Cincom

VisualWorks®

7.4 Release Notes

P46-0106-11

Part Number: P46-0106-11

Software Release 7.4

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 © 1999–2005 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 7.4

These release notes outline the changes made in the version 7.4 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 (7.2.1 release notes cover 7.2 as well).

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

Image and Engine Compatibility

Because of changes to the byte code set for 7.4, images saved using the 7.4 object engine can not be read by pre-7.4 engines. 7.4 engines are, however, fully backward compatible with pre-7.4 images.

Accordingly, pre-7.4 images that you edit and save using a 7.4 engine will no longer be readable by pre-7.4 engines.

Parcel Version Updated

Because of changes to the byte code set, the parcel version has been updated. This prevents parcels created in 7.4 or later, which use the extended bytecodes, from being loaded into earlier images.

Store Database Update

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

```
DbRegistry update74
```

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 7.4 image.

Several Limits Raised

Various limits to size restrictions have been lifted or raised. These are all described in the updated ImplementationLimits7x.pdf document.

Byte codes have also changed as a result.

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.

Mac OSX character spacing inaccurate for some fonts

Some Mac OSX fonts, noteably Lucida_Grande and Apple_Chancery, report inaccurate character spacing when used in VisualWorks. Text using this font may appear clipped within the bounds of a widget, the text insertion point may be one or more characters from the displayed cursor, or selected text may appear misaligned. This issue does not apply when the Mac OSX X11 engine is used. Avoid use of these fonts until this issue is resolved. To work around this limitation, Helvetica has been substituted for Lucida_Grande as the default system font family for the OSX Aqua look on native Mac OSX platforms. When available, the Lucida font is still used as this look policy system font on other platforms including Mac OSX X11.

Linux Startup Issue on X.org X11 Installations

A system hang condition has been observed on some Linux systems with X.org X11 support (AR48619). This appears to be related to fonts.

Refer to "HOWTO Xorg and Fonts" at

http://gentoo-wiki.com/HOWTO_Xorg_and_Fonts

Then consider specifying additional fonts, and ensure that FontPaths is defined in **`/etc/X11/xorg.conf`**, as described in that document.

For specifics of one user's fix, refer to Isaac Gouy's comment at:

http://www.parcplace.net/list/vwnc-archive/0511/msg00251.html

HPUX11 User Primitive Engine

The HPUX11 User Primitive engine does not run for as yet not understood reasons. This is covered by AR 49661. The engine appears to compile and link correctly but then exits prematurely after executing a few Smalltalk expressions, apparently without error. We hope to have this problem resolved by 7.4.1.

Limitations listed in other sections

- Limitation on Code Loading While Headless
- SNMP preview Limitations

2

VW 7.4 New and Enhanced Features

This section describes the major changes in this release.

Virtual Machine

New Platform VMs

Several VMs are introduced in this release, either as beta/preview or as unsupported. One platform is now supported, previously in preview. Refer to “64-bit VM” for more information about 64-bit VM issues. The VMs are summarized in the following table.

Virtual Machine	Bit size	Support status
solaris64	64-bit	Preview/beta
solaris86	32-bit	Unsupported
solaris86_64	64-bit	Unsupported
linuxx86_64	64-bit	Supported

A VM is considered to be in “preview” if it is intended to be fully supported in a future release, pending additional testing.

A VM is considered “unsupported” for a variety of reasons, including that: the platform itself is uncertain (as in many linux distributions which can be height variable); we are insufficiently experienced with a platform to be able to support it with confidence; customer demand is small enough that the cost of full support is not warranted.

64-bit VM

This release supports 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. It also introduces a beta of solaris64, which is 64-bit Solaris running on SPARC V9 platforms, and an unsupported beta of solarisx86_64, which is 64-bit Solaris running on x86-64/EMT86 platforms.

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/linuxx86_64`**, the Solaris SPARC engines are in **`bin/preview/solaris64`**, and the Solaris x86-64 engines are in **`bin/unsupported/solarisx86_64`**. The ImageWriter parcel's comment includes instructions for transforming images. You can find more information in **`readme.txt`** for each 64-bit vm.

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 add a number of additional 64-bit platforms. The highest priorities currently are:

- HP-UX PA-RISC 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.

Base system

Environment Variable Handling

This release changes how environment variables are looked up on Windows. In previous releases environment variables were only looked up in the Windows registry. In this release they are first looked up in the C environment. This allows one to override the registry values, for example by setting variables in a DOS window. Setting variables is still done by updating the registry.

Deployment and Startup Processing

This release makes some moderately significantly changes to Subsystem classes.

Runtime and Development Subsystems

This release introduces RuntimeSystem and DevelopmentSystem classes.

RuntimeSystem is particularly useful, as it allows applications to detect if the system is "ready" by including RuntimeSystem as a prerequisites. This doesn't mean that all subsystems have necessarily run, particularly since other things can have RuntimeSystem as a prerequisite, but it indicates that the basic system facilities are available. It also provides a uniform mechanism for testing if this is a development or a runtime image, with the isRuntime and isDevelopment methods.

DevelopmentSystem will only activate if we are not in runtime, and sets up the basic development environment (e.g. sets up the changes file if the image name has changed).

Note that DevelopmentSystem runs quite early in the startup process, and is not an indicator that the entire development environment is ready for use. Rather, if the image is in development mode, then DevelopmentSystem becomes a prerequisite for RuntimeSystem, and you should still test RuntimeSystem to determine if the image is ready for use.

DeploymentOptionsSystem is also introduced. This is a very early activating system that allows control over the startup process, e.g. implementing the **`-runtime`** and **`-development`** command-line options.

Subsystem Prerequisites Re-arranged

CacheFlushingSystem has been renamed to EarlyInterestNotificationSystem, to better reflect its purpose. Code that uses the old name can be made to work by loading the System-Subsystem Compatibility package.

Similarly, ExternalAccessSystem has been renamed to EarlyInstallationSystem.

Subsystem startup has been changed slightly for subsystems loaded during startup. When the system first starts, it makes a list of all subsystems, and then attempts to activate them all. However, some subsystems, notably ImageConfigurationSystem, can cause code to be loaded, and this code may contain new subsystems. Previously, these would be activated immediately. However, this could cause circularities if the loaded systems had ImageConfigurationSystem as a prerequisite, or if it was necessary for activation to wait until other code had also been loaded. The new mechanism will not activate them immediately if it is still in startup, and the startup mechanism will continue looping over the list of subsystems and activating them until no new ones have been created in the current pass. After system startup is complete, newly loaded subsystems will be immediately activated as before.

Reading the default configuration file ***imageName*.cnf** now happens before any other ImageConfigurationSystem actions (e.g. **-doit**) rather than afterwards. This makes it easier to load code from a configuration file and have a **-doit** on the command line to do something with that code.

Subsystem canActivate now checks if the subsystem or any of its prerequisites can activate. So, if something depends on, e.g., DevelopmentSystem, and we are in runtime mode, then it will automatically return false for canActivate. Previously it would not activate, but the canActivate test would return true.

Ordering Tool

A tool page has been added to the system browser to help visual subsystem startup dependencies and ordering. Load the Tools-StartupOrderingTool parcel. Then, when you select a Subsystem subclass an additional tab is provided in the source area that graphically shows the prerequisite dependencies.

Headless Image Changes

The Headless package is now included in the standard base image, so all headless options are available. The package can be unloaded it if it is not required.

Saving an image headless automatically disables the startup splash screen and herald sound.

Standard I/O Streams

Support for standard I/O streams (stdin, stdout, stderr) has been incorporated into the base image, in the Standard IO Streams package. This defines shared variables OS.Stdout, OS.Stdin, and OS.Stderr. If these streams are available from the operating system, then the variables will be set up appropriately during system startup.

