Cincom

Cincom Smalltalk™

Release Notes 7.5

P46-0106-13

SIMPLIFICATION THROUGH INNOVATION®

Part Number: P46-0106-13

Software Release 7.5

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 7.5

These release notes outline the changes made in the version 7.5 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

visual.im is ReadOnly

Over the years we have implemented various mechanisms to help users restore a clean image. These involved restoring a “clean” visual.im, either from install media or an installed backup. We have also recommended never saving work to visual.im, but also to a newly named image file.

While these facilities remain available and the recommendation holds, with this release we are making visual.im read-only, making the recommendation more forceful. The save dialog will be more helpful in encouraging you to create a new image file.

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.

Delay and Time Change Interaction

It has been noted, particularly on Windows systems, that changing the time clock adversely affects applications that are in a Delay. The results vary, but can be as severe as an image hang or crash.

The problem occurs if the system gets out of synchronization with network time, so that a large correction is necessary. The problem can be minimized by configuring windows to run a full NTP server, which changes time gradually, rather than the default SNTP server that corrects the time all at once.

Arbitrary changes to the clock will continue to cause problems with running applications in a Delay.

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.

Limitations listed in other sections

- SNMP preview Limitations
- IDNA preview Limitations
- WinCE Known limitations

2

VW 7.5 New and Enhanced Features

This section describes the major changes in this release.

Virtual Machine

Splash Screen and Sound

Displaying the splash screen and playing the startup sound is no longer the default. Instead, VisualWorks, or your application, just start directly.

If you want to display a splash screen or play a sound, you still have those options, as described in the *Application Developer's Guide*.

New Platform VMs

The macx and macx86 virtual machines represent a major reworking of the implementation for the mac OSX platform. The virtual machine is not yet a universal application, so two virtual machines have been provided.

Multi-core/CPU Issues

We have found that when Windows runs single threaded programs in multicore/multi-cpu computers, the performance of such programs can be severely degraded. At times, the impact is so high that programs run at less than half their potential speed.

The VisualWorks VM, like any other single threaded program, is also affected by this problem. Note this issue is caused by the Windows process scheduler, which frequently swaps the core/cpu in which the single threaded program is running. This leads to CPU cache thrashing, and thus the slowdown.

Something that can be done to avoid this performance loss under Windows is to use the tool **`imagecfg.exe`** to add information to the virtual machine executable so that Windows allocates only one core or CPU to it at the time the program is loaded. Once the .exe file has this additional information, Windows will cycle through the available cores/CPUs automatically each time the VM is loaded, so you only need to do this once. This behavior has been preliminary confirmed under Windows XP.

To do so for the VisualWorks VM, simply execute:

```
imagecfg.exe -u vwnt.exe
```

Make sure the VM executable file is not read only, because otherwise the operation will fail.

The **`imagecfg.exe`** tool comes with various Windows resources such as the Resource Kit, and is also referenced to be in the folder i386/support/debug. While I could not find a download link at Microsoft, an Oracle document has a link to the following page where you can get imagecfg.exe.

http://www.robpol86.com/Pages/imagecfg.php

We are investigating the issue under Linux and other Unix operating systems as well.

Base system

US Daylight Savings Time Change

Beginning in 2007, United States Daylight Saving Time will be brought forward by 3 or 4 weeks in Spring to begin on the second Sunday in March, and extended by one week in the Fall to end by the first Sunday in November. The VisualWorks 7.5 default Daylight Savings Time setting has been changed to this new schedule. As usual, the VisualWorks TimeZone may be modified from the "Time Zones" selection from the Settings Tool.

When converting a UTC time in VisualWorks to local time the TimeZone relevant to the year of time must be used such as in

```
aLocalTimeZone convertGMTSecondsToLocal: utcSeconds.
```

Because VW only uses the current OS clock time in conversion for

```
Time now
```

the user must, as always, be certain to provide the correct TimeZone instance.

Dependents and Collections

(AR50266: Dependents and event table access slows down collection growing)

In recent versions, the dependents collection has become a weak dictionary, and we've also added an event table weak dictionary. These dictionaries are protected by semaphores, which slows down access to them in general, and can bottleneck it if there are many accesses occurring at once.

The become: operation tries to preserve the dependencies. To do this, it checks the dependents of each of the objects and compares them. If become: is a rare operation this is a relatively small issue, but become: is used to grow OrderedCollections, Sets/Dictionaries and so forth. This means that the global dictionary access becomes a serious bottleneck.

To partially correct this situation, by default collections are not allowed to have an event table. The system collections do not signal any. If a subclass wants to signal events, it should implement local storage for them and the myEventTable/myEventTable: methods.

The CollectionDependentsBackwardCompatibility parcel, listed in the Compatibility section of the Parcel Manager, restores the previous behavior. Note that this can have significant effects on system performance and stability.

Also, some of the classes that commonly have dependencies in the base system have been given a dependents instance variable to further reduce reliance on the global collection. These classes are

- CodeModel
- BrowserNavigator
- Parcel
- ObjectMemory
- ExternalDatabaseConnection

Faster Browsing with Large Bundles

When using large bundles, the implementation of containedItems was very inefficient. This has been improved, and browser opening in these circumstances is now significantly faster.

Headless and Runtime Startup Changes

Formerly, if an image has been moved and then started up headless, the location of the change log was not updated, so it could either cause an error due to an invalid directory, or use the change log from the old directory. There was a bad assumption that a headless image is always also a runtime image, so the change log could be ignored.

This has been changed so that an image always has a change log, unless it is either in runtime mode or has had the change log explicitly removed (e.g., via SourceFileManager>>removeAllSources). If an image that should have a change log is running with a GUI, then it will prompt the user for action when the change log cannot be found in the expected location. If it is running headless, then it will use the default location, and if the file does not exist, will attempt to create it.

Also, the “save headless” operation would create a change log for the headless image automatically, which did not happen in previous releases. This behavior has been retained, because it is in line with the idea that a headless image can still be in development mode, and that an image can be started up either headless or headful via the command-line. However, in 7.4.1 not only would it create the change log, but it would set that to be the change log of the saving image. This has been corrected, along with the setting of the imagePrefix when saving headless. Saving headless or headfull now saves a completely independent image, and does not change anything about the saving image.

Finally, more operations have become safe to do in headless mode. This is part of a move towards making attempts to do graphical operations in headless mode do nothing, rather than throwing exceptions and causing the system to exit. In particular, this affects font operations, allowing operations that attempt to display text to run.

Stream Operations for Standard I/O

The StandardIOStream class has been made a subclass of Stream, and has more of the stream operations, including supporting text and binary mode and setting the line-end convention.

