerConnectionLimit” a #leftConnectionLimit:with: is triggered and the listener process is resumed. Both connection limits are configurable. The configuration parameters are called #lowerConnectionLimit and #upperConnectionLimit. Current defaults are set to 1000 (upper) and 900 (lower).

Within the limit zone, we have added a delay after each accept, which progressively grows as the number of connections approaches the upper limit. The growth is bounded by a configurable parameter called maxAcceptDelay (milliseconds). The default is set to 10ms. These delays can be turned off by setting maxAcceptDelay to 0. This is added into the reachingConnectionLimit:with: event.

We've also added an error handler in ConnectionListener to protect socket accept and connection creation. The error handler invokes #handleListenerError: message which in turn triggers a #listenerError:in: event. The later message takes the listener itself as a parameter, which is handy when one needs to restart the listener in case of error.

RemoteObject printing

By default, RemoteObjects used to print their host name. This entails an expensive reverse DNS lookup that may freeze the image for a couple of minutes. So, the default printing of RemoteObjects was changed to use the host address. The class variable PrintUsingHostAddress maintains a Boolean, that controls the employed printing strategy.

New message events

Two somewhat new events were introduced at the generic level of RemoteMessages. The event #evaluatingMessage:in: is triggered for every incoming message, that is, for every request coming into a server and for every reply coming back to the client. If there is an error during evaluation, an #evaluatingMessage:in:failedBecause: event is triggered.

Note that it is possible to signal an Error in an #evaluatingMessage:in: handler and thus prevent evaluation of an incoming request. This capability was introduced to support our current approach to message redirection, and in anticipation of supporting like mechanisms in SOAP header processors and CORBA interceptors.

Application Server

New Opentalk-Based Server Classes

Probably the most significant change in this version is the integration of the Opentalk and Application Server HTTP serving capabilities. When creating a server, the names of the possible server classes have changed, with WaveIPRequestBroker replacing IPWebListener and WaveHTTPRequestBroker replacing TinyHTTPServer. This change should be transparent to existing applications, but provides access to some new underlying facilities, as well as unifying the two separate HTTP serving components.

Use of New Settings Framework

With this release, application server settings are no longer in a separate settings window, but integrated with the other VisualWorks settings, using the facilities introduced in VW 7.1. Settings can be accessed interactively either using the "Web Settings" launcher menu, which will give a browser only on that sub-group of settings, or from the main settings browser.

New Settings File Format

The use of the new settings framework also means that settings are saved, like the other VisualWorks settings, in a new XML-based format. This replaces the format previously used in the .vwaverc files, although the new system will still read the old file format. Note that this does not yet replace all of the other configuration files. For example, the Web Toolkit .INI files for configuring web sites remain in their previous form for the moment.

New Load-Handling Improvements and Settings

With the use of the Opentalk HTTP server we have introduced a number of new settings for dealing with heavy load situations. In particular, it is possible to configure the server to limit the maximum number of simultaneous connections, and to introduce an optional delay between accepts. In a situation where the server is flooded with connections, this can help reduce the effect of the listener starving the processes which

are actually servicing requests. It is also possible to configure the priority of the listening and serving processes. These options are configurable using the settings mechanism described above. In addition to these changes, there has been a certain amount of performance optimization done, primarily in removing slow dynamic lookups of class names at runtime.

Adjusted Default Memory Sizes

After considerable performance tuning and load handling, and to accomodate modern machine capabilities, we have adjusted the default memory sizes. In particular, for dealing with large numbers of connections we have increased the size of the spaces for new objects, and increased the size of LargeSpace by a very large factor (250 times the default, 50MB total). This helps the garbage collector deal with the large number of socket buffers in use. The maximum total memory has been increased to 160MB, with the growth regime upper bound remaining at 2/3 of that value. For production use, it is recommended that you evaluate performance of your own application and available memory, and adjust these parameters accordingly. All of these values can be adjusted from the new settings windows or files, although it is worth noting that memory sizes at startup will only take effect after an image save and restart, and thus cannot be usefully set in the startup file).

Load Balancer Using Opentalk

The Load Balancer has been switched to use Opentalk rather than DST as its underlying communications mechanism. This eliminates the requirement to have DST loaded when using the load balancer, reducing footprint.

GIF Encoding by Default

With the expiration of the GIF-related patents, the system now defaults to GIF encoding for VisualWave's dynamically generated bitmaps.

Fixed Problem with Defaulting to Production Mode

In the last release, with the unification of the previous Wave Server and Wave Developer components, the entire system would always come up in production mode by default (as Wave Server did). This could be confusing, particularly to new users. The system now defaults to Debug mode.

Fixes for New Window Opening Types

Some of the new options for window opening policies (e.g. cascading) would cause walkbacks if you tried to open those windows as VisualWave applications. This has now been corrected.

Documentation for Perl Gateway

The Perl gateway, which was introduced with VW 7.1, now has documentation in the *Server Configuration Guide*. This gateway is now the recommended way to initially configure a server, as it is easy to install and performs well.

ISAPI Error Propogation

The way in which the ISAPI gateway reports errors has been improved. It reports an error page uniformly if the hostmap file cannot be found. It also handles more exceptions internally and writes them to the IIS log, rather than passing them up to IIS, which could treat them as an error caused by the DLL and terminate the process.

Possible Issues with IIS on Windows XP

We have observed some issues with the gateways using IIS on current versions of Windows XP. It appears that IIS is attempting to validate the full path of a request that uses a virtual directory, regardless of the settings. This does not appear to affect usage with a full path (e.g., **`.../scripts/cgi2vw.exe/...`**), and is not specific to a particular gateway.

Documentation

This section provides a summary of the main documentation changes.

