Cincom

# Cincom Smalltalk™

## VisualWorks 7.7

## Release Notes

P46-0106-15

SIMPLIFICATION THROUGH INNOVATION®

**Part Number: P46-0106-15**

**Software Release 7.7**



**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

## Chapter 4 Preview Components 4-1

# 1

# Introduction to VisualWorks 7.7

These release notes outline the changes made in the version 7.7 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

### 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.

# 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.

## Alt Shortcuts inactive for buttons and label mnemonics

A bug discovered late in the release makes keyboard shortcuts defined by label mnemonics inactive.

If an ActionButton, GroupBox, or Label has a mnemonic defined by an ampersand (&) character in its label, then pressing the keyboard shortcut for the mnemonic will not activate the labeled widget. For example, if an action button is labeled I&tem, then pressing <Alt>-<t> will not perform the button action. Likewise, focus will not move to the widget associated with the label mmenonic of a group box or label.

A patch to fix this is available from technical support.

# 2

# VW 7.7 New and Enhanced Features

This section describes the major changes in this release.

## Virtual Machine

### Ephemerons, Weak Objects and Garbage Collection

In releases prior to VisualWorks 7.7, it was generally not safe to send isActiveEphemeron: or isWeakContainer: to change the ephemerality or weakness of an object while the IGC w as marking objects. For VisualWorks 7.7, the IGC has been improved to recognize this condition and treat such objects accordingly.

For the purposes of this discussion, objects are distinguished into three mutually-exclusive categories:

- Strong objects
- Weak objects: these have strong instance variables, but weak indexed slots.
- Ephemerons: their instance variables are strong, except the first one.

    a Using primitive 468, isActiveEphemeron:.

    This primitive can be used to re-arm an ephemeron, with two restrictions.

    First, do not re-arm an ephemeron that answers false to isActiveEphemeron (primitive 466) before it receives mourn. If the ephemeron is re-armed in this state, the following two ill-effects may occur, depending on the state of the IGC and the finalization queue: (1) the ephemeron may receive mourn

after it is re-armed with a strong key, leading to incorrect ephemeron behavior; (2) the ephemeron may receive mourn twice.

Second, do not change an object's class to toggle its ephemerality while the IGC is marking objects.

b The VM implementation of the isWeakContainer: primitive assumes that applications will require flipping the weakness of a comparatively large object, e.g.: a one-million slot weak array. If high performance is a concern, applications should avoid flipping the weakness of numerous, tiny weak objects while the IGC is marking.

c Due to inherent race conditions arising from the interaction between the IGC and the finalization queue, it is possible for weak objects to be added to the finalization queue more than once. The net result is that, when the second occurrence of a weak object in the finalization queue is processed, the implementation of mourn may not find any tombstoned slots.

Therefore, weak collections must provide an implementation of mourn that tolerates the lack of tombstoned slots.

In the current VM implementation of the finalization queue, ensuring there are no weak object duplicates in the queue would require an onerous amount of computation. Since weak object duplicates in the finalization queue should happen sporadically at worst, implementing mourn so that it does not fail in the absence of tombstoned slots is the more efficient way to address duplicates in the queue.

## New Mirror Primitive

Several improvements were made to numOopsNumBytes in class ObjectMemory so that it works in the presence of objects that typically do not implement messages like class or basicSize in the usual manner (e.g.: proxies). Moreover, a new mirror version primitive for nextObject was added to class ObjectMemory. Primitive 163, nextObjectAfter:, works like nextObject (primitive 531) but instead of answering the object after the receiver it answers the next object after the argument.

## Semaphore Primitive Behavior Changed

In the past, the primitive setSem:forWrite: (primitive 685) would never fail on Windows. This has been amended in VisualWorks 7.7, so that if the primitive cannot register any given I/O semaphore associated with a file or socket descriptor, the primitive will fail with an allocation error return code.

## Improved Mouse Scroll Wheel Events

Previous versions of the VisualWorks virtual machine for Windows operating systems reported the wrong value for mouse coordinates in the event array. The array is supposed to contain 4 values: x and y values of the mouse relative to the window origin, and those same x and y values, but relative to the screen. However, what the Windows VM really reported were the screen values in place of the window origin values, and where the screen values ought to have been, the result of adding the two together. The Mac OS X VM had been written to match this. The X11 VM had the correct values. Code existed in the image to extract the window coordinates from the screen slot. And specifically, to note when the VM was X11, and replace the screen values with those of the window.

In 7.7, this was all fixed. The VMs now do the correct thing, and the various "adjustments" in the image were able to be removed.

The only negative fallout of this, is that a 7.6 image running on a 7.7 VM will have the wrong understanding between the two of where the values in the event array are at. In this situation, the parcel ScrollWheelEventsFor7-7VMs may be loaded into older images to adjust the code so it is consistent with the values reported by a 7.7 VM.

## 64-Bit VM Limitations

The current 64-bit VMs do not support perm space objects. Moreover, 64-bit VMs will refuse to create images with perm space objects (the snapshot primitive, index 405, will fail with a bad arguments error). Attempting to load a 64-bit image with perm space objects, or to promote old space objects into perm space, is not currently supported.

## Large Pixmap Performance on OS X

In release 7.7 (and prior releases to some extent as well), it has been noted that drawing a pixmap in a window graphics context, is quite a bit slower on Mac OS X than other operating systems with

comparably-equipped hardware. For very small pixmaps, the speed is comparable, but as pixmaps get larger (e.g. 500 x 500 pixels), drawing speed may be as much as 50 times slower.

Drawing time is not influenced by the actual area being drawn, but rather by the base size of the Pixmap being drawn. In other words, drawing a 100 x 100 Pixmap is *not* the same thing as drawing a 100 x 100 region (e.g. through clipping) of a 500 x 500 Pixmap. The second operation will be markedly slower.

In the event of performance issues, application programmers, may need to consider their usage of any larger Pixmap in drawing operations. The use of DoubleBufferingWindowDisplayPolicy is one such case. For some applications (the IDE's Merge tool for example), usage of DoubleBufferingWindowDisplayPolicy is generally not noticable. Other applications that do many frequent updates to the window (such as the open source BottomFeeder RSS Client), may encounter a marked performance degradation.

Cincom plans to address this issue in a future release.

# Base Image

## 64-Bit Support

Beginning with release 7.7, a 64-bit version of the base image is included in the /image subdirectory. This is now the preferred way to run the 64-bit version of VisualWorks.

It is still possible to use the ImageWriter to produce 64-bit images, but Cincom recommends using the base images included in the distribution.

Note that the 64-bit VMs do not currently support the reading or writing (snapshot) of perm space objects.

Appropriate memory policy settings for 64-bit may be different from those for 32-bit. You may need to experiment with size values.

Also, there is a known issue with the shipped 64-bit images, that they have a 0.0 for the initial size of large space (see ObjectMemory sizesAtStartup). You may want to increase this, and then save and restart the image.

## Floating Point Arithmetic

With this release, VisualWorks is now able to process IEEE special values such as NaN and INF by default. This means that the IEEE Math parcel is no longer needed when such values will appear in the image, regardless of whether NaN or INF are coming from the VM when it is configured to produce such values, or from external library calls such as via DLLCC. As a result, the IEEE Math parcel is now obsolete.

Furthermore, the underlying floating point arithmetic implementation has been carefully reviewed for all platforms. To the extent that it has been tested, the virtual machines for this release have consistent behavior across all implementations, with only two platform-specific exceptions. They are:

1 On HPUX, floorLog10 of really small numbers such as 10 raisedTo: -306 may be off by one. This effect is due to the platform's math library.

2 Depending on the platform, comparison of IEEE special values may offer different answers. For example, while NaN <= NaN is usually false, the AIX platform answers true for some of these comparisons.

Note that expressions which are expected to return NaNs may answer positive or negative NaNs. While this depends on the platform, the IEEE-754 standard does not interpret the meaning of the sign of a NaN, and therefore this is deemed to be IEEE-754 conformant behavior.

## New Time Zone API: SystemTimeZone

Release 7.7 includes a new timezone class, SystemTimeZone, which invokes operating system facilities to perform its tasks. These include conversion of timestamps between local and universal time, and conversion of timestamps to/from counts of seconds since the beginning of SmalltalkEpoch (1/1/1900). Using the OS facilities implies that the SystemTimeZone mirrors the current OS time zone configuration. Consequently, using SystemTimeZone as the default timezone in VW applications eliminates the need for specific time zone configuration, the applications will simply reflect any host system changes/updates automatically. For this reason we have also changed the default timezone setup in release image to use SystemTimeZone.

SystemTimeZone also motivated some general changes in the TimeZone API. The old APIs revolved around GMT, which was reflected in many of the selectors. However, time zone facilities across our supported platforms operate in terms of UTC. The primary difference between GMT and UTC is that UTC takes into account leap seconds (a corrective measure that was started in 1972) whereas GMT does not. Consequently we have deprecated the GMT-based selectors and added replacement selectors to avoid confusion. The new selectors are also aimed at making their function clearer and more convenient, e.g. by accepting Timestamps rather than second counts. This is especially relevant since correct conversion from Timestamps to second counts is non-trivial and best left to class TimeZone in the first place.

If you have an image build script you use to set the correct time zone, you can now remove that.

### New Methods

**universalToLocal:** *aTimestamp*

Convert *aTimestamp* in UTC to a timestamp in the local time zone

**localToUniversal:** *aTimestamp*

Convert *aTimestamp* in the local timezone to a timestamp in UTC

**timestampToSeconds:** *aTimestamp*

Convert a UTC timestamp to universal seconds since the Smalltalk Epoch

**secondsToTimestamp:** *seconds*

Convert universal seconds since the Smalltalk Epoch to a UTC timestamp

**secondsFromUTC**

compute the UTC offset right now

**secondsFromUTCAtLocal:** *aTimestamp*

Compute the UTC offset at *aTimestamp* in the local time zone

### Deprecated Methods

**convertGMT:** *seconds* **do:** *aDateTimeBlock*

**convertToGMT:** *seconds* **do:** *aDateTimeBlock*

**convertGMTSecondsToLocal:** *seconds*

**convertLocalSecondsToGMT:** *seconds*

**secondsFromGMT**

**secondsFromGMTAtLocal:** *aTimestamp*

These deprecated methods are still available, though we highly recommend switching to the new API to avoid timestamp-to-seconds conversion errors. The old methods generally call the new ones described above.

Note that using the new API on the old TimeZone implementations doesn't provide UTC precision, you'll still get the imprecise GMT-based computations as before. However, they still may be useful if an application needs to perform computations in several different time zones. Depending on the operating system, the SystemTimeZone may not be flexible enough to support that.

### Incompatibilities

When using SystemTimeZone, the deprecated APIs compute using universal seconds, not GMT seconds. Conversely, when using TimeZone, the new API computes using GMT seconds. The algorithm for converting timestamps to seconds and back again is defined by the operating system facilities. It is *not* the same as the algorithm that was previously defined in Timestamp>>asSeconds. The old algorithm is still available in the old TimeZone class.

When a Timestamp is outside the range of the operating system's time zone facilities, an Error is raised when attempting to do a conversion. This does not happen when using TimeZone or CompositeTimeZone, only when using SystemTimeZone. The ranges vary between operating systems, and the details can be found below in the discussion of limitations.

Timestamp>>nowUTC sets the milliseconds on the timestamp object as well (it did not do this before). Timestamps marshaled by converting them to seconds using the old algorithm may not match universal seconds and therefore data migration may be necessary before

switching to SystemTimeZone. If possible, it is better to marshal Timestamps using their individual components, instead of converting them to seconds.

### Limitations

Operating system time zone facilities are often limited in the range of timestamps that they can accept. These limits could be an issue for an application that needs to work with timestamps out of the range of the target operating system *and* that needs to convert the timestamps between time zones (including to/from UTC). In this case, the application may need to handle the Errors raised by the SystemTimeZone and take a corrective action. Unfortunately it is difficult to provide transparent fail-over to the old TimeZone facilities in sufficiently generic manner. There is also the issue of potential precision loss in the fail-over case, which may or may not be acceptable depending on circumstances.

A list of specific limits (that we are aware of) follows:

#### All Platforms

Time zone definitions are perpetually updated and SystemTimeZone is only as precise as your underlying platform is. If time zone information of the underlying OS is not up to date then time zone conversions may also be incorrect. For example there was a recent change in US DST rules in 2007, if the OS is not up to date, then Timestamps around the DST change times after 2007 may be off by an hour.

#### MS-Windows

Timestamp years before 1601 and after 30827 are not supported by 32-bit Windows. We are not aware of any changes in 64-bit Windows.

#### Unix (32-bit)

Timestamps before December 13, 1901 20:45:52 and after January 19, 2038 3:14:07 are out of range (because the second count at the OS API level is limited to 32-bits). The upper bound, especially, may be a fairly severe limitation. Our preliminary tests on 64-bit Linux show that these limits are eliminated (the count goes to 64 bits, extending the limits to few billion years in each direction), so moving to 64-bits could be a viable work-around for some.

#### HP-UX 11, Solaris 10

SystemTimeZone needs function timegm() for its operation. This function is not available on some systems (especially older versions). SystemTimeZone fails over to an officially recommended work-around (see the man pages for details), which uses mktime() while setting the TZ variable to UTC temporarily. Since setting the TZ variable has image wide impact, this workaround is protected by a mutex at the Smalltalk level, however it still has the potential to cause trouble when concurrent threaded DLL calls are involved. This is just a conjecture at this point though and may be highly platform dependent.

#### AIX 5.1, Solaris 8

These platforms lack even the calls required to set the TZ variable, consequently the above work-around isn't applicable. In this case, we have yet another fall-back where we take a difference between a local timestamp and its UTC equivalent to compute the UTC offset and apply that manually. This may potentially miss a leap second for timestamps right around the DST change-over.

#### AIX 5.2

In addition to the 32-bit Unix limitations noted above, we have observed that this platform refused timestamps before 1970. This limitation is documented but unexplained.

## Durations

Durations have been added to the core libraries. For details, see the *Dates and Times* chapter of the *Base Libraries* guide.

## Support for Timers and Timer-Based Delays

VisualWorks 7.7 includes the new class Kernel.Timer and a timer-based implementation of class Delay. Class Timer provides optional, image-level support for operating system timers.

#### Issues with the Previous Implementation

Our classic Delay implementation employs a queue of Delay objects and realizes them one at a time using available OS facilities. The Delay that is supposed to expire earliest is taken off the queue and an OS timer is set to fire at that time. When the timer fires the process waiting on the Delay is signaled and next Delay is taken off the queue. The Delays in the queue need to be sorted according to their

expiration time, so their duration is converted to an absolute time (Time now + duration) when they are activated (Delay>>wait) so that they can be queued up properly. The expiration time is then used again when a Delay's turn comes to be realized with an OS timer.

The problem is that if there is a system clock change after the expiration time is first computed (i.e. after the Delay is queued up), the Delay will be realized with incorrect expiration time. The resulting inaccuracy of the Delay directly corresponds to the magnitude of the clock change. There's also a potential issue with the currently active Delay, because we need the underlying OS timer to cope with the clock change correctly as well. So the underlying OS facility itself needs to be able to handle the system clock change as well. To compensate for the clock change for the queued Delays we would need following bits of functionality provided by the OS:

1 Be notified when the clock changes, so that we can correct all Delays queued up before the clock change

2 Be able to determine the actual clock change so that we know how to correct the queued up Delays. Assuming the currently active timer can fire correctly despite the clock change, we could try to infer that information with some level of accuracy (modulo any timer overruns and other overhead), by comparing the actual clock value when the timer fires with the target clock value of the active Delay.