Note that if running on Windows, these will only be available if using the console engine. Under MacOS9 or earlier, they are unavailable. In general, they may not be available, or may not go anywhere meaningful, if starting up an application other than from the command line.

Transcript Handling

Transcript handling while headless has been changed. It is now possible to have multiple simultaneous output channels for the Transcript. Included with the image are the standard Visual Launcher GUI, a file, and standard out. By default, when running headless, the Transcript goes to a file, and to stdout (if available). When running with a GUI, it goes only to the Visual Launcher. These options can be controlled programmatically, or if the TranscriptSettings package is loaded, then a Transcript settings page can also be used.

Exception Handling

Exception handling when headless has been changed to use the "Notifier current" mechanism rather than overriding the Notifier code. When starting up without a GUI, the Notifier current is set to be the HeadlessImage, and the previous value is remembered and reset when we start up with a GUI.

Limitation on Code Loading While Headless

If you attempt to load code into a clean **`visual.im`** while running headless, the image will crash with a headless error trying to update the mini change set manager in the visual launcher. For example, this occurs when executing:

```
<virtualmachine> visual.im -nogui -pcl HTTP
```

This also applies to any other image which has an open visual launcher and an empty default change set displayed in it.

To work around this, start up the image, make some code change, and save the image. Alternatively, you can remove the mini change set manager by removing the method miniChangeSetManager in

VisualLauncherToolDock. If this is done via file-in from the command line before other code is loaded, it will prevent the headless crash from occurring.

Suppressing Splash and Herald

The ability to suppress the splash screen and herald sound has been enhanced. The Runtime Packager now provides settings for these options. To disable the splash screen and herald sound programmatically, evaluate the following expression and save the image:

```
ObjectMemory registerObject: false withEngineFor: 'showHerald'
```

This image will now start without the splashscreen and sound.

Printing

VW no longer approximates eliptical arcs in PostScript, but instead uses the PostScript language arc function. All magnitudes and angles of arcs are now rendered to the precision of the PostScript printer.

Tools

Parcel Loading

The **Load Parcel Named…** menu item has been moved from the **Tools** menu to the **System** menu, and now uses an incremental search dialog rather than a simple prompter.

File Dialogs

This release adds support for specialized file and directory selection dialogs on all platforms. See subclasses of the class FileDialog in the Tools-Dialogs package.

The legacy file and directory selection API on the class side of Dialog now uses the new dialogs as well. To the best of our knowledge at the time of the release, applications relying on that API need no changes. However, in case of unforeseen problems, you can revert the Dialog class API to the old prompter-based behavior by evaluating the following expression:

```
Dialog.UseOldFilePrompters := true
```

By default as before, VisualWorks running on Windows uses Window common file dialogs. To change this and use Smalltalk-based dialogs on Windows as well, turn off the **Use native file dialogs** setting on the **Tools** page of the Settings Tool, or evaluate:

```
Dialog.UseNativeDialogs := false
```

The new file dialog framework is partially based on the file dialogs implementation contributed by Milan Cermak.

Advanced Tools

Documentation for Advanced Tools has been distributed to the new *Tool Guide* and *Basic Libraries Guide*.

Database

Miscellaneous changes

- Oracle connection pooling is provided in parcel OracleThapiCPEXDI. It works for Oracle 9.0 and later in multi-threaded mode.
- Sybase Threaded Interface is provided in parcel CTLibThapiEXDI.
- Methods for direct execution of SQL statement and connecting through SQLDriverConnect are added in ODBCEXDI.

Oracle Library Path

All hard-coded references to library directories for Oracle have been removed from all Oracle interface classes that used them (e.g., OracleLinuxInterface). Instead, all library directories are now read from the environment variable: on UNIX and Linux systems, LD_LIBRARY_PATH; on Windows, PATH.

Accordingly, it is essential to set the library path, as per the Oracle installation instructions.

This change resolves a problem loading the correct libraries on both 32-bit and 64-bit systems (AR 49088), and in a Multi-Oracle-Home situation.

Runtime Packager

This release includes number of bug fixes and improvements to Runtime Packager, and to deployment in general.

- The Runtime Packager GUI is now organized around packages and bundles, rather than around namespaces and class categories. This brings it more into line with the browsers, and also makes it easier to manage class extensions. There are a variety of smaller user interface improvements as well, including menu items that allow you to quickly keep or discard all of a package or bundle, or a package/bundle and all of its prerequisites.
- It is now possible to keep or delete namespaces and individual non-class items in namespaces and/or shared variables from the user interface.
- In general, the tracer's ability to reason about namespaces and shared variables is significantly improved. Class references in shared variables will be followed. In particular, this is useful for the Net framework, which used shared variables to keep track of handler classes for different protocols.
- The tracer is now aware of pragmas, and will attempt to include any references to it. This is necessarily conservative, since the pragma mechanism relies on reflection. The heuristics are that if a method being traced contains a pragma method, then include any senders of the pragma selector which also directly reference the class Pragma. If a class is being traced, automatically include all pragma methods that are part of that class. With this, the tracer can properly handle the settings framework.
- It is now possible to set a flag in an image so that it will suppress the splash screen and herald sound on startup. This is now an option when creating a runtime image, and is on by default. This is particularly useful for a headless image.
- The application startup when the image starts is now done using the Subsystem mechanism, as is the loading of code from the command line.
- Debugger probes are now automatically removed during the preparation of a runtime.
- By default, all OS and Window System looks are now kept, so the runtime images are portable. To minimize the image size, manually remove the looks that you don't want.

- By default, Subsystem and all of its subclasses are kept.
- There are new command-line options related to deployment and startup
  - **`-nogui`**, **`-gui`** and **`-headful`** are now available as synonyms for **`-headless`** and **`-headfull`**
  - **`-runtime`** and **`-development`** toggle whether an image is treated as a runtime or not (note that the "Loading" settings can be used disable these options, along with others that could be security risks in a deployed application, such as -filein and -pcl)
  - **`-listOptions`** will print all of the available command-line options to stdout (if available) and then quit the image
- The handling of dialogs and other windows when headless has been improved, mostly on a case-by-case basis. A Store image started up headless will not attempt to bring up a connection dialog. When a Runtime Packager image is in the process of the three-step save to optimize perm space usage, and is started up headless, it will not put up a notice about the three-step save but will simply start up and immediately save and quit.

WebService

Bindings for Collections

(AR#49739) The XML-to-Object binding for collections previously tested collection types only if the collections were kind of Collection class. It allowed passing a Struct as a valid Collection type. The marshaling exception did not provide meaningful explanation about the problem. Another side effect of the old implementation was not consistent approach how we marshal and unmarshal collections. It was possible to send and marshal an array but unmarshaling would always return an ordered collection if this type was specified in XML to object binding.

The fix checks and allows collection types exactly as they are specified in a XML-to-Object binding. The collection can have two descriptions, either

```
<sequence_of name="myCollection" >
    <..../>
</sequence_of>
```

or

```
<element name="anotherCollection maxOccurs="unbounded"/>
```

In either case the default type is OrderedCollection and argument values have to be passed as an OrderedCollection. Passing other collection types will raise the proceedable exception: WrongObjectType.

This change will break implementations which, for example, used Arrays instead of OrderedCollections. There are a few options to fix broken applications:

- Change the default collection type in XML to object binding specification to desired one. The <sequence_of> collection mapping allows specifying other than default collection type in the "smalltalkClass" attribute:

  <sequence_of name="myCollection" smalltalkClass="Array">

  Currently there is no support to provide other than default collection type for:

  <element name="anotherCollection maxOccurs="unbounded"/>

- Restore old collection validation rule:

  ManyRelation class useExactCollectionType: false.