Advanced Tools Guide

- Incorporated changes to the Profiler chapter
- Updated for changes to Benchmarks chapter

Application Developer's Guide

- Documentation of new Settings Manager usage (ch 1) and framework (ch 14).

COM Connect Guide

No changes.

Database Application Developer's Guide

- Removed obsolete OracleOR chapter (formerly chapter 10)
- Miscellaneous updates

DLL and C Connect Guide

- Documentation of support for MacOS X
- Miscellaneous updates.

DST Application Developer's Guide

No changes.

GUI Developer's Guide

- Restored brief discussion of model/view separation to chapter 2.
- Major revision to chapter 4, "Adapting Domain Models to Widgets."
- Restored information from the old Business Graphics guide to the Charts Widget description in chapter 9.
- Updated discussion of the List Widget in chapter 9.

Internationalization Guide

Miscellaneous updates.

Internet Client Developer's Guide

- Information added on international characters in messages and headers.
- Secure Socket Layer chapter removed an included in the Security Guide.
- Miscellaneous updates

Opentalk Communication Layer Developer's Guide

- Moved Opentalk-SOAP chapter to *Web Service Developer's Guide*
- Added new chapter on Opentalk Load Balancing support.
- Miscellaneous updates

Plugin Developer's Guide

No changes.

Security Guide

New document.

Source Code Management Guide

- Added information on PostgreSQL configuration
- Other minor additions

Walk Through

No changes.

Web Application Developer's Guide

No changes.

Web GUI Developer's Guide

No changes.

Web Server Configuration Guide

- Load Balancer Documentation updated for use of OpenTalk
- Documentation for use of Perl gateway
- Miscellaneous changes

Web Service Developer's Guide

- Added discussion of SOAP Header support.
- Additions to chapter 3, on WSDL.
- Additions to chapter 5, "Building Web Service Servers."

TechNotes

The following documents in the TechNotes directory have been updated:

VisualWorks ByteCode Set

- Updated and released as Technical Note

VisualWorks Memory Management

- Documentation of Object Engine memory management and garbage collection.
- Added guidelines for tuning large-memory applications.

Goodies

Default Package Namespaces

The DefaultPackageNamespaces pacel, in the **`goodies/parc`** directory, allows a package to specify a default namespace, using the #namespace property. If it does so, then

- class/namespace definitions default to that namespace rather than to Smalltalk
- class extensions defined in the package are compiled in the scope of that namespace, rather than the namespace of the class. This can greatly reduce the need for qualified names in class extension methods.

Note that, as with all goodies, this is unsupported code, and may still have issues in unusual situations.

3

Deprecated Features

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

GUI

PseudoEvent removed

(AR 46219) The class PseudoEvent and all its references has been removed. In a polling control environment instances of this class were used to wake up the window controller without waiting for a host event to arrive. In the new MultiProcUI control environment introduced in VW 7.1 dispatching instances of PseudoEvent were unnecessary and the time consumed to process it wasted since it was ignored. Application developers are advised to remove any reference of PseudoEvent or sends of #pseudoEvent or #isPseudoEvent in their control code when migrating to this release.

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.

DotNETConnect

DotNETConnect is a VisualWorks add-on that allows you to integrate Microsoft .NET components, defined in the .NET Common Language Runtime (CLR) libraries, into a VisualWorks application. This additional capability makes it easier for VisualWorks programmers to develop applications that coordinate well with other .NET applications.

DotNETConnect was developed by Georg Heeg eK and is licensed to Cincom.

For additional information, refer to the documentation in preview/DotNetConnect/Delivery/Doc/DotNETConnect.pdf.

Unicode Support for Windows

Extended support for Unicode character sets is provided as a preview, on Windows 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`**.

New GUI Framework (Pollock), Beta 3

Pollock remains in preview in 7.2.

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 SNMP

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

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

Usage

Initial Configuration

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

Broker or Engine Creation and Configuration

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

- several transport mappings,
- a PDU dispatcher,
- several possible security systems,
- several possible access control subsystems,
- a logically distinct marshaler for each SNMP dialect, plus
- an attached MIB module for recording data about its own performance.

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

```
SNMPEngine newUDPAtPort: 161.
```

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

```
newUdpAtPorts: aSet
| oacs |

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

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

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

Engine Use

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

OR-Style Usage

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

```
| broker3161 broker3162 oid ref return |

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

This expression returns:

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

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

Explicit Style Usage

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

```
| oid broker1 entity2 msg returnMsg |

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

which returns:

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

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

Entity Configuration

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

```
SNMP snmpv0CommandResponderAt: anInteger
```

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

MIBs

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

If you need to create an XML version of a MIB that is not provided, use the 'snmpdump' utility. It is a part of the 'libsmi' package produced by the Institute of Operating Systems and Computer Networks, TU Braunschweig. The package is available for download through http://www.ibr.cs.tu-bs.de/projects/libsmi/index.html, and at http://rpmfind.net.

Limitations

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

Port 161 and the AGENTX MIB

SNMP is a protocol used for talking to devices, not applications, and by default SNMP uses a UDP socket at port 161. This means that in the absence of coordination between co-located SNMP agents, they will

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

Opentalk

The Opentalk preview 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.

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.

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

Table 1 - Socrates scalar type to Smalltalk class mappings

Socrates Data type	Smalltalk Class
DOUBLE	Double
FLOAT	Float
INTEGER, SHORT, SMALLINT	Integer
NULL	UndefinedObject
NUMERIC	FixedPoint, LargeInteger
TIME	Time
TIMESTAMP	Timestamp

Table 2 - Socrates collection type to Smalltalk class mappings

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

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

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

Reference Support

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

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

Object Support

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

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

GLOs

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

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

Virtual Machine

IEEE floating point

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

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

OE Profiler

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

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.

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.

> **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.

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®