Cincom

Cincom Smalltalk™

VisualWorks®

Release Notes 7.10

P46-0106-20

SIMPLIFICATION THROUGH INNOVATION

Part Number: P46-0106-20

Software Release 7.10

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 7.10

These release notes outline the changes made in the version 7.10 release of VisualWorks. Both Commercial and Non-Commercial releases are covered.

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

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

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

http://www.cincomsmalltalk.com/

Product Support

Support Status

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

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

Product Patches

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

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

ARs Resolved in this Release

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

Distribution Designations and Non-crypto distribution

In compliance with U.S. security requirements, VisualWorks now has a commercial version without encryption code.

Installation directories names are also being updated, as follows:

<user-files>/vw7.10

Full commercial distribution

<user-files>/vw7.10pul

Personal use license (non-commercial)

<user-files>/vw7.10ne

No encryption distribution

where **<user-files>** is the selected installation directory.

VisualWorks on Vista or Windows 7

Microsoft Vista and Windows 7 operating systems impose additional restrictions on file permissions and applications. Accordingly, there are a few special considerations when using Windows.

Installing VisualWorks

You can install VisualWorks either as a regular user or an administrator, but only users belonging to the administrator group have write acess to the **Program Files** directory. Note that you can always install VisualWorks to other directories without complication.

When installing as an administrator, Windows will open a slightly cryptic prompt to confirm whether to run the Installer. Simply click **Continue** to proceed.

Note also that, when installing on 64-bit Windows 7, a dialog may display noting that the program might not have installed correctly. Our experience has been that the installation is correct, so click the option **This program installed correctly**, unless you know otherwise.

Uninstalling VisualWorks

If you installed VisualWorks to the **Program Files** directory, the install-uninstall shortcut in the **Start** menu will not work correctly. In this case, the Installer on the VisualWorks distribution CD must be used to uninstall the product.

Saving your Work

Since all directories under **Program Files** are write-protected, when working as a non-administrative user, your VisualWorks image files must be saved somewhere you have write privileges, such as your own **My Documents** directory.

Editing VisualWorks.ini

The **`bin/VisualWorks.ini`** file is installed as writable. After installation, all paths to all releases of VisualWorks in this file begin with the directory containing the VisualWorks home directory that you set when installing (so if you do not keep all your releases of VisualWorks within the same overall directory, you may wish to edit this file).

On a machine with more than one copy of VisualWorks installed, **`.im`** files will be assigned to the **`VisualWorks.exe`** of a given installation (unless they are not assigned to any). Double-clicking on a **`.im`** file opens the engine assigned by the **`VisualWorks.ini`** file of that **`VisualWorks.exe`**. If you already have an assigned version of VisualWorks on your machine, you can copy the more recent lines from your newly-

installed **VisualWorks.ini** file to the existing one. Alternately, you can reassign the **.im** file type to your newly-installed **VisualWorks.exe**. We recommend reassigning the file type by editing its path by hand. Using the **Browse** function the dialog provides to navigate to the new **VisualWorks.exe** often fails to work (the change of a file type from one program to another of the same name is not recognised as a change).

In addition, if VisualWorks is installed in the **Program Files** directory structure (as is done by the Typical installation), you must have administrator privileges to save changes. Being logged on to an administrator account is not sufficient, and you must **Run as administrator** the editor program, which might be VisualWorks or any text editor.

An alternative is to use the Launch Pad (Project Manager), which maintains its own version of **VisualWorks.ini** to select the appropriate object engine for an image.

Xtreams

Xtreams is a generalized stream/iterator framework providing simple, unified API for reading from different kinds of sources and writing into different kinds of destinations (Collections, Sockets, Files, Pipes, etc).

Xtreams is now included in the **xtreams** directory of the standard distribution. The code in this directory is supported by Cincom, while the code that remains in **contributed/xtreams** is not.

For instructions on usage, refer to the documentation at http://code.google.com/p/xtreams/.

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.

Purging Undeclared

When a parcel is only partially loaded, the purge of Undeclared will detect references in unloaded code. For example, if loaded parcel MyParcel contains method

```
FirstClassDefinedElsewhere>>extendingMethod
    ^SecondClassDefinedElsewhere
```

and both FirstClassDefinedElsewhere and SecondClassDefinedElsewhere are not loaded then:

- extendingMethod will also not be loaded but it will be in the parcel's unloaded code
- Undeclared *will not* contain FirstClassDefinedElsewhere (unless it is referenced in some other place)
- If SecondClassDefinedElsewhere and all loaded references to it are removed, and it gets added to Undeclared, then a purge of Undeclared will fail to remove it because of the reference in extendingMethod, even though that method is not loaded and there are no references to it in loaded code

This issue has existed since unloaded code was introduced, is recognised as undesirable and will be addressed.

Limitations on Windows of displayShape: methods

On more recent versions of Windows 7 it is not permitted to draw directly on the screen. Various methods in the Screen class in VisualWorks attempt to do so. This is implemented by creating an invisible window, drawing on it, and then destroying the window. However, this is a slow process, and methods that attempt to do animation using this primitive can become very slow and the animations can become effectively invisible. Use of these mechanisms is often used for animation of operations like dragging within a graphical editor. In such cases they can be replaced by graphics operations within that window, which will be much, much faster.

Operations like Rectangle>>fromUser: are not as slow because they can create the invisible window once, do numerous drawing operations, and then complete. But methods like the Screen>>displayShape:... family do not have this information. At the moment, use of these methods is discouraged while we examine possible solutions.

Publishing a Bundle as a Parcel

When **Publish as Parcel** is invoked on a bundle, we recommend selecting **Include bundle structure**.

If you choose not to select this, check that the bundle's own prereqs are what the parcel will need, since the prereqs of its subpundles will then not be assigned to the parcel (which may therefore show unexpected behaviour on load).

In the next release, the bundle structure will aways be saved when saving a bundle, and will no longer be an option.

2

New and Enhanced Features

This section describes the major changes in this release.

Virtual Machine

Large integer arithmetic improvements for big endian platforms

With the 7.10 release, we have newer and faster large integer primitives for big endian platforms. For Linux/PPC and OS X PPC, we sped up large integer primitives significantly by using specific instructions that change byte order on the fly. We also eliminated one-byte-at-a-time operation for SPARC, which resulted in additional speedups. In some cases, we measured up to 37% performance improvement because of this change alone.

Garbage collection performance improvements

We have made significant further improvements to the VM garbage collectors. The reduced overhead during garbage collection allows the VM to spend more time doing application work, resulting in faster execution of customer code.

There are two main changes. First, the scavenger is now significantly faster when there are numerous fixed and large objects in fixed and large space (respectively). Customers with fixed or large objects counts in the hundreds (or greater) should notice improved overall performance.

Second, the mark and sweep collectors have new data and OT (object table) compactors. We reviewed the implementation of the garbage collector data compaction algorithm, and significantly reduced its complexity. In addition, the OT compactor algorithm has

been rewritten using significantly less code, which should lead to faster execution. When taken together with other smaller optimizations, the new code runs one of our stress tests 21% faster. In addition, for 64-bits, we optimized how the OT is swept after the garbage collector's marking phase completes. In fact, we optimized it so much that we managed to optimize this step away completely. This enhancement will result in even faster garbage collector performance for 64-bit platforms, in addition to the improvement from the new data and OT compactors.

Improved handling of misaligned and different-endianness data

With the 7.10 release, we reworked how the VM handles misaligned data, as well as data stored in disfavorable endianness (for example, keep in mind that large integers are always stored in little endian format). The resulting VMs use significantly fewer instructions to perform the same operations. For instance, the Windows 32-bit executable is 2kb smaller just because of these changes. In general terms, fewer instructions leads to improved performance. Depending upon their use, applications that perform certain types of floating point arithmetic, or frequently accessing UninterpretedBytes, or heavily using DLLCC calls and callbacks, could see measurable improvements.

32-bit SPARC VMs now target v8 instruction set

With the 7.10 release, we have improved the overall performance for 32-bit SPARC virtual machines by targeting the v8 instruction set, as opposed to the v7 instruction set, at the compiler layer. One specific change that is very meaningful is that v8 introduced integer multiplication instructions. In general, the change means that the baseline performance of the virtual machine is improved across the entire binary. Some areas such as large integer multiplication will see a measurable performance improvement.

AIX virtual machines now compiled on AIX 6.1

With the 7.10 release, we have upgraded our AIX compilation environment to AIX 6.1. Our previous compilation environment, AIX 5.3, is no longer supported by IBM as of April 2012. In addition, all supported AIX versions require at least a POWER4 CPU. The VMs now take advantage of that requirement for the purposes of optimization, and so now they have the same CPU requirements.

Statically-linked Windows VM for deployment

Starting with the 7.10 release, we are providing a Windows VM binary specifically for single file deployment scenarios. This VM is statically linked to the relevant CRT, and so it does not depend on **`msvcrt100.dll`**. You can find this version of **`visual.exe`** under **`bin/win/static`** and **`bin/win64/static`**.

Changes to Image Signature Format

The header of the virtual image file, contains an "image signature" identifying the official release from which this image is descended; this signature does not change across image snapshots.

In VisualWorks versions prior to 7.10, the signature contained 4 bytes:

The meaning of the bit fields was as follows:

Byte 1:	M	M	M	M	M	M	M	M	— Major Number
Byte 2:	W								— Word Size
Byte 2:		m	m	m	m	m	m	m	— Minor Number
Byte 3:	R	R	R	R	R	R	R	R	— Release Number
Byte 4:						T	T		— Type of Release
Byte 4:								S	— Tailoring Status

In this release note, we will only concentrate on the major and minor numbers.

The major number is, when written in decimal, the version number of the release with the decimal point elided. For example:

"4.0" is 40 (0x28),
"4.1" is 41 (0x29),
"7.9" is 79 (0x4F).

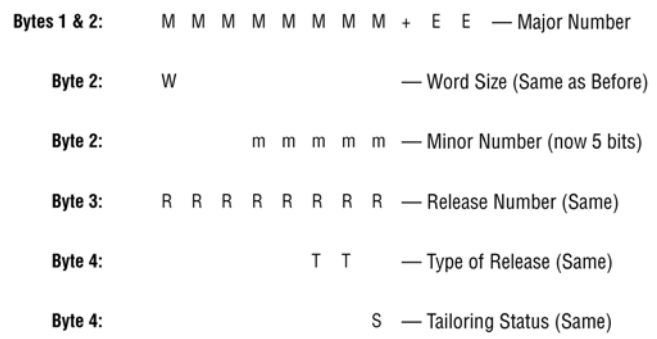
Because of this encoding scheme, the number after the dot could only take values from 0 to 9. Therefore, the current VisualWorks version number, 7.10, could not be represented.

For this reason, in VisualWorks 7.10, we have narrowed the "minor" field from 7 bits down to 5 bits wide, and assigned bits 5 and 6 of byte 2 to represent a second decimal place of the major-after-dot number, thus allowing values up to 39 to be represented. We have also introduced a Smalltalk API to interpret the bit-field structure of the image signature; starting with VisualWorks 7.10, client code is discouraged from directly decoding the bit-field structure of the 4 bytes.

For completeness, we are documenting the low-level format of the signature; however, the preferred way for applications to access this information is by calling the API instead of performing their own decoding.

Thus, here is the new bit-field format of the image signature:

Such that the meaning of the bit fields is now:



MMMMMMMM now represents the before-dot and the least-significant decimal place of the after-dot major number. EE (2 bits) represents the second decimal place of the after-dot major number. The full formula for the major number is now:

```
major = (M // 10) .
    EE * 10 + (M \\ 10)
```

The range of after-dot values is thus from 0 to 39.

Examples

Assuming W=0, mmmmm=0, the first two bytes of the signature for major "3.15" would be 35, 32 (0x23, 0x20): the least-significant decimal place of "15", 5, is encoded in byte 0 together with the before-dot "3", and the most-significant decimal place of "15" is 1, EE=01, encoded in bits 5 and 6 of byte 1: 16r00100000.

Similarly, "7.10" is 70, 32; and "5.12" is 52, 32.

As mentioned before, applications are discouraged from relying on these encoding details to interpret the encoded bytes; instead, they should call the API described in the next section.

Signature Decoding API

In order to facilitate the necessary code changes in applications that currently depend on the image signature's encoding, we have added the following three methods:

```
ObjectMemory class>>viReleaseSeries
ObjectMemory class>>viReleaseMajor
ObjectMemory class>>viReleaseMinor
```

Given a VI signature encoding for a hypothetical 7.10.3 release, the first message answers 7, the second message answers 10, and the third message answers 3. These methods can be used to reimplement variousclient functionality bits that currently depend on the VI signature'sencoding. To illustrate, here are some examples.

1. Using // and \\ to derive the series and the major values

The old code for this would be:

```
versionId := ObjectMemory versionId at: 5.
e cell addText: (versionId // 10) printString , '.' ,
                (versionId \\ 10) printString.
e newRow.
```

Henceforth, use the following code instead:

```
e cell addText: ObjectMemory viReleaseSeries printString , '.' ,
                ObjectMemory viReleaseMajor printString.
e newRow.
```

Note this code is also missing the minor field.

2. Creating version objects from the field values

The old code for this would be:

```
| versionID major minor |
versionID := ObjectMemory versionId.
major := versionID at: 5.
minor := versionID at: 6.
^self major: major minor: minor
```

Methods such as these should use the introduced API instead of decoding the VI signature bytes directly. At the very least, the code should interpret all the existing fields properly (e.g. the 2nd VI signature byte, accessed at index 6 above, also has the 64-bit flag bit).