  The default option value is "true".

- Use collection types as they were specified in XML to Object binding.

Creating Classes in Specified Namespace

(AR#49420) There was a problem that we were unable to create classes from WSDL in a non-default namespace when those class names were used in the system namespaces.

To resolve this, there is a new dialog added to the Wsdl wizard for "Create classes from a wsdl schema" option. The dialog displays two lists: a list of all classes to create and a list of classes with duplicate bindings. A user can choose create or not create classes or rename the duplicate bindings.

If the classes are created using WsdlClassBuilder there will be raised the WSDuplicateBindingsError exception for duplicate bindings. The exception parameter will be a Struct with the duplicate bindings list. The "exception proceed" instruction will not create classes from the duplicate bindings. New classes will be created if the exception resumes with #createAll parameter. A user can provide its own list of classes to create and return it in the #createClassesList Struct entry.

New Struct Class

This release has a new Protocols.Struct class. WebServices.Struct will be replaced by the Protocols.Struct in the 7.5 release. The Protocols.Struct class is not a Dictionary class extension anymore. See Opentalk notes for more comments about Protocols.Struct.

Opentalk Server Support

(AR#49125) This enhancement allows sending to Opentalk server query "..?wsdl" and the server returns a wsdl schema string.

WSDL Tool Enhancement

Beginning with this release the wsdl tool generates an addition mapping for operation selectors. This change fixed an old problem with selectors between a Wsdl and Opentalk clients. The selector map will be created for Opentalk client and server classes and used to find a Soap operation.

One-way Operations

(AR#49348) Added support for one-way operation.

Binding Wizard

We have added an XML-to-Object Binding Wizard. The tool allows creating XML to object bindings and testing marshaling/unmarshaling Smalltalk objects to XML.

Soap headers in Opentalk

(AR#48590) WSOpentalkServer is an abstract super class for an Opentalk server. The class initializes the RequestBroker and knows its service classes. The RequestBroker is responsible for setting Soap header processors and processing policy (SOAPProcessingPolicy). The Soap processing policy defines the SOAPInterceptorDispatcher which is going to send message processing events to each registered message interceptor. The SOAPMessageInterceptor invokes specific subclasses of WSHeaderEntryProcessor to process header entries for an operation.

The SOAPInterceptorDispatcher implements the header entry processing policy:

1. If a header entry has **`mustUnderstand=1`** or **`mustUnderstand=0`** attribute, targeted at SOAPProcessingPolicy>>role and has a header entry processor, the header entry will be processed with result that header entry is returned to client.

2. If a header entry has **`mustUnderstand=0`** attribute, targeted at SOAPProcessingPolicy>>role and doesn't have a header entry processor, the header entry will be ignored.
3. If a header entry has **`mustUnderstand=1`** attribute, targeted at SOAPProcessingPolicy>>role and doesn't have a header entry processor, the SoapMustUnderstandFault will be raised.
4. If a header entry is not targeted at SOAPProcessingPolicy>>role it will be ignored.

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

Subclasses must implement the following messages:

Instance methods:

processOutputHeader:reply:transport:
processing reply header on the client side (optional)

processInputHeader:transport:
processing request header on the server side

addOutputHeaderTo:transport:
add reply headers to the soap message on a server

addInputHeader:transport:
add request headers on a client

Class methods:

headerTag
Header entry node tag for the specific header processor.

header
The CtmHeader description. The description includes a Smalltalk type and exception types. For example in HPAuthenticationToken class:

```
header
    ^self ctmHeader
        smalltalkType: AuthenticationToken;
        addExceptionNamed: 'AuthenticationTokenException'
            smalltalkType: WebServices.AuthenticationTokenException;
        addExceptionNamed:  'WrongPassword'
            smalltalkType: WebServices.WrongPasswordException;
        yourself.
```

Opentalk client header support

WSOpentalkClient is abstract class for the user specific client class.

A customer should create a client class as subclass of WSOpentalkClient and implement the following methods:

Class methods:

- serverUrl
- port
- bindingName

Instance methods:

The **public api** category is the default for all methods used to access web services.

WSHeaderEntryProcessor 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.

If there is not a header entry, but the Wsdl schema describes 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.

When the message is sent, the specific header processor can add some input header entries by sending the message addInputHeader:transport:.

When the response is received, the specific header entry processor can add some processing to the header entry by sending the message processOutputHeader:reply:transport:.

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

- using a client:

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

- in a header entry processor

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

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

- a result, which is the body value. Access to a response header will be avaliable only in client header entry processors:

  ```
  SOAPMarshaler defaultReturnedObject: #result.
  ```

  or:

  ```
  client returnedObject: #result.
  ```

- an envelope, which is an instance of WebServices.SoapEnvelope. Having an envelope as a result allows you to get access to response header and body:

  ```
  SOAPMarshaler defaultReturnedObject: #envelope
  ```

  or:

  ```
  client returnedObject: #envelope.
  ```

- a SoapResponse. Having a WebServices.SoapResponse as a result can be helpful for debugging purpose:

  ```
  SOAPMarshaler defaultReturnedObject: #response
  ```

  or:

  ```
  client returnedObject: #response.
  ```

The header demo can be loaded from the Opentalk-SOAP-HeadersDemo. In this demo the CustomerServer class implements the Opentalk server.

The WebServicesDemo parcel has another sample (WSLD1TestSoapHeader class) that creates classes and header processors from a Wsdl schema.

Generating a Wsdl schema with Soap headers

To generate a Wsdl schema with Soap headers, perform the following steps:

1 Create a header processor class. The class has to be a subclass of Opentalk.WSHeaderEntryProcessor. For example:

```
Smalltalk.WebServices defineClass: #HdPrAuthenticationToken
    superclass: #{Opentalk.WSHeaderEntryProcessor}
    indexedType: #none
    private: false
    instanceVariableNames: ''
    classInstanceVariableNames: ''
    imports: ''
    category: 'Web Services'
```

2 Add the processor type and exception description to the class. For example, in HdPrAuthenticationToken, define class methods:

```
header
    ^self ctmHeader
        smalltalkType: WebServices.AuthenticationToken;
        addExceptionNamed: 'AuthenticationTokenException'
            smalltalkType: WebServices.AuthenticationTokenException;
        addExceptionNamed:  'WrongPassword'
            smalltalkType: WebServices.WrongPasswordException;
        yourself.

headerTag
    ^NodeTag new
        qualifier: ''
        ns: 'urn:LibDemo\AuthSearch'
        type: 'AuthenticationToken'
```

3 Add header pragmas to an operation, as in this method in WSLDAuthenticatedSearch:

```
authenticatedSearchByTitleWord: aString includeAffiliatedLibraries: aBoolean
    "Generated by WS Tool on #(January 24, 2003 10:33:22 am)"
    <operationName: #'AuthenticatedSearchByTitle'>
    <documentation: #'To be able to make this request the message
        should include authentication token and access level. The
        SearchByExactWord operation returns a collection of holdings or
        empty collection if no holdings found'>
    <addParameter: #'titleWord' type: #'String'>
    <addParameter: #'includeAffiliatedLibraries' type: #'Boolean'>
    <result: #( #Collection #'LDHoldingBook' )>
    "add input header to the operation "
    <inputHeader: #HdPrAuthenticationToken>
    <inputHeader: #HdPrAccessLevel>
    "adds output header to the operation"
    <outputHeader: #HdPrConfirmation>
    <outputHeader: #HdPrAccessLevel>

    ^(self searchByTitleWord: aString includeAffiliatedLibraries: aBoolean)
        values.
```

4 Use WsdlBuilder to create a wsdl schema with headers:

```
builder := WsdlBuilder new
            useDocument;
            buildFromService: WSLDAuthenticatedSearch
            classNamespace: 'WebServices'
            targetNamespace: 'urn:LibDemo\AuthSearch'.
builder printSpecWithSmalltalkBindingOn: aStream.
```

To see some samples you will need to load the WebServicesDemo parcel and look at methods:

```
WSLD1TestCreateWSDL>>testCreateSpec8DocWithHeaders
WSLD1TestCreateWSDL>>testCreateSpec9RPCWithHeaders
```

Tool Support for Soap headers

Added preview quality WsdlWizardWithHeaders parcel that allows creating Soap header processors and test Web services with headers.