There is another issue with this approach requiring additional support from the underlying OS. When there is a new Delay scheduled which would expire before the current active Delay, the active Delay timer needs to be cancelled and the Delay re-queued. For proper re-queueing we need to be able to determine the remaining time of an active Timer. Trying to compute this value from the clock exposes us again to the same sort of "clock change" issues.

An alternative queueing strategy using relative instead of absolute times is also possible, however it still requires the ability to get the "remaining time" of an active timer. Moreover this strategy has a new problem of accumulating overrun errors with Delays that are scheduled concurrently as it is unable to detect the overruns (unless the OS provides that facility as well). The accumulated overrun clears up at any point when the queue is emptied and there's no active or scheduled Delay, but any overlap between Delays propagates and increases the error.

Most of the recent OS versions provide timing services which can cope with system clock changes however they differ in terms of capabilities provided via their respective APIs. Consequently it is difficult to correct the behavior of the Delay queues outlined above, because at least one of the necessary capabilities is missing on most of our supported platforms. Moreover some of the platforms (e.g. Windows Vista and later) provide the equivalent of the Delay queues implemented at the OS level without any of the issues discussed above.

**New Class: Kernel.Timer**

In view of the various limitations with the previous implementation of class Delay, in release 7.7 we have decided to expose the underlying OS facilities to enable their direct use. A unified protocol reflecting the functional overlap of native timers across platforms is captured by a new class, Kernel.Timer. Timer runs an action every period after an initial period of waiting. The initial period can be a timestamp (absolute timer) or a duration (relative timer). A timer action can fork blocks, resume suspended processes or signal semaphores when it fires. The metaphor is this:

- #do: creates a new Process every time the Timer fires,
- #resume: resumes an existing (presumably suspended) process and,
- #signal: resumes an existing processes waiting on a provided semaphore.

If given a repeat period (#every:) a timer will keep firing indefinitely after the initial wait period. If a timer does not repeat, it stops itself after the first iteration.

Note that active (i.e. scheduled) timers can get garbage collected if not held strongly. In this case they will be de-scheduled when they finalize. However, traditionally, active Delays were held strongly. To mimic the same behavior, Timers that are initialized with Semaphores and Processes are automatically registered to prevent their garbage collection as well. Therefore the following timer will not be reclaimed until the action is performed:

```
Timer
    after: 10 seconds
    resume: [Transcript cr; show: 'Time is up!'] newProcess
```

However, this is not the case for the more general, block-based actions. This allows to take advantage of automatic reclamation when desired.

```
| timer |
timer := Timer every: 0.2 seconds do: [Transcript nextPut: $.; flush].
3 seconds wait.
timer := nil
```

Finally, Timers can be realized using either the classic VM facilities that were traditionally used to implement Delays, or they can be realized using native OS facilities if they are available. These facilities have different and platform-specific strengths and limitations (more detailed discussion of these can be found in the class comment of TimerSystem). Both kinds of timers can be used simultaneously. The default choice can be configured on the TimerSystem, however due to the number of limitations across all supported platforms (discussed below) we are currently keeping the default set to classic facilities. Specific choice for a given timer can be forced before the timer is activated (see #useNativeInterface and #useClassicInterface).

**Summary of Changes**

- New Timer class that can run off the classic delay primitives (default) or off native OS timers (where available).
- Delay remains API compatible (with some API deprecations), however internally it was re-implemented in terms of Timers. Any extensions to this class may need to be revisited in a future release.
- New convenient Timestamp>>wait
- new subsystem, TimerSystem, replaces the DelaySystem. Subsystems should not require DelaySystem as a prerequisite anymore, or they will not activate again on image startup. TimerSystem starts *after* EarlySystemInstallation (DelaySystem used to start before).
- (private) ClassicTimerSupport implements the "classic" delay primitive based alternative to the native OSSystemSupport calls for native timers.

**Known Limitations**

A "time-shift" can occur on all operating system when the system clock changes for any reason (e.g. NTP clock adjustments). Timers created using absolute time should reflect the system correction, whereas relative timers should not. Whether the above assumptions

can be satisfied depends on the capabilities of the underlying operating system. Most recent operating systems can compensate for system-clock changes, so timers backed up by the native facilities should fire correctly. This was the primary motivation to develop the ability to use native timers.

On the other hand native timers are subject to OS limits, whereas the "classic" timers are practically unlimited (subject to available memory). Some observed, platform-specific limitations follow, however anyone considering to use native timers under strict timing requirements should consult the documentation (or the vendor) about the limitations of their specific version of the OS, hardware, etc. Our primary focus during development was Windows, Linux and OS X, however we were not able to come up with a suitable native solution for Mac OS X. It seems that at this point OS X does not officially support POSIX timers and whatever else we tried just was not good enough.

On Windows we use two separate APIs: TimerQueues on Windows XP and prior versions, and Threadpool Timers on Windows Vista and later versions. In our ad-hoc tests we were able to push both kinds of timers up to tens of thousands of concurrently active timers. However the TimerQueue timers exhibited the "overrun accumulation" error discussed above in the context of the delay queues. Consequently we don't recommend using native timers as the default timer type on Windows XP and earlier to avoid the overrun issue. The accumulated overrun error is directly proportional to the number of overlapping timers and it doesn't go away until such moment that there are no active timers. Another observed anomaly with TimerQueue timers were timer under-runs, i.e. timers firing a some millisecond earlier than they should (as far as we could measure). These were small enough to be potentially just a side-effect of timers with coarser resolution on given platform (Windows XP). The Threadpool timers don't seem to exhibit any of these problems.

On Linux we use the POSIX APIs. These timers seem to be robust however we weren't able to take it much above a thousand concurrently active timers (in both 32 and 64 bits) at least on the Red Hat (RHEL) platforms we use internally. It might be possible to tune these limits with some kernel parameters.

Other Unix platforms (Solaris, HPUX, AIX) do support POSIX timers, however they do not support callback based even delivery (they require using Unix signals instead). We intend to look into signal based solution in future releases to extend our native timer support to these platforms as well.

Furthermore, current implementation of the native timers is also affected by a VM limitation regarding concurrent foreign callbacks. Current foreign callback implementation in the VM ignores foreign callbacks if:

1 There aren't any unused thread info structures available in the VM's thread pool and

2 the internal structures used to record the foreign thread's associated Semaphore and Process objects are full and need to be extended.

In practical terms this means that you can lose foreign thread callbacks when you have many timers firing at once. This does not mean you cannot have many active timers, they just cannot all fire at once.

As a workaround, it is possible to increase the thread pool size using the existing #primGetThreadLevels, #primSetThreadLimit:lowTide: methods of ProcessorScheduler (see DLLCC documentation, page 5-15). Note however that thread slots cannot be "reserved" just for timers, they have to be shared with any other threaded activity, thapi calls, database callbacks, etc. Therefore overall thread requirements need to be considered when configuring the limits for a specific application. Another complication is that after a thread limit change the VM won't pre-allocate all the required space until it is actually needed. However the new slots can only be allocated by the main VM thread which cannot synchronize with foreign callbacks. So, to make the workaround effective, a corresponding number of real OS threads has to be actually attached to that many smalltalk processes simultaneously (Process>>attachToThread) to create the pressure to actually allocate the slots. After this initial push the processes and threads can be released.

Finally, when the image is saved with active timers in progress, the timers are susceptible to time-shifts regardless of their underlying infrastructure. This is because the time that passed between the image save and subsequent restart has to be extrapolated from the startup time and a timestamp saved during the snapshot. This elapsed down-time is then required for rescheduling of active timers

on startup. Any discrepancy caused by intervening system-clock adjustments will necessarily be reflected in the newly scheduled timers. For example if the image is saved with a timer waiting for 1 hour and the operating system clock is then shifted forward by 30 minutes, then 25 minutes later the image is resumed, the timer in question will fire in 5 minutes instead of the expected 35 minutes. Another variation of this problem is if the system clock is shifted backward by 30 minutes, the timer is not extended by an extra 30 minutes, it will continue to count down its remaining 35 minutes.

## Evaluable Symbols

Known sometimes as the "SymbolValue" add-on, VisualWorks 7.7 includes support for using unary symbols as shortcuts for BlockClosures which send the related message to a given object. Consider the following expressions:

```
(1 to: 4) select: [:each | each odd]
#('Fred' 'George' 'Ginny') collect: [:each | each size]
a := (4 + 3) ifNotNil: [:value | value negated]
```

Each block closure is of the form that it takes one argument, and the block sends a single unary message to one argument and returns that value. In this common case, these may be replaced with the symbol of the selector. The three previous expressions can now be written as:

```
(1 to: 4) select: #odd
#('Fred' 'George' 'Ginny') collect: #size
a := (4 + 3) ifNotNil: #negated
```

This change does not involve any changes to the compiler. It is simply that Symbol now implements the value: method, which allows it to be used in the same places where a BlockClosure would be sent the value: method.

Class Symbol also understands cull: for further BlockClosure compatibility, with the same interpretation as value:.

## Binary Selectors Longer than Two Characters

In conformance with the ANSI Smalltalk specification, VisualWorks 7.7 allows binary selectors of more than two characters. This can also change the interpretation of certain code constructs. The most common case where this shows up is with literal negative numbers.

For example, consider the case of:

```
3--4
```

In the current VisualWorks, this is parsed as 3 - -4, with the result of 7. But with extended binary selectors, it might be possible to define a #-- binary selector, and we have no way of telling what was intended. ANSI says that in such cases, the leading - on a negative number literal must be preceded with whitespace.

The most realistic case where this can arise is with point construction. 3--4 is unusual. 1@-1 is not. With the new system, this would need to be written with a space, as 1@ -1.

If these changes cause problems for your code, you have a number of options. One is to fix the code. Using the rewrite editor in the Refactoring Browser, you can detect all such constructs in code and have them changed automatically. In fact, simply running the formatter on code, in any version, will insert the spaces after binary selectors. An example that runs the rewrite rule is also listed below.

Another option is to define binary messages that match the possible problem selectors. For example, we can define:

```
Number>>@- aNumber
"Return a Point constructed by interpreting the receiver as the x value and
the argument aNumber as the negated y value."
    ^Point x: self y: aNumber negated
```

Now 3@-4 will be parsed as 3 @- 4, rather than 3 @ -4. But the end result will be the same.

An example to find these problems, expressed as an SUnit test is:

```
testBinaryMessageSeperation
    | aLintRule environment |
    aLintRule := Refactory.Browser.ParseTreeLintRule
            createParseTreeRule: (Array
                with: '(``@a `p: `#l) `{:node | | arg |node selector isInfix
and:ortcut[arg := node arguments first. node selectorParts first stop + 1 =
arg start and: [(arg source at: arg start) = $-]]}')
            method: false
            name: ''.
    environment := Refactory.Browser.PundleEnvironment
            onEnvironment: Refactory.Browser.BrowserEnvironment new
            pundles: (List
                with: (Store.Registry bundleNamed: '***PROJECT TOP
BUNDLE***')).
    (Refactory.Browser.SmalllintChecker new)
        rule: aLintRule;
        environment: environment;
        methodBlock: [];
        run.
```

```
"aLintRule problemCount isZero ifFalse: [aLintRule openEditor].
self assert: aLintRule problemCount isZero
```

If such methods are found, they could be manually formatted, or you could also use the System Browser's rewrite tool, with the following search rule:

```
(``@a `p: `#l) `{:node | | arg |
node selector isInfix and:  [arg := node arguments first.
 node selectorParts first stop + 1 = arg start and: [(arg source at: arg
start) = $-]]}
```

And this replace rule:

```
``@a `p: `{``@a stop + 1 = `#l parent selectorParts first start ifTrue: [
  `#l addReplacement: (RBStringReplacement
     replaceFrom: ``@a stop + 1
     to: ``@a stop
     with: ' ')].
  `#l addReplacement: (RBStringReplacement
        replaceFrom: `#l start
    to: `#l start - 1
    with: ' ').
  `#l}
```

## Deprecation Support

A framework to note deprecation of methods has been added to the core system. There is no tool support for actually deprecating methods or querying them, other than through programming them explicitly. One does so by adding:

```
self deprecated: #(...).
```

to the method, usually as the first statement. What goes in the argument array is arbitrary, but is intended to be a literal array of key-value strings. For example:

```
self deprecated: #(#version '5' #sunset '9' #use 'anotherMethod').
```

Using an array of simple values here, allows tools which search for given deprecation patterns to be written, and also allows different intents to be communicated.

What happens when a deprecated: message is sent is determined by the shared variables defined in the Deprecated class.

## Object>>initialize Added

An empty #initialize method has been added to class Object. Subclasses may now call:

```
super initialize
```

without worrying if they are the hierarchy uppermost implementation or not. Subclasses must still implement a new (or other instance creation method) which actually invokes initialize.

## ExternalReadAppendStream>>flush

ExternalReadAppendStream on sockets and pipes does not flush on read anymore.

Previously, reading from an ExternalReadAppendStream (ERAS) on a socket would first flush any pending writes into the socket before continuing with the read operation. This behavior was largely inherited from ERAS on a file where it is actually useful, to allow ERAS on a file read what it itself previously wrote. For example, if a read tries to read 4000 bytes, and there are only 1000 bytes in the file, but the write stream has 3000 bytes pending to be written, then those 3000 should be flushed so that the read can read them. But if the ERAS is on a socket, then the read and write "pipes" are completely independent, i.e. flushing 3000 bytes from the write stream will not automatically make them available to the read stream.

In fact this hidden interaction of the read and write side of the ERAS can be outright harmful in a very common socket streaming scenario where applications use separate processes for reading and writing a socket stream. Concurrent use of the ERAS often left the stream in an inconsistent state. Commonly seen symptoms of these issues were spurious data integrity failures on an SSL wrapped socket stream, caused by the SSL layer detecting the stream inconsistency thanks to its built-in integrity protection mechanisms. Another observed manifestation of this issue was spurious duplication of data written into the socket stream, caused by the reader process interrupting the writer process during a flush. Consequently the same data ends up being flushed twice, once by the reader process and once by the writer process.

Since there isn't a compelling reason to flush on read in the case of socket (or pipe) streams, this behavior was removed to make the streams behave better in the above mentioned common scenarios. Note, that this however does not make the stream "thread-safe" in an

absolute sense. An application may still need to protect an ERAS with a mutex in a more complicated scenario involving multiple-processes (e.g. multiple concurrent writer processes).

A consequence of this change is that applications now cannot rely on the socket stream being flushed by a read operation. An explicit #flush (or #close) is now required for any pending data to move from the write buffer into the socket. A missing flush is likely to manifest itself with the application getting "stuck" waiting for incoming data, simply because a previously written data that triggers the response is still sitting in the outgoing buffer waiting to be flushed.

One possible way to detect these cases is to override senders of #readsCanOverlapWrites in ERAS replacing the following line:

```
ioBuffer readsCanOverlapWrites ifTrue: [self flush].
```

with something like:

```
ioBuffer readsCanOverlapWrites
   ifTrue: [self flush]
   ifFalse: [writeStream position isZero
               ifFalse: [DebuggerService interruptAllUserProcesses]].
```

This would freeze all processes when an ERAS read (or position seek) detects un-flushed data on the write side, which might be a case of missing flush. Of course such situation isn't necessarily always an issue if there are separate processes reading and writing concurrently. You may choose a less draconian action as well, however the above might be necessary if you need to analyze a more complex situation involving multiple processes.