3. Decoding the first byte as a version check

The old code for this would be:

```
(ObjectMemory versionId at: 5) >= 76
   ifTrue: [self newFunctionality]
   ifFalse: [self oldFunctionality]
```

Use the following code instead:

```
(ObjectMemory viReleaseSeries >= 7
   and: [ObjectMemory viReleaseMajor >= 6])
      ifTrue: [self newFunctionality]
      ifFalse: [self oldFunctionality]
```

4. Decoding the first byte as a series check

Here is the old code:

```
(ObjectMemory versionId at: 5) >= 70
    ifTrue: [^self newFunctionality].
(ObjectMemory versionId at: 5) >= 50
    ifTrue: [^self oldFunctionality].
(ObjectMemory versionId at: 5) >= 30
    ifTrue: [^self olderFunctionality].
^self evenOlderFunctionality
```

Use the following code instead:

```
ObjectMemory viReleaseSeries = 7
    ifTrue: [^self newFunctionality].
ObjectMemory viReleaseSeries = 5
    ifTrue: [^self oldFunctionality].
ObjectMemory viReleaseSeries = 3
    ifTrue: [^self olderFunctionality].
^self evenOlderFunctionality
```

Base Image

Enhancements to Time, Timestamp, and Duration classes

VisualWorks 7.10 includes changes to classes Time, Timestamp and Duration. Precision in Time and Timestamp has been increased to the nanosecond level, and Duration now answers #milliseconds, #microseconds, and #nanoseconds in the style of Time and Timestamp (the previous behavior has been preserved as well).

Instance variable names in Timestamp now mirror those in Time and Duration for clock-related variables, and mirror those in Date for calendar-related variables.

In practice, this means that the accessors for clock-related instance variables in Timestamp have changed from singular form (#hour, #minute, #second, #millisecond) to plural form (#hours, #minutes, #seconds, #milliseconds). In addition, #microseconds and #nanoseconds are now available.

The previous singular form of accessors in class Timestamp are maintained for backward compatibility with existing software.

Because of the way time beyond milliseconds is stored, both microseconds and nanoseconds are derived values that are assembled as requested. Accessing the underlying instance variable partialNanosecond requires caution. Examine the instance methods #milliseconds and #nanoseconds for more information.

Because Time and Timestamp objects may be created and answered in several places in the VisualWorks class hierarchy, not all instantiation situations have code to provide significant data for sub-second instance variables. In these cases, the value for those sub-second components will be zero.

A change has been made to the manner in which class Duration answers sub-second time values. This affects the accessors #milliseconds, #microseconds, and #nanoseconds. Answers are now in the form used by Time and Timestamp.

For example, let's consider a number of nanoseconds, say 3004005. This would be expressed as a number of seconds in decimal notation as:

```
0.003004005
```

In this example, 0.003004005, the 3 is in the units position for milliseconds, the 4 is in the units position for microseconds, and the 5 is in the units position for nanoseconds. In previous versions this would yield:

```
(Duration fromNanoseconds: 3004005) milliseconds  => 3
(Duration fromNanoseconds: 3004005) microseconds  => 4
(Duration fromNanoseconds: 3004005) nanoseconds  => 5
```

In VisualWorks 7.10, these answers are:

```
(Duration fromNanoseconds: 3004005) milliseconds  => 3
(Duration fromNanoseconds: 3004005) microseconds  => 3004
(Duration fromNanoseconds: 3004005) nanoseconds  => 3004005
```

The pre-7.10 answers are available from a Duration using the compatibility methods #millisecondCount, #microsecondCount and #nanosecondCount.

Enhanced ClassBuilder schema migration

Release 7.10 includes new a mechanism to assist when simultaneously adding and renaming instance variables in a class. This is an infrequent-use and special-purpose tool, a little manual, so it needs to be used with care.

The semantics of the mapping between old instance variable names and new ones has been reversed. Instead of BehaviorBuilderRecord>>mapFrom:to: creating a mapping from old instance variable indices to new indices, it now creates a map from new indices to accessor blocks, which may do an instVarAt: on the old object, or may compute the value some other way.

Both details — the fact that the indices of the map are now relative to the new object, not the old one, and the fact that the map now contains blocks and not integers — may trip up customers who might have delved deeply enough into ClassBuilder to make their own use of those maps or who might have overridden some of the methods modified here.

A class variable on BehaviorBuilderRecord allows mapping of new instance variable names — if the instance variable name is already found in the old object, this class variable will not be consulted, so it can't be depended on if an instance variable has the same name but needs a new type signature. The class variable is intended to support migration of all subclasses of the affected class, so you don't need to specify migration schemas for all subclasses of an affected class.

The class variable supports renaming of an instance variable (just use the new instance variable name as the key and the old one as the value) or arbitrary other transformations (use the new instance variable as the key, and a one-arg block as the value — the argument of the block will be the old instance from which we need to compute the new inst var's value).

As an example, here is the migration schema for the recent changes to class Timestamp:

```
BehaviorBuilderRecord.ClassRemap :=
   (Dictionary new
      at: Timestamp fullName
      put: (Dictionary new
               at: 'hours' put: 'hour';
               at: 'minutes' put: 'minute';
               at: 'seconds' put: 'second';
               at: 'milliseconds' put: 'millisecond';
               yourself);
      at: Time fullName
      put: (Dictionary new
               at: 'milliseconds' put: [:inst | 0];
               at: 'partialNanosecond' put: [:inst | 0];
               yourself);
      yourself).
```

Using class Timestamp as an example, the general steps to use this tool are:

1 Assign the value of BehaviorBuilderRecord.ClassRemap as shown above. This provides the schema for the transformation.

2 Load the new version of code that includes the classes with new definitions, whose existing instances in the running image are to be migrated.

3 After successful completion of step 2 above, set the value of BehaviorBuilderRecord.ClassRemap to nil (as it was before you started).

4 Then save the image (or do other build activities as appropriate to your situation).

GUI

Screen>>nativeGUIPolicy

To support the new UISkinning system (currently in Preview), VisualWorks needs to know the standard fonts for the runtime platform, as well as various colours. These are provided by an object returned from the Screen>>nativeGUIPolicy method. The returned object is a subclass of NativeGUIPolicy, which is a protocol specification class. We intend to enhance this mechanism in the future to specify the font and color preferences for e.g. code editing, along with a preferences editor for this information. The key detail being that this information is orthogonal to the skin selection.

MacOSXLookPolicy has been modified to use this mechanism for fonts, which makes that look and feel more usable on X11.

Commands

The commands system is the first release of a mechanism to move shortcut keys out of menus and widgets and into a centralised repository. It allows for platform-specific specification of shortcut key bindings to abstract commands, represented as Symbols. An editor for key bindings is provided in the look and feel section of the Settings Tool. The key class is CommandRegistry — all commands are defined in class-side methods of this class, grouped by protocol, which is reflected in the editor. For details, look at the class comment to CommandRegistry.

The menu system has been enhanced to allow you to specify a command wherever you can specific a shortcut key. We encourage you to adopt this mechanism.

As part of the command system various improvements have been made to menu and keyboard handling. The most noticable is that we now support the Mac OS X command key, and we display Mac OS X shortcuts in menus using the appropriate characters (when on OS X).

Note that there is a known issue in menu handling, particularly related to the edit menu items in several tools. In a number of cases, the cut/copy etc menu items aren't redirected to the currently focused widget, but instead are handled directly by the model. Thus we have not bound commands to those items because the shortcut key behaviour is then very annoying — your shortcuts don't operate on the current widget. This will be corrected in an upcoming release using a general command-specific solution for this problem.

Tools

One-Step Merge

The Merge Tool now has two operational modes. The new behavior is a one-step merge, where the Merge Tool automatically performs your requested merge in a single operation, without opening the Merge Tool UI unless it determines that there are conflicts which you will need to resolve manually. This is now the default. The original behavior is still available. In this mode, the Merge Tool UI opens and you can inspect all the version selections before you perform the merge and then the publish operations. To choose this behavior, open the Settings Tool and select the **Merge Policy** setting **Always open MergeTool before performing Merge** in the Store Settings.

It's important to understand that this new one-step operation will *never* publish new versions to Store. You must still explicitly publish any and all changes following the completion of the merge operation. If the Merge Tool detects conflicts it cannot resolve by tracing the parentage of the participating versions, it will open the Merge Tool UI and you will need to satisfy all the identified conflicts before performing the merge there.

As always, you may use the Comparison Tool to review the changes in packages or bundles before you do the merge, by selecting **Compare with image** from the Store Published Items list, or by simply comparing specific package versions against each other. You may also view the

packages touched by a one-step merge operation in your image, prior to publishing any packages, by selecting **Spawn > Unpublished Packages** from the menu in the browser's packages list. You can then use the **Store > Compare with Parent** menu item to open the Comparison Tool on the currently unpublished changes in your image.

Under most conditions, the Merge Tool can easily determine the correct versions to merge into your image, and the one-step operation can reduce the amount of complexity that must be handled, especially by new users, using the more intuitive Comparison Tool to review the effects of change. However, we are aware that the Comparison Tool does not easily handle comparison of very large packages or bundles at present. If you regularly merge large components or if you require the explicit detail presented in the Merge Tool UI, please select the original Merge Tool behavior in the VisualWorks settings.

Framework-oriented Refactorings

A new refactoring, RenameObjectAndMethodsRefactoring, will offer to rename a method whose selector fits certain patterns and contains the name of the class that is being renamed. (This occurs only when renaming a class; name spaces and shared variables are not checked.) The pragma #classNameInSelectorPatterns lets frameworks register patterns they wish renamed in this way. The Glorp framework's extension method, RenameObjectAndMethodsRefactoring >> classNameInSelectorPatternsGlorp, shows an example of using this pattern (see the Glorp release notes).

Database

Restore the ability to set bindInput before prepare

(AR 66878) In VisualWorks 7.10, developers using the EXDI can evaluate #bindInput: either before or after the #prepare:, even when the binding list contains active sessions. On Oracle, binding variables for running procedures or functions must happen after the prepare, otherwise an error will be triggered.

Support for Unicode on DB2

(AR 67563) In VisualWorks 7.10, we have added full Unicode Support to the DB2 connect. What that means is that, developers can use Unicode table and column names, include Unicode strings in their SQL statements, bind Unicode strings to host variables, and even input and output Unicode strings to and from stored procedures.

Here is how the new implementation works for you:

1 If you don't set the main encoding to Unicode (either UTF16 or UCS_2), it will work as before, all existing applications will continue to work.

2 If you don't set the main encoding to Unicode but want to use Unicode columns, you can do so as well, but when binding String values, you have to set binding templates for the binding values to indicate which Strings should be encoded as Unicode. This will provide the flexibility for users to develop "mixed" applications.

3 If you set the main encoding to Unicode (either UTF16 or UCS_2), no binding template is necessary, all connect strings, SQL statements, error messages, insert and retrieved Strings will be treated as Unicode.

Binding of Float values in Oracle

(AR 65723) With VisualWorks 7.10, the binding of Float values has reverted to its pre-7.7.1 state. In VisualWorks, Float objects have a precision of 6 digits. Binding this to a NUMBER(p,s) column of greater width made Oracle generate extra trailing digits, thus:

```
SELECT ... WHERE 1.23456 = ?
```

could fail when ? was bound to 1.23456, because these added digits did not match. To avoid this, following an Oracle recommendation, release 7.7.1 of VisualWorks converted Floats to Strings as they were bound. However, this conversion did not handle locale, a serious problem for some customers, and had a performance cost. It has therefore been backed out. Instead, customers can choose to pass FixedPoint (or String) values when binding to specific numeric columns. The following paragraphs provide additional infomation and examples.

When Oracle converts a 32-bit Float to a targetted NUMBER(p,s) column format, extra digits are generated in the conversion process if p exceeded the 6 digits of precision that a Float offers. This behavior is expected, but can have annoying and misleading effects. For example, in VisualWorks:

```
Float pi asFixedPoint: 5
```

returns 3.14159s, which contains all 6 accurate digits from the Float.

```
Float pi asFixedPoint: 10
```

returns 3.1415927387s, but the trailing digits are misleading in their apparent precision.

```
Double pi asFixedPoint: 10
```

returns 3.1415926536s, whose trailing digits are more reflective of the value at that precision.

Exactly the same thing happens in the Oracle database. When a NUMBER(10,8) column stores a Float, the value will likewise acquire these trailing digits.

Oracle's suggested solution — that Float values be transmitted in string form so the digits can be placed into the NUMBER() column directly, without undergoing conversion — causes the following problems:

- With internationalization, the printed number needs to account for the Oracle client locale which may differ from the image locale. This has impacted some customers.
- As of Oracle 10g, there are new column types, BINARY_FLOAT and BINARY_DOUBLE which are naturally suited to receive native Float data. For these target columns, it is better to transmit Float values directly.

The new method Float>>asFixedPoint returns a FixedPoint value that contains only 6 digits. This avoids locale issues and can be sent by customers to values being bound. For example, inserting to a table defines as:

```
CREATE TABLE video_purchase(
    id integer NOT NULL,
   customername VARCHAR(50),
   title VARCHAR(50),
   price NUMBER(10,5),
   CONSTRAINT video_purchase_pk PRIMARY KEY (id))
```

the code could be:

```
INSERT INTO video_purchase VALUES(?,?,?,?) (10, 'Joe Bloggs', 'Its a Wonderful Life', 200.6s)
```

or, for non-literals:

```
200.6 asFixedPoint: 5
```

which evaluates to 200.60001s, or:

```
200.6 asFixedPoint
```

which evaluates to 200.600s.

```
discountedPrice := 200.6.

INSERT INTO video_purchase VALUES(?,?,?,?) (10, 'Joe Bloggs', 'It's a Wonderful Life', discountedPrice asFixedPoint)
```

One may of course, use:

```
 asFixedPoint: n
```

where n is <= 6, the number of digits to the right of the decimal point. Float>>asFixedPoint returns a FixedPoint number with up to 6 non-zero digits, equivalent to using Float>>printString.