To create Soap header processors load the parcel and start the WsdlWizard.

Select the first option "Expose an application as a web service"

Select an operation you want to add a header and press "Add Header.." button.

Create a new header processor and add it to the selected operation or all operations.

WebService Demonstration Code

The WebServicesDemo parcel, as shipped, includes a dependency on a parcel that is not included with the distribution. When loading the demo, and warned of the dependency, select to load the parcel anyway.

WebServicesDemo is intended to demonstrate features of using WebServices library. The following examples function in the absence of the missing code.

Create Smalltalk Classes from a Wsdl Specification

```
builder := WsdlClassBuilder readFrom: schemaDoc readStream.
builder package: 'WSTestLDOpentalk'.
builder opentalkServerName: 'OpentalkServerDoc'.
serviceClass := builder createServiceClasses first.
serverClass := builder createOpentalkServerClass.
proxyClientClass := builder createOpentalkClientClasses first.
```

Create a Wsdl specification from Classes Defining that Service

```
serverClass := OpentalkServerSrvcGenPublicDoc.
builder := WsdlBuilder new.
builder
    useDocument;
    buildFromOpentalkServer: serverClass
        classNamespace: 'WebServices'
        targetNamespace: 'urn:LibraryDemo\srvcGeneral\doc'.
stream := String new writeStream.
builder printSpecWithSmalltalkBindingOn: stream.
```

Create and Run an Opentalk-SOAP Server

```
builder := WsdlBuilder new.
builder
    useDocument;
    buildFromOpentalkServer: OpentalkServerSrvcGenPublicDoc
    classNamespace: 'WebServices'
    targetNamespace: 'urn:LibraryDemo\srvcGeneral\doc'.
stream := (String new: 2048) writeStream.
builder printSpecWithSmalltalkBindingOn: stream.
server := OpentalkServerSrvcGenPublicDoc new.
client := OpentalkClientWSLDSrvcGeneralPublicDoc new.
main := server services first key library.
book :=  main ownedHoldingsColl first.
acquisitionNumber := book acquisitionNumber.
"...find the book by existing number"
book1 := (client holdingByAcquisitionNumber:  acquisitionNumber) first.
```

Net Clients

- Base64StreamEncoder and TimedPromise classes are moved from Net namespace to Protocols and into a new parcel, Protocols-Common.
- All classes from the URISupport parcel are moved to the Smalltalk.OS namespace.
- The Net namespace is not loaded to the base image anymore.
- If you have any problem loading your extensions to URI classes you will need to load the NetNamespace parcel.
- The Net namespace defines the compatibility variables for URI support classes. Please note, that the compatibility aliases in the Net namespace are meant to help with transition to the new names and will likely be removed in some future release. We highly recommend switching to the new names as soon as possible

Security

ASN.1

There were significant developments in the ASN.1 framework. It was reorganized in multiple packages:

- ASN1-Support: ASN1 namespace, OID and SMINode, Entity and Module, BitString and Stuct, ...
- ASN1-Constraints: Constraint hierarchy
- ASN1-Types: Type and TypeWrapper hierrarchy
- ASN1: encoding support, TLVStreams (BER/DER) and EncodingPolicies

Since the framework lives in its own namespace the Asn1 prefix on all class names was dropped. The names of classes representing ASN.1 types were changed to match the ASN.1 conventions, so for example ASN.1 'INTEGER' is represented by class INTEGER and ASN.1 'SEQUENCE OF' is represented by class SEQUENCE_OF. Similarly the names of the constraints and marshaling streams were simplified as well.

A lot of new supporting API was added, and will be described in the following text.

Modules

One of the major changes is adding support for ASN.1 modules. This change eliminates the global hardwired type repository. Modules are completely independent and multiple ASN.1 applications can coexist without having to worry about name clashes with others. Modules will eventually support importing and exporting of types; however, that is not supported yet. Of course there is nothing preventing an application to make its module globally accessible. In fact the X509 framework now exposes it's module via X509Object.ASN1Module.

This change had a profound impact on the type construction API. The old #register:* constructors were not feasible anymore, and therefore were removed. In most common scenarios the module instance will have to be involved in construction of type definitions. We made a conscious attempt to make the new API reasonably concise and flexible, and hopefully the result is easier to use than the original API.

Here's the ASN.1 type description for X509.TBSCertificate as an example. It takes advantage of the module instance being accessible via shared variable ASN1. To keep the example reasonably short, some supporting types are not defined here:

```
( ASN1 SEQUENCE: self asn1TypeName )
    addElement: #version type: #Version tag: 0
        tagging: #explicit default: 0;
    addElement: #serialNumber type: #CertificateSerialNumber;
    addElement: #signature type: #AlgorithmIdentifier;
    addElement: #issuer type: #Name;
    addElement: #validity type: #Validity;
    addElement: #subject type: #Name;
    addElement: #subjectPublicKeyInfo type: #SubjectPublicKeyInfo;
    addOptionalElement: #issuerUniqueID type: #UniqueIdentifier
        tag: 1 tagging: #implicit;
    addOptionalElement: #subjectUniqueID type: #UniqueIdentifier
        tag: 2 tagging: #implicit;
    addOptionalElement: #extensions type: #Extensions tag: 3
        tagging: #explicit;
    mapping: self;
    retainEncoding: true.

( ASN1 INTEGER: #Version )
    constraint: ( ASN1.Asn1ConstraintEnumerated
        with: #v1 -> 0
        with: #v2 -> 1
        with: #v3 -> 2 ).
```

```
ASN1 INTEGER: #CertificateSerialNumber.

ASN1 BIT_STRING: #UniqueIdentifier.

( ASN1 SEQUENCE: #Extensions OF: #Extension )
    constraint: ( ASN1.Asn1ConstraintSize lower: 1
        upper: SmallInteger maxVal ).
```

Here are the changes described in a bit more detail. Modules are represented with instances of a new class, Module. All Types now maintain a reference to their module and acquired associated API, to maintain the bi-directional association. That said, the type instances are still basically fully functional without having a module assigned. So you can create a type on the fly, for example:

```
(INTEGER name: #X) constraint: (Constraint from: 10 to: 50); yourself
```

and use the type to validate or marshal objects. It's mostly operations involving lookup of other types by name where a module is necessary, as it provides the context for this lookup. For this a Module provides the same sort of operations as the Asn1Type.TypeRegistry API used to: #register:/#unregister:/#find: ...

There are two main avenues to building types in a module:

1. Module>>new: <aTypeSpec> named: <aSymbol> and a suite of associated "standard" type constructors provided in the 'definitions' protocol on Asn1Module, e.g. #SEQUENCE:, #INTEGER:, #SET:OF:, etc.
2. all Types understand #name: <Symbol> in: <Asn1Module>.

Also, all structured types have new convenience API for adding elements (protocol "accessing - element constructors"). Arguments that represent types can be specified using type name or a type instance; moreover, basic types can be specified using the type class.