StreamEncoder Error Policies

It is now possible to specify an error policy for a streamEncoder. If it encounters a character that it cannot encode, rather than using error:, it will delegate to the error policy. This was previously used in the VisualWorks Application Server to encode characters using the

XML/HTML convention of "&<code point>;" when serving web pages. (e.g. the Microsoft back-single-quote, if emitted into an iso character set without that character, would print as "&2019;".

User Interface

Text Editors May Be Read-Only and Selectable

It may be necessary to programmatically alternate the state of an input field or text editor between read-only and editable. This is possible by sending the message readOnly: to the editor's controller:

```
aTextEditorController readOnly: true.  "Be read-only"
aTextEditorController readOnly: false. "Be editable"
```

When the editor is made read-only this way, however, one may not bring the widget into focus by tabbing to it or select and copy its text. In VisualWorks 7.5 one may now send the message selectable: to the controller to determine whether the editor is read-only:

- The editor may receive focus by the keyboard.
- Its text may be selected and copied.
- A menu to copy, select all, or search text can be opened.

By default, a read-only editor is not selectable, but can be set to be:

```
aTextEditorController selectable: true.
```

This says that, if the text is read-only, allow the editor to receive focus or its contents to be selected and copied.

Invalid Characters in Window Labels

In previous versions, if a window label was set to a string that included characters outside the operating system's character set, this would cause a walkback. Characters that cannot be encoded are now replaced with a '?'.

Database

Fixes to isSafeAsQuery

In previous versions of the Object Lens, the isSafeAsQuery method (used for testing if a compiled method is suitable for being used as part of a query specification) relied on the methods peekForInstVarRead and

peekForInstVarWrite. These methods were always problematic, and were removed as part of a byte-code re-arrangement in 7.4 to remove size restrictions on compiled methods.

The isSafeAsQuery method has now been rewritten to use a more reliable mechanism, using InstanceVariableSearch.

TIMESTAMP datatype support in OracleEXDI

We have added support for the TIMESTAMP datatype. According to Oracle, “The new TIMESTAMP datatype is almost identical to DATE and differs in only one way: TIMESTAMPs can represent fractional seconds. The granularity of a TIMESTAMP value can be as little as a billionth of a second, whereas DATE variables can only resolve time to the second.”

Connection Pooling support in the ODBCEXDI

The following example demonstrates how to use use connection pooling in the ODBC EXDI. From the example, you can see that, on the server side, only two connections are created. The third connection makes use of the connection released by the second connect attempt.

```
| conn1 conn2 conn3 sess1 sess2 sess3 ans1 ans2 ans3 |

" Set the connection pooling on."
ODBCConnection connectionPooling: true.

" Set the connection pooling to relaxed match, the default is strict match."
ODBCConnection cpStrictMatch: false.

conn1 := ODBCConnection new.
conn1 username: 'userid'.
conn1 environment:'connectionString'.
conn1 connect: 'pwd'.

sess1 := (conn1 getSession).
sess1 prepare: 'select * from testtb'.
sess1 execute.
ans1 := sess1 answer.
ans1 upToEnd inspect.

conn2 := ODBCConnection new.
conn2 username: 'userid'.
conn2 environment:'connectionString'.
conn2 connect: 'pwd.
```

```
sess2 := (conn2 getSession).
sess2 prepare: 'select * from testtb'.
sess2 execute.
ans2 := sess2 answer.
ans2 upToEnd inspect.
conn2 disconnect.

conn3 := ODBCConnection new.
conn3 username: 'userid'.
conn3 environment:'connectionString'.
conn3 connect: 'pwd'.

sess3 := (conn3 getSession).
sess3 prepare: 'select * from testtb'.
sess3 execute.
ans3 := sess3 answer.
ans3 upToEnd inspect.
```

Store

Merge Tool Update

There has been a significant overhaul of the Merge Tool interface, including adding some extra features to better detect possible class moves and renames. The interface has been changed to allow a tree view, and to allow choosing resolutions at any level in the tree hierarchy. There are now options to hide/show items based on various criteria, including the ability to hide modifications that were made only in the image version and not in the version being merged in. This is on by default, greatly reducing the number of items that need to be looked at in a typical merge. The menus have also been re-organized, and a toolbar has been added.

WebService

XML Complex Type Mappings

Beginning with this release we stopped mapping the XML complex type to a collection. For example:

```
<complexType name="Items">
<sequence>
<element name="item" maxOccurs="unbounded"/>
</sequence>
</complexType>
```

is now mapped by default to a Smalltalk object, as are all other complex types:

```
<object name="Items">
<element name="item"  maxOccurs="unbounded"/>
</object>
```

This is a settable option that can be changed in WebServices.XMLTypesParser at the instance or class level.

Note that sending

```
XMLTypesParser useSequenceMapping: true
```

will restore the old (7.4.1) behavior.

Porting WS applications from 7.4.1 to 7.5

If a WS application has client classes derived from WSOpentalkClient, methods with pragma <selectorMap> should be renamed to selectorMap.

For example, in 7.4.1 MyOpentalkClient might have a class method:

```
urnMyChoiceDemoselectormap_TestChoiceService
    <selectorMap>
    ^'<ns:selectorMap xmlns:ns="urn:visualworks:operationSelectors">
    <ns:binding name="TestChoiceService"
  namespace="urn:MyChoiceDemo\selector\map"/>
    <ns:operation name="GetTestClass" selector="getTestClass:"/>
  </ns:selectorMap>'
```

In 7.5, this method should be renamed:

```
selectorMap
    ^'<ns:selectorMap xmlns:ns="urn:visualworks:operationSelectors">
    <ns:binding name="TestChoiceService"
  namespace="urn:MyChoiceDemo\selector\map"/>
    <ns:operation name="GetTestClass" selector="getTestClass:"/>
  </ns:selectorMap>'
```

Server classes derived from the WSOpentalkServer do not use a method to hold the selector map. The server will create the selector map every time it starts from the service class operation pragmas.

Net Clients

HTTP streaming improvements

We have done a significant overhaul of our HTTP infrastructure in this release cycle. The primary motivation was to add the ability to handle large attachments. Until now attachments had to be recreated in memory, in many cases. Now, most cases are handled by allowing streaming of attachment contents directly to and from files. Other enhancements also provide better control of other HTTP protocol features, such as message chunking.

This new implementation uses layers of stream wrappers handling different aspects of message transport processing. Things like compression, character encoding, HTTP chunking, etc are processed by their own wrappers.

On the receiving side, all HTTP attachments are now saved to an external file by default. This feature is configurable through the HttpBuildHandler>>saveAttachmentAsFile: aBoolean flag. If the option is set to false the attachment will be saved into an internal stream.