By sending #asFixedPoint before entering Float data into a database column such as:

```
cnum NUMBER(20,8)
```

you can prevent arbitrary trailing digits from being written to the database, and so allow an equality comparison expression such as:

```
match := 1.2345s.       "or match := 1.2345 asFixedPoint"
session read: MyClass where: [:each | each cnum = match].
```

where the resulting SQL:

```
SELECT ... FROM ...
WHERE t.cnum = match
```

would otherwise fail because a Float 1.2345, written to cnum, would have unguessable trailing digits.

Support for BINARY_FLOAT and BINARY_DOUBLE data types on Oracle

(AR 66008) According to Oracle, "The BINARY_FLOAT and BINARY_DOUBLE data types represent single-precision and double-precision floating point values that mostly conform to the IEEE754 Standard for Floating-Point Arithmetic.

Prior to the addition of these data types with release 10.1, all numeric values in an Oracle database were stored in the Oracle NUMBER format. The new binary floating point types do not replace Oracle NUMBER. Rather, they are alternatives to NUMBER that provide the advantage of using less disk storage".

In VisualWorks 7.10, we've added support for these two data types. The following example code illustrates their use:

```
"Connect to an Oracle database"
conn := OracleConnection new.
conn username: 'username';
password: 'password';
environment: 'ORACLEDBLaterThan10'.
conn connect.

"Drop the test table if it existed."
sess := conn getSession.
sess prepare: 'DROP TABLE TestFloatDouble'.
sess execute.
ans := sess answer.
ans := sess answer.

"Create the test table."
sess := conn getSession.
sess prepare: 'CREATE TABLE TestFloatDouble (cnum number(20, 6),
cnum1 number(20, 8), cfloat binary_float, cdouble binary_double)'.
sess execute.
ans := sess answer.
ans := sess answer.

"Insert some test data."
sess := conn getSession.
sess prepare: 'INSERT INTO TestFloatDouble VALUES (3.1415926,
3.1415926, 3.1415926, 3.1415926)'.
sess execute.
ansStrm := sess answer.
ansStrm := sess answer.

sess prepare: 'INSERT INTO TestFloatDouble VALUES (?, ?, ?, ?)'.
sess bindInput: #(3.1415926 3.1415926d 3.1415926 3.1415926d).
sess execute.
ansStrm := sess answer.
ansStrm := sess answer.

"Retrieve and verify the test data."
sess := conn getSession.
```

```
sess prepare: 'select * from TestFloatDouble'.
sess execute.
ans := sess answer.
ans upToEnd.
```

Support for nanosecond-resolution Timestamp on Oracle

(AR 66830) While Oracle's Timestamp supports nanosecond resolution, in previous releases of VisualWorks, we only supported down to milliseconds. In VisualWorks 7.10, we've added support for nanosecond-resolution Timestamp objects in OracleEXDI.

The following example illustrates how to use this feature:

```
"Connect to an Oracle database"
conn := OracleConnection new.
conn username: 'username';
password: 'password';
environment: 'oracleDB'.
conn connect.

"Drop the test table if it existed."
sess := conn getSession.
sess prepare: 'DROP TABLE test_timestamp'.
sess execute.
ansStrm := sess answer.

"Create the test table."
sess := conn getSession.
sess prepare: 'CREATE TABLE test_timestamp (
                    ts TIMESTAMP(9),
                    cid INTEGER)'.
sess execute.
ansStrm := sess answer.

"Insert some test data."
ts := (Timestamp new)
      year: 2005;
      month: 11;
      day: 30;
      hour: 5;
      minute: 25;
      second: 30;
      millisecond: 998;
      partialNanosecond: 123456.

sess := conn getSession.
sess prepare: 'INSERT INTO test_timestamp values (?, ?)'.
```

```
sess bindInput: (Array with: ts with: 1).
sess execute.
ans := sess answer.

"Retrieve and verify the test data."
sess := conn getSession.
sess prepare: 'select * from test_timestamp'.
sess execute.
ansStrm := sess answer.
ansStrm upToEnd.
```

Support for scrollable cursors on ODBC

(AR 65538) In VisualWorks 7.10, we've added support for scrollable cursors. The following example demonstrates usage:

```
"Connect to the server."
conn := ODBCConnection new.
conn username: 'username';
    password: 'password';
    environment: 'DSN'.
conn connect.

sess := conn getSession.

"Drop the test table if existed."
sess prepare: 'DROP TABLE TESTTABLE';
    execute;
    answer;
    answer.

"Create a test table."
sess prepare: 'CREATE TABLE TESTTABLE(
    cid int ,
    name varchar(30)
)';
    execute;
    answer;
    answer.

"Set the number of recrods being inserted."
loopCount := 10.

"The SQL used to do inerst."
sql := 'INSERT INTO TESTTABLE VALUES (?, ?)'.

"Insert some test data."
sess prepare: sql.
```

```
1 to: loopCount
    do: [:i | sess bindInput: (Array with: i with: ('test', i printString));
        execute;
        answer;
        answer.
].

"Select the test data to verify."
sess := conn getSession.
sess scrollable: true.
sess blockFactor: 3.
sess prepare: 'Select * from TESTTABLE'.
sess execute.
ans := sess answer.
[ (val := ans next) isNil ] whileFalse: [
    Transcript cr; show: (val at: 1) printString.
].

sess := conn getSession.
sess scrollable: true.
sess blockFactor: 3.
sess prepare: 'Select * from TESTTABLE'.
sess execute.
ans := sess answer.
ans moveTo: 10.
[ (val := ans previous) isNil ] whileFalse: [
    Transcript cr; show: (val at: 1) printString.
].
```

Support for DB2's DBCLOB data type on ODBC connect

(AR 66261) In VisualWorks 7.10, we've added the support for DB2's DBCLOB data type to ODBCEXDI.

The following example demonstrates its usage:

```
"Connect to DB2 with initial Unicode encoding."
conn := ODBCConnection new.
conn encoding: #UCS_2.
conn
        username: 'username';
        password: 'password';
        environment: 'DB2DSN'.
conn connect.

"Drop the test table if existed."
sess := conn getSession.
sess prepare: 'drop table test_dbclob'.
```

```
sess execute.
sess answer.

"Create the test table."
sess := conn getSession.
sess prepare: 'create table test_dbclob (ncl DBCLOB(4m))';
   execute;
   answer.

"Insert test data."
sess := conn getSession.
sess prepare: 'insert into test_dbclob values(?)';
bindInput: (Array with: (String new: 300000 withAll: $A));
   execute;
   answer.

"Retrieve the test data."
sess := conn getSession.
sess prepare: 'select * from test_dbclob' ;
   execute.
ans := sess answer.
ans upToEnd.
```

Support for setting connection attributes by value on ODBC

(AR 66530) For historical reasons, when setting and getting connection attributes, ODBCEXDI only accepted attribute symbols as arguments. With VisualWorks 7.10, the EXDI will henceforth accept both attribute symbols and attribute values as arguments.

The following code examples illustrate how to set and get connection attributes:

```
"Establish the connection."
conn := ODBCConnection new.
conn
   username: 'username';
   password: 'password';
   environment: 'DSN'.
conn connect.

"Set and get by symbols."
conn setConnectionOptionExternal: #SQL_TXN_ISOLATION value: 2.
conn getConnectionOptionExternal: #SQL_TXN_ISOLATION.

"Set and get by numbers."
conn setConnectionOptionNumber: 108 value: 2.
conn getConnectionOptionNumber: 108.
```

When processing SQL on ODBCEXDI, only report errors, not warnings

(AR 66894) When processing single or multiple SQL commands at the session level, ODBCEXDI will report errors, but not warnings.

Support for password expiration warning with grace period on ODBC

(AR 67020) In the VisualWorks 7.10 ODBCEXDI, both errors and warnings will be caught at the connection level; warrnings will be reported as Notifications.The following code illustrates how to catch and handle warnings:

```
[conn := ODBCConnection new.
   conn environment: 'ODBCDSN';
      username: 'username';
      password: 'password';
      connect]
   on: Exception
   do:   [:exception |
         Dialog warn: exception errorString.
         (exception isKindOf: Notification)
            ifTrue: [exception pass]
            ifFalse: [exception return: nil]].
```

Of course, users have to modify the sample code to meet their own needs.

Support nanosecond-resolution Timestamps on ODBC

(AR 67157) In VisualWorks 7.10, the ODBCEXDI is able to handle Timestamp data types with nanosecond resolution. Different databases may support different degrees of accuracy for their Timestamp data types, but VisualWorks will support accuracy to a nanosecond.

The following examples for Oracle and SQL Server illustrate nanosecond-resolution support. Note that while Oracle's Timestamp can have nanosecond accurary, SQL Server's datetime2 only supports accuracy to a resolution of 100 nanoseconds:

```
"Connect to Oracle."
conn := ODBCConnection new.
conn username: 'username';
   password: 'password';
   environment: 'oracleDSN'.
conn connect.

"Create a test table."
sess prepare: 'create table test_timestamp (cid integer, ts
```

```
timestamp(9))'.
sess execute.
ans := sess answer.
ans := sess answer.

"Insert some test data."
ts := Timestamp now.
ts partialNanosecond: 123456.
sess := conn getSession.
sess prepare: 'insert into test_timestamp values(1, ?)'.
sess bindInput: (Array with: ts).
sess execute.
ans := sess answer.
ans := sess answer.
"Select the test data to verify."
sess := conn getSession.
sess prepare: 'select * from test_timestamp'.
sess execute.
ans := sess answer.
res := ans upToEnd.

"Connect to SQL Server."
conn := ODBCConnection new.
conn username: 'username';
    password: 'password';
    environment: 'SQLServerDSN'.
conn connect.
"Create a test table."
sess prepare: 'create table test_datetime2 (cid int, cdatetime datetime,
cdatetime2 datetime2(7))'.
sess execute.
ans := sess answer.
ans := sess answer.

"Insert some test data."
ts := Timestamp now.
ts1 := ts copy partialNanosecond: 123400.
sess := conn getSession.
sess prepare: 'insert into test_datetime2 values(1, ?, ?)'.
sess bindInput: (Array with: ts with: ts1).
sess execute.
ans := sess answer.
ans := sess answer.

"Insert some test data."
ts := Timestamp now.
ts1 := ts copy partialNanosecond: 123456.
```

```
sess := conn getSession.
sess prepare: 'insert into test_datetime2 values(2, ?, ?)'.
sess bindInput: (Array with: ts with: ts1).
sess execute.
ans := sess answer.
ans := sess answer.

"Select the test data to verify."
sess := conn getSession.
sess prepare: 'select * from test_datetime2'.
sess execute.
ans := sess answer.
res := ans upToEnd.
```

Support nanosecond-resolution Timestamps on DB2

(AR 67285) In VisualWorks 7.10, DB2EXDI can handle Timestamps with nanosecond precision. Here is an example:

```
" Connect to DB2."
conn := DB2Connection new.
conn
        username: 'username';
        password: 'password';
        environment: 'DB2'.
conn connect.

"Drop the test table if is exists."
sess := conn getSession.
sess prepare: 'drop table test_timestamp'.
sess execute.
sess answer.
sess answer.

"Create a test table."
sess := conn getSession.
sess prepare: 'create table test_timestamp (cid integer, ts timestamp)'.
sess execute.
ans := sess answer.
ans := sess answer.

"Insert some test data."
ts := Timestamp now.
ts partialNanosecond: 123456.
sess := conn getSession.
sess prepare: 'insert into test_timestamp values(1, ?)'.
sess bindInput: (Array with: ts).
sess execute.
```

```
ans := sess answer.
ans := sess answer.

"Select the test data to verify."
sess := conn getSession.
sess prepare: 'select * from test_timestamp'.
sess execute.
ans := sess answer.
res := ans upToEnd.
```

Limitation with DB2EXDI: file names can only be English when using LOB file references

(AR 67990) In VisualWorks 7.10, we've added Unicode support to the DB2EXDI, but there is a limitation such that, when using LOB file references, only English file names will work. This, even when the main encoding of the connection is set to Unicode. We are working to remove this limitation in a subsequent release.

PostgreSQL Configuration Settings

The current latest version of PostgreSQL is 9.2.4. During the 9 series, PostgreSQL changed the default settings of some settings, two of which required Glorp changes.

bytea_output

The default value of **`bytea_output`** is now **`hex`**, not **`escape`**. In VisualWorks 7.9.1 and earlier, users of Store (or of blobs in their postgres applications) had to reset this to **`escape`**.

standard_conforming_string

The default value of **`standard_conforming_string`** is now **`on`**, not **`off`**. In VisualWorks 7.9.1 and earlier, users of Store (or of strings containing the \ character in their postgres applications) had to reset this to **`off`**.

In VisualWorks 7.10, these resets are no longer necessary; Glorp handles either value of either of these settings.

PostgreSQL Type Mappings

The PostgreSQLEXDISession class >> from*Block methods provide translations for reading column values of various types. These are coded to defaults chosen to be suitable for the vast majority of applications. Users of particular applications may wish to check them, and occasionally change them to suit specific requirements. The

default fromTimestampBlock method has been updated to handle microseconds, a level of granularity that both Postgres and VisualWorks 7.10 Timestamps can understand. The fromTimestampBlock does nothing with timezones by default, but it now contains commented-out code showing one of various ways in which timezones could be mapped.