Asn1Module overrides #doesNotUnderstand: to allow fetching types by using their name as a message sent to the module instance. It also allows to use a name of an already existing type as a single keyword message, to mimic behavior of messages for standard types like #SEQUENCE:, #BOOLEAN:, etc. Here's an example of creating type B derived from type A:

```
module INTEGER: #A.
module A: #B
```

An equivalent way to create the instance without using the #doesNotUnderstand: feature is:

```
module new: #A named: #B
```

Similarly you can also get an instances of basic type via #doesNotUnderstand, as in:

```
module INTEGER
```

However names of constructed types will signal an InvalidTypeReference error, because such type definition would not be fully specified and it would be an error to use it as a real type, in a sequence element for example.

Most type definitions should now print themselves in a shortened ASN.1 style way to aid debugging. Types also add a full ASN.1 syntax equivalent of themselves, as an inspector attribute, labeled '-ASN.1'.

Subtyping

This release further enhances support for ASN.1 subtyping, which is basicly constraining an existing type, e.g:

```
X ::= INTEGER (70..140)
Y ::= X (50..100)
```

Subtypes are modeled as instances of SubType. Regular Type instances are always used as roots of the subtype 'parent' chain and all derived types are modeled as SubType instances. This model is more faithful to the ASN.1 notion, because there possibilities of what you can do with a subtype are limited. Modeling subtypes as full Type instances would allow creation of definitions that aren't valid from the point of view of ASN.1. This model also allows to define subtypes before supertypes which wasn't previously supported.

So back to our initial example. Using the new API it will look like this

```
tX := (INTEGER name: #X) constraint: (Constraint from: 70 to: 140).
tY := (tX named: #Y) constraint: (Constraint from: 50 to:100).
```

The 'tX named: #Y' construct creates an instance of SubType, names it #Y and makes tX its parent. There's also #named:in: to assign the newly created type to a specified module as well.

Type tagging is modelled as subtyping as well. Take for example

```
X ::= [5] IMPLICIT INTEGER
Y ::= [6] IMPLICIT X
```

To be able to model this, Types now maintain their #tag and #tagging. The above example would be expressed as follows

```
tX := (INTEGER name: #X) tag: 5; tagging: #implicit; yourself.
tY := (tX named: #Y) tag: 6; tagging: #implicit; yourself.
```

There's yet another benefit to attaching tags to types. With that we don't need to tie tags to elements. An element with a tag is modelled as a simple element with tagged 'anonymous' type. So let's take the following example

```
X ::= [5] INTEGER
S ::= SEQUENCE {e    [3] X}
```

This is now described as

```
tX := (INTEGER name: #X) tag: 5.
tS := (SEQUENCE name: #S)
      addElement: #e type: tX tag: 3;
      yourself.
```

What happens is that the #addElement:type:tag: method creates an "anonymous private" subtype on the fly '(tX name: nil) tag: 3' and uses it as the type of the element #e.

Type Equivalence

The previous release started defining equality for Types as a "structural equivalence" of type definitions. This makes it non-trivial to satisfy the standard Smalltalk contraints imposed on #= and #hash. Type equivalence carries most of the complexity and performance penalties of general DAG equivalence and using #= and consequently #hash for this purpose is dangerous. So we renamed all the #= implementations in Type and Constraint to #equals: and removed all the #hash definitions. Therefore, #= and #hash will now default to the identity based ones from Object. We've developed the #equals: methods a bit further than #= was in previous release in an attempt to emulate the notion of "type compatibility" as described by the semantic model of ASN.1 (p.121 of Dubuisson).

Constraints

AbstractConstraint was renamed to Constraint and new instance creation helpers were added in a protocol ''construction,' present on both the class and instance sides. A complex constraint can now be created as follows:

```
((Constraint size: 10) & (Constraint values: #(1 2 3 4))) -
        (Constraint type: INTEGER new)
```

Constraints also try to print themselves using standard ASN.1 syntax.

Struct

In an effort to address some of the issues of exploitation of MessageNotUnderstood handling we have made Struct a proper nil subclass.

There's now a new package Protocols-Common that contains a nil subclass Protocols.ProtoObject. This is a trimmed down version of Object with a reduced set of basic capabilities (comparing, printing, inspecting, class membership, some system primitives). The recognized set of selectors is chosen so that there's is only minimal probability of conflict with application domain selectors (intentionally avoiding things like #value, #size, #name, etc). Note, however, that in order to satisfy some of the more prevalent expectations of the environment it does define #=, #==, #hash and #class.

Protocols.Struct is subclass of ProtoObject. Its protocol supports Smalltalk style accessors (via doesNotUnderstand:), Dictionary style accessing and a subset of Dictionary protocol, structural equivalence and copying. Again intentionally avoids selectors that are likely to clash with application domains, so for example the 'name' is accessed via #structName. Struct grows transparently by fixed amount (currently 5).

ASN1.Struct is now a subclass of Protocols.Struct. Its extensions are explicit support for encoding retention ('enconding' inst var) and a change in equality semantics. Two ASN1.Structs with equal set of elements are equal even if their particular element order doesn't match. This is to support the diverse set of ASN.1 encodings.

TypeWrapper

TypeWrapper is also converted into a nil subclass. There's a new generic Protocols.MessageForwarder subclass of nil with absolutely minimal protocol. It has only #doesNotUnderstand: and couple of supporting methods prefixed with $_ to minimize the chance of clashes. TypeWrapper is a subclass of Protocols.MessageForwarder. It's again striving to minimize the potential of selector clashes. We've switched to accessing protocol using $_ prefix (so #value or #type won't be a problem anymore). It defines #yourself so that things don't get unwrapped unintentionally (losing the attached type information). It extends printing to provide visual hints of wrapper presence.

We have also fixed up comparison for TypeWrappers to allow comparison of wrapped and unwrapped values, so the following should answer true:

```
5 = (5 withAsn1Type: INTEGER new)
```

We've also cleaned up the constructors and added permissibility check to most of them. TypeRealizationError is now signaled when an invalid wrapper is created, for example:

```
5 withAsn1Type: BOOLEAN new
```

The instance accessors (#_type:, #_value:) can be used to avoid the checking if necessary.

We have renamed #asAsn1Type: to #withAsn1Type:. The former was misleading since it doesn't convert an Object into and ASN1.Type but simply wrapps it in a TypeWrapper.

Choice

Choice is also a wrapper around an object that is used as an actual value for a CHOICE type. Its main role is to disambiguate the cases where it's not clear which of the CHOICE alternatives is represented by given object. In this release we have made Choice a TypeWrapper subclass to provide the same kind of transparency and for more consistency in general. Choice accessors were also replaced with accessors prefixed with $_ (to reduce the chance of clashes with application domain). We've added an #asAsn1Choice:[symbol:] constructors to encapsulate and simplify creation of Choices.

The previous release was strictly forcing use of Choice instances for all CHOICE types. While they are definitely necessary to disambiguate some cases (e.g. a choice with four integral options), it's definitely sometimes desirable to represent choices as real classes (e.g. X509.Name). Also in many cases there will be no ambiguity (e.g. a choice with a boolean, integral and string option). In these cases forcing the wrapping in a Choice instance might be perceived as gratuitous. Therefore we have added provisions allowing arbitrary objects to be used in place of Choice for marshaling purposes as long as they implement #asn1ChoiceType and #asn1ChoiceValue (see for example X509.Name). The default implementations in Object make the marshaling machinery do a guessing game where it simply iterates over the CHOICE elements using the #permits: logic. The first one that succeeds is it. If it's the wrong one you may either get a marshaling error or it may marshal successfully but not as intended. Note, however, that in many cases (e.g., X509.Name) there is no need to guess and the #asn1ChoiceType can point straight to the right type, so users can avoid the perfomance penalty and ambiguity with little effort.