## BOSS and 64-Bit Support

In order to make BOSS maintenance easier, the current implementation has been heavily refactored. All application entry points into BOSS remain the same.

In addition, boss32 (version 7) has been made compatible with 64-bit images, and runs bit-identically to the old implementation running on 32 bit images.

Since 32-bit images have different SmallInteger sizes, and do not have SmallDoubles at all, these 64-bit objects must be handled with care. This is done by converting SmallDoubles and some SmallIntegers to their 32-bit Double and LargeInteger counterparts while writing. Since SmallIntegers and SmallDoubles with equal values would be

identically equal on 64-bit images, the conversion process keeps track of individual values and replaces them with a single converted value when writing 32-bit BOSS streams.

There is an analogous counterpart to this arrangement when a 64-bit image reads a 32-bit BOSS stream. Since there is a mandate that any integer that can be represented as a small integer must be expressed so, the 32-bit BOSS reader for 64-bit converts incoming LargeIntegers to SmallIntegers when appropriate. An equivalent process is applied to Doubles so that they become SmallDoubles when possible.

## BOSS Performance of Writing Single Objects

When writing one object at a time, performance can be dominated by two global object lookups for Smalltalk and Processor. A switch has been added to the class side of AbstractBinaryObjectStorage to control whether these lookups should be cached or not. The default is not to cache, which is the old behavior. The switch is called shouldUpdateRegistryObjects. Tests of this facility suggests that their run time has been reduced by a factor of two.

## Using BOSS to Write Compiled Methods

Previously, a failure to retrieve source code when writing methods with BOSS would result in notifiers from class SourceFileManager stating errors such as "the parcel folder is invalid" or "the sources file is invalid". These warnings have been turned into an exception by AR 55940, and now BOSS squelches such warnings while writing objects.

## Compiler Changes

In release 7.7, the compiler has been modified to no longer generate the short compiled code form.

Short compiled code was an optimization which allowed to encode bytecodes of a method (or a block) as one or two SmallIntegers instead of a normal ByteArray instance, assuming the entire body of the bytecodes would fit into those. There's a reasonably large number of compiled code instances in an average image that fit the criteria (on the order of tens of thousands of those). Consequently this form of encoding provided memory savings that were significant enough back when short compiled code was introduced that the cost of increased complexity in various parts of the system that this causes was worth it. With the increasing abundance of memory in today's

computers the space saving argument is growing weaker. At the same time the SmallInteger-based representation is getting more complicated with the arrival of 64-bit platforms. Therefore we decided to abandon the short compiled code form. We have done the following to achieve that:

- Converted all the existing short compiled code instances in the shipping images to normal form (using a ByteArray).
- The compiler does not produce short compiled code anymore.
- Any short compiled code stored in parcels (or binary published versions in Store) will be converted on load.

Based on some informal polls and measurements we've conducted the average increase of the memory footprint of an image seems to be somewhere between 1.5% to 2% of the original image size. It also seems that most of the increase could be clawed back by sharing identical bytecode arrays among the compiled code instances and we are considering providing the option to do so in some form in the future. In the meantime it's worth noting that RuntimePackager already provides the ability to do that as part of the packaging process.

## API for Formatting Selectors

The typo 'formating' was fixed in the following three selectors:

```
InputFieldView>>shouldUseFullFormating
PrintConverter>>hasSpecialFormating
InputFieldView>>renderTextFor:useFullFormating:
```

In release 7.7, these have been renamed as:

```
useFullFormatting:
hasSpecialFormatting
renderTextFor:useFullFormatting:
```

Application code that uses this API needs to be changed.

## Custom Memory Policies

The standard installed MemoryPolicy is set so that memory is requested from the operating system in 1 MB blocks. If an application regularly creates objects that are larger than 1 MB, it is possible for memory fragmentation to develop if an application runs over an extended period of time. The result is that the virtual machine may

refuse to instantiate a moderately large object (e.g. a 1 MB string) even though there seems to be enough free memory to allocate the object.

A custom memory policy can be made to ensure that the blocks of memory requested from the operating system are larger than the application's largest objects.

While these modifications can be made directly to the MemoryPolicy>>setDefaults method, or by executing code in a workspace, the preferred approach is to create your own subclass of MemoryPolicy that overrides setDefaults and has new settings in it after calling the super version of the method.

For details, see "Creating a Custom Memory Policy" in the VisualWorks Memory Management technical note, **`/doc/TechNotes/vwMemoryMgmt.pdf`**.

The code below is an example of setting the block size requests to be 16 MB via workspace code:

```
ObjectMemory
    installMemoryPolicy:
        (MemoryPolicy new setDefaults;
            preferredGrowthIncrement: (2 ** 24);
            growthRetryDecrement: (2 ** 15);
            freeMemoryUpperBound: (2 ** 25)).
```

The call to installMemoryPolicy: is critical to ensure that the old MemoryPolicy is removed from the system and the new one is hooked in. If you make changes to MemoryPolicy>>setDefaults, then a new instance of MemoryPolicy should be passed to ObjectMemory class>>installMemoryPolicy:. If you take the preferred approach of creating your own subclass of MemoryPolicy, then an instance of your subclass should be passed in. In the latter case, the setting of preferredGrowthIncrement, growthRetryDecrement, and freeMemoryUpperBound shown above will be unneeded if you have those values set in your subclass's implementation of setDefaults.

If testing shows that the errors go away after changing the setting of the installed MemoryPolicy, the changes should be included and installed during your application's start-up. Alternately, the changes could be installed in a base image of your application via workspace code, and the image saved.

In images that create objects of sizes less than 16 MB, it may be better to set ObjectMemory currentMemoryPolicy preferredGrowthIncrement to 16 megabytes by modifying

MemoryPolicy>>setDefaults. This change will create larger old space segments, and thus fragmentation will be addressed more effectively by the compacting garbage collection.

Unfortunately, the current memory monitoring tools make it difficult to see all the (at times subtle) interactions between the variables referenced in MemoryPolicy>>setDefaults. For instance, if preferredGrowthIncrement is 16 MB, then growthRetryDecrement should probably be larger than ten thousand. In addition, the freeMemoryUpperBound amount should be roughly twice as large as preferredGrowthIncrement. This modification makes it much more difficult for the image to enter into a tight “get more memory” then “release the memory just allocated” cycle. Avoiding this situation may be important because the memory policy could decide to do a garbage collection before each grow, resulting in stiff performance penalties. There may be other such interactions that may need to be monitored to ensure optimal performance.

For the sake of illustration, and assuming no other values need to be changed, the values in MemoryPolicy>>setDefaults could be modified like this:

```
preferredGrowthIncrement := 2 ** 24.
growthRetryDecrement := 2 ** 15.
..
freeMemoryUpperBound := preferredGrowthIncrement * 2.
```

To install the new memory policy, evaluate:

```
ObjectMemory installMemoryPolicy: MemoryPolicy new setDefaults
```

The workspace code below was used to test these new settings. Before the changes, a **No space available** exception is raised because of memory fragmentation. After the changes, the workspace code will still fail, but the cause will be because VisualWorks is truly out of memory.

There is a technical explanation after the following code that explains the test failure when the standard VisualWorks MemoryPolicy is installed:

```
mutex := Semaphore forMutualExclusion.
garbage := OrderedCollection new.
ObjectMemory garbageCollect.
maxGarbage := ObjectMemory currentMemoryPolicy
memoryUpperBound
   - ObjectMemory current oldDataBytes
   - ObjectMemory current permDataBytes
   bitShift: -20.
```

```
threadA := [
    [
        | litter |
        [garbage size > maxGarbage] whileTrue:
            [mutex critical: [garbage removeFirst]].
        litter := ByteArray new: 524288.
        mutex critical: [garbage add: litter]
    ] repeat
] forkAt: 40.
threadB := [
    [
        | litter |
        [garbage size > maxGarbage] whileTrue:
            [mutex critical: [garbage removeFirst]].
        litter := ByteArray new: 1048576.
        mutex critical: [garbage add: litter]
    ] repeat
] forkAt: 40.
threadK := [
    [mutex critical: [garbage size = maxGarbage]] whileFalse:
        [(Delay forSeconds: 1) wait].
    threadA terminate
] fork
```

Note that after a bit, threadB fails. At that point, in the particular evaluation, a compacting garbage collection was done after which there was about 200 MB of free space. Apparently, there is no 1 MB memory chunk anywhere because:

```
ObjectMemory current availableContiguousOldSpace
```

evaluated to 69592. Note that, even though:

```
ObjectMemory current oldDataBytes
```

evaluates to 337028888 (meaning about 200 MB of free space),

```
ObjectMemory current oldBytes
```

evaluates to 501426932. In other words, there is free space, it appears to be really fragmented, and the image cannot grow any further. These values can be use as a discriminating factor to diagnose memory fragmentation problems.

As an additional note, during testing you may find that:

```
ObjectMemory current oldDataBytes
```

returns a value that is smaller than:

```
ObjectMemory current availableContiguousOldSpace
```

This is because oldDataBytes answers the number of bytes used in old space, as opposed to the size of old space. For the size of old space, execute:

```
ObjectMemory current oldBytes
```

Now, as stated above, the above workspace code will always fail, but the reasons change depending on the settings of the installed MemoryPolicy. The following data was collected after evaluating the workspace. The first run used the standard VisualWorks MemoryPolicy. The second run was done after restarting the image, and then making the suggested changes to the settings of the MemoryPolicy. It was gathered after the workspace failed, but before closing the error walkback window.

Without any changes to the MemoryPolicy:

```
ObjectMemory current availableContiguousOldSpace. 69,592
ObjectMemory current oldDataBytes. 406,396,476
ObjectMemory current oldBytes. 517,991,652
```

With the suggested changes to the MemoryPolicy:

```
ObjectMemory current availableContiguousOldSpace. 413,612
ObjectMemory current oldDataBytes. 516,494,976
ObjectMemory current oldBytes. 519,015,292
```

Using the standard VisualWorks configuration, less oldDataBytes is used up, in part, because there is less availableContiguousOldSpace. This is because memory fragmentation has become a significant problem. If the suggested changes are made to the MemoryPolicy, then more of the oldDataBytes is used, and there is more availableContiguousOldSpace.

Finally, if you have not directly modified the values in MemoryPolicy>>setDefaults, you can reset the installed MemoryPolicy to use the system defaults by executing the following code:

```
ObjectMemory installMemoryPolicy: MemoryPolicy new setDefaults
```

## Resolution of Font Issues under Fedora

Fedora 11 does not, by default, install the X.org fonts, and thus doesn't have any of the fonts which VisualWorks wants to use for system fonts (in particular: Helvetica).

To resolve this issue, the 7.7 release of VisualWorks includes changes to the TextAttributes class methods, to add the name of a font that is included in the Fedora distribution. Installing the X.org fonts will provide the missing fonts as well.

## ExternalInterface Classes in Store

Release 7.7 includes a number of changes in the way ExternalInterface classes are handled by Store. Previously, ExternalInterface classes made loaded packages appear unreconciled to their versions in Store in the following cases:

- Each ExternalInterface subclass imports an associated shared variable, named by appending 'Dictionary' to the class' name. All ExternalInterface dictionaries are now imported privately on all routes of creation and loading. Previously, it was possible for the private modifier to be flipped in some cases, creating a spurious non-reconciliation.
- Previously, these shared variables were generated into the package containing the namespace of the owning class. Thus loading a package with an ExternalInterface subclass could make another package, containing the namespace of that class, look dirty. In VW7.7, these dictionaries are generated into their classes' packages.
- Once generated, these shared variables are saved to Store on publish, just like other shared variables. There is no requirement to upgrade existing code to relocate these shared variables to match the new defaults. The following remarks will help you upgrade if and when you decide to do so.
- If you have saved ExternalInterface subclasses with non-private imports of their shared variables, you will see that changed to private when you next have occasion to publish the package.
- If you have already moved a shared variable from its namespace's package to its class' package (the logical place for it) then you will no longer see spurious overrides of it between its name space's package and its class' package. (No upgrade is needed.)
- If you have previously saved a shared variable in its name space's package (the easiest option until now), then, since the name space's package is loaded before the class' package, no new shared variable is generated. There is no need for you to change any such existing code until and unless you wish to do so. At any point you can move such a shared variable from the package of its name space to the package of its class and publish.

- If the move is between two top-level packages, you can load the new versions into other images whether or not you are loading over old versions.
- If you publish a move between two packages in a bundle and you then load the new version into another image that has the old version loaded, you must load the later package, then the earlier, before loading the whole bundle. (Otherwise the atomic load of the bundle would not recognise that the delete of the shared variable in the earlier package had any obligation to relink its data to the shared variable added in the later package. Nor would it regard the class as having changed, so it would not repopulate the externals dictionary by recompiling the class' methods.)

# GUI

## MenuItem label API

MenuItem>>label is no longer constrained to return a String object, but may return a Text object, if a Text has been set as the label. This is particularly important for VisualPart subclasses that may use MenuItem instances to compose their labels, so that any emphasis provided to the MenuItem label will not be lost.

Creating MenuItems with Text labels can only be accomplished programmatically, not through the UI Painter tools.

An application which programmatically may have set Text objects as the labels of MenuItems and then expected to query the asString equivalent, would no longer get a String.

Customers should examine their code to ensure that any tests to MenuItem>>label such as

```
aMenuItem label = 'Some string'
```

are changed to

```
aMenuItem label asString = 'Some string'
```

or similar to ensure their tests continue to work. MenuItems may have their label set programmatically to an instance of Text but a MenuEditor cannot as yet open, set, or install a resource with an item containing a Text label.

## List Modernization/Improvements

A number of updates have been done to the list widget, in an attempt to "modernize" its appearance and interaction behavior:

- List widgets have had their default lineGrid increased slightly
- They now show (and allow selection of) the lower most item in the list, even if it is only partly visible due to clipping
- List widgets no longer toggle selection when selecting an already selected item. The item will remain selected unless the control key is pressed (command key on Mac OS X) when selecting, in which case it may be toggled off. This behavior does not apply to lists which have been configured using the UIPainter to disable the **Use Modifier Keys for Multi Select**
- A right-button click which opens a menu on a list, will first select the item below the mouse pointer first
- When a programatic change is made to the selection(s) of a (Multi)SelectionInList, the associated view will attempt to make at least one of the new selections visible if none currently is.

## Focused Widget Handles Shortcuts First (Not ApplicationModel)

In VisualWorks 7.6 and previous releases, it's been the case that we don't resolve "shortcut keys" in the proper order. They enter the scope of the window, and are instantly siphoned off into the menu bar. If the menu bar does not handle them, they are then passed to the current keyboard consumer.

We now believe this is incorrect (and think other widget kits bear us out, e.g <http://devworld.apple.com/documentation/Cocoa/Conceptual/EventOverview/EventArchitecture/EventArchitecture.html#//apple_ref/doc/uid/10000060i-CH3-SW11>), that the resolution goes first to the current keyboard consumer and then is passed back up the widget tree.

In 7.7, the keyboard handling has been updated to be done like mouse events. The current keyboard consumer is asked for its handlerForKeyboardEvent: (analogous to handlerForMouseEvent:), and on a non-nil return, it is asked to handleKeyboardEvent: (analagous to handleMouseEvent:). And then the menu bar (if appropriate) is negotiated the same way. Finally, so that old keyboard consumers continue to work, even though they haven't implemented a proper handlerForKeyboardEvent: yet, they are dispatched again. The very focused result is that when you press Ctrl-D or any text editor bound shortcut in the debugger, it will go to the text editor first.