Attachment files are saved in a directory, which is by default named **`http-temp-files`** located in the image directory. To change the default directory use the following expression:

```
HttpBuildHandler defaultUploadDirectory: 'myDirectory'.
```

HttpBuildHandler creates a file name based on the Content-Disposition filename parameter. If a file with this name already exists a new name will be generated. In all cases the framework raises a notification, AttachmentFilename, allowing the user to override the file name on the fly. Here is an example:

```
[ response := client executeRequest: request
    ] on: AttachmentFilename
      do: [ :ex | ex resume: 'http-temp-files\MyTempFile.txt' ].
```

On the sending side, the main enabler for the uploading of big files is our new support for automatic chunking of outgoing messages. HTTP chunking is a transport encoding which splits the message in a number of smaller “chunks” and writes each chunk in sequence with a size indicator for individual chunks rather than for the entire message. Not having to compute the actual byte size of the entire message up front allows us to write even very large messages efficiently.

Here is how it is done. When writing a message, the stack of stream wrappers includes a ChunkedWriteStream, which collects the message body bytes in a buffer until the buffer is filled. The buffer size is settable and the default size is 4K. If the message body fits entirely into a single buffer, the message won't be chunked, but will be sent as is and the message header will include the “Content-length” field with corresponding byte size value. If the body is longer than the buffer, then when the ChunkedWriteStream is about to write the first chunk into the underlying stream it notifies the higher levels of the framework which add a "transfer-encoding: chunked" header field instead and the body will be written out in the chunked format.

As was mentioned, the chunk size is settable. To set the size globally (in bytes) send a message:

```
ChunkedWriteStream class>>defaultWriteLimit: aNumber
```

To set the chunk size individually for each message use the following pattern:

```
printer := HttpPrintHandler new.
printer chunkSize: aNumber.
printer writeMessage: anHttpMessage on: aWriteStream.
```

or

```
client := HttpClient new.
client chunkSize: aNumber.
response := client executeRequest: aHttpRequest.
```

This implementation also includes a new option that allows you to control compression of the outgoing messages. Setting the option to true will transfer the message in gzip format ("Transfer-Encoding: gzip"). The default value is false. Use the following pattern to enable gzip compression of messages:

```
client := HttpClient new.
client useGZipTransfer: true.
[ response := client executeRequest: request ]ensure: [ client close ]
```

or:

```
printer := HttpPrintHandler new.
printer useGZipTransfer: true.
printer writeMessage: httpMessage on: stream.
```

A compressed, chunked message will be sent out using both of these transfer encodings and will include the following header fields:

```
Transfer-Encoding: gzip
Transfer-Encoding: chunked
```

The body will be compressed and then split into chunks of the specified size.

> **Note:** Now that while we support automatic chunking of outgoing messages, it may cause some troubles with older HTTP 1.0 servers that do not support chunking. Sometimes the servers simply do not respond at all and the outgoing request will simply time out. Currently the only workaround we have is to increase the chunk size (using any of the methods mentioned above) to a size large enough to accomodate the entire message, so that it is not chunked. This, of course, means that the entire message will be constructed in memory first, as was the case in previous releases, and you will not get the benefits that come with chunking. We hope to come up with somewhat better workarounds in later releases.

SMTP Client

This release includes a new SMTPClient class. The class implementation is based on RFC2821 and includes methods to invoke the all client commands and authentication. Currently supported authentications are : login ('AUTH LOGIN') and plain ('AUTH PLAIN').

There are two utility messages to send a mail message:

send: *aMailMessage*

This message does the following:

1. Open connection
2. Login
3. Send the message
4. Quit and disconnect

sendMessage: *aMailMessage*

This message can be used to send a few mail messages over the same connection. The client has to be connected and authenticated before the utility is used. After the message is sent the client changes the state from transaction to authenticated. After all messages have been sent, the connection must be closed by sending a quit message.

Depending on the useAuthentication setting, the utilities will start the communication session with the EHLO or HELO commands. Based on the server reply for the EHLO command the client selects a supported authentication scheme and sends authorization information to the server.

If the hand shake is successful, the client is moved to the “authenticated” state. The useAuthentication option is true by default, which means the client always tries to start an authenticated session.

Examples

- Sending an authenticated message over regular connection:

```
smtpClient := SMTPClient host: 'smtp.cincom.com'.
smtpClient user: (NetUser username: 'username' password: 'password' ).
smtpClient send: message.
```

- Sending an authenticated message over SSL connection:

```
smtpClient := SMTPClient host: 'smtp.cincom.com'.
smtpClient user: (NetUser username: 'username' password: 'password' ).
smtpClient useSecureConnection.
smtpClient send: message
```

- Sending a non-authenticated message over regular connection:

```
smtpClient := SMTPClient host: 'smtp.cincom.com'.
smtpClient  useAuthentication: false.
smtpClient send: message
```

- Sending a few messages

```
smtpClient := SMTPClient host: 'smtp.cincom.com'.
smtpClient user: (NetUser username: 'username' password: 'password' ).
smtpClient connect.
[smtpClient login..
    smtpClient send: message1.
    smtpClient send: message2.
    ] ensure:  [smtpClient  quit ]
```

SimpleSMTPClient class is now deprecated, and eventually will be removed.

NetClient Closing Protocol Changes

We have changed POP3>>disconnect in 7.5. The method disconnects the client from a server but not sends quit anymore which means all started transactions won't be completed if the disconnect was executed without first sending quit.

We tried to clean up the net clients API to be more consistent for all clients.

Most Net Clients support this protocol:

- #connect - establish socket connection with a server
- #login - a client authenticates itself
- #close - a client signs off and closes socket connection #disconnect - close socket connection

The IMAPClient protocol is different:

- #logout - the connection is being terminated, and the server will close the connection
- #close - command permanently removes from the currently selected mailbox all messages that have the \Deleted flag set, and returns to authenticated state from selected state.

The following examples illustrate the differences.

SMTPClient:

```
smtpClient := SMTPClient host: 'smtp.cincom.com'.
smtpClient user: (NetUser username: 'username' password: 'password' ).
smtpClient connect.
[smtpClient login.
    smtpClient send: message1.
    smtpClient send: message2.
    ] ensure: [smtpClient close ]   <===== change
```

POP3Client:

```
pop3Client := POP3Client new.
pop3Client user: (NetUser username: 'username' password: 'password' ).
pop3Client hostName: 'hostname'.
pop3Client useSecureConnection.
[pop3Client connect] on: Security.X509.SSLBadCertificate do:
    [ :ex | ex proceed].
    [pop3Client login; list ] ensure: [pop3Client close ]  <===== change
```

Mail Clients SSL Connection

Some mail clients now support two types of connection: NetConnection, which is stream wrapper for a regular connection; and SSLConnection, which provides a connection over SSL.

Three new packages contain the mail client support for an SSL connection: POP3S, SMTPS.