Note that the two selectors mentioned above are replacing #valueType and #value used for similar purpose, because the former selectors are too prone to name clashes with other frameworks.

However there still will be cases that will be ambiguous (e.g. a choice of simple types). In these cases the users are encouraged to employ Choices wrapping the objects. Using Choice instances is also efficient because the Choice embeds all the necessary type information so there's no need to do the #permits: trials.

To support more consistency we've added #useChoice flag to CHOICE type which instructs the marshaler to wrap unmarshaled value in a Choice instance. This way users can control which CHOICEs are meant to employ Choice instances and which aren't. Since the Choice instances prevent ambiguities, the default is to use those. Users have to turn the Choice usage off explictly for specific CHOICE instances.

Note that having the marshaler use Choice instances doesn't prevent mapping to user classes, as is demonstrated by X509.Name, but you need to do the unwrapping manually.

Enumeration

Enumeration is almost the same thing as TypeWrapper, so it's now a subclass of TypeWrapper which also gives it more transparency and consistency with TypeWrapper and Choice. For more consistency with the TypeWrapper hierarchy we've replaced Enumeration>>valueSymbol with #_symbol and Enumeration class>>type:valueSymbol with #type:symbol:. We have added permissibility checking to Enumeration constructors and added #asAsn1Enumeration: constructor to Integer and Symbol.

Similarly to CHOICE>>useChoice, there's now also ENUMERATED>>useEnumeration flag which controls if enumerations are unmarshaled as bare Integers or as Enumeration instances. The default is true to preserve backward compatibility.

The changes also required updates in ENUMERATED. Only one extension marker is allowed, as per the spec. We added support for auto-numbering of elements, so it is now possible to add just bare symbols and not only associations (symbol -> integer). Auto-numbering kicks in lazily when needed. Modification of an existing type will be handled properly as well. We have fixed ENUMERATED>>includes: and dropped #asENUMERATED as it cannot create valid Enumeration instances.

INTEGER

Similar changes were required in INTEGER. We have replaced the enumeration constraint with explicit support for value "identifiers". The constraint approach wasn't quite appropriate because the identifier are not constraining, i.e. a value is not made invalid just because it's not mentioned amogst identifiers. For this we have also added supporting API: #identifiers, #identifiers:, and #identify:as:.

Marshaling

The encoding retention flag was moved from marshaler level to type level, allowing to fine tune retention behavior. Applications can now selectively enable retention for specific types where it is useful and avoid the overhead for the rest. The accessor for this flag is Type>>retainEncoding:.

Moreover the encoding retention support was refactored into a pluggable EncodingPolicy component of the marshaling stream. The EncodingPolicy also serves as a dummy "do nothing about encodings" policy and the existing encoding support is represented by the RetainEncodings subclass. This allows people to turn encoding support on and off easily. EncodingPolicy support was moved up to BERStreamBasic, therefore pretty much any ASN.1 stream can now retain encodings, or do something else at the various encoding interception points if provided with customized EncodingPolicy. An interesting example of such custom policy is the PrettyPrinter which pretty-prints indented structure of encoded entities onto a stream as it marshals them. The streams now have their #defaultEncodingPolicy. The defaults are set so that only the DERStream defaults to RetainEncodings for backward compatiblity.

We've made numerous fixes in the marshaling logic. Most notably marshaling of implicit/explict tags had a lot of problems before.

There was also an issue with distinguishing the cases where nil represents a NULL value and where it represents absence of an optional element. This is the case of X.509 AlgorighmIdentifier, where element "parameters" is defined as ANY OPTIONAL. Some algorithms (the RSA signing ones) prescribe that their parameters should contain ASN.1 NULL value. Marshaling machinery has no chance to distinguish if nil means absence (as with the DSA algorithm) or presence of NULL. Therefore to express presence in such case the marshaling machinery now expects a nil wrapped in a type wrapper (with type ANY in this case, but it could be a CHOICE or NULL as well). Type ANY now allows general use of TypeWrappers to direct marshaling of ANY to type-in-hand rather than typeless marshaling.

X.509

X509 classes were moved to a new namespace, X509. Consequently the X509 prefix was removed on most of the classes. To improve backward compatibility we've added an alias Security.X509Certificate pointing to X509.Certificate. Most of the other classes are basically private so the rename shouldn't cause too much trouble. The other public class X509Registry kept its name, so it should work the same provided your application adds the import of the X509 namespace.

The most significant development in this release is switching to the new automatic mapping support of the underlying ASN.1 framework. It effectively removed all the marshaling code from the X509 framework and automatically handles not only decoding but also encoding of certificates. So it is now possibly to generate X.509 certificates from VisualWorks. The changes to the certificate APIs are minimal, just enough to provide simple way to create a certificate. Most of the X509 entities comprising a certificate are now created lazily upon access. X509Name now also provides simple creation API via #add: method taking an association with the attribute type name and value expressed as an association. We also needed to add API to actually sign a certificate, #signUsing:, which takes an instance of signing algorithm as the parameter. The algorithm has to be set up with the keys upfront so that it can do its job when #sign: is invoked. Access to various algorithms is provided via specific constructors on X509AlgorithmIdentifier. Here is how to generate a self-signed certificate

```
cert := Certificate new.
name := Name new
        add: 'C' -> 'US';
        add: 'L' -> 'Cincinnati';
        add: 'O' -> 'Cincom Systems';
        add: 'OU' -> 'Cincom Smalltalk';
        add: 'CN'-> 'Test CA';
        yourself.
cert
    serialNumber: 1000;
    issuer: name;
    subject: name;
    notBefore: Date today;
    notAfter: (Date today addDays: 7);
    publicKey: aPublicKey;
    forCertificateSigning;
    yourself.
cert signUsing: (
    DSA new
        privateKey: aPrivateDSAKey;
        yourself).
marshaler := DERStream on: (ByteArray new: 100).
marshaler marshalObject: cert withType: cert asn1Type.
marshaler contents
```

The X509CertificateFileReader can now read both OpenSSL PEM files and the ca-bundle.crt file. There is a slight change though, since X509.Certificate doesn't have the privateKey instance variable (to reduce the

chance of somebody sending out the certificate along with the private key by accident), both keys and certificates are now returned by the reader in the same OrderedCollection in the same order as they were in the file.

The X509Registry has been simplified as well. There were a few unused shared variables that were causing some confusion. All the associated API has been redirected to the default registry instance.

SSL

Recent improvements in the X.509 framework allowed us to eliminate a long standing limitation. The server now passes the list of all CA names from its SSLContext's registry and the CertificateRequest message now reads and writes those. With this change a VisualWorks server can properly advertise the list of all recognized certificate authorities.

Opentalk

Protocols-Common

To reduce duplication of functionality among the various networking frameworks, we are introducing a new package, Protocols-Common, gathering useful components and extensions to share. Currently it contains things like Struct, TimedPromise, Base64StreamEncoder, etc. To avoid dependencies on any of the specific frameworks a new namespace, Protocols, has been created and imported into the existing namespaces. Consequently some of the classes were moved to this new namespace. Some of the components underwent further development and clean up.

There's now a new class, Protocols.ProtoObject. It is a nil subclass meant to be a trimmed down version of Object with a reduced set of basic capabilities (comparing, printing, inspecting, class membership, some system primitives). The recognized set of selectors is chosen so that there's is only minimal probability of conflict with application domain selectors (intentionally avoiding things like #value, #size, #name, etc). Note, however, that in order to satisfy some of the more prevalent expectations of the environment it does define #=, #==, #hash, and #class.