As a result of this, we found that Ctrl keys really need, when being bound to a DispatchTable to include the #(#control) modifiers array.

## Rearchitect ProtocolItemNavigatorPart>>iconFor:

In release 7.7, ProtocolItemNavigatorPart>>iconFor: has been reworked to use toolListIcon and to be more pluggable.

The iconFor: method has been frequently patched by customers, in order to make their own tools work properly. To simplify this, the method has been changed to use a <method tag> based mechanism.

If your tools patch iconFor:, you may want to consider removing the patch, and using the new mechanism.

## Keyboard Shortcut Changes

In an attempt to make the VisualWorks text editors more standards compliant, some default key mappings have been changed:

- Ctrl-F is no longer mapped as "insert ifFalse:" but instead invokes the Find dialog
- Ctrl-L which used to invoke the Find dialog, no longer does
- Ctrl-T is no longer mapped as "insert ifTrue:"
- Ctrl-G is no longer mapped as "insert :=", but instead invokes the Find again operation
- Ctrl-Shift-G invokes the Find previous again operation
- Ctrl-Shift-F now inserts ifFalse:
- Ctrl-Shift-T now inserts ifTrue:

The VisualWorks distribution includes the unsupported goodie MagicKeys which you can use to add to these changes, or restore your old favourites if you wish.

## Removal of Alt/Meta Key Shortcuts for Text Editor

Text editor had default Alt and Meta key shortcut mappings which paralleled many of the common control-key commands. For example, Alt-C, Alt-X, and Alt-V were mapped the same as Ctrl-C, Ctrl-X, and Ctrl-V (copy, cut, and paste). These have been removed.

Applications that still want these extra short cuts may load the EdtingWithAltAndMetaKeys parcel to restore the functionality.

### Character Position Tab Stops

Tab stops may now be set by character position from the left margin rather than by pixel distance. Text alignment by character position within document margins is much more useful and common in typesetting.

Set character position tabs by sending the message #useTabPositions: in place of #useTabs: to the instance of TextAttributes or VariableTextAttributes used to define formatting for your ComposedText object. Similarly, if you wish to define character position tabs for a Document send #useTabPositions: in place of #useTabs:. Specify an array of character positions for each tab.

### Application Labels

Application labels may now be obtained from an instance based accessor method. If you select **Supplied by Application** in the Painter Tool and provide a selector name in the **Message** field, the content of the label will now be obtained at runtime from an instance accessor method in your application, if any is defined. If an instance accessor method is not defined the label content will be still be obtained by the usual means from a class accessor method in your application, if it is defined, a definition appearing in the UIBuilder labels dictionary, or the ApplicationModel's DefaultLabels shared variable.

## Tools

### Updated Icons

Many of the VisualWorks IDE icons have been updated to be modern alpha blended icons. Unfortunately, not everything in the IDE has been updated, but most of the more prominent tools have. Parts of the IDE that have been updates include:

- Main application icon
- Error notifier icon
- Main launcher icons (some older tools that place icons in the launcher dynamically may place older icons there)
- Code Browser
- Inspector
- Workspace

- Debugger
- Store Merge Tool
- Store Published Items
- Store Publish Dialogs
- Object icons that show up in the Inspector and Code Browser (browse the package Tools-IDE-ListIcons too see these gathered in one spot)

## Minimized Windows under OS X

On Mac OS X, windows that have been minimized to the dock no longer show a magnified application icon in the dock; rather they do the "standard" thing by showing a miniaturized version of the window contents with a small application icon superscripted to it. This is done by simply ignoring the icon installation primitive on the OS X platform. Applications that wish to have the old behavior, need to arrange for the primitive to be called. See the methods Window>>setIcon:mask: and Icon>>installOn:.

## New Prerequisites Interface

The IDE tools for viewing and manipulating package prerequisites have been completely redone. The new functionality is discussed in the “Specifying Prerequisites” section of the “Organizing Code in Store” chapter of the *Source Code Management Guide*.

Because the new IDE provides a WYSIWYG interface for manipulating component prerequisites, it exposed a number of long standing issues in the legacy model of prerequisites. Historically, these were kept in two seperate property vectors, one called #developmentPrerequisites (used when loading from Store) and #deploymentPrerequisites (used when loading via Parcel or when loading binary from Store). VisualWorks 7.7 now stores this information in a single property called #prerequisiteDescriptions. This is a sequence of arrays, where each array is 2n list of key-value pairs. This scheme allows us to specify richer information and to support future enhancements (or for customers to add their own data to a PrerequisiteDescription). First-class PrerequisiteDescription objects are synthesized from these property arrays.

This new scheme is mostly backwards compatible, with one caveat: when an older package is encountered with no #prerequisiteDescriptions, the information is inferred from the former two properties (#deploymentPrerequisites and

#developmentPrerequisites). Unless, a change is actually made to the prerequisites via the tools, no #prerequisiteDescriptions will be added. This means that pre-7.7 components with the #prerequisiteDescriptions property may safely be loaded.

When a change is made to #prerequisiteDescriptions, the old values are updated accordingly. This means that edits made to the prerequisites of a component, will be seen when loading the component into older vesions of VisualWorks.

The single caveat is that if changes are made to the prerequisites of a component using 7.7, and then the package is loaded into 7.6 or earlier and further changes to the prerequisites are made there, then the two prerequisites specifications will be out of sync.

## Inspector Enhancements (Bulletproofing and Proxies)

Various interactions between object and inspector have been refined to be safer, in particular to allow proxy objects to live more robustly within the inspector, and to avoid malformed objects from wrecking the inspector.

Much of this is accomplished via the new tool API aBlockClosure>>toolSafeIn: aDuration else: anAlternativeBlock. This method allows a BlockClosure to be evaluated with an upper limit of aDuration before it is interrupted and anAlternativeBlock's value is used. Further aBlockClosure's evaluation has all errors trapped, and for good measure, the process doing the computation is watched for what looks like infinite recursion.

## Browser Enhancements

In release 7.7, the System Browser's code editing pane includes an additional right-click menu item, **Add method**, which appears when the cursor is positioned inside the name of a method that the browser knows to be unimplemented for the receiver, and when the receiver is something that we can reasonably infer. If the receiver is a literal, or self, super, or a class name, the menu item will open a new code editing pane in the appropriate receiver to allow you to define that new method.

Note that this menu item only appears when the cursor is inside the relevant method name and not when you simply right-click on the name if the cursor is elsewhere.

## UIPainter Auto-loaded from Browser Edit Button

When a developer attempts to **Edit** a UI resource (window spec, image, menu), a confirmation dialog confirms it is **OK** to load the UIPainter package so that editing may take place, rather than simply informing them it isn't loaded.

## MiniChangeSetManager Removed as Default Item in Launcher

MiniChangeSetManager is no longer placed by default in visual.im as a dock item in the Launcher window. A parcel by the same name exists and may be loaded by those who wish to use MiniChangeSetManager as a dock item from the Launcher.

## Store Progress Dialogs

Uses of the NotifierDialog that opens to show progress during Store-related transactions have been removed. Rather than opening a separate window, an "in window" widget is now temporarily added to the originating window to give feedback to the Store load, reconcile, and publish operations.

See references to StoreProgressOverlay for examples of how to include Store feedback in your own tools.

## Initial Database Links

Our core parcels and Base VisualWorks packages no longer have database links to Cincom's internal databases embedded in them. (Some contributed parcels are supplied with links to the Cincom Open repository or other public databases, since you can access these databases and so links to them have some information value to you.)

Thus, if your process of preparing base images for your own use includes publishing the base to your local repository, the base packages in your reconciled images will show links only to your own database(s).

## Refactoring Interaction Changes

Some changes have been made to various code browser refactoring options:

- **Remove** and **Safe Remove** menu options coerced into one menu option. If the item removal has no system effects, it is removed directly. If there are references to the removed item, these are

noted in a confirmation dialog, which also allows the references to be browsed.

- Renaming a method, for which there are all multiple implementations of the same name, will now raise a confirm dialog that allows the developer to edit the changes, so that they can a) rename none (**Cancel**) b) rename all c) rename some by tuning the change set with the show changes dialog. Also, the new method name dialog, allows the developer to reorder arguments by drag and drop as well as use the arrow buttons at the right.
- **Add** and **Remove Parameter** dialogs now allow the developer to modify the new method selector. Additionally, they may reorder the arguments. They provide additional warnings and information about the target selector (such as if it affects polymorphic signatures, etc).

## Restore Original RBFormatter

In VisualWorks 7.6, the original RBFormatter was replaced as the default formatter by the RBConfigurableFormatter. 7.7 reverses this and makes the RBFormatter once again the default formatter. This formatter has been found to be nearly the same as the formatting style described by Kent Beck in "Smalltalk Best Practice Patterns". It is also faster than the configurable variant.

The RBFormatter now has some limited settings for the most commonly found formatting preferences. It allows developers to set:

- the frequency of periods (minimal, or at the end of the methods, or even at the end of blocks)
- the number of blank lines (minimal, or below method comments, or even more)
- the amount of white space (minimal, or after parentheses, or even more)

A separate parcel, RBConfigurableFormatter, provides the RBConfigurableFormatter for those wishing to use it.

## Format on View and Format on Save Options

The browser code editing panes may now be set to format on save (accept) or even format when viewing. Set the options in the Settings tool under the formatting section. This setting does not yet extend to code edits made in the debugger.

## MethodDefinition>>toolListIcons is Now Extensible

In addition to adding your own class side toolListIcon methods for your own classes, to give them icons in the code browser and inspector, you can now add icons that show up conditionally for methods in the method list pane.

Add a method to MethodDefinition which returns either nil or a VisualComponent (preferably of 16x16 size). Include the method tag <icon: N> where N is a number that gives the precedence of your conditional icon. For examples, look in the icons method category of MethodDefinition.

## Browser List of Instance Variables is Consistent with Inspector

In 7.6, the inspector was changed to show instance variables in alphabetical order, with bold emphasis applied to those values that were defined in the receiver's class. This same scheme is now used in the browser for the Inst Var list.

## One-Shot Breakpoints

There is now a menu option to add "single shot" breakpoints. These act as a breakpoint on the first invocation, and then return to normal code execution. They remain inserted after the first invocation, but are disabled. You may reenable them for further single shot breakpoints.

Customers who use the ExtraEmphasis goodie will see new icons for breakpoints as well.

## Refactoring Browser Menu Caching Removed

Refactoring Browser menus are always computed dynamically now. Tools that extend those menus no longer need to send flushMenus. Additionally, 20+ browser menu actions have been moved from menu specs to tagged method implementations.

## Changes to Browser Package Name Annotations

Package name annotations in the System Browser such as **+**, **-**, and ***** have been removed. The **+** annotation on the package icon shows if the package (or bundle) has changes that would be lost if it is not published in one or more repositories, regardless of which repositories it is reconciled against.

Package names are also annotated with a number (displayed in orange) indicating a change count. That is, the sum of "deltas" against the original version, where a delta can be a change, an add, a removal, a rename, etc.

## Better Detection of Embedded URLs in Text Editors

Various HTTP escape sequences are now better handled by the automatic link highlighter.

## Inspector IEEE Floating Point Decomposition

Inspectors of floating point objects (Float, Double, SmallDouble), now have a pseudo attribute that shows the IEEE-754 fraction decomposition of the receiver.

## Source Files Manager

In the past, class SourceFileManager could open warning windows when either parcels or the source file could not be accessed. In release 7.7, this mechanism has been changed so that the warnings are opened via a new subclass of Warning, SourceFileManagerWarning. BOSS has been changed to handle this warning so that writing compiled methods for which no source code is available does not result in several warnings coming up.

## Known Limitations of the ImageWriter

Beginning with release 7.7, a 64-bit version of the base image is included in the /image subdirectory of the standard distribution. This is now the preferred way to run the 64-bit version of VisualWorks.

Using the ImageWriter, it is also possible to convert an existing 32-bit into a 64-bit image. Currently, this process has a number of limitations, including:

- Object identity is subject to change. For example, Doubles may become SmallDoubles and LargeIntegers may become SmallIntegers.
- Object hashing is subject to change too. For example, a set of large integers may become a set of small integers. Since SmallInteger>>hash returns self, then such collections need to be rehashed. This is done during image startup, but reading collections via BOSS may require special handling. Furthermore, instances of HandleRegistry are assumed to contain objects the

hash values of which do not depend on identityHash values being constant across the conversion.

Currently, the ImageWriter will not rehash instances of HandleRegistry or any of its subclasses. This means that a newly created 64-bit image will have to be able to clean out any of the handle registries upon startup. Since HandleRegistry is a hashed collection based on equality, and since identityHash changes between 32- and 64-bit images, leaving handle registries as they were on a 32-bit image assumes that no handles are sending the message identityHash for the purposes of calculating their hash values.

## Advanced Tools Profilers and Stack Spills

The profilers in the Advanced Tools parcels have been updated to provide statistics about the number of stack spills observed during execution.

To explain, recall that the VisualWorks' VM uses a native stack to execute Smalltalk processes. When the space allocated to this stack runs out, however, the VM allocates stack frames inside the image as would happen in Smalltalk-80. This is not done in order to continue execution from the image space. Instead, the goal is to make room in the native stack space to resume execution there.

When native stack spills occur frequently, it leads to significant inefficiencies because allocation of execution frames is very expensive compared to allocation in the native stack space. Furthermore, an additional side effect is that when numerous stack spills occur the VM will have to garbage collect the many allocated stack frames when they become unused.

The amount of memory the VM will allocate to the native stack can be tuned at startup. This is done via a float multiplier stored at index 4 of this array:

```
ObjectMemory sizesAtStartup
```

The default value is 1.0, meaning 20kb. However, measurement results obtained with an SUnit benchmark suite indicate that, for processes that have an average stack depth of between 20 and 50, it is much better to set this multiplier to the maximum amount of concurrent Smalltalk processes // 5.

In other words, if you have 1000 concurrent processes that are expected to execute simultaneously, the native stack size multiplier should be set to 200.0. To make the change become effective, evaluate:

```
ObjectMemory sizesAtStartup: modifiedArray
```

then save the image and restart it. The performance improvement factor can easily reach 10x. However, the multiplier should not be increased too liberally because it will slowly become counterproductive due to the management associated with it.

As always, measuring is a good thing. To make the evaluation of this multiplier easier, the profilers in the Advanced Tools parcels have been updated to notify the number of stack spills observed during the measurement. Note that a few spills are generally harmless, and that it is their excessive occurrence what harms performance the most.

Two other figures are also available in the profilers. The first one is the count of mark stack overflows. This informs the number of times that the new space was not large enough to hold the mark stack during a mark and sweep garbage collection pass. When this happens, the garbage collection goes through the equivalent of finishing a portion of the job and then restarting from where it left. Depending on the application this can lead to slower garbage collection operation. If justified by measurements, the size of the new space should be adjusted by increasing indices 1 and 2 of the sizesAtStartup array.

Finally, there is also a count for weak object list overflows. When these occur, the native code cache space is not enough to hold weak objects and ephemerons during a mark and sweep garbage collection pass, and as a result some of them will not be queued for finalization. The value of this figure is the number of objects that could not be added to the list, and it can be used to drive the tuning of the native code space size when necessary.

## Menu UI Compatibility

In VisualWorks 7.3, an optional MenuUICompatibility parcel was introduced to address incompatibilities of VisualWorks menu bar menus with host OS menu feel requirements. With release 7.7, this functionality has been integrated and the parcel removed.