To start using the SSL connection, the mail client should be sent a useSecureConnection message.

Examples

- SMTPClient

```
client := SMTPClient new.
client user: (NetUser username: 'username' password: 'password').
client hostName: 'smtp.com' portNumber: '9090'.
client useSecureConnection.
[ client connect ] on: Security.X509.SSLBadCertificate do: [ :ex | ex proceed].
client send: self message
```

- POP3Client

```
pop3Client := POP3Client new.
pop3Client user: (NetUser username: 'username' password: 'password' ).
pop3Client hostName: 'hostname'.
pop3Client useSecureConnection.
[pop3Client connect] on: Security.X509.SSLBadCertificate do:
      [ :ex | ex proceed].
[pop3Client login; list ] ensure: [pop3Client close ]
```

The useDefaultConnection method will change the client connection to the regular socket connection.

The useSecureConnection method will check if there is a class that handles the SSL connection and, if there is no class loaded, will try to load the corresponding SSL package.

Renamed Classes

While making these changes, we renamed some classes to provide a more consistent pattern for the net client class names:

- All class prefixes from the POP3 package were capitalized as the following:
  - From Pop3Client to POP3Client
  - From Pop3State to POP3State

  For backward compatibility we still preserved Pop3Client and Pop3Mailbox which are derived from POP3Client and POP3Mailbox. The classes are commented as obsolete.
- Renamed HttpConnection to HttpExternalConnection
- Renamed HttpStreamHandler to HttpConnection
- Renamed HttpsStreamHandler to HttpsConnection

Also note that the DeliveryMonitor class has been moved from the Mail package to the NetConfigTool package.

Security

Automatic seeding of DSSRandom

(AR50362 and AR51416) DSSRandom has occasionally thrown an exception on startup. It was easy handle the exception, but it was a symptom of a deeper, more serious problem. The problem is that we don't know how good (from the security point of view) is the auto-generated seeding that we use to seed the default random generator. In fact, we suspect that it is not nearly as good as it should be. Suppressing the exception just drives this problem further under ground.

Unfortunately, we don't have a complete solution for this problem at this time. Any good solution will require robust seed file management and some sort of entropy gathering facility. So at this point we have done the following:

- We cleaned up the superficial issues and added a new warning (AutogeneratedSeed) that will be signaled every time the seed is auto-generated. The purpose of the warning is to alert the users that the quality of the seeding may not satisfy their security requirements. The warning will stay in place until we can be reasonably sure that the quality of the seeding is satisfactory (in terms of the DSS standard).
- Where available, we will attempt to seed the generator from randomness sources provided by the operating system. Specifically we will attempt to access the CryptGenRandom facility on MS Windows platforms and the /dev/urandom on Unix platforms (see DSSRandom class>>osGeneratedSeed). If it succeed there will be no warning, but if this fails in any way we will fallback on the existing auto-generation facility with the warning.

The new warning can be somewhat disruptive if the OS facilities fail, but that's the purpose. Users have the following options to deal with this warning:

- Seed the generator explicitly before using it (see class comments)
- Handle the AutogeneratedSeed warning (it is proceedable)
- Turn the warning off globally with

```
DSSRandom seedWarning: false.
```

Note that turning off or proceeding the warning doesn't solve the issue of using a low quality seed, however that might be acceptable in some circumstances.

Simplified API for SSLContext certificate setup

An SSLContext of a server needs to be set up with a certificate and a private key that will be used for authentication. The same applies to the client's SSLContext when the client is expected to authenticate as well.

There are several kinds of certificates and keys, and the existing API was rather crude, requiring the user to pick the right method depending on the type and specific properties of the certificate (e.g: #rsaCertificatePair:, #rsaSigningCertificatePair:, #dsaCertificatePair:, etc). However all this information is captured in the certificate itself, therefore it was possible to add a much simpler API that will figure all that out for the user.

It is now possible to add any (supported) kind of certificate using SSLContext>>certificate:key:, which does the right thing.

Note that the certificate argument can be either a single certificate or a sequence of certificates representing the chain of certificates from the server certificate to the certificate of the root certificate authority (CA) that issued it. Note also that the root CA certificate may be safely omitted from the chain as it is not needed for the authentication process. The intermediate CA certificates may however be necessary for the other party to successfully validate the server certificate. Same applies on the client side when client authentication is used.

New Implementation of X509 Name Class

In SSL, whenever one party is being authenticated, the subject name from its certificate is processed by a "subject validator" block. The purpose of the block is to confirm or deny that the name matches the entity the authenticating party is trying to reach. Until this release the argument of the block was a Dictionary. This Dictionary was obtained by transforming the Security.X509.Name object from the subject field of the certificate into a Dictionary object.

However representing a Name as a Dictionary is just not good enough, especially since the Name can have multiple associations with the same key. Moreover a Name can provide more convenient APIs than a Dictionary. For example instead of

```
[ :name | (name at: 'C') = 'USA' and: [ name at: 'OU' = 'Cincom Smalltalk' ] ]
```

we can have

```
[ :name | name country = 'USA' and: [ name organzationUnit = 'Cincom
    Smalltalk' ]]
```

or if you prefer the short versions

```
[ :name | name C = 'USA' and: [ name OU = 'Cincom Smalltalk' ]]
```

To keep this change reasonably backward-compatible, we added at: and at:put: protocol to Security.X509.Name, so that most of the old style blocks still work fine.

Opentalk

STST and Firewalls

Previously there were severe limitations imposed on the STST protocol when sending requests to a broker behind a firewall. With properly configured localityTest (available as of VW 7.4) the broker is able to accept the request; however, it wasn't able to export new objects with a valid, publicly accessible address. So any communication would have to be restricted to passing all arguments by value (as opposed to passing by reference). This is functionally equivalent to capabilities provided by protocols like SOAP, but it severely restricts the transparency and freedom that the STST protocol is capable of.

In order to enable usable pass-by-reference semantics, the broker has to advertise its external (firewall) address in the object references that it generates. This will provide the clients with a valid (firewall) address to connect to. The clients brokers will then connect to the firewall and the firewall will forward those connection requests to the server broker behind it. Of course, the firewall has to be properly configured to do so (it is the same setup as with any other server that is exposed to the Internet).

It is now possible to configure the broker for this type of setup. All that needs to be done is, when the broker is being created via

```
BrokerConfiguration newAt: anAddress
```

the address parameter must be the external (firewall) address through which it can be accessed. The same applies to the various instance creation methods on the class side of BasicRequestBroker.

Note that if the server broker needs to send a request back to client, the roles are reversed. For this to happen the client has to export an object and provide accessible address in the corresponding object reference. If the client is sitting behind a firewall, it will have to be configured similarly as the server, with an external access address and with the firewall configured to forward to the client. Otherwise the server won't be able to

establish a connection back to the client. It is not yet possible for the server to reuse the previously established connection from the client to deliver requests from the server.