The EXDI layer supports unbound and bound mappings for all databases. In the special case of PostgreSQL, this makes virtually no difference down at the driver layer, so in the Glorp application's use of it, PostgreSQLPlatform >> supportsBinding returns true but PostgreSQLPlatform class >> useBindingIfSupported returns false. (The latter value can be overridden — globally, for all databases accessed by Glorp in that image — by setting Glorp.DatabasePlatform.UseBindingIfSupported := true.) Some coding errors that prevented Timestamps, FixedPoints and Doubles from being bound are now corrected, and the code that diverted such types down the unbound path even if binding was otherwise being used has been removed. (Dates and Times are still diverted to unbound handling; this will cease in 7.11.)

Automatic Conversion of Double and Float values to FixedPoint

(AR 68172) Occasionally there is a need to convert Float or Double values to FixedPoint, as described in the section "Binding of Float values in Oracle". The existing method, Float (or Double) >> asFixedPoint: (scale) has been complemented by two new base image methods, #asFixedPointOfPrecision: and #asFixedPoint. The latter simply calls the former using the default precision parameter, 6 digits for Floats and 14 digits for Doubles.

These methods automatically figure out what scale (digits after the decimal point) is necessary to achieve the desired precision (the number of significant digits). For example, in the first line below it is evident that asking for a particular scale isn't very helpful for the large value, 1.3e14. In the second line, requesting 10 digits of precision, also looks curious.

The trailing digits are artifacts from converting a binary value into a decimal number, similar to repeating decimal remainders after division. Although the extra digits aren't harmful, and they help to preserve the original value, a simpler output can often serve just as well, as shown in the last example (which is really just a send of

#asFixedPointOfPrecision: 6). In that example, the FixedPoint result has the 6 leading digits rounded, so the trailing nines get rounded up to zeros.

```
1.3e14 asFixedPoint: 5
    returns    129999994617856.00000s

1.3e14 asFixedPointOfPrecision: 10
    returns    129999994600000s

1.3e14 asFixedPoint
    returns    130000000000000s
```

The following three lines show how each of these conversions, in this case, appears to restore the original value, and does so within the accuracy of the printString method.

```
129999994617856.00000s asFloat
    returns    1.3e14

129999994600000s asFloat
    returns    1.3e14

130000000000000s asFloat
    returns    1.3e14
```

Actually, the default precision of 6 for Float, and 14 for Double rounds the last digit, so the original value often cannot be restored exactly, although it will always be very close. This effect can be seen with the method Float>> ulpsAwayFrom:, which indicates how close the mantissa bits are.

The results below indicate perfect restoration at precision 9 and 10, for the value 1.3e14 and pi, but 1.7e14 restores even from precision 5.

```
(5 to: 10) collect:
    [:i |   1.7e14 ulpsAwayFrom:
            ((1.7e14 asFixedPointOfPrecision: i) asFloat)]
    #(0 0 0 0 0 0)

(5 to: 10) collect:
    [:i |   1.3e14 ulpsAwayFrom:
            (1.3e14 asFixedPointOfPrecision: i) asFloat]
    #(-1 -1 -1 1 0 0)

(5 to: 10) collect:
    [:i |   Float pi ulpsAwayFrom:
            ((Float pi asFixedPointOfPrecision: i) asFloat)]
    #(-30 11 -1 0 0 0)
```

In summary, the default precision should be adequate, but often a higher precision provides greater fidelity.

Glorp

Simplified APIs

The duplicate Glorp APIs of read:* readManyOf:* and returningManyOf:* have been reduced to read:* (the other methods remain, deprecated). Similarly readOneOf:* is now the method to use, returningOneOf:* being deprecated.

Enhanced Support for #groupBy: and #retrieve:

Glorp now supports multiple groupBy: fields and computed retrieve: expressions (e.g. retrieve: [:each | each stockQuantity * 3]).

Improved Code Comments

Code comments for bundles, packages and API methods have been improved.

Additional Code Examples

The examples in doc/GlorpGuide.pdf are now in the parcel GlorpGuideExamples.

Refactoring of Glorp Models

Renaming a class will offer to rename also any methods whose selectors match descriptorFor<ClassName>: or classModelFor<ClassName>: (see the release note “Framework-oriented Refactorings”).

Date arithmetic allowed in WHERE clause

In Smalltalk, date and time arithmetic can be performed between dates and durations, e.g.:

```
employee startDate + 5 days >= (3 apr: 2012)
```

In SQL, simple integer expressions will parse, e.g.:

```
SELECT * FROM EMP t1 WHERE (t1.STARTDATE + 25) >= TO_DATE
('2012-4-3', 'YYYY-MM-DD')
```

SQL supports adding an integer to a date, meaning incrementing the date by that number of days.

In VisualWorks 7.10, a Glorp WHERE clause can now include arithmetic of Date objects. You can use both constant and field expressions, or dates added to or subtracted from integers, where the integer expression represents a number of days.

For example:

```
session read: Employee where:
    [:each | each startDate + 1 > Date today].
```

or you can use field expressions, like:

```
session read: Employee where:
    [:each | each startDate + each orientationDays < Date today].
```

Here, orientationDays is integer-valued.

You can also use duration values, e.g.:

```
session read: Employee where:
    [:each | each startDate + 5 days >= Date today].
```

DB2 does not allow integers, so you should use duration type expressions. We have therefore added Duration arithmetic functions to DB2Platform >> initializeFunctions.

Security

Smalltalk-based fall-back for Xtreams-Crypto

The Security-Xtreams package now provides complete Smalltalk-based fall-back for Xtreams-Crypto.

We've completed the integration of our pre-existing Smalltalk-based cryptographic library into the new Xtreams-Crypto APIs. The primary benefit is that if the implementations based on external libraries causes difficulty, the Security-Xtreams package provides a full fallback solution that doesn't require any external resources. This also fully restores the capabilities provided by the Smalltalk library in the context of the new APIs.

The default implementation selection mechanisms were further enhanced to allow discriminating on a per-algorithm basis. For example, it can be tuned to pick an external library for AES encryption, but use the Smalltalk implementation for RC4.

Changes to TLSCipherSuite API

The API to access predefined TLSCipherSuite constants has changed.

Cipher suites are generally constant and new instances are never created in the normal course of use. Instead, predefined instances are kept in global variables under their standard name, starting with the TLS_ prefix (e.g. TLS_RSA_WITH_RC4_128_SHA). In previous releases, the constants would be accessed by sending the suite name to the TLSCipherSuite class as a message. In this release, this approach has been replaced with a more straightforward solution in which each cipher suite name is also declared as a shared variable in the Xtreams.* namespace. Thus, the constants can be referenced directly as any other global.

Improved TLSSessionCache implementation

The main goal of this change was to allow code such as the Net Clients package to take advantage of TLS session resumption. For example, when executing this code:

```
'https://www.google.com' asURI get
```

The first time, there's a full TLS handshake. The second time, the previously-negotiated session is resumed transparently (assuming the server agrees to resume it). To achieve this, we had to improve the default TLSSessionCache implementation. (The details may be found in the class comments for TLSSession and TLSSessionCache, and in the expanded section on 'Session Resumption' of the package comment. The outcome is that now there are two global default caches. One is a client cache maintained by Net.HttpsConnection (enabling the capability mentioned above). The secure mail protocols do not cache sessions by default, but the same capability is available there. The second cache is the default server side cache on TLSContext, so that it can be reused for multiple server contexts.

The caches aren't light (barely useful) things anymore. They cannot be created and thrown away carelessly. Consequently the easy instance creation methods on TLSContext cannot just create them as before. With these changes, global caches make a more sensible default.

The computational savings from session resumption aren't that impressive on the client side. The vast majority of TLS interactions on the net use RSA handshaking with small public exponents, which really doesn't strain the clients as much. We've seen maybe a 20%

saving from resumption for the client. However it makes significant difference on the server side. The 2048 bit keys that are the norm today can tax a server significantly (multiplied by the number of connections handled). That said, session caching isn't free either, and if clients aren't taking advantage of it when it's available, it can be a wasted cost to the server too.

Code Management and Store

Modification required for Version 7.9 Store Repositories

In VisualWorks 7.9, Store Package and Bundle comment and property columns were incorrectly set to have a default value of 0. This could cause problems with the replicator.

To change this with Oracle, the following code should be run:

```
ALTER TABLE TW_BUNDLE MODIFY commentID DEFAULT NULL;ALTER TABLE TW_BUNDLE MODIFY propertiesID DEFAULT NULL;ALTER TABLE TW_PACKAGE MODIFY commentID DEFAULT NULL;ALTER TABLE TW_PACKAGE MODIFY propertiesID DEFAULT NULL;
```

For PostGreSQL, the code to change these looks like this:

```
ALTER TABLE TW_BUNDLE ALTER COLUMN commentID DEFAULT NULL;ALTER TABLE TW_BUNDLE ALTER COLUMN propertiesID DEFAULT NULL;ALTER TABLE TW_PACKAGE ALTER COLUMN commentID DEFAULT NULL;ALTER TABLE TW_PACKAGE ALTER COLUMN propertiesID DEFAULT NULL;
```

For DB2, the code to change these looks like this:

```
ALTER TABLE TW_Bundle ALTER COLUMN commentID SET DEFAULT NULL;ALTER TABLE TW_Bundle ALTER COLUMN propertiesID SET DEFAULT NULL;ALTER TABLE TW_Package ALTER COLUMN commentID SET DEFAULT NULL;ALTER TABLE TW_Package ALTER COLUMN propertiesID SET DEFAULT NULL;
```

For SQLServer you must use a database administration tool and drop the DEFAULT constraint(s).

Note: SQLite does not support ALTER syntax. Once created, tables are modifiable.

TRACE field mistakenly created with not NULL constraint

In VisualWorks 7.9, Store tables are incorrectly created with the trace field not set to null. This can cause the Store garbage collector to fail for any database created with release 7.9.

Whereas other tables in the Store schma have a trace field, only those for the tables: TW_BUNDLE, TW_PACKAGE, TW_CLASSRECORD, TW_METHOD, TW_NAMESPACERECORD and TW_DATAELEMENT are affected by the Store garbage collector.

To change this in Oracle the following code should be run:

```
ALTER TABLE TW_BUNDLE MODIFY (TRACE NULL);ALTER TABLE
TW_PACKAGE MODIFY (TRACE NULL);ALTER TABLE
TW_CLASSRECORD MODIFY (TRACE NULL);ALTER TABLE
TW_METHOD MODIFY (TRACE NULL);ALTER TABLE
TW_NAMESPACERECORD MODIFY (TRACE NULL);ALTER TABLE
TW_DATAELEMENT MODIFY (TRACE NULL);
```

In PostgreSQL the code to change these, looks like this:

```
ALTER TABLE tw_bundle ALTER COLUMN trace DROP NOT NULL;ALTER
TABLE tw_package ALTER COLUMN trace DROP NOT NULL;ALTER
TABLE tw_classrecord ALTER COLUMN trace DROP NOT NULL;ALTER
TABLE tw_method ALTER COLUMN trace DROP NOT NULL;ALTER
TABLE tw_namespacerecord ALTER COLUMN trace DROP NOT
NULL;ALTER TABLE tw_dataelement ALTER COLUMN trace DROP NOT
NULL;
```

In DB2 the code to change these, looks like this:

```
ALTER TABLE TW_BUNDLE ALTER COLUMN TRACE DROP NOT
NULL;ALTER TABLE TW_PACKAGE ALTER COLUMN TRACE DROP NOT
NULL;ALTER TABLE TW_CLASSRECORD ALTER COLUMN TRACE DROP
NOT NULL;ALTER TABLE TW_METHOD ALTER COLUMN TRACE DROP
NOT NULL;ALTER TABLE TW_NAMESPACERECORD ALTER COLUMN
TRACE DROP NOT NULL;ALTER TABLE TW_DATAELEMENT ALTER
COLUMN TRACE DROP NOT NULL;
```

In SQLServer the code to change these, looks like this:

```
ALTER TABLE TW_Bundle ALTER COLUMN trace int NULL;ALTER TABLE
TW_Package ALTER COLUMN trace int NULL;ALTER TABLE
TW_ClassRecord ALTER COLUMN trace int NULL;ALTER TABLE
TW_Method ALTER COLUMN trace int NULL;ALTER TABLE
TW_NameSpaceRecord ALTER COLUMN trace int NULL;ALTER TABLE
TW_DataElement ALTER COLUMN trace int NULL;
```

Web Services

Porting Web Services applications to VisualWorks 7.10

In VisualWorks 7.10, the Opentalk Web services parcels have been moved to the **/obsolete** directory. If your application uses an Opentalk client or server, you need to install these obsolete components. The Opentalk-SOAP and Opentalk-WS-Support parcels have a few minor fixes, but otherwise they are almost identical to the versions in the 7.9.1 release. Existing Web Services applications will work using these Opentalk parcels, without any changes to your application code. However, the Web services tool no longer supports generating Opentalk client and server classes.

Summary of the changes to Web Services in 7.10:

- Class WsdlClient has been enhanced with support for interceptors, and now replaces the Opentalk client. The WSDL client provides authentication, SOAP header processing via interceptors, and both HTTP and HTTPS connections.
- The Opentalk server has been replaced by the new SiouX server. SiouX is a basic HTTP server with a simple responder mechanism for connecting higher-level services to it. Using responders allows running Web Services and other web requests from the same host and port. SiouX also has HTTPS support built in.
- SiouX includes a tool to manage servers.