The compatible behavior integrated from MenuUICompatibility includes:

- On Windows, a mouse drag (i.e., mouse button hold and move) to a menu item with a submenu will not close the menu or submenu upon button release.
- On Windows only, menus highlight disabled menu items; other platforms skip past disabled menu items entirely.
- MacOS X menus will not wrap highlight of menu items for key up/down navigation.
- Unix platforms do not highlight menu items while moving the mouse cursor over an item; the mouse button must be down to do this.
- Except on Unix, submenus open after about a 0.3 second delay when the mouse cursor rests on their parent item.
- For the Motif and Windows look, a menu and its submenu should remain open after a mouse click and release on a menu item that opens a submenu.
- On Mac platforms, moving the mouse outside a menu to the right should not close all submenus open.
- On Mac OS X, <Tab> and <Shift><Tab> navigate and open menus right and left in the menu bar.
- On Mac OS X, a menu and all its submenus will not close prematurely if the up or down arrow keys are used to navigate to a menu item with an unopened submenu.
- For MS Windows and Motif menus, a menu item that opens a submenu cannot become a selection itself and then close; only a menu item without a submenu may be the menu selection.

# Database

## Oracle EXDI: Statement Caching

The OracleEXDI now supports statement caching. This feature is useful when you execute the same SQL statement multiple times. When statement caching is enabled, an existing prepared statement handle will be reused, thus improving performance.

By default, statement caching is disabled. This feature was added in Oracle 9.0.0 and later. The VisualWorks EXDI for Oracle is compatible with both pre-9.0.0 versions and later.

To enable and use statement caching:

```
| conn sess |
conn := OracleConnection new.
“Check to see whether statement caching is supported by the installed
Oracle client.”
conn supportStatementCaching
    ifFalse: [self error: 'Does not support caching.'].
“Enable statement caching.”
conn useStatementCaching: true.
conn environment: 'oracleDB';
    username: 'username';
    connect: 'password'.
“Set the size of the statement cache.
conn setStatementCacheSize: 30.
sess := conn getSession.
 1 to: 20 do: [:i|
    sess prepare: 'INSERT INTO testtb VALUES( ?, ''test'' )'.
    sess bindInput: (Array with: i).
    sess execute.
    ansStrm := sess answer].
```

## Oracle EXDI: Adjustable Buffering for LOBs

When reading/writing LOBs, the size of the buffer is now settable from an instance of OracleSession. You can adjust the buffer size based upon the lengths of LOB you are handling.

For example:

```
| conn sess |
conn := OracleConnection new.
conn username: 'userid';
    password: 'pwd';
    environment: 'OracleDB'.
conn connect.
conn begin.
sess := conn getSession.
"Better set this, otherwise the returned LOBs will have the default size:
4000 bytes."
sess defaultDisplayLobSize: 1130729.
"Use the default lobBufferSize 32768."
sess answerLobAsValue.
sess prepare: 'select * from testlob'.
sess execute.
ansStrm := sess answer upToEnd.
"Increase the buffer size, it will improve performance for Large LOBs."
sess lobBufferSize: 524288.
```

```
sess prepare: 'select * from testlob'.
sess execute.
ansStrm := sess answer upToEnd.
sess disconnect.

conn rollback.
```

## Support for OEM Encoding

In release 7.7, support for OEM code pages has been included in the Oracle, ODBC, MySQL, and CTLib EXDIs. These code pages are most often used under MS-DOS-like operating systems. Examples include: 437 - The original IBM PC code page; 737 - Greek; and 775 - Estonian, Lithuanian and Latvian.

OEM encoding must be explicitly enabledvia the connection object. Thus, this addition to the Oracle EXDI has no effect on existing customer code.

For example, to use OEM encoding with an Oracle database:

```
| conn sess |
conn := OracleConnection new.
conn username: 'username';
  password: 'password';
  environment: 'env'.
conn connect.
"Drop the test table."
sess := conn getSession.
sess prepare: 'drop table test_umlauts';
   execute;
   answer.
"Create a test table."
sess prepare: 'create table test_umlauts (cid number, cname
varchar2(50))';
   execute;
  answer.

"Turn on OEM encoding."
conn turnOnOEMEncoding.

"Insert test data using OEM encoding."
sess prepare: 'insert into test_umlauts values(10, ''ä, ö, and ü'')';
   execute;
  answer.

"Turn off OEM encoding."
```

```
conn turnOffOEMEncoding.

"Insert test data using normal encoding based upon NLS_LANG."
sess prepare: 'insert into test_umlauts values(11, ''ä, ö, and ü'')';
  execute;
 answer.

"Turn on OEM encoding."
conn turnOnOEMEncoding.

"Retrieve the test data, record with cid=10 will be displayed correctly."
sess prepare: 'select * from test_umlauts' ;
  execute.
ans := sess answer.
ans upToEnd.

"Turn off OEM encoding."
conn turnOffOEMEncoding.

"Retrieve the test data, record with cid=11 will be displayed correctly."
sess prepare: 'select * from test_umlauts' ;
  execute.
ans := sess answer.
ans upToEnd.
```

## ODBC EXDI: Enhanced Data Type Support

In release 7.7, the new SQL Server data types varchar(max), nvarchar(max) and varbinary(max) are supported.

## ODBC EXDI: Improved Connection Reliability

In this release, the handling (setting/getting) of connection options is improved.

## ODBC EXDI: Support for Multiple Active Result Sets (MARS)

When using older SQL Server ODBC drivers, sharing a connection among multiple sessions has long been an issue. Attempting to do this causes problems for multi-thread applications.

The native SQL client provided with SQL Server 2005 seems to provide a way to get around it: MARS (Multiple Active Result Sets).

MARS is available for use if you set a connection attribute SQL_COPT_SS_MARS_ENABLED to be SQL_MARS_ENABLED_YES. Note that attribute must be set before connecting to a data source.

With a multi-threaded (multi-session) application that shares the same connection, MARS is supported through interleaving. With multiple connections and a single session on each connection, MARS is supported via parallel execution.

For example:

```
| sem rc xif conn b1 b2 b3 |
"Multiple sessions sharing one connection."
sem:= Semaphore forMutualExclusion.
conn := ODBCThreadedConnection new.
"Getting the interface."
xif := conn class xif.
"Setting the attribue before connecting."
rc := conn setConnectionOption: #SQL_COPT_SS_MARS_ENABLED
value: (xif SQL_MARS_ENABLED_YES).
"Getting the attribue to verify."
rc := conn getConnectionOption: #SQL_COPT_SS_MARS_ENABLED.
"Connecting to a data source."
conn username: 'username';
   password: 'password';
   environment: 'DSN'.
conn connect.
"Creating a Block."
aBlock := [:tableName || sess ansStrm |
sess := conn getSession.
sess prepare: 'select * from ', tableName.
sess execute.
ansStrm := sess answer.
(ansStrm == #noMoreAnswers) ifFalse: [
[ansStrm atEnd] whileFalse: [ |row|
row := ansStrm next.
sem critical:
   [Transcript show: tableName,': '.
   Transcript show: row printString; cr.]]].
].
"Run the Block in multiple threads."
b1 := aBlock newProcessWithArguments: #('dept').
b2 := aBlock newProcessWithArguments: #('emp').
b3 := aBlock newProcessWithArguments: #('curtest').
b1 priority: 30.
b2 priority: 30.
b3 priority: 30.
b1 resume.
b2 resume.
b3 resume.
```

## MySQL EXDI: Refactored Connection and Session Classes

Prior to VisualWorks 7.7, there were 3 connection classes in the MySQL EXDI: MySQLAdminConnection, MySQLHybridConnection and MySQLConnection, and 3 session classes: MySQLAdminSession, MySQLHybridSession and MySQLSession.

With the current release, these have been refactored such that there are now only one of each: MySQLConnection and MySQLSession. This is simpler and easier to use; it is also consistent with the architecture of the other EXDIs.

## DB2 EXDI: Default LOB Size can be Specified

Prior to release 7.7, the DB2 EXDI used a shared variable to specify the size to retrieve LOB values, and the default value was 16M. There were several problems with this implementation:

1 The default value is too big. The ideal way to handle LOBs is to use their locators. We can set the default size to 4000 bytes as we did to OracleEXDI, users can reset it to a bigger value based upon their needs.

2 Once the shared variable is reset, it will affect all of the sessions, it will be better if each session can set its own retrieval LOB size.

3 In a multi-thread environment, resetting the shared variable may cause problems if without protection.

After the change, default LOB size can be set for each session. The following is an example of usage:

```
| connection session ans res clob blob clobLength blobLength |
connection := DB2Connection new environment: 'env';
        username: 'username';
        password: 'pwd';
        connect.

"Create the test table."
session := connection getSession.
session prepare: 'CREATE TABLE TestLob (a CLOB(2m), b BLOB(2m), c
INT)'.
session execute.
session answer.

"Input test data."
connection begin.
session := connection getSession.
session prepare: 'INSERT INTO TestLob (a, b, c) VALUES ( ?, ?,
```

```
?)'.clobLength := 1130720.
blobLength := 1130720.
clob := String new: clobLength withAll: $a.
blob := ByteArray new: blobLength withAll: 1.
session bindInput: (Array with: clob readStream with: blob readStream
with: 2).
session execute.
session answer.
connection commit.

connection begin.

"Retrieve 4000 bytes of data."
session := connection getSession.
session answerLOBAsValues.
session prepare: 'select * from TestLob'.
session execute.
ans := session answer.
res := ans upToEnd.
clob := (res at: 1) at: 1.
clob size.

blob := (res at: 1) at: 2.
blob size.

"Retrieve the values of the whole LOBs."
session1 := connection getSession.
session1 maxLongData: 2000000.
session1 answerLOBAsValues.
session1 prepare: 'select * from TestLob'.
session1 execute.
ans := session1 answer.
res := ans upToEnd.
clob := (res at: 1) at: 1.
clob size.

blob := (res at: 1) at: 2.
blob size.

connection rollback.
```

## DB2 EXDI: Fetch Multiple LOBs in One Execution

In release 7.7, it is now possible to retrieve multiple LOB files at once.

For example, let's assume that we have a folder called Test, which contains a text file called LOBFileReference.test and a binary file calc.exe which is the Windows calculator. In the example code shown below,

these two files are inserted into the LOB columns of a DB2 table, and then retrieved to the same folder using different file names. By comparing the files, you will find that the retrieved files are identical to the ones we inserted.

```
| aConnection aSession clobFileRef blobFileRef |
aConnection := DB2Connection new environment: 'env';
username: 'username';
password: 'pwd';
connect.
aSession := aConnection getSession.

"Create a test table."
aSession prepare: 'CREATE TABLE TestLOBFileRef (a CLOB(32k), b
BLOB(1M), c INT)'.
aSession execute.
aSession answer.

"Inserting the files into the test table."
aConnection begin.
aSession prepare:'INSERT INTO TestLOBFileRef (a, b, c) VALUES (?, ?,
?)'.
clobFileRef := DB2LOBFileReference forCLOB:
'C:\Test\LOBFileReference.test'.
blobFileRef := DB2LOBFileReference forBLOB: 'C:\Test\calc.exe'.
aSession bindInput: (Array with: clobFileRef with: blobFileRef with: 1).
aSession execute.
aSession answer.
aConnection commit.

"Retrieve the LOB files to the same folder using different file names."
aConnection begin.
clobFileRef := (DB2LOBFileReference for:
'C:\Test\LOBFileRefOutputFile.test') overwriteFile.
blobFileRef := (DB2LOBFileReference for: 'C:\Test\calcOutput.exe')
overwriteFile.
aSession := aConnection getSession.
answer := aSession prepare: 'select a, b from TestLOBFileRef where c=1';
answerLOBAsFileRef: (Array with: clobFileRef with: blobFileRef);
execute;
answer.
answer upToEnd.
aConnection rollback.
```

# Store

VisualWorks 7.7 includes the first phase of Store modernization. The main goal for the VisualWorks engineering team with regard to Store in release 7.7 was to make it faster and amenable to future extension.

We researched many of the possible ways of accomplishing this, and in doing so, realized that a fundamental problem was that Store had created its own database interface built directly upon EXDI, making it impossible to take advantage of any database specific optimizations while remaining portable across database platforms.

Instead of trying to create a more abstract, optimization capable interface, we chose to re-host the Store database interface on the existing open source Glorp framework. Glorp already knows how to do optimized queries and supports a large number of databases. This decision brought with it the advantages of Glorp, and as it continues to grow, Glorp support of underlying database specific optimizations that it adopts are automatically gained by our Store interface.

Fortunately, the Glorp community had already done a large amount of preliminary work to interface Store to Glorp. That work was the starting point of our Store work.

In release 7.7, we have exploited the two most important speed up opportunities enabled by Glorp: reading/loading code from a repository, and writing to a repository.

By basing our Loading system on Glorp we were able to add a project that was already underway: the "atomic" loading of code, which is already a feature of parcels. Thus, we took an already well-understood and progressing atomic loading system and added Glorp access to it. The result is the new Store/Glorp Atomic Loader, which we refer to simply as the *Atomic loader*.

VisualWorks 7.7 includes new settings for Store, that allow you to control and even turn off the Atomic loader. By default it is turned on. If you turn it off, you will then invoke the previous loader, which does not use Glorp, nor have any optimizations. Internally, the whole VisualWorks engineering team has used the Atomic loader since January of 2009. In our and our testers' experience, this new loader can actually load code that the old loader could not. Of course, it doesn't hurt that it will do so faster than its predecessor.

For the general Store user moving to 7.7, the difference under the hood for Store access will be invisible. For those users that have extended Store in ways that have changed the underlying Store schema, the code that they use will need to be ported to the new Store code base so it can take advantage of Glorp. For those who may have hand coded non-schema based optimizations, we suggest using the new loader. We expect that such optimization changes may no longer be needed.

## A Bit of Detail

For those who want to understand how the Atomic loader works, the following discussion may be of interest.

The old loader simply took an ordered, one-thing-at-a-time approach to loading code from a repository. First, for a given package, either standalone or in a bundle, it would gather up any name spaces, compile them directly into the image, then move on to classes, then on to shared variables, and finally methods. Each individual item changed the image as soon as it was brought in. If the loader had a problem resolving something, it pretty much gave up, often leaving your image in an intermediate and barely useful state.

The Atomic loader instead takes the approach of splitting the compiling and installing of code into two distinct phases: compile, and load.

In the first phase, compiling, the Atomic Loader compiles units (packages or bundles) of code into what we have come to call a *shadow environment*. While it still follows the same order of battle as the old loader, by putting compiled objects into a shadow environment, no meaningless or uncompilable code becomes the current definition of any class, method, shared, or name space in your image.

Once the loader has successfully compiled all the code, it then goes into the second phase: installing. The loader iterates over the shadow environment, installing the code in an ordered fashion from the shadow environment into the image, which for contrast with shadow, we sometimes call sunshine. The biggest gain we get here is if the loader determines that it can not cleanly install *all* the code in your load request, it will (optionally, under settings control) ask you if you want it to continue. This gives you the opportunity to say "no, don't possibly corrupt the image" with the problem code. New settings allow you to choose to be notified, to just always go ahead, or abort the load without changing the image.

## Atomic Loading and Early Install

The new loader often can load a whole bundle atomically. There are some things that you should be aware of with regard to the "atomicity" of the new Atomic Loader.