Protocols.Struct is subclass of ProtoObject. In the spirit of our previous incarnations of Struct, its protocol supports smalltalk style accessors (via a #doesNotUnderstand: override), Dictionary style accessing via #at: and #at:put:, and a reduced subset of Dictionary protocol. As before it defines equivalence in terms of structural comparison. Again it intentionally

avoids selectors that are likely to clash with application domains, so for examplike the 'name' is accessed via #structName. Struct grows transparently by fixed amount (currently 5).

There is also a new Protocols.MessageForwarder. It is again a subclass of nil but with absolutely minimal protocol. It has only #doesNotUnderstand: and couple of supporting methods prefixed with $_ to minimize the chance of clashes.

Improving firewall traversal capabilities of Opentalk-STST

Opentalk-STST checks the receiver of each incoming request to verify that it is indeed a local object. However this check is based on IP addresses which creates a problem with firewalls or with any other network setup involving NAT (network address translation). A server in a DMZ zone is at least one firewall away from the internet and so the chances that the IP address in the request receiver's reference will match the IP address of the host that runs the server is virtually zero. As a consequence, Opentalk servers were unnecessarily hostile to firewalls and internet. However an Opentalk server should be able to work at least as well as a WS/SOAP server if you avoid passing objects by reference (which is exactly what WEB/SOAP communication boils down to). Therefore we have added configurable locality check support, see AdaptorConfiguration>>localityTest:. The check is expressed as a configurable 2 argument block which takes an ObjRef and an Adaptor instance and answers true if the reference is local to that adaptor and false otherwise. This allows users to use customized locality tests that can take into account any external access points of the broker.

Configuration of process prioirities

We've added configuration parameters for priority of the rest of background processes employed by Opentalk.

ConnectionAdaptorConfiguration>>listenerPriority: is for the listener process employed by connection oriented brokers. Its responsibility is waiting for incoming connection requests and setting up Transport instances to handle the connections. There's only one listener process per broker. Note that connection-less brokers don't use listener processes.

TransportConfiguration>>serverPriority: is for server processes. There's one server process associated with each Transport. There's usually single Transport instance for connection-less brokers. Connection-oriented brokers employ one Transport instance for each established connection.

Each transport uses a server process to read incoming messages, unmarshal them and dispatch them for execution. Note that actual execution is performed by worker processes.

Message Interceptors

This release introduces message interceptors, a fairly common pattern employed by many middleware frameworks for distributed computing (e.g. CORBA) to allow applications to observe and possibly intervene with processing of remote messages. While there aren't many fundamental differences between message interceptors and broker event handlers, which have been available with Opentalk for a while, there are certain advantages to modelling these handlers as objects, rather than add hoc blocks hooked into the broker events. For example it is much easier to pass state from one interception point to another in the context of an interceptor object. More detailed description of the new message interceptors follows.

BasicObjectAdaptor now maintains a ProcessingPolicy, which is configured as a processingPolicy aspect of AdaptorConfiguration. The main responsibility of ProcessingPolicy is to provideInterceptorDispatcher to any incoming/outgoing request. By default a new instance of InterceptorDispatcher configured with fresh set of MessageInterceptor instances is created to handle each request/reply pair. However there's nothing preventing the policy to optimize by reusing dispatcher/interceptor instances if the processing is stateless. The kinds of dispatchers and interceptors used depend on the policy which is currently configured with dispatcher class and a sequence of interceptor classes to use. These will be different for different protocols as the interception points might be different for different protocols, consider for example message envelope processing in SOAP. Users may or may not use their specialized class of dispatcher (if some pre/post processing is necessary before/after interceptors get invoked) and usually will provide custom interceptor classes filling in actual processing at specific interception points. Interceptors have a back pointer to their dispatcher which allows interceptors to reflect on other interceptors in case such coordination is necessary. Therefore the dispatcher also serves a a sort of "processing context" for the interceptors.

To preserve backward compatibility with old configurations that don't specify processing policy in their configuration, there's now MarshalerConfiguration>>defaultProcessingPolicyClass which allows to default to the right type of Interceptors/Dispatcher which can deal with protocol specific events. For example SOAP brokers need

SOAPInterceptorDispatcher which is aware of the message envelope events. The policy default is delegated to marshaler configuration because it is the one configuration component that reflects the application protocol.

The same interceptor is expected to process corresponding request and reply. Interceptors are assigned to a request as soon as it is created on the client or server side, before any interception points are reached. Similarly each reply gets the dispatcher from its corresponding request as soon as possible as well. On the client side the reply has to be first matched with the corresponding request. If the request is not found then a new dispatcher is obtained from the processing policy so that the interceptors can process events of the orphaned reply.

Multicast broker improvements

There was a bit of confusion in the multicast brokers with regards to the function of the transport #accessAddress. The most prominent presentation of the confusion was that you couldn't restart an mcast broker on some platforms (e.g. Windows) because once the broker started, the accessAddress was set to the multicast group address and on subsequent restart the broker tried to bind its socket to the mcast address. Windows XP and possibly other platforms refuse that (although Stevens says that on some platforms it is a valid way to restrict a UDP socket to receive multicast packets only, as oposed to unicast packets targetted at the same port number). To address this issue we have separated the notion of group address from the access address. Access address now represents only the address to which the broker socket is bound ("any" if not specified). Group address is the address of the group that the broker belongs to. BcastTransport now caches its group address in a dedicated instance variable, #groupAddress, while McastTransport's group address is the first of its mcast addresses (there's usually only one).

New #mcastAddresses, #ttl and #loopBack parameters were also added to McastTransportConfiguration, so that users can configure those. The default multicast address is now configurable on the class side of McastTransport.

DST

New implementation of weak dictionaries

We had reports of serious thrashing of the WeakKeyDictionaries in some specific circumstances. In particular this would happen with DSTtypeAny.CORBATypeMap under heavy load involving a lot of incoming objects marshaled as type Any. We narrowed it down to overly aggressive

shrinking strategy in the existing implementation of WeakKeyDictionary. The new strategy allows to fine tune the growth parameters and defaults to somewhat less aggressive shrinking. It also allows turning shrinking off altogether which is the most efficient approach in many scenarios. For more details about tuning the parameters, please refer to the class comment on WeakKeyDictionary.

Since this is a critical component of the DST machinery we decided to also keep the old implementation as OldWeakKeyDictionary and OldWeakValueDictionary and added configuration APIs throughout DST (in the DSTObjRef hierarchy and DSTtypeAny) allowing to plug in custom weak collections for each specific purpose. This will allow users to roll back to the old strategy in case the new one fails to match the performance of the old one in some unforeseen scenario. Note that this API allows to do this in fully forward compatible way and also opens up the possibility to use completely customized solutions.

Extended Message Interceptor API

Based on customer requests we have extended some of the interceptor events to include more parameters. Here is a sample of current sequence of interception events as logged in the transcript using the DSTSampleComputeService example:

```
clientInvokePreSend: a GIOPMessage
        context: an ORBContext
        target: a (remotable) "DSTSampleComputeServiceInterface"
        operation: ... ::carmichaelNumber
        parameters: #(2465)

preSendMessage: a GIOPMessage
        context: an ORBContext
        target: a (remotable) "DSTSampleComputeServiceInterface"

preReceiveMessage: a GIOPMessage
        context: an ORBContext
        target: a DSTSampleComputeService

targetInvokePreReceive: a GIOPMessage
        context: an ORBContext
        target: a DSTSampleComputeService
        parameters: DSTsignature (2465)

targetInvokePostReceive: a GIOPMessage
        context: an ORBContext
        target: a DSTSampleComputeService
        result: true
```

```
postReceiveMessage: a GIOPMessage
    context: an ORBContext
    target: a DSTSampleComputeService
    reply: a GIOPMessage

postSendMessage: a GIOPMessage
    context: an ORBContext
    target: a (remotable) "DSTSampleComputeServiceInterface"
    reply: a GIOPMessage

clientInvokePostSend: a GIOPMessage
    context: an ORBContext
    target: a (remotable) "DSTSampleComputeServiceInterface"
    operation: operation ... ::carmichaelNumber
    result: true
```

(Full operation names will occur where ellipses are shown.) The new extended events call the old ones by default to preserve backward compatibility with older interceptors.