The basic steps to port an Opentalk-based client and server to 7.10:

1 Having installed the obsolete components from the release media, load the Opentalk-WS-Support package into the image.

2 Load your Web Service application.

3 Create an instance of the server or client to initialize the registries (WsdlBinding and XMLObjectBinding).

4 Load the WSDLWizard parcel.

5 Use the WSDLWizard to regenerate new client and/or responder classes.

The tool generates:

- The client class derived from superclass WsdlClient.

- The responder class derived from superclass SOAPResponder.
- Client interceptor classes if the WSDL has SOAP headers. The client interceptors are derived from ClientMessageInterceptor class. The client interceptors are described in the client's #processingPolicy class method.
- Responder interceptor classes if the WSDL has SOAP headers. The responder interceptors are derived from class SOAPMessageInterceptor. The responder interceptors are defined in the #inteceptors class method.

If you port an application that uses SOAP header from version 7.9, the interceptor methods can be copied to the new client and server interceptors without any changes. For example, the implementation of the callback #sendingRequest:in: should be copied to a client interceptor and #receivingRequestEnvelope:in: to a responder interceptor.

Applications running in VisualWorks 7.8 and earlier could have header processors. amd porting these to interceptors will require changes. More specifically, the header processor method #addInputHeader:transport: should be re-implemented as the client interceptor method #sendingRequest:in:. The header processor method #processInputHeader:transport: should be re-implemented as the responder interceptor method: #receivingRequestEnvelope:in:.

The code for running a Web service application in 7.10 has changed as well, e.g.:

```
HTimeResponder addToServer.
client := WebServices.HTimeClient new.
value := client timeNow.
HTimeResponder flushResponders.
```

Enhance WsdlClient to support interception points

(AR 65222, 50157) These ARs introduced the following changes in Web Services support:

- The WSDL client has been enhanced to support interceptor points. Since class WsdlClient now supports interceptors, authentication and persistent connection the Opentalk client is deprecated.
- Henceforth, the Web Services tool only creates a WSDL client. The option to create Opentalk classes has been removed.

- Existing Opentalk-based client classes will continue to work without any changes, but we encourage developers to upgrade their applications to use the new WSDL clients. The Opentalk-SOAP and Opentalk-WS-Support packages have been moved to the **/obsolete** directory, but they can be loaded in the image as before.
- The Web Services tool creates two interceptor classes for each SOAP header: client and responder interceptors. The SOAPProcessingPolicy with SOAPMessageInterceptor classes serve receiving request and sending response callbacks on the server. The ClientProcessingPolicy with ClientMessageInterceptor classes serve sending request and receiving response callbacks by WsdlClient and its subclasses.
- The tool creates an application responder derived from class SOAPResponder. The responder is added to the SiouX HTTP server. The application responder knows the WSDL schema, X2O binding, service map and, if the WSDL has SOAP headers, the interceptors as well.
- The SiouX server uses a collection of responders to process the SOAP messages. The default is class SOAPResponder which is derived from NetHttpResponder. Class NetHttpResponder uses the Net HTTP framework. The responder receives Net.HttpRequest and sends Net.HttpResponse objects.

To update your application to use a WSDL client:

- Create WSDL clients using the Web Services Wizard or WsdlClassBuilder.
- If your Opentalk client used interceptors, create a #processingPolicy class method for the new client and describe the processing policy and interceptors there.

For example, if your existing Opentalk client describes processing policy in a #newBroker method as:

```
newBroker
| configuration |
configuration := BasicBrokerConfiguration new
    adaptor: (ConnectionAdaptorConfiguration new
                isBiDirectional: false;
                processingPolicy:
                    (DemoProcessingPolicy new
                        interceptorClasses:
                            (OrderedCollection with: DemoHeaderInterceptor);
```

```
                    yourself);
                transport:
                    (#{HTTPTransportConfiguration} value new
                    marshaler: (SOAPMarshalerConfiguration new
                    bindings: (OrderedCollection with: self class binding);
                    yourself))).
    ^configuration newAtPort: self port.
```

The new WSDL client needs to implement the following class method:

```
processingPolicy
^DemoProcessingPolicy new
    interceptorClasses: (OrderedCollection with: DemoHeaderInterceptor);
    yourself
```

The old interceptor classes should work with the new WSDL client classes without any changes, even if the interceptor callback parameters were changed from Opentalk.SOAPRequest and Opentalk.SOAPReply to WebServices.SoapRequest and WebServices.SoapResponse.

To illustrate, we can use the Web Services Demo interceptor: DemoHeaderInterceptor. Here, the interceptor callback #sendingRequest:in: accepts an instance of Opentalk.SOAPRequest sent by the Opentalk client as its first parameter, and WebServices.SoapRequest sent by the WSDL client:

```
sendingRequest: aRequest in: aTransport
    (aRequest needsHeader: #AccessLevel)
        ifTrue:
            ["Add the #AccessLevel header to the request"
            (aRequest headerFor: #'AccessLevel')
                value: self policy accessLevel].
    (aRequest needsHeader: #AuthenticationToken)
        ifTrue:
            [(aRequest headerFor: #'AuthenticationToken')
                value: (WebServices.AuthenticationToken new
                        userID: self policy userID;
                        password: self policy password;
                        yourself).]
```

For each SOAP Header interceptor, the Web Services Tool creates:

- Client interceptors derived from class WebServices.ClientMessageInterceptor that provide the following callbacks:
    - #sendingRequestEnvelope:in:

- #sendingRequestDOM:
- #sendingRequest:in:
- #receivingReplyEnvelope:in:
- #receivingReply:in:

- Responder interceptors derived from class SOAPMessageInterceptor that provide the following callbacks:
  - #receivingRequestDOM:
  - #receivingRequestEnvelope:in:
  - #sendingReplyEnvelope:in:
  - #sendingResponseDOM:

The Web Services tool creates a responder class derived from class SOAPResponder. The SiouX Server maintains a collection of SOAPResponder objects. The SOAPResponder is responsible for processing SOAP message for a specified WSDL binding address. The responder accepts an HTTP request in the #acceptRequest:method:version:connection: method and checks if there is a WSDL binding port for this path. If the path is found, the responder returns a HttpResponse object with a marshaled SOAP response. If the responder returns false the server suggests the HttpRequest to the next responder. The SOAPResponder can also define its own processing policy. To add a processing policy to an application responder, implement the class method #interceptors and set the processing policy in #defaultProcessingPolicyClass:.

To illustrate, using the TimeResponder example:

```
"Load the WSDL schema, create responder instances, and add them to a
server"
TimeResponder addToServer.
"Create a client and invoke the service"
client := TimeClient new.
value := client timeNow.
"Release the responders from the cache and remove them from the
server"
TimeResponder flushResponders.
```

To access a responder instance use the SOAPResponder class>>responders method. This method loads the WSDL, initializes WsdlBinding and X2O bindings, creates instances of the application responder and caches them in the class variables #responders. A responder instance is created for each <soap:address> element. Note

that creating responder instances is time consuming because it may require loading an imported component from the Internet. If a WSDL schema is changed, use the SOAPResponder class>>flushResponders method to reset cached instances. Use the SOAPResponder class>>addToServer method to find an existing SiouX server with a matching IP Address, or create a new server and add the responder to it. If the server was just created, it will start automatically.

Net Clients

Support for UTF-8 encoded Filenames for FTP

(AR 67475) In the release, support for the FEAT command (RFC 2389) has been added.

In order to permit clients to determine which new commands are supported by a particular server, without trying each possible command, the FEAT command requests the server to list all extension commands, or extended mechanisms, that it supports.

Right after the connection is established, the FTP client sends the FEAT command to the server, processes the response, and collects the server features in the #features instance variable. If the FTP server support UTF-8 encoding, the FTPClient optionally applies this encoding, depending on the state of the #useUtf8 instance or class variable (by default, it is true).

To set the option for an instance of FTPClient:

```
FTPClient new useUtf8: aBoolean.
```

To set the option for all new instances:

```
FTPClient useUtf8: aBoolean.
```

To create a directory with non-ASCII characters in its name, the client can send a request:

```
ftpClient := FTPClient host: ‘ftpServer’.
ftpClient makeDirectory: 'test P?íliš žlu?ou?ký'.
```

UTF-8 encoding/decoding works for class FTPClient only. Class FTPUrl doesn’t yet support UTF-8. URL encoding and decoding issues will be addressed in the next release.

Shared variables use values from various Net settings

(AR 65132) In this release, the following changes appear in the Net Clients framework:

- Created the Net.PISettings hierarchy for each client PI. The settings hold #retries, #timeout and #delaySeconds parameters.
- Removed shared variables from class NetClient.
- Changed the Net pages in the VisualWorks Settings Tool. Added pages with #retries, #timeout and #delaySeconds parameters for each client.
- Removed the URI Support page from the Settings Tool.
- Changed the default for the URI auto-load option. Any missing parcels will now be loaded by default.

Add a default User-Agent field value to Net Settings

(AR 64757) This release includes the addition of an #addUserAgentFieldValue option to class HttpWritingOptions. By default, the option is disabled (set to false). If the option is set to true, a User-Agent field will be added to every HTTP client request. The default value for user agent field is current system version (i.e., the value of SystemUtils version).

To enable or disable the user-agent field:

```
HttpWritingOptions addUserAgentField: aBoolean
```

To set the value of the user-agent field"

```
HttpWritingOptions class>>userAgentFieldValue: aString
```

For example:

```
client := HttpClient new.
client writingOptions
      addUserAgentField: true;
      userAgentFieldValue: 'VisualWorksAgent'.
response := client get: 'http://en.wikipedia.org/wiki/
File:Flag_of_Canada.svg'.
```

MIME supports parsing extended parameters (RFC 5987)

(AR 53867) This release includes new support for extended header field parameters from the RFC 5987 specification.

For example:

```
value := ContentDispositionField readFrom: 'Content-Disposition: form-
data;filename*=iso-8859-1''en''%A3%20rates' readStream.
```

where the value filename is decoded as '£ rates'.

Extend URL support for data: and about: protocols

(AR 52346) This release includes two new classes in the URISupport package: DataURI and AboutURI.

Class DataURI represents a URI with the protocol data:, representing a resource provided by the URI data itself. For details, see: http://en.wikipedia.org/wiki/Data_URI_scheme

A DataURI can return a resource based on its mime type, e.g.:

'data:text/plain,Hello World' asURI resource.

The shared variable MimeTypes holds BlockClosure objects that handle mime type resource resolution. If you load the ImageReaders package, the image/* mime types are mapped to Image objects. You can use DataURI class>>mimeTypes to get the dictionary and add your own handler code (or replace the existing handlers) for different mime types.

Class AboutURI represents a URI with the protocol about:, representing a resource provided by a local image service. The protocol is unofficial but is used widely by web browsers.

http://en.wikipedia.org/wiki/About_URI_scheme

The about: URI is essentially a services lookup mechanism. The 'data' portion of the URI is handled by the Services shared variable, and you can add your own unique services to the dictionary by calling AboutURI class>>services. The services can be anything and are not governed by any specification.

The framework provides a simple registry of "services" for about: that return an object:

blank

Empty string

version

Details about this version of VisualWorks

license

Cincom's license string for VisualWorks

logo

The VisualWorks logo image

about

The services dictionary

asset?name

Look up an object in the Assets class hierarchy

Examples:

```
'about:version' asURI resource inspect.
'about:logo' asURI resource inspect.
'about:asset?DebuggerIcons/pause' asURI resource inspect.
```

Ensure consistent error handling while parsing header values

(AR 51090) This release includes changes in the code for parsing malformed header fields:

- Parsing malformed headers reraise all Errors as proceedable InvalidHeaderField notifications.
- The InvalidHeaderField default action creates an UnparsedHeaderFieldValue object and assigns it to the header value. This UnparsedHeaderFieldValue object holds information about the parsing error, stream and error position.
- Printing malformed header fields prints the original header field source.
- Writing malformed header field raises a proceedable WritingInvalidHeader exception. The exception parameter is a stream from the #writeOn: method.
- Parsing header fields with malformed parameters will raise a proceedable InvalidParameterValue exception. If the exception is chosen to proceed, the invalid parameter is skipped.
- Parsing a correct network address creates a NetworkEntityDescriptor object. If a network address is malformed, the header value will be an UnparsedHeaderFieldValue.

For example, to parse a sender field with an empty network address:

```
field := HeaderField readFrom: 'Sender: <>' readStream.
```

The address field returns an instance of UnparsedHeaderValue. To print this header, e.g.:

```
field printString
```

This returns the string: 'Sender: <> (unparsed)'.

When writing this header to a stream:

```
(field writeOn: String new writeStream)
```

When the address is empty, an WritingInvalidHeader exception is raised.

DotNET Connect

Support for Visual Studio 2012

The DotNETConnect Code Generator now also generates makefiles suitable for Visual Studio 2012.

64-Bit Support

DotNETConnect can now also be used in the 64 bit version of VisualWorks.

Improved exception handling in delegates

You can now use all standard Smalltalk exception handling techniques for errors inside delegates. You can raise exceptions in the delegate code and catch, retry or resume them with an exception handler outside the delegate. See class DelegateTests in the parcel DotNETSUnitTests for examples.

COM Connect

Documentation

This section provides a summary of the main documentation changes.

Basic Libraries Guide

Minor updates.

Tool Guide

Expansion and revision of chapter on the Rewrite Tool. Minor revisions to discussion of Unit Testing.

Application Developer's Guide

No changes.

COM Connect Guide

Several updates.

Database Application Developer's Guide

Updates to EXDI chapter and discussions of Unicode support.

DLL and C Connect Guide

No changes.

DotNETConnect User's Guide

Several updates.

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 changes.

Security Guide

No changes.

Source Code Management Guide

Minor changes.

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

Extensive revision for 7.10.