Another limitation is that currently the port number that is opened on the firewall for the purpose of forwarding traffic to an STST broker must be the same as the port number used by the broker on the physical host that it runs on. So if the open port on the firewall is 4242 the broker will also bind to port 4242 on its host machine. Consequently this port number must be available on the host.

Finally, note that there is a slight overlap of this new functionality with previously available support for binding to specific network interfaces. Previously if the address parameter to the broker creation methods wasn't a wildcard address (0.0.0.0) or completely unspecified (via #newAtPort:), it was expected to be an address of one of the network interfaces of the broker's host or localhost (127.0.0.1). The broker would then bind to that specific interface instead of all available interfaces. Specifying any other address would result in broker's failure to start. Now it is possible to specify some other address in which case the broker will expect it to be an external (firewall) address. In this case the broker will bind to all local interfaces and advertise the external address in all object references that it generates. For this the broker has to be able to distinguish a local address from an external one. However, there's currently no platform independent way to query for all the host interfaces and to obtain their IP addresses. Opentalk attempts to work around this limitation by obtaining the host name from the operating system and converting it to an IP address using reverse DNS lookup (see GenericProtocol class>>hostAddress). That yields satisfactory results in many cases, provided that the local DNS is configured properly and the host has single network interface. However it cannot handle multiple network interfaces or misconfigured DNS. Therefore we've added GenericProtocol class>>hostAddresses, which is an explicit, manually configured list of local network interface addresses. If the dynamic IP address discovery fails in some configuration, the user will have to configure this list manually. In most cases this shouldn't be necessary.

Bidirectional Connection Support (STST)

Bidirectional connection is a new capability that will be useful for Opentalk clients attempting to access a server from behind a firewall. Previously the server was unable to send requests back to the client unless that client was exposed through the firewall to incoming connections (which is usually blocked by client firewalls). This new capability allows the server to reuse a previously established connection

initiated by the client (which is usually allowed by client firewalls) for requests sent from the server to the client (e.g. callbacks to client objects). It also allows for better management of networking resources on heavily loaded servers, as there won't be a need for two separate connections per client when there are requests flowing in both directions.

Bidirectional support is controlled by picking the right type of adaptor for broker configuration. ConnectionAdaptor means the broker is bidirection capable, and AsymmetricConnectionAdaptor means it is not.

Currently only STST provides bidirection support. Choosing the ConnectionAdaptor for any other protocol will make no difference, the connections will still be asymmetric (Note that many of the protocols, such as HTTP, require that anyway). It is safe to mix bidirection capable and incapable brokers arbitrarily; they will simply fall back into the non bidirectional connection mode. Only two bidirection capable brokers can take advantage of bidirectional connections.

CAUTION! There is an issues that maintainers of bidirection enabled public servers need to be aware of. If the client brokers aren't explicitly set up with distinct access points, then they will most likely pick up their local host IP. The usual case is going to be that the IP will be from one of the private network ranges, e.g. 192.168.1/24. If several clients are in this situation, it may happen that they end up advertising the same address/port for their objects. The server will not be able to distinguish them. It will be most likely be the first client broker connecting the server that will get associated with that address under the bidirectional mode and all requests to that address/port will then be routed to that broker. This means that messages potentially meant for other brokers with that address will be routed to the wrong broker. Without bidirection support, the request attempts from the server will simply fail in these cases, but with bidirection mode they may just happen to work with unpredictable, possibly catastrophic results. So it is important to think about this when enabling the bidirection mode for a server. Probably the simplest work-around is simply requiring the client brokers to be configured with their external firewall address. That will guarantee uniqueness. There's no need to make any holes in the client firewalls, because requests will take advantage of bidirectional connections, it's just to make sure all the clients can be distinguished on the server side.

Application Server

New Perl CGI Gateway Script

This version of the gateway script adds support for requests that are sent in the “chunked transfer encoding,” without specified Content-Length. This can happen when the body of the request is large or compressed, e.g., when sending a large file attachment with the request. Also, note that as of VW 7.5, the HttpClient sends any request that exceeds a configurable limit (8K by default) in chunked transfer encoding.

However, in our testing we have observed some anomalies in the server behavior when dealing with chunked or zipped requests.

- Apache2 can handle “chunked transfer encoding” and this version of the Perl script works with that. However Apache2 seems unable to handle “zipped transfer encoding,” whether chunked or not.
- With IIS we have been unable to get chunked requests through the CGI interface to work at all. Non-chunked requests, which do specify full Content-Length should work however.

For fuller details on using the new Perl relay with Apache or IIS, refer to **`/waveserver/waverelays/perl/readme.txt`**.

The old Perl script is available, but has been moved to **`obsolete/waveserver/waverelays/perl/visualworks.pl`**.

Test Script for CGI Gateway Installation

Configuring web servers to relay requests to Smalltalk can be difficult to debug. In particular, the CGI relay has a configuration file which must be set up in the correct location, in a directory indicated by an environment variable, with the right permissions. All of these conditions may be different for the user ID with which the web server attempts to access the file, making them difficult to check.

In this release, we add a script, **`cgi2vw-test.pl`**, in the $(VISUALWORKS)/waveserver/waverelays directory. This is a simple perl script, which can be installed in the appropriate directory for CGI scripts. Once installed, accessing http://hostname/cgi-bin/cgi2vw-test.pl should give a report on whether the configuration file is visible and readable, and the settings of environment variables. This can be very helpful in setting up a relay.

Note, of course, that it is also possible to use other relays, including the perl relay which is simpler to configure and more efficient under Apache with mod_perl. It is also possible not to use a relay, and simply to set up a proxy to Smalltalk. For more information, see Chapter 5, "Using Front-End HTTP Servers" of the Web Server Configuration Guide.

Split Out URI-Encoding

The handling of URI-encoding has been split out into a separate package (URIEncoding). The intent is to change the base image encoding/decoding methods on the class side of URI to use this package as well. Apart from removing duplication, this also handles using different underlying encodings for the characters rather than encoding them by Unicode code point. However, there are some semantic incompatibilities that still need to be addressed for that, so for the moment we've simply split them out from the Application Server packages.

StreamEncoder Overrides

The streamEncoder overrides to provide an error policy object for streams have been incorporated into the base image, and removed from the Wave-Server-Base package.

Error Handling on Cookie Decode

In attempting to determine the encoding of a request, the server tries various different encodings. If there is an error in a particular encoding (most common if trying to utf-8 decode a non-utf8 string) then an exception handler catches that and concludes that that definitely wasn't the right encoding to try. However, there is no exception handler around the decode of cookie data, so the attempt to decode could result in a walkback.