Even using the old loader, some considerations were needed. For instance, if you created a new parser or scanner using the AT Parser Scanner utility, the package in which your new scanner or parser is defined must be *separate from* a package that has code that uses it. This has not changed for the Atomic Loader. In fact, when the Atomic Loader sees such code, it will compile it, but before continuing with other dependent packages, it does what we call an *early install*. When this happens, it finishes compiling the package, then goes into an install phase for everything compiled but that hasn't been installed up until then, including the package that triggered the early install. It then returns to Atomic behavior for any remaining packages or bundles.

Why is this being done? Well, our shadow environment is not what is commonly called a *sandbox*. It is *not* possible without a sandbox to execute code that lives in the shadow environment. In a shadow environment we can do a lot of cool things, such as looking up and reflecting upon objects in the shadow environment. A sandbox provides an independent execution environment for objects in a shadow world. Without the addition of a sandbox though, we can not ask any of our objects in the shadow environment to execute. Creating a sandbox for VisualWorks is an interesting idea, but way beyond the scope of the Store loading system and therefore the early install behavior became an essential behavior of the Atomic Loader.

Besides the parser or scanner situation just described, there are a few other cases that can cause an early install.

Parcels are loaded in their own separate Atomic fashion. Any parcel which might be a prerequisite of a bundle or package will cause an early install of any package or bundle prerequisite that has up to that point been compiled but not yet loaded.

A package that is saved binary will cause an early install when it is found as part of a bundle or as a prerequisite package. In effect, a package saved binary is loaded by Store using the parcel loader.

Finally, if you add an extension method to ExternalInterface, we detect that fact and do an early install. Note that if you need to add a method directly to the ExternalInterface class instead of on one of your own

subclasses, it needs to be in a package that *does not* have code that uses the method you are adding. Again, without a sandbox, one can not execute code in the shadow environment.

If you encounter a situation where you believe that you need to force an early install, we hope you will document it and send us that information. You aren't stuck though. You can force any package or bundle to install early *by adding a property to that bundle or package* named: #installBeforeContinuing. The value for that property is ignored by the loader, the presence of the property is all that matters. Thus, it is *not dynamic based on the value of the property*. If the property is there, early install will take place. The value of the property, if any, is completely ignored. You might choose to make it a string with a brief description of why it is needed, and to be sure to remove the property should it no longer be required.

## The Future Looks Bright

In upcoming releases of VisualWorks, we will extend the use of Glorp objects until all uses of the old hand-crafted Store database objects are no longer being made.

One of the things we have planned for the next release is the ability for the loader to, if needed, load objects that are in the shadow environment in an order different from that in which they were seen. Where both the old loader and current Atomic Loader installs one package at a time, thus making a bundle's structure into a hard-coded prerequisite order, in the future the loader will detect "out of order" objects in the shadow world, and install them without regard to the order of packages in the bundle, but of course into the correct packages. This will remove the need to validate the order of packages and sub-bundles within an atomically loadable bundle.

Also planned for the next release is a whole new package and bundle comparison interface using only Glorp-based Store objects. Faster meaner and leaner, you will also be able to browse repository packages and bundle versions using a new Refactoring Browser based user interface.

The old hand-crafted Store database classes will not be removed from the system until after all existing Store behaviors are moved to Glorp over upcoming releases of VisualWorks. There will be plenty of time for you to migrate any extensions of the old system to the new Glorp-based design, but *you are encouraged to begin to do so now*.

One of the existing features of Glorp we will take advantage of in the future is its ability to detect schema versions, and fallback to old schema behavior or, on command, migrate a database to a new schema, all the while with careful coding on our part to not prevent existing users with non-Glorp images from accessing and updating a Store database which may or may not have schema changes.

Once the whole Store system is moved to Glorp, we intend to move Store forward as a whole, incorporating robust improvements such as Configuration and Security systems. We fully plan to implement these systems as extensions, allowing the detection and fallback to the current schema without adversely affecting users who choose to take their time updating, or who may choose not to use such systems at all.

## Repository Indexing

With this release, Store now automatically creates indexes for all appropriate tables in the Store schema when a new Store database is created.

In order to install these indexes in an existing Store database, first be sure that any existing database indexes on all of the Store tables are removed. You may need to have your database administrator do this for you. Then you (or your database administrator) can execute:

```
Store.DbRegistry update77
```

This will then create the new indexes.

If you have existing indexes in your Store database, and do not remove the existing indexes before executing this update, you are likely to create duplicates, with different names, but possibly the same index values. This isn't harmful, but it may reduce the effectiveness of the new indexes.

## Table Spaces Settings

Table Spaces are now settings instead of being hard coded. Changing these should only be done before creating the physical database table spaces. Changing them after the actual tables and spaces are created will cause data to be unavailable.

Note also that this information is global and is not tied to a single database connection. Thus changing these values will cause the system to expect the same table space names to be used across all connections.

### Settings Reorganized

In this release, a number of elements on the Store **Settings** page have been changed:

- We removed the **StoreII** page.
- We removed the setting (but not the code) for **Debug Store II**
- We renamed the **Connection** page to **Loading Policies**
- We moved the **Load Atomic** setting from the main Store page to the **Loading Policies** sub page
- We moved the **Reconnect After Start** setting to the main Store page
- We made the main Store page show as Radio buttons instead of a Drop Down List
- We moved the **Create Overrides in Bundles** setting to the **Loading Policies** sub page
- We moved the **How Should Load Errors be Handled** setting to the **Loading Policies** sub page.
- We moved the **Copy Load Transcript** setting to the **Loading Policies** sub page.
- We moved the **Allow Binary Loading** setting to the **Loading Policies** sub page.

## WebServices

### WSDL: Support for Empty <import/> Elements

According to XML Schema (http://www.w3.org/TR/xmlschema-0/#import) both the namespace and schemaLocation attributes in an <import> element are optional. This is why a bare <import /> information item is allowed. This simply allows unqualified reference to foreign components with no target namespace without giving any hints as to where to find them. It is up to the application to interpret those references however it sees fit. Consequently our implementation interprets these references as ones to be resolved in any accompanying schemas that do not specify their target namespace.

We will collect all schemas without the target namespace in a WSDL document and use those to resolve unqualified types in schemas with <import> element without namespace declaration. We will create

XMLObjectBindings from these schemas with target namespace set to nil, but these bindings won't be registered in the global XMLObjectBinding registry. However these bindings will be part of the document's #importedBindings. Unqualified references will be resolved using the first binding with unspecified target namespace from the #importedBindings collection that has a node with the same type as the type of the unqualified reference.

## Type Validation for Serialization and Deserialization Blocks

During the process of encoding/decoding, a value may fail some simple validation rules. In this event, the following resumable exceptions are now raised.

### XMLDecodingError

Raised while decoding XML strings into objects (simple types: arithmetic, boolean, date, time, url, byte) if there is a problem to decode an XML string into a corresponding Smalltalk object.

### XMLEncodingError

Raised while encoding objects (simple types: arithmetic, boolean, date, time, URL, byte) into XML strings if an object type is different than an XML schema type.

### DecodedIntegerOutOfRange

Raised if an XML string decoded into an integer that is out of the expected XML datatype range. The default action for DecodedIntegerOutOfRange returns a parameter which is an XML string with encoded integer value and the decoded integer value.

### EncodingIntegerOutOfRange

Raised if the integer value to be encoded is outside of the range of the expected XML datatype. The default action for EncodingIntegerOutOfRange returns an integer encoded as an XML string.

### DecodedInvalidString

Raised if a decoded string has invalid characters (violating the restrictions of the XML string type). The error is resumable and the default action returns the XML string as it was received.

**EncodingInvalidString**

Raised if a string to be encoded includes invalid characters (violating the restrictions of the XML string datatype). The error is resumable and expects a corrected string as the resumption value.

## Support for XML Union Types

The 7.7 release includes support for creating XML to Object binding specifications for <xsd:union> elements, and for marshaling/ unmarshaling the union elements in to a proper simple type.

The UnionMarshaler holds union member type marshalers, and tries to use them in the order in which they appear in the definition until a match is found. The evaluation order can be overridden by using xsi:type.

Example:

```
schema := '<?xml version="1.0" encoding="utf-8" ?>
 <xsd:types xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
<xsd:schema xmlns:s0="http://www.xignite.com/services/union"
targetNamespace="http://www.xignite.com/services/union">
  <xsd:element name="Coat">
  <xsd:complexType>
   <xsd:sequence>
   <xsd:element name="size" type="s0:size" />
   <xsd:element name="color" type="xsd:string" />
   </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
 <xsd:simpleType name="size">
  <xsd:union>
  <xsd:simpleType>
          <xsd:restriction base="xsd:integer"/></xsd:simpleType>
    <xsd:simpleType>
    <xsd:restriction base="xsd:token">
     <xsd:enumeration value="small"/>
    <xsd:enumeration value="medium"/>
     <xsd:enumeration value="large"/>
   </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
 </xsd:simpleType>
</xsd:schema>
</xsd:types>'.
```

To create a XML to Object binding:

```
bindingSpec := XMLTypesParser readFrom: schema readStream.
```

The binding specification will include union type as:

```
'...
<union name="size">
      <simple baseType="xsd:integer"/>
      <simple baseType="xsd:token">
         <enumeration value="small"/>
         <enumeration value="medium"/>
         <enumeration value="large"/>
      </simple>
   </union>
..'.
binding := BindingBuilder
            loadFrom: bindingSpec canonicalPrintString readStream.
```

To marshal an object with a union type:

```
manager := XMLObjectMarshalingManager on: binding.
manager useInlineType: true.
coatMarshaler := b marshalers
   detect: [:mx | mx tag type = 'Coat'] ifNone: [self assert: false].
struct := Struct new.
struct
   size: 6;
   color: 'red'.
xml := manager marshal: struct with: coatMarshaler.
xml:
'<ns:Coat xsi:type="ns:Coat" xmlns:ns="http://www.xignite.com/services/
union" xmlns:ns0="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <ns:size xsi:type="ns0:integer">6</ns:size>
   <ns:color xsi:type="ns0:string">red</ns:color>
</ns:Coat>'
```

To unmarshal XML into an object:

```
object := manager unmarshal: xml with: coatMarshaler.
```

## Internationalization

VisualWorks 7.7 includes an extensive Locale system based upon CLDR (Unicode Consortium's Common Locale Data Repository) locales. From the set of 15 locales included with VisualWorks 7.6 (the so-called "legacy locales"), coverage has been extended to include

more than 400 locales based upon data contained in CLDR 1.6.1 (the "CLDR locales"). In addition, VisualWorks 7.7 includes greatly expanded Unicode support, especially for Windows platforms.

## Enhanced UTF-8 Support

VisualWorks 7.7 includes enhanced support for UTF-8 locales and character input.

However, some limitations still exist with copy and paste on Linux platforms. More specifically, the VisualWorks paste buffer is now in 'default' encoding, but the VM still treats it as ISO-8859-1 regardless of the current locale setting. This can cause problems when pasting non-ISO-8859-1 text to another application or vice versa, unless VisualWorks is running in an ISO-8859-1 locale (which is now pretty uncommon).

## CLDR-based Locales

In VisualWorks 7.7, care has been taken to allow both the CLDR and legacy locale systems to exist in the same image, with the caveat that only one of them may be active at a time. By default, the CLDR Locales are active in the 7.7 image.

### Using the Legacy Locale System

The general steps are: change Locale.LocalePoolType. Set its value to be #vwlegacy to use the old locales, and #vwcldr to use the new locales. For example:

```
Locale.LocalePoolType := #vwlegacy.
Locale initLocaleSubsystems.
Locale initialHookup.
```

Note: this is a fairly invasive change to the running VisualWorks image. It is not recommended that applications do this dynamically as part of normal operations. Rather, it is suggested that the image be prepared and saved with the needed locale system active and initialized.

### Locale API

Legacy locales are represented by instances of class Locale, while CLDR locales are represented by instances of class CompositeLocale (which themselves hold instances of LocaleLocalizationComponent and LocaleEncodingComponent).

To make the transition between legacy and CLDR locales easier, the class side of Locale remains the point to which to direct messages that address the general locale API. When using CLDR locales, messages are forwarded by class Locale to class CompositeLocale for execution. Some deprecated methods (see below) answer self shouldNotImplement in a CLDR locale environment.

The following Locale class methods are safe to use with either legacy or CLDR locales, and for CLDR locales represent the recommended way to set and inquire about locales, both system and per-process:

**availableLocales**
**current current:**
**named:**
**preferredLocaleName**
**setDefault setSystemLocaleTo:**

The method Locale class>>current will answer either the Locale set for the current process, or if no locale has been set for the process, will answer the global system locale (Locale.CurrentLocale). The current locale of a process may be changed by sending the desired instance of Process the message locale: aLocaleThing (where aLocaleThing is either an instance of Locale or CompositeLocale, depending upon whether legacy or CLDR locales are active). To change the locale of the currently running process, use Locale current: aLocaleThing.

To change the overall system locale, use Locale setSystemLocaleTo: localeNameSymbol.

The implementation of class Locale provides the flexibility for new processes to inherit your current locale setting or to control the setting of a process' locale on a process-by-process basis.

**General Considerations**

The following points should be considered when working with locales:

- Legacy locales should work as usual if that is the user's choice, at least through the 7.7 release cycle. Eventually they may be phased out, but there is no firm timeframe for this at present.
- Customer applications should not mix locales of both types in one running image. Hence, the system has not been coded to reset the locale subsystem type dynamically while running.
- Although not specifically included in the CLDR, we have included a #C locale equivalent to the #C locale present in the legacy locales. This is for continuity (many people use this as their standard locale, for differing reasons), and also because some

internal facilities store timestamps, times and dates using the formatting of the #C locale so as to be culturally-independent of the actual locale in use.

- The image should allocate a reasonable default locale for you, based on the setting in your operating system.
- The locale names in the CLDR set are more extensive and more like what you will see in your operating system's list of locales. In most cases the name of your default locale will change. For example, the US locale in the legacy set is #us and in the CLDR set, it's #en_US (for English, anyway. Spanish in the USA would be #es_US, for example).
- There are now (again, in the CLDR set of locales) both culturally-specific (or territory-specific, if you will) locales, such as #en_US, and culturally-neutral, or language-only, locales such as #en.
- CLDR locales are represented by instances of class CompositeLocale, which is composed of a LocaleLocalizationComponent and a LocaleEncodingComponent. They are somewhat more opaque than before, in the sense that there is no API currently provided to play with subcomponents of a locale, such as overriding the TimePolicy with an arbitrary one. You are encouraged to create custom locales if you need them.
- Locales in the CLDR environment are created dynamically, rather than all being instantiated at once and held by the system. That was trivial with 15 locales and not so much now.
- For now, the API continues to rely on class Locale. E.g., even if you are using CLDR locales, your code can still send Locale availableLocales to receive a list of the locales available. Note that the list will differ between locale subsystems, as you would expect. Locales may be retrieved with Locale named: as before.
- Timestamp, Time, and Date displays in textual format will now use native characters and formats. Russian timestamps now show in Cyrillic, for example. There are some limitations in this version, for example there is not yet extensive East Asian (the so-called CJKL) character set support.
- The characters you can display have a lot more to do with font selection than code pages, as code pages aren't used with native Unicode apps, in Windows anyway. Some naming of locales may change in the near future, specifically the Windows ones that still reference the vestigial code pages in their names, because this

may be misleading, causing you to think you are limited to the character code points in the named codepage.

- For the locales #de, #el, #es, #fi, #fr, #nl, #pt, and #se there are instances by that name in both the Legacy and CLDR sets. This is yet another reason why it is not intended that both kinds of locales be concurrently instantiated. Also note, that although the names are the same, the locales represented by those names differ in the two environments. In the Legacy environment, these are actually a combination of language and territory, while in the CLDR environment, the locales by these names are culturally-neutral, or language-only, locales.