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.

Virtual Machine

Web Application Server

VisualWave/WebToolkit designated as Legacy Components

With the introduction of AppeX in release 7.10 and the broad use of Seaside for web application development, the two older VisualWorks web application frameworks are now designated as legacy components. This is not a big change in any practical sense. It does not change the support status of the components. Unlike obsoleting it does not signal any imminent plans to remove these components from the product. However, it is meant to steer developers away from using these frameworks for new applications. Going forward, we do not intend to develop these frameworks further, beyond what may be required by support obligations.

The technologies represented by VisualWave and the Web Toolkit have been in decline for a number of years now. At present, a second generation of technologies has emerged (JS/HTML5) since the JSP/ASP/Servlet models provided by the Web Toolkit (which is the more modern of the older VisualWorks web application frameworks).

Given the fairly dynamic lifecycle of web technologies, we're now pursuing a solution that should allow seamless mixing of various server technologies. This should enable incremental transition from older technologies to newer ones at a pace that suits our customers. We hope to be able to provide a reasonable migration strategy for at least some of the applications built with WebToolkit in a subsequent release. However, we currently do not plan to enable the same for VisualWave applications.

VisualWave Load-Balancing and Wave-VRML now obsolete

The load-balancing and VRML components of VisualWave are obsoleted in this release, with the goal of removing them altogether in the following release. These components have seen very little use by our customers and removing them from the product allows us to focus on the components that are now in demand.

CGI gateways obsolesced in favor of Reverse-proxying

VisualWorks 7.10 includes a technical note (**`/doc/TechNotes/Using-Server-Proxies.pdf`**) that describes how to configure some popular web servers (Apache and IIS) to use reverse proxying to pass requests to a VisualWorks web server. This configuration can be used for any framework relying on HTTP (Seaside, Web Services, Wave/WebToolkit, etc.) and should be more than an adequate replacement for the CGI relays that were used for this purpose to date. Consequently, we intend to obsolete the relays in the following release, and encourage our customers to use reverse-proxying configurations instead.

Opentalk

Opentalk-HTTP and Opentalk-CGI obsolesced

Opentalk-HTTP was used for Web Service and Seaside servers. In this release this function has been supplanted by SiouX, a new web server framework. The new framework is simpler in many respects as it doesn't have to cater to Opentalk's request broker architecture. It

also provides additional capabilities that the previous solution didn't. For example, a SiouX server can host multiple services of various kinds on the same address/port, and a single server can serve multiple ports at the same time. SiouX servers also provide support for HTTPS. The change should be largely transparent as the server framework is mostly hidden underneath the Seaside and Web Services layers.

Opentalk-CGI was used as a communications relay for third-party web servers (Apache, IIS) through VisualWave's CGI relay framework. We are moving away from this solution and are henceforth recommending the use of a reverse proxy. In 7.10, the VisualWorks documentation includes a technical note (**`/doc/TechNotes/Using-Server-Proxies.pdf`**) describing how to set up a reverse proxy with common third-party web servers (Apache and IIS).

WebServices

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.

Web Application Server

SiouX: A New Web Server

In VisualWorks 7.10, we're introducing a new general-purpose web server, SiouX. There are/were several HTTP servers in the product, supporting various server solutions employing the HTTP protocol. SiouX is meant to replace most of them and finally enable the ability to run different kinds of services from the same IP address and port, as well as the ability to serve multiple ports at the same time. It also brings the capability to directly serve connections secured by SSL/TLS protocol (HTTPS).

SiouX also serves as a platform for our, currently experimental, re-implementation of HTTP using Xtreams (see: **`preview/www/SiouX-Http`**). However it also provides a fully supported HTTP back-end based on the proven HTTP implementation from the Net framework (see: **`www/SiouX-Net-Http`**). This implementation is largely equivalent to what used to be provided by Opentalk-HTTP, just without the complications imposed by the Opentalk architecture. Note that both back-ends can be used simultaneously in the same server. Even on the same connection requests can be handled by either back-end, the Responder selected to handle a request determines which back-end is used.

Seaside and Web Services are already re-hosted on top of SiouX-Net-Http. With Seaside it is largely transparent. In Web Services the tooling generates a lot of additional infrastructure, which would previously be based around Opentalk. With SiouX the infrastructure is a lot simpler, but also very different. It is still possible to run Web Services created with previous releases in the 7.10 release, relying on Opentalk-HTTP (now moved to **`/obsolete`**). However, it should be fairly straightforward to regenerate the service infrastructure in the new release to get a SiouX-based solution that can take advantage of the benefits (like being able to run multiple services off the same address/port). Main goal of preserving the ability for the two solutions to coexist in the same image was to allow for an easier, incremental transition.

SiouX also facilitates creation of custom data services, web APIs, etc. Just create your own subclass of the SiouX Responder that takes an incoming request and creates a response and register it with a server under a specified url prefix. The Responder superclass determines which back-end is used (Net or Xtreams) and which kind of request/response objects will be used.

Finally, there's also a preview of a new web application framework, AppeX. Unlike Seaside, it is aimed at supporting new-style web application development which moves a lot of the application logic to the client side, leveraging the Javascript runtime in the browser and the vast ecosystem of Javascript libraries.

SiouX also comes with server configuration tool allowing to create/configure/start/stop/destroy servers in the image (SiouX-Tools). For more information, please refer to the package and class comments.

Here's the list of available packages:

Directory **`www/`** (supported):

SiouX-Server

The core framework

SiouX-Server-Secure

Adds support for secure connections (HTTPS)

SiouX-Tools

Server configuration tools

SiouX-Net-Http

Net framework based HTTP implementation

SiouX-Net-Examples

Example responders for the Net based back-end

Directory **`preview/www`** (preview):

SiouX-Https

Xtreams based HTTP implementation

SiouX-Examples

Example responders for the Xtreams based back-end

AppeX: A New Framework for Developing Web Applications

In this release we are including a preview of new framework for developing highly interactive, modern web applications that utilize the capabilities of web browsers supporting HTML5 and Javascript (ECMAScript 5.1). It is based on SiouX, our new general purpose web server.

AppeX builds on the experience and insights gained from our previous web application frameworks. We recognize the importance of our products' ability to support modern web application development, and this represents our commitment to provide developers with the tools and frameworks to remain competitive in the web application marketplace.

Although AppeX is currently offered as a preview, it is a full-featured framework providing automatic web session management, asynchronous client/server AJAX communication, server to client push notification of events using the HTML5 EventSource object, and more.

AppeX is completely library-neutral; there are no requirements to use any particular Javascript library, but it enables developers to use whichever library they wish. Developers who prefer direct manipulation of the client DOM can do so right out of the box. Those who prefer to work with more friendly DOM API libraries (jQuery, Mootools, Twitter Bootstrap, etc.) can easily include them if they wish, and write the Javascript code using the library of their choice. One of our goals is to provide web developers with the ability to write web

applications in the scripting language and API with which they are already familiar, while integrating them with the rich set of tools included in the VisualWorks IDE.

For server-side code, developers will use the Smalltalk classes they are already familiar with. Writing web request-handling methods is as simple as tagging a method with a pragma declaring the path to the request, while the framework takes care of the rest, including session management and event push notifications. The request handlers can serve various content types, from plain text, to JSON-encoded binary data, to the content of both static and dynamically-generated files.

For client-side code, developers can write object-oriented Javascript code directly in the VisualWorks System Browser, and the AppeX framework pushes the resulting Javascript code library to the client's web browser at runtime.

We have included numerous examples of simple web applications, with and without the use of established Javascript libraries such as JQuery, as well as an example of serving a completely Javascript-free, static HTML web site.

Loading AppeX

AppeX consists of four parcels currently included in the **`preview/www`** directory. For development, you will need to load AppeX-Tools which includes all the prerequisite parcels. The AppeX-Examples parcel has several example web applications that demonstrate, at different levels of detail and/or complexity, various techniques for writing web applications with AppeX. Finally, AppeX-Tests contains a small suite of automated tests that can be used to unit-test AppeX functionality and explore the AppeX framework further.

The AppeX web application framework is built using the new SiouX HTTP server. (For details on how to set up, configure, and maintain SiouX servers and responders, please refer to comments for the SiouX-Server package.)

An AppeX web application typically consists of at least two classes: a Smalltalk server class that runs in an image on a server host, and a Javascript client class which, although written in the Smalltalk IDE, is used to generate Javascript code that is downloaded and executed by a web browser on a client machine. Unlike some web development tools, AppeX does not use a mapping mechanism whereby you would write code in Smalltalk and let the framework translate it to Javascript. The client code is written in Javascript, and the code you see in the IDE is what you get in the client web browser.

Usage

The two key classes that form the AppeX web application framework are ApplicationServer and ApplicationClient.

ApplicationServer is an abstract class that implements all the server-side behavior of the web application. Its name should not be confused with the usual meaning of the term “application server” — it only indicates that this class (and its subclasses) implements an application’s behavior, and that it resides on the server. When writing an HTML5 application using this framework, developers will have to create a subclass of ApplicationServer to handle client requests, and the corresponding client-side code in a subclass of ApplicationClient.

ApplicationClient is an abstract class that implements the client-side application behavior. Each concrete subclass that you define contains the Javascript code required to run in the client’s web browser. ApplicationClient is a subclass of JavascriptClass (discussed under “Implementation Details”, below), and it is represented on the client as an object accessible as $t.application. Only a single instance exists in a client window, and it is linked to a unique instance of the ApplicationServer subclass on the server. Thus, a pairing of a Javascript object on the client and a Smalltalk object on the server constitutes a single web session. If an application chooses to serve only static HTML files and requires no client-side Javascript code, it is not required to have an ApplicationClient subclass.

Web Application Development Tutorial

To help developers start developing HTML5/Javascript based web applications using AppeX, the following paragraphs explain how to write a simple 'Hello World!' web application.

This tutorial illustrates the key steps in building a web application, and introduces the behavior of the participating classes. The example is very simple and does not illustrate any user interaction or session-dependent behavior. The main goal is to show how a web application client/server class pair is created, how the service methods are implemented on the Smalltalk side, and how they are invoked from the Javascript client side.

To begin, subclass ApplicationServer, e.g., HelloWorldServer:

```
Smalltalk defineClass: #HelloWorldServer
    superclass: #{AppeX.ApplicationServer}
    indexedType: #none
    private: false
    instanceVariableNames: ''
```

```
    classInstanceVariableNames: ''
    imports: ''
    category: ''
```

Subclass ApplicationClient, e.g., HelloWorldClient:

```
Smalltalk defineClass: #HelloWorldClient
    superclass: #{AppeX.ApplicationClient}
    indexedType: #none
    private: false
    instanceVariableNames: ''
    classInstanceVariableNames: ''
    imports: ''
    category: ''
```

Implement the method HelloWorldServer class >> #applicationClientClass to return class HelloWorldClient:

```
applicationClientClass
    ^HelloWorldClient
```

Optionally, you can implement a HelloWorldServer class >> #htmlTitle method to return a String that will be the title of the starting HTML page. It can be later changed programmatically by the Javascript code in HelloWorldClient.

Create a simple pragma configuration method for the creation of a server that will accept request for your web application. Extend class SiouX.Server in the package that contains your application classes, with a class method like the one below. You may have to use a port other than 8888 if it is already in use:

```
SiouX.Server class >> myDemoServer: aServer
    <server: 'Demo HTTP Server'>
    aServer listenOn: 8888 for: HttpConnection
```

Here, the pragma is: <server: 'Demo HTTP Server'>,

Implement a pragma configuration method to define a responder to be added to the server. The responder will dispatch requests with a path matching the one in the path: argument of the pragma. Note that the path must start with a forward slash.

```
HelloWorldServer class >> helloWorldResponder
    <server: 'Demo HTTP Server' path: '/hello-world'>
```

To start the server, evaluate the following code in a workspace:

```
(SiouX.Server id: 'Demo HTTP Server') start
```

Alternately, you can use the <Operate> context menu in the System Browser to create/destroy a server, or to register/deregister the application with the server. These menu picks are available whenever you select an AppeX pragma method in the browser.

At this point, you should be able to access this example application from a web browser, e.g.: http://localhost:8888/hello-world. It will show the default HTML content inherited from ApplicationServer, but there are no message sends from the client to the server other than the session link establishment and some framework-related messages to register interest in server events.

In the following steps, we can create a service method to return some data from the server to the client, and a client function to send a message to the server, invoking the service method.

Implement a class-side method in HelloWorldServer that returns the 'Hello World' greeting:

```
HelloWorldServer class >> #getGreeting
    <plainText: 'getGreeting'>
    ^'Hello World!'
```

Implement an instance method in HelloWorldClient that builds the full DOM on the client. Note that you will write a Javascript function body (without the 'function' keyword). The JavascriptCompiler will translate the code below into an annotated method named #buildHtml that returns the full string of the function definition:

```
buildHtml() {
    // Create an HTML paragraph element
    var element = document.createElement("p");

    // Send a SYNCHRONOUS message named "getGreeting" to the server.
    // The name of the message is the same as the #plainText: pragma
argument in HelloWorldServer class >> #getGreeting method.
    // On the server side, the method name and the pragma argument do
not have to be the same.
    var response = this.messageToServer("getGreeting", null, {async:
false});

    // Retrieve the greeting from the response
    var greeting = response.object;

    // Create an HTML text node and append it to the paragraph element.
    element.appendChild(document.createTextNode(greeting));

    // Append the paragraph element to the document body
```

```
    document.body.appendChild(element);
}
```

Refresh the browser at http://localhost:8888/hello-world to see a Hello World! greeting generated by your application.