Spaces After Colons in Headers

The relevant specifications allow but do not require a space after the colon in HTTP headers, and the server did not provide one. However, Internet Explorer can fail in some circumstances if a space is not present. So now we put one in.

COM Connect

Configurable Type Library Path for Deployed COM Server Images

Prior to this release COM Connect server images that require a type library have a crippling problem: the type library path is hard coded in the server image and that path typically becomes invalid when the image is deployed outside the development environment. As of this release, distributing a COM Connect server image with a type library has become easier. If the type library cannot be found several different attempts are made now to resolve its path before reporting an error. The type library file resolution strategy, in order of precedence, is now:

1. Use the type library path assigned in the image.
2. If the type library is registered, use the path in the Windows registry.
3. Search for the type library file in the designated default COM directory. Other directories may be specified to search for the type library in place of the default COM directory using the command line option '-typeLibPath'.
4. If the image is not headless and is a development image or includes the command line option '-ComDebug' then open a dialog requesting the type library path from the user.
5. All attempts to locate the library have failed. Report an error that the type library file cannot be found.

Exceptions Logged During COM Automation Calls

The COMTraceManager and COMTraceViewer are useful aids for tracing COM interface function calls made to and from your application as it executes. Anytime an exception occurs during a COM callback this trace information becomes indispensable. This release enhances the COMErrorReporter to now robustly log the call stack and maintain server operation following an exception during a COM callback. To enable and designate a log file for COM callback exceptions in an image, use the new COM->Error Notifications page from the Settings Tool. From this page you may also choose any of the alternate Runtime Packager (if loaded) error notifier classes to manage exception logging so that more details may be recorded but at the risk of suspending server operation.

Self Registration/Unregistration in VW COM Server Images

COM Connect server images may now register or unregister their interfaces with the Windows registry via the command line. Self registration means there is no longer a need to need to maintain a separate .reg file that is likely to be misplaced or outdated for the class PROGID, CLSID, type libraries, or OE and image path. Registration may be updated anytime the COM server VM or image files change location. Also, it adds for the first time a simple means for the server to be unregistered so it can be removed.

Class Registration Declaration

To manage registration for a Smalltalk class to be published through COM Automation first add a class-side comRegistrationSpec method to that class. This class method should answer an instance of COMAutomationRegistration set for the attributes of the class registered: type libraries, major and minor version numbers, version independent PROGID and description, and CLSID.

```
comRegistrationSpec
    "Returns the specification which allows registering the component
    from the command-line"

    ^COMAutomationRegistration new
        versionIndependentDescription: 'VisualWorks All Data Types
            Example';
        versionIndependentProgID: 'VisualWorks.AllDataTypes';
        typeLibraries: self typeLibraries;
        majorVersion: 1;
        minorVersion: 0;
        clsid: self clsid;
        yourself
```

A single COM server image may of course publish several classes at once. Each class implementing a comRegistrationSpec method should be added to a registration list COMSessionManager manages for command line registration:

```
COMSessionManager addRegistrationSupportForClass:
    AutomationAllDataTypes.
```

If a class to be published is not included in the COMSessionManager registration list it may still be registered/unregistered if its full unambiguous name is included as a command line parameter.

The COM Automation Wizard will auto-generate a comRegistrationSpec method on its Reg File page for any class that is being published. Simply complete all prior pages and fields and press the **Self Registration** button. This button will auto-generate the method and also add the class to the COMSessionManager list of classes for command-line registration.

Command Line Registration Examples

- Use the '-RegServer' command line option to register or re-register interfaces for the classes added to COMSessionManager for registration support. Look for type libraries in the c:\typelibs directory.

```
visual comserver.im –RegServer –typeLibPath c:\typelibs
```

- Unregister all classes in the server image added to COMSessionManager.

```
visual comserver.im –UnregServer
```

- Explicitly register only the SmalltalkCommanderCOMObject class assuming its type library is located in the default location.

```
visual vwcomsrvr.im –RegServer
Examples.SmalltalkCommanderCOMObject
```

New Command Line Options

The following new command line options are available this release for a COM Connect image:

-COMPath c:\myComPath:

Set the specified directory as the default COM directory.

-ComDebug:

Interactively query the user as needed to:

1. Open a debugger rather than terminate the server on an exception.
2. Correct a missing type library path.

-typeLibPath c:\typeLibPath1 c:\typeLibPath2:

Search one or more specified directories in place of the default COM directory for a missing type library.

-RegServer COMClass1 COMClass2:

Register zero or more specified COM classes with the Windows registry. Use the full unambiguous name for any class specified as a parameter. If no class parameters are specified register all classes in

the COMSessionManager registration list. Any class to be registered should implement a class comRegistrationSpec method.

-UnregServer COMClass1 COMClass2:

Unregister zero or more specified COM classes from the Windows registry. Use the full unambiguous name for any class specified as a parameter. If no classes are named as parameters unregister all classes in the COMSessionManager registration list. Any class to be unregistered should implement a class comRegistrationSpec method.

Documentation

This section provides a summary of the main documentation changes.

Basic Libraries Guide

General updates and corrections.

Tool Guide

General updates and corrections.

Application Developer's Guide

- New chapter on the Announcement framework.
- General updates and corrections.

COM Connect Guide

No changes

Database Application Developer's Guide

No changes

DLL and C Connect Guide

No changes

DotNETConnect User's Guide

Updated for 7.5

DST Application Developer's Guide

No changes

GUI Developer's Guide

No changes

Internationalization Guide

No changes

Internet Client Developer's Guide

No changes

Opentalk Communication Layer Developer's Guide

No changes

Plugin Developer's Guide

No changse

Security Guide

No changes

Source Code Management Guide

General updates

Walk Through

No changes

Web Application Developer's Guide

No changes

Web GUI Developer's Guide

No changes

Web Server Configuration Guide

No changes

Web Service Developer's Guide

Updates and reorganization

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.

Security

DH class method p:q: is deprecated

This method was based on an invalid assumption that the generator parameter g is produced deteministically and therefore can be reliably derived from the parameters p and q. However, properly it should be a random choice from a specified interval and thus the assumption is not valid. Without this assumption this API is not feasible anymore and is therefore being deprecated. Use any of the alternative instance creation methods instead.

DLL and C Connect

Due to the reorganization of the sources for the Windows engine in 7.2 (refer to “Dynamically loadable user primitives” in the 7.2 release notes), the file prbckref.lib is no longer required or provided, as stated on p. 147 of the document. Please disregard that paragraph.

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.

UUID Support

Support for UUID generation, using a number of different mechanisms, is now included in the UUID parcel, in **`preview/`**.

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 (e.g., running DbRegistry update74).
- Ability to create the database and install the tables all from within Smalltalk, as described in the documentation.
- No need to use the Windows Control Panel to create the Data Source Name.