### Deprecated Locale API

When using CLDR locales, the following methods on the class side of Locale will answer self shouldNotImplement:

**listLocaleNamesFor:**
**localeMap**
**mapLocaleName:**
**set:**

These methods are deprecated because they deal with facilities in the legacy locale set that have been changed in the CLDR locale set, viz., setting or changing the system locale or working with the mapping of names between OS locale names and image locale names, which is accomplished differently when using CLDR locales.

### CLDR Locale Per-process Locale Inheritance

When using CLDR locales, starting a new process will cause that process' locale setting to be set to that of the process that created it. If there is no per-process locale set when the new process is created, it will be created with its locale set to nil, which will cause Locale current to answer the global system locale.

This means that an application may effectively set the locale to one different from the locale answered by the operating system (which is used to set the global system locale and encoding). With the addition of CLDR locales, a new primitive has been introduced to the VM that reports more accurately than before as to the language, territory and encoding set within the operating system's active locale.

More comprehensive handling of per-process locales with regard to dependency notification to processes when the root locale from which the processes inherited their locale setting has changed is planned for a future release.

If the system locale has been changed with setSystemLocaleTo:, dependents (fonts, etc.) will be notified when the locale is changed using either legacy or CLDR locales.

**Testing Locales For Equality**

When using legacy locales, all 15 locales are instantiated and held in the active image, regardless of which locale you are using. When a Locale instance is retrieved using Locale named:, an object reference is returned to one of these locales. obtaining an additional reference by again using Locale named: will yield instances which compare as the same object with ==.

When using CLDR locales, the same scenario will yield two distinct object references to instances of CompositeLocale which do *not* compare as the same object with ==. (The instances of LocaleLocalizationComponent and LocaleEncodingComponent held by the CompositeLocales would test as the same objects with ==, however.) For this reason, instances of CompositeLocale implement the = method, which should be used instead.

As always, Cincom welcomes your feedback and comments. Comments regarding the facilities that have been made available with this phase of the project will be given priority in consideration over requesting extended facilities beyond those provided in this round. Naturally, though, we are open to suggestions for features you wish to see in VisualWorks 7.8. Development on this framework is ongoing, and we'd like to know what matters most in your use of locales.

## Locale-driven Formatting of Date/Time/Timestamp Values

Since the addition of Locales to VisualWorks, an unintended use of the formatting policies in TimestampPrintPolicy, often through the use of PrintConverter, has become established. This is the use of a textual representation of a Date, Time, or Timestamp object as intermediate storage, with the intention that an object may eventually be recreated from the information contained in the textual representation.

This did not pose any particular issues in VisualWorks prior to the inclusion of Locales based upon the CLDR. This is because the legacy locales all include milliseconds in their print policy formats, so that there was never any mismatch in the formats between a format (typically #long or #medium) used to create a textual representation and the format (either #editing or #timeEditing) used to interpret that same textual representation in a UI canvas input field, for example.

Thus creating a textual representation of a timestamp would yield a string that could be read back to create a timestamp with the same time values down to and including the milliseconds field.

CLDR formatting is intended to produce textual representations for culturally-appropriate display and does not pay any particular attention to the parallel need to sometimes create a textual representation that can be used to create a time thing (object) with the same resolution as the original object from which the textual representation was created.

So if you take a Timestamp and print it out using the #editing format, and read it back, the milliseconds are not preserved when using CLDR-based locales.

For programmatic use, the preferred pattern would be this (using the #C locale rather than "Locale current"),

```
| timestamp converter string result |
timestamp := Timestamp now.
converter := Locale named: #C.
string := converter printAsTime: timestamp policyNamed: #medium.
result := converter readTimestampFrom: string readStream.
self assert: result = timestamp.
```

Enhancements in future versions of VisualWorks will address the need to functionally separate these two somewhat different uses for textual representations of Time objects.

## Add-on Support Parcels

Changes to the Locale system in VisualWorks 7.7 have required changes to the set of add-on support parcels for internationalization features, and these impact their usage. The affected parcels are:

### AllEncodings

This parcel is now obsolete and has been removed from the distribution for VisualWorks 7.7. Most of the functionality has been rolled into CLDR locales. Some features may operate slightly differently in VisualWorks 7.7 compared to 7.6. Customer applications that specify AllEncodings as a prerequisite should be able to remove the prerequisite when using CLDR locales.

### JapaneseLocale

Provides customizations of the environment to enable a more usable environment for Japanese language contexts. Usable both with CLDR locales and legacy locales.

**JapaneseEncodings**

Supplies Japanese encodings. Usable both with CLDR locales and legacy locales.

**Internationalization-Asia**

Used with legacy locales only. Not used with CLDR locales, but not known to be harmful.

**Internationalization-Encodings**

Supplies Chinese and Korean encodings that are too large to be included in images that don't need them. Usable both with CLDR locales and legacy locales.

# Net Clients

## Custom MIME Handlers

When processing a complex multipart message there are number of different things that you may want to do with individual parts. The default behavior is to read all part bodies and create internal streams out of them, unless the part is an attachment in which case you can choose to store it directly into an external file instead (see the saveAttachmentsAsFiles option). But what if you need a more sophisticated decision process, or what if you don't want either of those to happen and you just want to skip and ignore certain types of parts, MimeParserHandlers (reading) and MimeDispatcherHandlers (writing) are structured in a way that allows easy overriding of the default behavior in custom subclasses. And if the existing structure of the parsing/writing process doesn't fit, you can subclass the Parser/Dispatcher as well, and redefine the process completely.

The only difficulty with this setup is that builders/parsers and writers/dispatchers are highly transient, they generally exist to process a single message only and then discarded. Consequently they are often hidden inside higher level components handling entire conversations. For example, HttpClient may need to exchange several messages with the server to arrive at the final response to an HTTP GET and therefore creates handlers on the fly. The handlers are now created according to a blue-print provided by "options". DispatcherHandlers are built from WritingOptions and ParserHandlers are built from ReadingOptions. All the various options relevant to reading/writing of messages (previously scattered around various classes) are now gathered in corresponding Options

classes. The options also define which class of WriteHandler/BuildHandler should be used, in fact the options object is used as a factory for handlers, allowing it to pre-configure the handler according to its various settings. HttpClient provides access to its options through the #writingOptions and #readingOptions accessors. Various option setting methods are still provided for backward compatibility.

As an example, let's create a custom build handler that will simply skip any attachments that are not *.txt files (instead of saving them into files or internal streams). We'll make a new AttachmentFilteringHandler sub-classing HttpBuildHandler and redefine the following method:

```
processAttachment: aBody from: aStream
| fn |
    fn := aBody parent fileName.
    ('*.txt' match: fn)
        ifTrue: [^super processAttachment: aBody from: aStream].
    "We still need to read the part contents even if we intend to discard it,
    so that any following parts can be processed properly"
    aStream binary.
        [   [ aStream atEnd ] whileFalse: [ aStream next ].
            ]   on: Error
                do: [ :ex | self handleStartBody: aBody exception: ex ].
    aStream text.
    "And we need to give the dummy part some content."
    aBody parent removeFieldAt: 'Content-type'.
    aBody source: 'Attached file was discarded.' readStream.
```

To test the new handler let's pretend we're receiving an HTTP response with two attachments:

```
| sent received pipe |
(sent := HttpResponse code: '200')
    contents: 'downloading files';
    addFileAttachment: (Filename fromComponents: ('$(VISUALWORKS)/
net/SMTPS.pcl' tokensBasedOn: $/)) asFilename;
    addFileAttachment: (Filename fromComponents: ('$(VISUALWORKS)/
readme.txt' tokensBasedOn: $/)) asFilename.
pipe := ByteArray new writeStream.
sent writeOn: pipe.
pipe := pipe contents readStream.
received :=
    AttachmentFilteringHandler new
        saveAttachmentsAsFiles: false;
        readFrom: pipe.
received parts
```

The result should be 3 parts. The second should be the one for SMTPS.pcl with its contents discarded. The third one should have its full contents since it is a *.txt file. To be able to use the new handler with HttpClient or the Opentalk HTTP server a corresponding ReadingOptions subclass has to be created as well, since both of these facilities use the options object as a factory for the builder. The simplest variant in this case is to subclass HttpReadingOptions as AttachmentFilteringOptions and add a single method:

```
newBuilder
    ^AttachmentFilteringHandler with: self
```

The new class is then used as the argument for the #readingOptions: message, for example:

```
HttpClient new
    readingOptions: AttachmentFilteringOptions new
    get: 'http://....'
```

Note that for more complex customization the custom options class is meant to serve as the holder of any additional parameters that the customized process might require. Custom write handlers and writing options can be used analogously.

## Streaming of Generated Content

The groundwork for being able to stream MIME message/part bodies from sources of unknown (potentially large) size has already been included in previous releases. Among other things it allows VisualWorks to process arbitrarily large file attachments efficiently. The underlying assumption was that these externalized sources/ targets are represented by streams of various kinds. This works well for static, pre-existing data, however it is not so suitable for information that has to be generated on the fly.

In the 7.7 release, we have extended the capabilities of the MIME framework to allow creating MIME entities using blocks as their bodies. The block embodies the content computation process. It takes one argument which will be the fully set up stream stack (chunking, zipping, boundaries, etc.) on top of the output stream (socket, file, etc) when the block is evaluated. The block is expected to write the body contents directly into that stream. Obviously this has to happen after any preceding headers and content (in case of multipart messages) is already written.

Here's an example:

```
| output |
output := String new writeStream.
Net.MimeEntity new
    contents: [:stream | stream nextPutAll: 'Hello'];
    writeOn: output.
output contents
```

This produces the following output:

```
Content-type: text/plain;charset=utf-8
Hello
```

By the time the block runs producing the “Hello” content of the body, the header has already been written in to the output stream. Notice that by default entity with block contents will be set up for textual content (specifically content-type: text/plain;charset=utf-8). For any other content type the entity has to be set up with corresponding content-type values before it is written. The stream that the block receives will be set up according to the content-type value.

Here is an example with generated binary content:

```
| output |
output := String new writeStream.
Net.MimeEntity new
    contentType: 'application/octet-stream';
    contents: [ :stream | stream nextPutAll: #[1 2 3 4 5 6 7] ];
    writeOn: output.
output contents
```

yielding:

```
Content-type: application/octet-stream
Content-transfer-encoding: base64
AQIDBAUGBw==
```

The block-based solution scales up to multipart messages as well. There can be parts with static content before or after the dynamically generated part. Or several parts of a single multipart message can be expressed as blocks which are then evaluated in sequence when their turn in the structure of the message comes.

To accommodate the new type of body contents and to make it easier to work with some previously supported body types we decided to extended the capabilities of MimeEntity>>contents:. The extended method makes it significantly more convenient to create complex messages. It can now handle following types of arguments:

- String - as before allows you to specify textual content, defaulting to content-type: text/plain;charset=utf-8 (MimeEntity class>>defaultTextCharset)
- ByteArray - allows to specify binary content, defaulting to content-type:application/octet-stream (will be base64 encoded for transfer)
- BlockClosure - content is generated by the block when the message is being written, defaults to textual content type
- (readable) Stream - content is read from the stream when the message is being written, content type depends on stream's response to isByteStream
- and SequenceableCollection of an arbitrary mix of these types to allow creating multipart bodies

Note that the contents: method only attempts to set content-type if it was not previously set, allowing to override the defaults that way.

With these changes it is fairly easy to build complex MIME messages, for example the following:

```
| output |
output := String new writeStream.
Net.RFC822Message new
    contents: (Array       “multipart message”
        with: #[1 2 3 4]
        "simple binary content"
        with: [:stream | stream nextPutAll: 'Hello']"generated text"
        with: (Array          “nested multipart entity”
                with: [ :stream | stream nextPutAll: 'and one more']
    "more generated text"
                with: 'and some streamed text' readStream
    with: (ByteArray withAll: (50 to: 100)) readStream)
 "and some streamed bytes"
        with: 'Hello' "simple text content" );
    writeOn: output.
output contents
```

yields:

```
Content-type: multipart/mixed;boundary="=_vw0.22176418836311d_="
This message is in Mime format
--=_vw0.22176418836311d_=
Content-type: application/octet-stream
Content-transfer-encoding: base64
AQIDBA==
--=_vw0.22176418836311d_=
Content-type: text/plain;charset=utf-8
```

```
Hello
--=_vw0.22176418836311d_=
Content-type: multipart/mixed;boundary="=_vw0.60014504874132d_="
--=_vw0.60014504874132d_=
Content-type: text/plain;charset=utf-8
And one more
--=_vw0.60014504874132d_=
Content-type: text/plain;charset=utf-8
And some streamed text
--=_vw0.60014504874132d_=
Content-type: application/octet-stream
Content-transfer-encoding: base64
MjM0NTY3ODk6Ozw9Pj9AQUJDREVGR0hJSktMTU5PUFFSU1RVVldYW
VpbXF1eX2BhYmNk
--=_vw0.60014504874132d_=--
--=_vw0.22176418836311d_=
Content-type: text/plain;charset=utf-8
Hello
--=_vw0.22176418836311d_=--
```

Block-based bodies also allow full control over transmission of the generated content. The next chunk of what has been written so far can be pushed out by simply flushing the provided stream. Note that a chunk is also pushed out when the written content reaches the pre-configured chunkSize without flushing.

This may be illustrated in the following example:

```
| output |
output := String new writeStream.
(Net.HttpResponse code: '200')
    contents: [:stream |
        (31 to: 40)
            inject: 30 factorial
            into: [:factorial :next |
                stream flush.
                factorial * next
                    printOn: stream;
                    yourself]];
    writeOn: output.
output contents
```

which yields (highly useful) part of the factorial sequence, where each value is packaged up and pushed out in its own chunk:

```
HTTP/1.1 200 OK
Content-type: text/plain;charset=utf-8
Transfer-encoding: chunked
22
```

```
822283865417792281772556288000000
24
263130836933693530167218012160000000
25
868331761881188649551819440128000000
27
295232799039604140847618609643520000000
29
10333147966386144929666651337523200000000
2A
37199332678990121746799944815083520000000
2C
13763753091226345046315979581580902400000000
2D
523022617466601111760007224100074291200000000
2F
203978820811974433586402817399028973568000000
30
81591528324789773434561126959611589427200000000
0
```

We can use this example to demonstrate a "Comet style" (in web parlance) web application using the contributed Opentalk-Web. Simply load the parcel and inspect the result of the following bit of code:

```
| action |
action := [:req | [ :stream || query from to |
    query := req httpRequest urlEncodedDataFrom: req url query.
    from := (query detect: [:asc | asc key = 'from'] ifNone: ['from' -> '1'])
value asNumber.
    to := (query detect: [:asc | asc key = 'to'] ifNone: ['to' -> '50']) value
asNumber.
    (from to: to)
        inject: (from - 1 max: 1) factorial
        into: [ :factorial :next |
            stream flush; cr.
            0.25 seconds wait.
            factorial * next
                printOn: stream;
                yourself ] ] ].
((WebAdaptorConfiguration new
        addExecutor:
            (WebDo
                if: [:req | req path last = 'factorial']
                do: action);
        transport: (HTTPTransportConfiguration new
                marshaler: WebMarshalerConfiguration new );
        soReuseAddr: true
```

```
        ) newAtPort: 4242
    ) start
```

If you then point your web browser at http://localhost:4242/factorial?from=50&to=100 and if the web browser is able to process incoming chunks as they come, you should see factorial results popping up on the web page at a rate of four lines per second, i.e. the page shows the partial results while the rest are still being computed on the server. This is a moral equivalent of Comet (without any Javascript or even HTML involved). Note that the adaptor should be stopped before closing the inspector (using the <Operate> menu on self in the inspector).