You can now stop and destroy the demo server:

```
(SiouX.Server.Registry at: 'Demo HTTP Server') release
```

This concludes the AppeX tutorial.

Other Features

For brevity and simplicity, we have omitted some AppeX features from the tutorial. We will briefly describe them below, and indicate where to find more information. More complete documentation for AppeX will be available in a subsequent release of VisualWorks.

Content Pragmas

Each request handler method must declare a pragma that indicates what kind of content it returns, and the path relative to the application's base URI. The pragmas are defined in ApplicationServer class >> #requestHandlerPragmas. Each of the pragmas is also the name of a corresponding class method in class ApplicationServer, e.g., the #html: pragma is linked to the ApplicationServer class >> #html: method. During request dispatch, the request URL is used to look up a pragma matching the path of the URL, and the request is dispatched to the method which declares the matching pragma. Request handler methods implemented on the application's class side represent session neutral requests, and those implemented on the instance side represent requests tied to one of the sessions in the class session cache. For each path, you can implement a request handler method either on the instance or on the class side, but not both.

Customizing the Application's HTML Document Structure

Each web application is assumed to have a main HTML document that is the entry point into the application. The HTML document loads the corresponding Javascript libraries, Cascading Style Sheets, and it runs the scripts that initialize the application. By default, this HTML document is returned from the #htmlDocument class method of your application server class, and is indicated by the use of the #html: pragma with the application's root path (e.g., <html: ''>).

To make the structure of the main HTML document flexible and easily configurable, we define two html part pragmas — #head: and #body: (see ApplicationServer class >> #htmlPartPragmas). Developers can include various parts in the generated <head> and <body> elements of the HTML document by implementing methods with #head: or #body: pragma declarations, using a numeric position indicator of where the part should be in the document relative to other parts. Lower position indicators mean the part will be included closer to the beginning of the element, e.g., the content returned from a method declaring <body: 100> will be before that of declaring <body: 200>. Naturally, all the content declared with the #head: pragma will come before any declared with the #body: pragma, regardless of their respective relative positisions. All HTML part methods using these pragmas must return strings that will together form the final HTML document. For examples of how this works, examine the senders of #head: and #body: in ApplicationServer or its subclasses.

HTML Templates

To further customize the content of the main HTML document, its portions may be declared as 'template tokens' that get expanded at runtime and in the context of a particular request, application name, etc. Developers can include an HTML token in the form of {{token-name}} throughout the HTML parts, and write methods with #htmlToken: pragma where the pragma argument is a matching 'token-name'. This mechanism is modelled after the mustache template (http://mustache.github.io). If a template does not have a matching #htmlToken: pragma on the server side, it is the developer's responsibility to write corresponding Javascript code to expand the token on the client, which is beyond the scope of these Release Notes.

Server Event Push Notification

AppeX provides an automatic push notification of events that occur on the server or in other client sessions. It is implemented using the Javascript EventSource object connected to a streaming response at the path get/server-events. An application client registers interest in a server event with Javascript code in the form:

```
this.onServerEvent(<event-name>, <callback>);
```

On the server, events are pushed to the clients by executing:

```
self announceServerEvent: <serverEvent>
```

where <serverEvent> is an instance of ServerEvent with an event name, an optional sessionId of the session that originated the event, and optional event data.

This mechanism should handle most situations in which a client needs to be notified of server events. However, developers can write their own, using an EventSource object on the client, and writing an #eventStream: request handler method on the server. Note that the #eventStream: response must be a one-argument block that accepts an Xtreams.WriteStream in the argument, and the data in the streaming response must conform to the EventSource 'text/event-stream' format.

Implementation Details

There is more to the AppeX framework than the ApplicationServer and ApplicationClient classes described above. Additional classes with which developers should become somewhat familiar are:

ApplicationResponder
JavascriptCode
CoreCode
JavascriptClass

Their roles in the framework and their mutual relationships are briefly explained below. To learn more about each of these classes, refer to their respective class comments.

ApplicationResponder

This belongs to the SiouX.Responder class hierarchy. Its sole purpose is to determine if an HTTP request should be handled by the application server class to which it was given and, if so, to dispatch the request to an ApplicationServer class for domain-specific request handling. You should never have to subclass ApplicationResponder, as all the domain-specific behavior is implemented by subclasses of ApplicationServer.

JavascriptCode

A class that implements abstract behavior for developing, maintaining and generating Javascript from within the Smalltalk image. It is not necessary to create instances of these classes, as they serve as placeholders for Javascript code.

JavascriptClass

A class that represents a Javascript constructor function used to generate a 'class' in the web browser. The framework translates instance methods in JavascriptClass to JS functions and binds them to the constructor prototype on the client.

CoreCode

Subclass of JavascriptCode. This class generates the core javascript global objects used to build a web application. The code generated by CoreCode must be the first set of Javascript statements in the generated JS.

We wish you good luck in developing modern, feature-rich web applications with AppeX!

Glorp

Reading DescriptorSystems from Existing Databases

Several new packages are available to assist with reading descriptor systems from existing databases.

GlorpAtlasSystemsInVW

If a Glorp model has been prepared in ObjectStudio's Modelling and Mapping tools and saved to a parcel or package, load this utility to enable loading that component into a VisualWorks image.

GlorpAtlasClassGeneration

This package extends the GlorpActiveRecord utilities. Like ActiveRecord, it automatically reads the schema to create a descriptor system that maps between them, but it can read from schemas that do not adhere to the ActiveRecord pattern, and it can generate classes that match the tables.

GlorpAtlasSystemGeneration

This supports generating a DescriptorSystem subclass, with classFor<Name>:, descriptorFor<Name>: and tableFor<Name>: methods, from a descriptor system. (Typically, this will be a descriptor system created when a DB's metadata is read by GlorpActiveRecord or GlorpAtlasSystemGeneration.)

For detailed API descriptions, see their package comments. The GlorpAtlasTests package provides some usage examples. Load them in VisualWorks only; ObjectStudio's modelling and mapping tools provide a UI on a more powerful version of the same capability.

UISkinning

UISkinning is a project that improves the look and feel architecture in VisualWorks. It provides a look and feel named Skinned that can take on a number of 'skins'. The supplied skins are:

Native

Only available on Mac OS X 10.5 and above, and Windows with theming support. In practice, this excludes Windows Server versions. This skin uses the Mac OS X artfile or the Windows UxTheme DLL to provide a high-fidelity native look for those platforms

Default

This is a platform-agnostic look and is available on every platform. It has a flat, geometric design.

Default Red

Identical to Default except with the addition of a strong red tint to all elements. This skin is meant for use when you are running more than one instance of VisualWorks and you need to easily distinguish the windows.

Default Green

As per Default Red, but the tint is green.

UISkinning employs a delegate/strategy architecture to provide different looks, rather than inheritance. This means that widgets are not subclassed. Support for different feels is provided purely within the widget — each widget is a 'super widget' in this respect. There are some characteristics of the look and feel that are not determined by the skin, but rather by the underlying platform. For example, font selection is not part of the look, but is provided by a new mechanism in the base image: Screen>>nativeGUIPolicy. Similarly, feel is determined primarily by the underlying platform. Thus the Default skin looks slightly different on each platform w.r.t. fonts, menu accelerators etc, and obeys platform-native feel conventions.

The ability to have e.g a Windows look and feel running on Mac OS X or X11 is not available for obvious technical reasons (the native support is not available). The focus of UISkinning is on adaptation, not simulation.

We anticipate UISkinning being a fully supported part of VisualWorks in 7.11. It is the future of look and feel support for VisualWorks and will be the focus of all of our efforts in this area. It is our intention to get as close to 100% native look and feel on Windows and Mac OS X as possible.

Being preview, we explicitly make no guarantees about compatibility with an upcoming release version in any sense e.g. the class hierarchy will change.

Regardless of the above caveats, UISkinning covers most of widgets and is usable for development. The most obvious exclusion is support for table header elements being buttons, which affects the sort order controls.

Text2

Text2 is a new multilingual text layout and rendering framework for VisualWorks. In release 7.10, it is part of the base image. Text2 comes with a new model layer which supports new capabilities not previously supported in the product, such as:

- Adornments — strike through, jagged underlines, double underlines and more
- Actions — clicking on a piece of text could activate a hyperlink
- Annotations — pop up extra information as the mouse moves over a section of text
- Variable font sizes in the same document
- Vertical alignment within lines
- Bidirectional content with support for embedded direction marks
- Unicode font measuring and rendering
- Images in documents
- Lists both bulleted and numbered, with international numbering schemes
- Extensible design to add new features

- Sparse editing history, to support large editing and unlimited undo/redo
- Multilingual unicode word and sentence detection rules

The model comes with new widgets too:

DocumentView

View document objects rendered to the screen or a printer.

DocumentEditor

An interactive multi-line text editor widget with extensive customizability.

InputEditor

A single line input editor to compliment the document editor.

Details on how to use these new features are expanded on in class comments found on Text2.Document, Text2.Flow and UI.DocumentView.

For customers familiar with original Text framework, they know that to get only some of these features they must use the unsupported ExtraEmphasis package. Text2 offers these same kind of capabilities plus a great deal more and it's part of the product, ready to use from the UIPainter today.

Universal Start Up Script (for Unix based platforms)

This release includes a preview of new VW loader that runs on all Unix and Linux platforms. This loader selects the correct object engine for an image, based on the image version stamp. Formerly, the only loader of this sort was for Windows.

The loader consists of two files and a readme in preview/bin. Installation and configuration information is provided in the readme.

This loader is written as a standard shell script which allows it to be used to launch VW on virtually any Unix based platform. This opens up the possibility of having a centrally managed site-wide installation of an arbitrary set of VW versions allowing users to simply run their images as executables without any user specific setup required. The loader figures out which version of VW and which specific VM is needed to run the image being launched, using information provided in the INI file).

For installations using only final releases (not development build releases), a single entry line in the INI file for each VW version will suffice to serve any Unix based platform for which a VM is available at the specified location.

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.

BOSS 32 vs. BOSS 64

The current implementation of BOSS (boss32, version 7), does not accomodate 64-bit small integers and small doubles natively. Also, it does not support extremely large objects that are outside the implementation limits for 32 bits. Furthermore, since the implementation of identityHash is not equal in 32 and 64 bit images, identity based collections may require special handling when moving them across images of different bit size.

A preview implementation of boss64 (version 8) has been implemented for this purpose. This implementation is an addition to the existing BOSS parcel, and is called BOSS64.

The new BOSS implementation has been structured so that there is a new factory class that takes care of choosing the proper reader for either boss32 or boss64 without user intervention, and a similar factory arrangement that chooses either boss32 or boss64 as the output format depending on the image BOSS is running on.

More concretely, until now application code would have referred to BinaryObjectStorage to write BOSS streams in boss32 format:

```
BinaryObjectStorage onNew: aStream
```

Referencing the class BinaryObjectStorage64 instead will result in BOSS streams in boss64 format:

```
BinaryObjectStorage64 onNew: aStream
```

Finally, referencing AbstractBinaryObjectStorage will choose either boss32 or boss64 depending on the image in which the code is running:

```
AbstractBinaryObjectStorage onNew: aStream
```

Moreover, referencing the abstract factory class for reading,

```
AbstractBinaryObjectStorage onOld: aStream
```

will automatically determine the format of the stream and choose the appropriate reader accordingly:

Execution environment	Selected reader
32-bit image, 32-bit BOSS stream	BOSSReader
64-bit image, 32-bit BOSS stream	BOSSReader32
64-bit BOSS stream	BOSSReader64

Existing code making reference to classes already present before these changes will not be affected, and they will still rely on existing boss32 behavior.

Also, although boss64 streams can be written by 32 bit images, 32 bit images should write BOSS streams in 32 bit format because 64 bit images can read these BOSS streams while doing all the necessary conversions.

64-bit Image Conversion

The ImageWriter parcel is still capable of converting arbitrary 32- bit images to 64-bit images. However, due to an unresolved race condition, occasionally it may create an image that brings up one or more error windows. These windows can safely be closed, and if the 64- bit image is saved again, they will not return.

However, they may be problematic in a headless image or an image that has been produced by the Runtime Packager. For such cases, re-saving or recreating the original 32-bit image, and then converting it again may avoid the race condition. Alternatively, converting the image to 64 bits before applying the Runtime Packager or making the image headless may also be helpful.

ImageWriter empties all instances of HandleRegistry or its subclasses. Since these classes have traditionally been used to register objects which must be discarded on startup, emptying them during the image

write is safe. But if your code is using HandleRegistry or a subclass to hold objects which are intended to survive across snapshots, ImageWriter may disrupt your code. Running ImageWriter before initializing your registries may solve this problem. We would also like to know more about how you use HandleRegistry, in order to improve ImageWriter's ability to transform images without breaking them.

Tools

With the Trippy basic inspector now being much more robust, work was done on integrating this with the debugger. Nicknamed Diggy, it has not been finished yet, but may be loaded from **`preview/parcels`**.

At this writing, this causes the inspectors located in the bottom of the debugger (the receiver and context fields inspectors) to be basic trippy inspectors (not the entire diving/previewing inspector, just the basic tab). This makes operations between the two the same, and provides the icon feedback in the debugger. The stack list of the debugger also shows the receiver type icons.

Cairo

Overview

VisualWorks 7.8 includes ready-to-use Cairo libraries for use on MS-Windows (Windows 2000 and newer) and Apple Mac OS X (10.4 and newer, PowerPC or Intel processors). The prebuilt libraries are built from 1.8.8 stable release sources. It also includes version 7.7.1 - 1 of the CairoGraphics parcel which binds to said libraries.

Developers on MS-Windows and Mac OS X, should be able to simply load the CairoGraphics parcel and begin using Cairo.