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

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

Store for Supra

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

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

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

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

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

StoreForSupra installation instructions

1. Install Supra SQL
2. Create a Supra database
3. Use XPARAM under Windows to set the following
   - set Password Case Conversion = Mixed
   - set Supra tuning variable SQLLONGVARCHAR = Y
   - set Supra tuning variable SQLMAXRESULTSETS = 256

4 Start the Supra database

5 From the SUPERDBA user, create the Store administration user with DBA privileges.

- User ID BERN is recommended, password is your own choice.
- Sample SQL for creating the Store administration use:

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

6 Load the StoreForSupra parcel.

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

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

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

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

New GUI Framework (Widgetry), Feature Set 3 (In Progress)

Widgetry (formerly called “Pollock”) remains in preview in 7.5.

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 Widgetry.

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.

Widgetry

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. Widgetry 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. Widgetry 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. Widgetry will redo the Drag/Drop framework as a state machine. It will also use the Announcement 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: WinXP, Win95/NT, Win98/2K, MacOSX and Motif

- Better hotkey mapping

  A whole new hotkey system has been developed. It separates the definition of a keystroke from the action that it invokes. It also recognizes different keystrokes for different platforms, each being able to be assigned to the same action. With this, on Windows and X11 platforms, Control-C can be defined for "Copy" and on MacOSX Command-C. Hotkeys are no longer mapped through a dispatch table, and can be modified for each pane or window, on the fly at runtime.

- 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 Widgetry, the specs

will be able to be saved to disk, as well as loaded from disk at runtime.

- Unit Tests

  Widgetry will, and already does, have a large suite of unit tests. These will help maintain the quality of the Widgetry 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. At this time, there are over 29,000 unit tests.

- New Metaphor

  The Widgetry framework is based on a guiding metaphor; "Panes with Frames, with Agents and Artists." See "The New Metaphor: Panes with frames, agents, and artists" 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 Widgetry, 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 Widgetry, a pane, at its simplest, is akin to the existing VisualComponent. There is a Pane class. A pane may have subpanes. Widgets are kinds of panes. A Window is also a kind of pane, but it will remain in its 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 Pane.

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: bounds and inner bounds. The bounds represents

the whole outer bounds of the pane. The inner 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 bounds is the whole outer bounds of the pane, while the inner 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 bounds are the outer bounds of the pane, and the inner bounds is the 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 bounds that is larger than the inner 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 TextPane 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. Widgetry 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

Announcements

The change/update mechanism will be taking a back seat to the new Announcement framework (System-Announcements in the parcels/ directory). The framework replaces ValueModel with a new class, ObservableModel in the new Observables framework (Model-Observables in the parcels directory). Internal to the Widgetry 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.

UserInterface Class

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.

Window Classes

The Widgetry ScheduledWindow has two subclasses: ApplicationWindow and DialogWindow.

- ScheduledWindow is a full-fledged handler of all events, not just mouse events like the current ScheduledWindow.
- ApplicationWindow will be allowed to have menus and toolbars; ScheduledWindow and DialogWindow will not.
- ApplicationWindow and DialogWindow will know how to build and open UserInterface specifications, ScheduledWindow will not.
- Conversely the UserInterface will only create instances of ApplicationWindow and DialogWindow.

So, What Now?

Widgetry now has all of its widgets, and most (but not all) of the internal turmoil in the APIs and guts of the system are completed. All of the behavior to match the features of Wrapper are completed.

Feature Set 3, when completed, will be the Production release of Widgetry, and will be a full replacement for our current Wrapper framework.

Besides matching or surpassing all of the Wrapper framework's features and behaviors, it contains the following major features and changes:

- Announcements: Replacing the Symbol based Trigger Events with Object based Announcements
- Attributes: These will provide type safe dynamic values for most widget attributes.
- Toolbar Toggle Buttons (and Grouped Buttons)
- Toolbar InputField and Toolbar DropDownList
- A new "User Draw" Pane named Canvas
- DoIt attribute for Text
- Grid support of Tree as a Model, as well as display of a TreeView in any 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 Winter 2007/2008. Once this is done, the VisualWorks Tools will begin being migrated to Widgetry.

Security

OpenSSL cryptographic function wrapper

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

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

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

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

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

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

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

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

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

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

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

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

library for your platform, see http://www.openssl.org/source. Note that the library is usually included with many of the popular Linux distributions, in many cases this package should just work.

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

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

Opentalk

The Opentalk preview provides extensions to 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.

Opentalk HTTPS

This release includes a preview of HTTPS support for Opentalk. HTTPS is normal HTTP protocol tunneled through an SSL protected socket connection. Similarly to Opentalk-HTTP, the package Opentalk-HTTPS only provides the transport level infrastructure and needs to be combined with application level protocol like Opentalk-XML or Opentalk-SOAP.

An HTTPS broker must be configured with a SSLContext for the role that it will be playing in the SSL connections, i.e., #serverContext: for server roles and #clientContext: for client roles. Also, the authenticating side (which is almost always the client) needs to have a corresponding validator block set as well. The client broker will usually need to have the #serverValidator: block set to validate server certificates. The server

broker will only have its #clientValidator: block set if it wishes to authenticate the clients. Note that the presence or absence of the #clientValidator: block is interpreted as a trigger for client authentication.

Here's the full list of all HTTPSTransportConfiguration parameters:

clientContext
The context used for connections where we act as a client.

serverContext
The context used for connections where we act as a server.

clientValidator
The subject validation block used by the server to validate client certificates.

serverValidator
The subject validation block used by the client to validate server certificates.

Note that the same broker instance can be set up to play both client and server roles, so all 4 parameters can be present in a broker configuration. For more information on setting up SSLContext for clients or servers please refer to the relevant chapters of the *Security Guide*.