Application Server

- Fixed a memory leak in the ISAPI gateway
- Changed the perl gateway to use SERVER_ADDR rather than SERVER_NAME, which is needed when the server is doing NAT.
- Added logic for HEAD request processing in general, and specifically when serving files. Corrected some minor problems with expiry dates and cache control.
- Support for server-side comments (<%-- ... -->) was missing

Documentation

This section provides a summary of the main documentation changes.

A general change that is beginning to work its way into the PDF documentation is that page numbering is changing to a chapter-page format. This is done to accommodate those users who print the documentation, reducing the need to print chapters that have not changed.

Another change will be the post-release launch of the documents in web page format. Watch the Cincom Smalltalk site for the launch.

Advanced Tools Guide

Obsoleted. Content has been distributed to the new *Tool Guide* and *Basic Libraries Guide*, or deleted, as appropriate.

Basic Libraries Guide

Mostly a spin-off from the *Application Developer's Guide*, but including material from other documents and new material as well, this document describes the content and use of the most common and foundational libraries, for purposes of application development.

Tool Guide

Mostly a spin-off from the *Application Developer's Guide*, but including material from other documents and new material as well, this document describes the VisualWorks tools and their use in application development.

Application Developer's Guide

Changes to the ADG continue the trend of reorganization to make this more of a user's guide than the all-purpose manual that it has been in the past.

- Further updated to reflect the current use of packages/bundles for representing code organization
- Several "library" chapters have been spun out into the *Basic Libraries Guide*.
- Major revision of the graphics chapter, including inclusion of animation facilities
- Various smaller updates.

COM Connect Guide

No changes

Database Application Developer's Guide

Minor updates

DLL and C Connect Guide

Several small changes

DotNETConnect User's Guide

Updated to reflect improvements made for 7.4

DST Application Developer's Guide

No changes

GUI Developer's Guide

Various small changes

Internationalization Guide

No changes

Internet Client Developer's Guide

No changes

Opentalk Communication Layer Developer's Guide

- New chapter on processes and scheduling
- Updates to broker configuration

Plugin Developer's Guide

No changse

Security Guide

- Limitations for SSL updated
- New ASN.1 chapter added

Source Code Management Guide

No changes

Walk Through

Major update, including a new chapter creating a Hello World example.

Web Application Developer's Guide

No changes

Web GUI Developer's Guide

No changes

Web Server Configuration Guide

Minor updates

Web Service Developer's Guide

No changes

TechNotes

The following documents in the TechNotes directory have been updated:

ImplementationLimits7x
This document has been updated for 7.4.

Goodies

The traditional "goodies" directory has been renamed to "contributed," to reflect better that the contents are not "toys," but serious contributions to the product. The subdiretory structure has also been flattened, eliminating the "parc" and "other" distinction.

In this diretory is a wide range of software contributions, from tool enhancements, full evaluation copies of applications, utilities, and even some toys and games.

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.

4

Preview Components

Several features are included in a **`preview/`** and available on a “beta test,” or even pre-beta test, basis, allowing us to provide early access to forthcoming features. Several are described in the following sections. Browse the directory contents for last minute inclusions.

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.

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`**.

Store Previews

Store for Access

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

The enhancements include:

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

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

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

Store for Supra

In order to allow Store to use Supra SQL as the repository, the StoreForSupra package provides a slightly modified version of the Supra EXDI, and implements circumventions for the limitations and restrictions of Supra SQL which are exposed by Store. The Store version of Supra

EXDI does not modify/overide anything in the base SupraEXDI package. Instead, modifications to the Supra EXDI are achieved by subclassing the Supra EXDI classes.

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

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

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

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

StoreForSupra installation instructions

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

- Sample SQL for creating the Store administration use :

  ```
  create user BERN password BERN DBA not exclusive
  ```

6. Load the StoreForSupra parcel.
7. To create the Store tables in the Supra database, run the following Smalltalk code from a workspace (You will be prompted for the Supra database name, the Supra administration user id and password.)

   ```
   Store.DbRegistry goOffLine installDatabaseTables.
   ```

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

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

New GUI Framework (Pollock), Feature Set 2

Pollock remains in preview in 7.4.

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, 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. Pollock has done away with Controllers. 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?

Pollock now as all of it's widgets, and most (but not all) of the internal turmoil in the APIs and guts of the system are completed. Most of the behavior to match the features of Wrapper are completed. Major changes and additions to the Feature Set 2 release include:

- WinXP Look and Feel
- New Keyboard Hot Key System
- Removal of Controllers
- Action framework for Hot Keys and Menus
- Text can load and save to XML
- New Frame bounds API (see comment on AbstractFrame)

Feature Set 3, which is the Production release of Pollock, will reach the stage of being a full replacement for our current Wrapper framework. Besides matching or surpassing all of the Wrapper framework's features and behaviors, it will contain the following major features and changes:

- Announcements: Replacing the Symbol based Trigger Events with Object based Announcements

- ValueSuppliers: These will provide type safe dynamic values for most widget attributes.
- Toolbar Toggle Buttons
- Toolbar as a fist class Pane
- DoIt attribute for Text
- Grid support of Tree as a Model, as well as display of a TreeView in the first column
- Grid support of ANY pane in ANY cell
- Drag Drop from ANY pane to ANY pane

The target for Feature Set 3/Production is currently the Summer/Fall of 2007. Once this is done, the VisualWorks Tools will being migrated to Pollock.

Security

OpenSSL cryptographic function wrapper

The OpenSSL package provides access to some of the encryption functions of the popular OpenSSL library (http://www.openssl.org). The functions currently available include ARC4, AES, DES and Blowfish, with support for the usual padding, and encryption modes. The API of this wrapper is modelled after the native Smalltalk encryption classes so that they can be polymorphically substituted where necessary. Since these classes use the same name they have to live in their own namespace, Security.OpenSSL. The intent is that each set of classes can be used interchangably with minimal modification of existing user code.

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

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

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

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

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

The current version should support usual OpenSSL installations on Windows and Linux (possibly other Unixes too, but that was not tested). There is only one interface class, with platform specific library file and directory specifications in it. You may need to change or add these entries for your specific platform. If you get a LibraryNotFoundError when trying to use this package, you need to find out what is the correct name of the OpenSSL cryptographic library on your platform and where is it located, and update the #libraryFiles: and #libraryDirectories: attributes of the OpenSSLInterface class accordingly. More information can be found in the *DLL and C Connect User's Guide* (p.51). To obtain the shared library for your platform, see http://www.openssl.org/source. Note that the library is usually included with many of the popular Linux distributions, therefore in most cases this package should just work.

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.

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.

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 ]
```

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.

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 **`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.

SmalltalkDoc is presently in preview status. When loaded, the Documentation editor may be used in the System Browser, and SmalltalkDoc content may be saved in file-out format. It is currently not possible to publish SmalltalkDoc content to a Store repository, though this is in development and planned for a subsequent release.

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

5

Microsoft Windows CE

WinCE devices have been supported since 7.3. Because separate documentation has not been developed or provided elsewhere, this section repeats the information provided in the releases here from the previous release.

Supported Devices

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 interprets holding the pen for approx 1.3 seconds as a right button press to open the operate context menu. This behavior 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 behavior.

.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.

6

Installer Framework

The Installer Framework has moved from goodies to the packaging directory. Until full documentation can be provided, the following notes are provided.

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.