Developers on X11 platforms may also use the CairoGraphics parcel, but will need to make sure VisualWorks have libcairo.so available in their library path. Most up to date Linux versions ship with Cairo binaries.

What is Cairo?

The main project page found at http://cairographics.org/ states:

Cairo is a 2D graphics library with support for multiple output devices.Currently supported output targets include the X Window System, Quartz, Win32, image buffers, PostScript, PDF, and SVG file output.

Cairo is designed to produce consistent output on all output media while taking advantage of display hardware acceleration when available (e.g. through the X Render Extension).

The cairo API provides operations similar to the drawing operators of PostScript and PDF. Operations in cairo including stroking and filling cubic Bézier splines, transforming and compositing translucent images, and antialiased text rendering. All drawing operations can be transformed by any affine transformation (scale, rotation, shear, etc.).

Cairo is implemented as a library written in the C programming language, but bindings are available for several different programming languages.

Cairo is free software and is available to be redistributed and/or modified under the terms of either the GNU Lesser General Public License (LGPL) version 2.1 or the Mozilla Public License (MPL) version 1.1 at your option.

The CairoGaphics parcel is a VisualWorks bridge to the Cairo graphics library. It is maintained as an open source project, hosted in the Cincom Public Repository.

Drawing with Cairo

This section describes a simple overview of drawing with Cairo with VisualWorks. It is not exhaustive, but rather demonstrative.

Getting a Cairo Context

A Cairo context is the object which defines transactions that draw things on a given surface. Cairo may be used to draw on 3 different types of VisualWorks surfaces: Windows, Pixmaps, and Images. For the first two one usually has a VisualWorks GraphicsContext object in play for the surface. To help manage resources efficiently, we use a while: aBlock pattern to create Cairo interface objects and release them efficiently.

```
aVWWindowGC
    newCairoContextWhile: [:cr | "...cairo work goes here..."].
aVWPixmapGC
    newCairoContextWhile: [:cr | "...cairo work goes here..."].
aVWImage
    newCairoContextWhile: [:cr: | "...cairo work goes here..."].
```

By convention, in the Cairo community in large, as well as in VisualWorks code, a Cairo context variable is always called a cr. Using this convention increases the likelihood that other Cairo programmers (both Smalltalk and for other language bindings) will understand your code.Cairo also has its own built in Surface type called an ImageSurface. These are like VisualWorks Pixmaps, but are managed by Cairo. They are created with a format code and an extent.

```
cairoImage := ImageSurface
    format: CairoFormat argb32
    extent: 100@100.
cairoImage
    newCairoContextWhile: [:cr | "...cairo work goes here..."].
```

Note: The dual-threaded VMs for 10.5 and 10.6 are not compatible with the CairoGraphics on OS X. The problem involves drawing on Pixmaps. On 10.6, the Pixmaps are blank, and on 10.5, this operation wll quickly crash the image.

Setting the Source

In VisualWorks parlance, the source is somewhat analogous to the paint of a GraphicsContext. Cairo operators will draw pixels from the source onto the target surface. Sources may be simple color values, linear and radial gradients, and other cairo surfaces.

Setting a simple ColorValue

```
cr source: ColorValue red
```

Setting an alpha channel weighted color

```
cr source: (ColorBlend red: 0.9 green: 0.2 blue: 0.3 alpha: 0.5).
```

Setting a vertical green to blue linear gradient

```
gradient := LinearGradient from: 0 @ 0 to: 0 @ 10.
gradient addStopAt: 0 colorBlend: ColorValue green.
gradient addStopAt: 1 colorBlend: ColorValue blue.
cr source: gradient.
```

Setting a radial translucent orange to yellow gradient

```
gradient := RadialGradient
                from: 0 @ 0
                radius: 0
                to: 0 @ 0
                radius: 100.
gradient addStopAt: 0 colorBlend: (ColorBlend orange alpha: 0.5).
```

```
gradient addStopAt: 1 colorBlend: (ColorValue yellow alpha: 0.2).
cr source: gradient.
```

Setting the file background.png as the soruce

```
surface := ImageSurface pngPath: 'background.png'.
cr source: surface.
```

Defining Shapes

Shapes in Cairo are defined by paths. A path is more than a simple polyline. A path is composed of a series of move, line, bezier curve, and close commands. They do not need to be contiguous. Defining a path does not actually cause it to be rendered to the context.

The following examples are a sample of the path creation methods found in the paths and handy paths method protocols of the CairoContext object.

Simple line from 0,0 to 40,50

```
cr
    moveTo: Point zero;
    lineTo: 40 @ 50.
```

Two disjoint rectangles

```
cr
    rectangle: (10 @ 20 corner: 30 @ 40);
    rectangle: (110 @ 120 extent: 40 @ 40).
```

Closed right triangle with leg length of 30

```
cr
    moveTo: 5 @ 5;
    relativeLineToX: 0 y: 30;
    relativeLineToX: 30 y: 0;
    closePath.
```

Cincom Logo in unit coordinate space

```
cr
    moveTo: -1 @ 0;
    arcRotationStart: 0
        sweep: 0.75
        center: 0 @ 0
        radius: 1;
    relativeLineTo: 0 @ -2;
    arcRotationStart: 0.5
        sweep: 0.25
        center: 1 @ 0
```

```
        radius: 1;
    lineTo: 1 @ 0.
```

Filling and Stroking Shapes

With a source set and a path defined, you can stroke and/or fill the shape. The messages stroke and fill can be sent to a cr. In both cases, the path is reset by the call. The messages strokePreserve and fillPreserve, cause the path to remain in effect even after the operation.The stroke width may be set with the strokeWidth: aNumber method. Stroking is a solid line unless a dashes: anArrayOfLengths offset: aNumber is set.

Draw a blue rectangle with a thick dashed red outline

```
cr
    rectangle: (10 @ 10 extent: 40 @ 40);
    source: ColorValue blue;
    fillPreserve;
    source: ColorValue red;
    strokeWidth: 3;
    dashes: #(1 2 3 4) offset: 0;
    stroke.
```

Additional Operators

Stroke and fill are the most common operators performed on a context. There are others that may be used:

paint

Like fill, but requires no path. Simply fills the entire clip region.

paintAlpha: *aNumber*

Like paint, but applies a uniform alpha adjustment during the operation.

maskSurface: *aSurface*

Paints the current source using the alpha channel of aSurface.

clip

Renders nothing, but intersects the current clip with the current path and clears the path. Use clipPreserve if path resetting is not desired.

Affine Transformations

Cairo uses a transfomation matrix at all levels of drawing. The matrix is described as:

XX XY Xo
YX YY Yo

Given two input values, Xi and Yi, the new values Xn and Yn are computed as follows:

Xn = Xi•XX + Yi•XY + Xo
Yn = Xi•YX + Yi•YY + Yo

The easist way to manipulate the matrix of a context is to use the modifyMatrix: method which takes a single argument block as its argument. For example, to adjust the matrix to be a centered unit coordinate space of the receiver view:

```
cr modifyMatrix:
        [:matrix |
        matrix
            scale: self bounds extent;
            translate: 0.5 asPoint].
```

Matrices may be modified with methods such as translate:, rotate:, scale:, etc. See the transformations method category of class Matrix. Any of these modifications are cumulative to the receiver.

Individual elements may be set as well using accessors such as xx:. The matrix can be returned to an initial unity state by sending initIdentity to it.

Patterns (source surfaces, gradients, etc) have their own matrices as well and also respond to the modifyMatrix: method. It is important to remember when using a pattern's matrix to modify its appearance, the matrix is applied on the source side, where as the context matrix is applied on the target side. In other words, pixels are extracted from the source pattern through the inverse of the matrix. One might translate: 10@20 a cr context to cause things to shift to the right 10, and down 20. But to achieve the same end result by modifying the source's matrix, and leaving the cr's untouched, one would use a translate: -10@-20. Use the reciprocal of any scaling factors in the same way.

The Context Stack

Cairo supports a drawing context stack. This allows one to take a "snapshot" of the current context stake, make changes for further operations, and then at some point "rollback" to the snapshot. The API used is saveWhile: aBlock. These may be nested.

They are particular useful with transformation operations. Consider the following example, which decides a 12-sided equilateral polygon centered around the point 50,50.

```
cr translate: 50 @ 50.
0 to: 1
    by: 1 / 12
    do: [:rotation | cr saveWhile:
        [cr
        rotation: rotation;
        lineTo: 50 @ 0]].
cr closePath
```

Grouping Operations

A final pattern of interest is the groupWhile: aBlock pattern. A group in Cairo terminology refers logically to a series of operations that are buffered to a temporary surface, which then may be used as source for a one time paint operation. This can be used to implement "double buffering" but may also may be used to assemble complex graphis that require multiple surfaces to piece together (e.g. two overlapping linear gradients, one in the vertical direction, one in the horizontal direction).

Deploying VisualWorks with Cairo Support

MS-Windows

The Cairo library is contained in a single cairo.dll file which is placed alongside the VisualWorks virtual machine. Simply include this dll along with your virtual machine executable, and you should have access to the Cairo library.

Mac OS X

Cairo is embedded in the application bundle. It is a set of 3 dylib's placed in a Frameworks directory which is coresident with the MacOS directory found in the application bundle directory structure. If you simply deploy the visual.app application bundle, you shouldn't need to do anything. If you assemble your own application bundle, you will

need to ensure that the Frameworks directory contains the 3 dylibs and is a parallel directory to whatever directory the virtual machine executable exists in.

Ongoing Work

The CairoGraphics package is an ongoing work. It is maintained in the Cincom Public Repository. If you find bugs or want to provide enhancements, please do so, publishing your work on a branch version.In the future, Cincom hopes to be able to support Cairo prebuilt libraries on all of its various supported platforms, making it a true piece of the VisualWorks cross-platform strategy, and allowing the VisualWorks IDE to begin to take advantage of the possibilities Cairo offers.Cairo is a 2D vector graphics library. It has rudimentary support for rendering character glyphs from platform fontsets. But it does not pretend to offer any of the higher level CairoGraphics preview services one usually needs to work with text (layout, measuring, etc). A sister project to Cairo called Pango is under investigation by Cincom engineers to also be used in a cross platform fashion, in the same way Cairo is being considered.

Grid

A Grid widget combines elements of the Table and Dataset widgets for a simpler and more flexible interface of viewing and editing tabulated forms. This release includes a preview of a new Grid, based on the Grid from the Widgetry project. It currently supports the following features:

- Multiple row sort by column with or without a UI
- Multiple and single selection options by row or individual cell
- Interactive row or column resize
- Scroll and align column, row, or cell to a particular pane position (e.g., center, left, right top, bottom)
- UIBuilder canvas support

Planned features include:

- A SelectionInGrid model. Currently one may directly access, add, remove, and change elements of the Grid. Direct access will always be available.
- Drag-and-drop rows or columns to add, remove, or sort elements

- A tree column
- Completion of announcements, trigger events, or callbacks
- Specific OS look support for column headers. Currently only a default look is supported.
- The column and row headers may be set to not scroll with the Grid.

Further information on usage and supporting classes with examples appears in /preview/Grid/grid.htm.

Store Previews

Internationalization

MessageCatalogManager supports multiple locales simultaneously

The PerProcessCatalogs package, in preview (preview/parcels), extends the existing MessageCatalogManager facilities to support simultaneous use of multiple locales. This is mainly needed for application servers where different clients may require server side processing to use different catalogs depending on the client locale.

For languages with different variants in different territories, e.g. different english dialects #en_US, #en_GB, etc. the application may require some messages to be localized differently, e.g. 'color' vs. 'colour'. At the same time many other message are common among the territories. To support this efficiently the Locale will search for the translations in the most specific catalogs first and then look for less specific ones if that fails. For example an #en_US Locale may subsequently look into catalogs for en_US, en and finally C.

Opentalk

The Opentalk preview provides extensions to VisualWorks and the Opentalk Communication Layer. It includes a preview implementation of SNMP, a remote debugger and a distributed profiler.

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

Distributed Profiler

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

Installing the Opentalk Profiler in a Target Image

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

- Opentalk-Profiler-Core

Installing the Opentalk Profiler in a Client Image

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

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

Opentalk Remote Debugger

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

The remote debugger is contained in two parcels.

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

```
RemoteDebuggerClient startMonitor
```

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

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

```
RemoteDebuggerClient stopMonitor
```

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

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

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

Opentalk CORBA

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 ]
```

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"
```

MatriX

Polycephaly2 renamed to MatriX

In VisualWorks 7.10, a new name was chosen to avoid unnecessary discomfort caused by the original name. The first version of Polycephaly is not being renamed as it will be obsoleted in favor of MatriX.

MatriX

MatriX (formerly known as Polycephaly) provides simple mechanisms for spawning multiple running copies of a single image and using those to perform various tasks in parallel. This is primarily useful when attempting to utilize hosts with multi-core CPUs. The images are spawned headless and are connected with the main controlling image through their standard I/O streams, which are wrapped with BOSS so that arbitrary objects can be sent through.

There are two versions of MatriX in preview at this time. MatriX (polycephaly2.pcl) is a direct descendant of Polycephaly 1 (polycephal.pcl), however there are some key differences between the two frameworks.

Under the hood, Polycephaly 1 used BOSS to marshal objects, while MatriX uses the Xtreams ObjectMarshaler. These two marshalers may handle object edge cases differently.

Polycephaly 1 used pipes to communicate with local images, while Polycephaly 2 uses sockets. On Unix platforms, these are domain sockets, which should be secure — but on Windows they may be TCP sockets, which may NOT be secure.

For more information and examples, refer to the package comments.