This example shows how to set up a secure Web Services broker as a client:

```
context := Security.SSLContext newWithSecureCipherSuites.
broker :=
    (BasicBrokerConfiguration new
        adaptor: (
            ConnectionAdaptorConfiguration new
                isBiDirectional: false;
                transport: (
                    HTTPSTransportConfiguration new
                        clientContext: context;
                        serverValidator:
                            [ :name | name commonName = 'Test Server' ];
                        marshaler: (
                            SOAPMarshalerConfiguration new
                                binding: aWsdlBinding)))
    )   newAtPort: 4242.
```

This example shows how to set up a secure Web Services broker as a server:

```
context := Security.SSLContext newWithSecureCipherSuites.
"Servers almost always need a certificate and private key, clients only when
client authetication is required."
"Assume the server certificate is stored in a binary (DER) format."
file := 'certificate.cer' asFilename readStream binary.
[ certificate := Security.X509.Certificate readFrom: file ] ensure: [ file close ].
"Assume the private key is stored in a standard, password encrypted PKCS8
format"
file := 'key.pk8' asFilename readStream binary.
[ key := Security.PKCS8 readKeyFrom: file password: 'password' ] ensure:
      [ file close ].
context certificate: certificate key: key.
broker :=
      (BasicBrokerConfiguration new
          adaptor: (
              ConnectionAdaptorConfiguration new
                  isBiDirectional: false;
                  transport: (
                      HTTPSTransportConfiguration new
                          serverContext: server;
                          marshaler: (
                              SOAPMarshalerConfiguration new
                                  binding: aWsdlBinding)))
      )   newAtPort: 4242.
```

This release also includes a toy web server built on top of Opentalk as contributed code, and is not supported by Cincom. It is, however, quite handy for testing the HTTP/HTTPS transports without having other complex infrastructure involved. So here is another example how to set up a simple secure web server as well:

```
| resource ctx |
resource := Security.X509.RegistryTestResource new setUp.
ctx := Security.SSLContext
      suites: (Array with: Security.SSLCipherSuite
            SSL_RSA_WITH_RC4_128_MD5)
      registry: resource asRegistry.
ctx rsaCertificatePair: resource fullChainKeyPair.
(Opentalk.AdaptorConfiguration webServer
          addExecutor: (Opentalk.WebFileServer prefix: #('picture')
              directory: '$(HOME)\photo\web' asFilename);
          addExecutor: (Opentalk.WebFileServer prefix: #('ws')
              directory: '..\ws' asFilename);
          addExecutor: Opentalk.WebHello new;
          addExecutor: Opentalk.WebEcho new;
          transport: (Opentalk.TransportConfiguration https
                serverContext: ctx;
                marshaler: Opentalk.MarshalerConfiguration web )
) newAtPort: 4433
```

Once the server is started, it should be accessible using a web browser, for example https://localhost:4433/hello.

Distributed Profiler

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

Installing the Opentalk Profiler in a Target Image

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

- Opentalk-Profiler-Core

Installing the Opentalk Profiler in a Client Image

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

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

Opentalk Remote Debugger

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

The remote debugger is contained in two parcels.

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

```
RemoteDebuggerClient startMonitor
```

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

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

```
RemoteDebuggerClient stopMonitor
```

The Opentalk-Debugger-Remote-Target parcel loads support for the image that is expected to be debugged remotely. To enable remote debugging this image has to be “attached” to a monitor, i.e., to the image that runs the remote debugger UI. Attaching is performed with one of the “attach*' messages defined on the class side of RemoteDebuggerService. Use detachMonitor to stop forwarding of unhandled exceptions to the remote monitor image.

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

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 asserts 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.

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 corresponding read/write/print methods
- the supported profiles are IIOPProfile and IOPMultipleComponentProfile, and anything else is treated as an IOPUnknownProfile
- the only supported service context is CodeSet, and anything else is treated as an IOPUnknownContext
- however it does not support the codeset negotiation algorithm yet; correct character encoders for both char and wchar types can be set manually on the CDRStream class
- the supported tagged components are CodeSets, ORBType and AlternateAddress, and anything else is treated as an IOPUnknownComponent

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

- there is a new IIOPTransport and CDRMarshaler with corresponding configuration classes
- these transport and marshaler configurations must be included in the configuration of an IIOP broker in the usual way
- the new broker creation API consists of the following methods
  - #newCdrIIOPAt:
  - #newCdrIIOPAt:minorVersion:
  - #newCdrIIOPAtPort:
  - #newCdrIIOPAtPort:minorVersion:

- IIOP proxies are created using Broker>>remoteObjectAt:oid:interfaceId:
- there is an extended object reference class named IIOPObjRef
- the LocateRequest capabilities are accessible via
  - Broker>>locate: anIIOPObjRef
  - RemoteObject>>_locate
- LocateRequests are handled transparently on the server side.
- A location forward is achieved by exporting a remote object on the server side (see the example below)

Examples

Remote Stream Access

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

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

“Locate” API

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

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

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

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

Transparent Request Forwarding

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

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

Listing contents of a Java Naming Service

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

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

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

List Initial DST Services

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

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

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 significant 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.

Internet Browser Plugin

The new VisualWorks Plugin is available as a preview for VW 7.5.

Beginning with the VisualWorks Plugin for VW 7.5, we provide both an ActiveX Control for Internet Explorer and a Gecko Plugin for use in Gecko 2.0 enabled browsers such as Netscape 8, Mozilla 1.7.5+ and Firefox 1.0, 1.5 and 2.0. For the time being, the Plugin is still only available on Windows platforms.

The Plugin parcel for VW 7.5 is not backward compatible with any of the old Plugin parcels. Please see preview/plugin/readme.txt for more information.

International Domain Names in Applications (IDNA)

RFC 3490 “defines internationalized domain names (IDNs) and a mechanism called Internationalizing Domain Names in Applications (IDNA) which provide a standard method for domain names to use characters outside the ASCII repertoire. IDNs use characters drawn from a large repertoire (Unicode), but IDNA allows the non-ASCII characters to be represented using only the ASCII characters already allowed in so-called host names today. This backward-compatible representation is required in existing protocols like DNS, so that IDNs can be introduced with no changes to the existing infrastructure. IDNA is only meant for processing domain names, not free text” (from the RFC 3490 Abstract).

Limitations

The current implementation in VisualWorks

- doesn't do NAMEPREP preprocessing of strings (currently we just convert all labels to lowercase)
- doesn't properly implement all punycode failure modes
- needs exceptions instead of Errors
- needs I18N of messages

USAGE

You can convert an IDN using the IDNAEncoder as follows:

```
IDNAEncoder new encode: 'www.cincom.com'
    "result: www.cincom.com"
```

or

```
IDNAEncoder new encode: 'www.cìncòm.com'
    "result: www.xn--cncm-qpa2b.com"
```

and decode with

```
IDNAEncoder new decode: 'www.xn--cncm-qpa2b.com'
    "result: www.cìncòm.com"
```

This package also overrides the low level DNS access facilities to encode/decode the hostnames when necessary. Here's an example invocation including a Japanese web site.

```
host := (String with: 16r6c5f asCharacter with: 16r6238 asCharacter), '.jp'.
address := IPSocketAddress hostAddressByName: host.
    "result: [65 99 223 191]"
```

The host name that is actually sent out to the DNS call is:

```
IDNAEncoder new encode: host
    "result: xn--0ouw9t.jp"
```

A reverse lookup should also work, however I wasn't able to find an IP address that would successfully resolve to an IDN, so I wasn't able to test it. Even our example gives me only the numeric result:

```
IPSocketAddress hostNameByAddress: address
    "result: 65.99.223.191"
```

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.