# Glorp

VisualWorks 7.7 includes a number of additions and enhancements to Glorp. Almost 50 ARs have been incorporated in this component, ranging from simple bug fixes to significant additions. For example, Active Record and the new migration features offer developers an easier way to analyze and upgrade their databases. Both are used in Cincom's new WebVelocity product.

Notably, Glorp now powers VisualWorks' own source code control tool, Store. There is increased platform support for Oracle, SQLite, SQLServer, MySQL, Postgres, and DB2 support is now available (in preview).

## Logins and Store Connection Profiles

Store connection profiles (the entries in the **Store** > **Connect to…** dialog) now set up Glorp logins when they connect to Store databases.

These logins are configured to suit database performance for Store on the various supported platforms: in particular, all Store database platforms that return true to supportsBinding

- Also set useBinding: true, except for PostgreSQL
- Also set reusePreparedStatements: true, except for SQLServer and InterBase / Firebird

These settings are *not* guaranteed to be in force if you use Glorp to connect to these databases for any other purpose. Therefore, if you use the same database platform for Store support and for your

application, be sure to set these same values (or other values if your application prefers them) when you create your application's Glorp logins.

## Active Record

In release 7.7, Glorp includes support for Active Record. In particular, the API for persisting objects has changed.

In several popular implementations of Active Record, a save method is used to persist objects. Without Active Record, the usual method for saving objects with Glorp is to use register: inside a unit of work. With Active Record, you should send bePersistent directly to the domain object, followed by commitUnitOfWork. This sequence will produce the expected behavior.

## Migrations

Release 7.7 includes support for migrating database schemas. This is necessary when you need to change your application's domain model during the course of development.

The migration API is intended for use with Active Record designs. While Active Record does not require that developers build their own descriptor systems, for migration it may be necessary to explicitly specify the descriptor system using a method in the domain model class.

Generally, you begin by cloning your schema class. For each class/table in your schema, there will be a method that describes it (e.g. tableForMYTABLE:). When migrating, you'll have two separate descriptor classes, each with its own tableForMYTABLE: method.

Glorp provides a mechanism to analyze the differences between two versions of a table, and to generate a migration script. This may be turned into a single method which contains Glorp code to alter the first version of the table so that it matches the second version of the table. The conversion methods can be generated for both directions of conversion (upgrade or revert).

A migration is initiated by: DescriptorSystem>>createMigrationChangeSetFor:. This generates the conversion script from one descriptor system to the other. Rather than requiring that the developer write the script manually, Glorp performs this step automatically. For complex schemas, though, it may be necessary to tune the script by hand.

### Information Schemas

In release 7.7, Glorp now includes support for reading a database vendor's "information schema". This is required by Active Record, and makes it possible to create a meta-analysis of an entire database.

Information schemas are somewhat vendor specific, but Glorp now supports this feature for Oracle, SQL Server, Postgres, MySQL, and DB2 (the last is currently in preview). To afford this capability, numerous small enhancements were made to the platform-specific code in Glorp.

For a minor example, the MySQL TIMESTAMP column type turned out to be more limited than the DATETIME type, which then became the new Glorp default column type for Smalltalk Timestamp objects. (Eg., the range of a TIMESTAMP is 1970-2038 AD, whereas DATETIME range is 1000-9999 AD.)

## WebSphere MQ Interface

None submitted in this release.

## Seaside Support

### Seaside 3.0alpha5

Release 7.7 of VisualWorks includes Seaside 3.0a5, upgraded from Seaside 2.8 in the previous release.

### Default Encoding now UTF-8

In VisualWorks 7.7, the codec is set to UTF-8 instead of the default NULL used in Squeak/Pharo. This may be incompatible with applications written for Seaside 2.8 or written for Seaside ported from Squeak/Pharo.

To make your application work correctly, do not encode strings as UTF-8 in your application code; Seaside will now do this for you when writing to a web page and reading requests from a web browser.

### jQuery Support

In this release, support for jQuery is provided as a separate library.

### Comet Support

Release 7.7 of VisualWorks includes support for Comet.

## DLLCC

### Browser Support to Identify and Stub Missing DLLCC Definitions

Historically, when you compile a method like:

```
<C: MyReturnType someFunc() >
```

if there was no method that defines MyReturnType it would add a MyReturnType method that typedef'ed MyReturnType as void* (or as an empty struct or enum if it was declared so). This "feature" has caused headaches for some time. One doesn't always want this helpful feature being invoked from the bowels of the compilation process. This behavior has been removed.

The Code Browser now has a class menu option -- **Generate Missing Type Definitions** -- that shows up on ExternalInterfaces when they indeed are missing these messages. It can be used to then generate them.

### Solaris and the C Heap

When using the external C heap to allocate memory, it is the usual expectation that freeing such memory explicitly or implicitly via the gcMalloc: mechanism will result in memory being returned to the operating system. This is not so in Solaris. According to the manual page for the function free(), freed memory blocks will be still reserved for the application and will not be returned to the operating system.

This has practical consequences, particularly when using the gcMalloc: mechanism. Consider an application that makes heavy use of garbage collectable pointers to the external heap. Since significant time may elapse between GC invocations, there is the potential for a buildup of such garbage collectable pointers. Solaris happily accomodates this by increasing the amount of memory allocated to the virtual machine. After the GC runs and collects the garbage collectable pointers, however, Solaris will keep all the necessary memory reserved for the virtual machine.

If the C heap has managed to grow past the amount of virtual memory still available, then trying to execute an external process from the image will be no longer possible. This is because fork() makes a copy of the invoking process, and since the (now mostly

empty) C heap makes the footprint of the virtual machine too large to fit in the amount of available virtual memory, the attempt to copy the virtual machine process fails.

This is a known issue for Solaris, and there exist a number of workarounds such as the ones described here.

http://www.steubentech.com/~talon/dlmalloc.html

http://g.oswego.edu/dl/html/malloc.html

Note that Cincom does not necessarily endorse nor recommend any such workaround. Customers are advised to use caution with regards to this matter.

## Flag to Ignore ExternalErrorNoThreadFound

In release 7.7, a global flag has been added that can be used to ignore ExternalErrorNoThreadFound.

ExternalErrorNoThreadFound can occur if an external thread that invoked a CCallback and into which the callback attempts to return to no longer exists. This can happen if an image is saved while the callback is in progress. When the image is restarted the CCallback attempts to return to the external thread, which results in an unhandled exception in the foreign callback process.

Because this all happens in a background system process that developers normally cannot control, it is difficult to handle the resulting unhandled exception. That's why we have added a guard clause in #primReturnToThread:for: which allows suppressing this particular exception.

This behavior can be set via the IgnoreExternalErrorNoThreadFound shared variable in class CCallback. The default is to suppress the exception. Setting the value of the shared variable to false will restore the previous behavior, allowing the exception to bring up an error notifier.

## Objective-C Runtime Support

Release 7.7 of VisualWorks includes support for interfacing to an Objective-C runtime (versions 1 & 2). This provided as a set of ExternalInterface classes in the OS-MacOSX package. The API is packaged in class AbstractObjectiveCRuntime.

Most entry points in this API are the same between versions 1.0 and 2.0. In the common cases where some difference exists, class AbstractObjectiveCRuntime provides an API to swap between the two implementations, e.g.: #getClassName:, #getMethodName:, #getAllClasses, #getAllMethodsForClass:, #getSuperclass:.

For examples of use, see the comment for AbstractObjectiveCRuntime.

Another package, ObjectiveCConnect, is not finished yet but is in /contributed for release 7.7. ObjectiveCConnect does automatic interface generation (as required), similar to DLLCC, except better in the respect that it doesn't actually install classes that you need to publish.

**Memory Management**

When using this API, your application must take charge of memory management, and adhere closely to its invariants.

Objective-C includes garbage collection, but the VisualWorks ExternalInterface does not currently enable it. At present, there is no mechanism for maintaining strong references to objects until the VisualWorks garbage collector is done with them.

If your application calls an object with 'alloc' 'new' or 'copy' in its name, then the object is your responsibility. You must #release it when you're done with it. (You do not have to #retain it).

Any message you send that returns an object *may* be an object that was created just for you, but is managed by the object you called from. The API doesn't make it explicit whether this has happened or not. For example, an object is made for you when you send #objectEnumerator, but not #sharedApplication.

What happens is this: since the object you're calling the message on "owns" what it made, it is also response for releasing it. Often this simply means the object is placed in to an *autorelease pool*. Every running thread must have an autorelease pool. Without it, the application will crash. The VisualWorks VM sets up an autorelease pool for the Smalltalk thread on startup. But this is not enough, because we neither have a way nor an appropriate time to send #drain to it.

An object ends up in the autorelease pool by sending #autorelease to it. In a regular Cocoa application, the main thread's event loop drains the pool between events. When you wish to optimize the creation/destruction of objects you can create your own autorelease pool.

Since our VM runs on a thread that is not the main thread, there is no event loop to drain the autorelease pool. Consequently, if you do not set up and drain your own release pool, you'll be leaking memory.

The method autoreleaseWhile: will create an NSAutoreleasePool and drain it when the block has finished executing, handling the most common case of memory management for Objective-C.

Setting up and draining an autorelease pool is easy:

```
pool := NSAutoreleasePool alloc init.
"do stuff"
pool drain.
```

When you create an autorelease pool, it "nests" inside the existing autorelease pool. This is important, because it *strongly warns us* that we cannot have multiple Smalltalk green threads talking to Objective-C at the same time in any way shape or form.

Things like Open/Save dialogs are safe because they only deal with singletons. A message like #focusedWindow is safe because it only deals with objects that already exist.

Code that might be interrupted by another Smalltalk green thread that could create its own autorelease pool or call #autorelease -- this must be placed inside a critical section.

**Limitations**

The Objective-C runtime facilities require some initialization when an image is started. In release 7.7, this occurs later than it really needs to. Thus, it is possible to invoke code at startup time which, if it accesses the ObjectiveCRuntime shared variable, can find that it is not yet initialized. Any code that runs before the SystemEventInterest #returnFromSnapshot may encounter this situation. Command line options such as -doit, -filein, etc., are evaluated after that point, so those are safe. This limitation will be resolved as early as possible in in the next release.

**Q & A**

**When do I need to send #retain:?**

If you have received an object from Objective-C and intend to hold on to it beyond the autorelease pools existance.

**When do I need to send #release:?**

To any object you have #retained or any object you created; when you are done with it.

**When do I need to create/drain an autorelease pool?**

Any time you're interacting with Objective-C in a non-trivial way (e.g.: almost always).

**When do I need to become uninterruptable?**

Any time you've created an autorelease pool, for the period until you have #drained the autorelease pool.

**How do I tell if a particular method uses #autorelease and therefore I need an autorelease pool?**

Send #retainCount to the object you get back, then do it again and see if it goes up by 1.

**Can't I just throw #retain everywhere instead of dealing with autorelease pools?**

No, if the object is in an autorelease pool, retaining does not take it out of the autorelease pool.

**Can't I just throw #release everywhere and cross my fingers?**

Yes, if your goal is to crash the runtime.

## Objective-C Utility APIs

The following utility APIs have been added for sending messages to Objective-C objects:

**object:perform:**

**object:perform:with:**

**object:perform:with:with:**

**object:perform:with:with:with:**

**object:perform:withArguments:**

The following utility APIs are used to end messages to Objective-C objects to be run in the main thread:

**object:performOnMainThread:**

**object:performOnMainThread:with:**

**object:performOnMainThread:with:with:**

**object:performOnMainThread:with:with:with:**

**object:performOnMainThread:withArguments:**

# Documentation

This section provides a summary of the main documentation changes.

Note that we changed the documentation source format in 7.7. While we have attempted not to lose any content or formatting, some errors have nearly certainly occurred. Please notify us of serious lapses.

## Basic Libraries Guide

- The “Dates and Times” chapter has changed to “Chronology.”
- Added discussion of new classes Duration and Timer.
- TimeZone discussion has been updated.
- Timestamp has been expanded.

## Tool Guide

- Updates to System Browser for new package/bundle system.
- Update and expanded Unit Testing chapter.

## Application Developer’s Guide

- Rewrite of the Weakness and Finalization section.
- Updates to System Browser for new package/bundle system.
- Corrections to the BOSS discussion

## COM Connect Guide

Miscellaneous updates and corrections.

## Database Application Developer’s Guide

Miscellaneous updates and corrections.

### DLL and C Connect Guide

No changes.

### DotNETConnect User's Guide

Miscellaneous updates and corrections.

### DST Application Developer's Guide

No changes

### GUI Developer's Guide

Minor updates.

### Internationalization Guide

This guide is significantly out of date, and will be updated for the next release. In the meantime, refer to the release notes for changes in 7.7.

### Internet Client Developer's Guide

Significant updates and corrections, particularly relating to refactorings and improvments to MIME and mail support.

### Opentalk Communication Layer Developer's Guide

Updated for 7.7.

### Plugin Developer's Guide

No changes.

### Security Guide

No changes.

### Source Code Management Guide

Miscellaneous updates and corrections.

### Walk Through

Updated for 7.7

### Web Application Developer's Guide

No changes.

### Web GUI Developer's Guide

No changes.

### Web Server Configuration Guide

Minor updates.

### Web Service Developer's Guide

Removed discussion of UDDI, no longer supported.

# 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

### WinCE Engines Dropped

In release 7.7, the WinCE virtual machines for both ARM and x86 platforms are no longer supported. The VisualWorks Installer no longer includes an installation option for these engines, and their corresponding READMEs have been removed as well. If this presents an issue for your development effort, please contact Cincom.

### SGI IRIX Engines Dropped

In release 7.7, the virtual machines for the Silicon Graphics IRIX platform is no longer supported. The VisualWorks Installer no longer includes an installation option for this engine, and its corresponding README has been removed as well. If this presents an issue for your development effort, please contact Cincom.

# Base Image

## IEEE Math

With this release, VisualWorks is now able to process IEEE special values such as NaN and INF by default. This means that the IEEE Math parcel is no longer needed when such values will appear in the image, regardless of whether NaN or INF are coming from the VM when it is configured to produce such values, or from external library calls such as via DLLCC. As a result, the IEEE Math parcel is now obsolete.

For details, see: Floating Point Arithmetic.

## OS/2 and MacOS 9.x Platform Support Removed

Platform-specific code for filenames, sockets, fonts, and OS interfacing, specific to Macintosh (pre-MacOS X) and OS/2 have been removed from the base image.

# GUI

## NotebookWidget gone

The Notebook widget has been removed from the system. Parcels Obsolete-Notebook and Obsolete-UIPainter-Notebook provide the deprecated functionality.

## Subcanvas Obsolesced

The class Subcanvas is obsolete. It is left as an empty stub class in the image for backwards compatibility reasons, but its new method has been modified to return an instance of its CompositePart superclass.

## MacOS 9 Look and Feel Removed

The legacy MacOS Look and Feel classes were removed. They may be loaded and used still using the UILooks-Mac parcel.

# Tools

## RBSmallDictionary Gone

RBSmallDictionary, a Dictionary implementation optimized for small sets, has been replaced by the system CompactDictionary.

# Net Clients

## Class SimpleSMTPClient Obsolesced

Class SimpleSMTPClient has been marked as obsolete since release 7.5 of VisualWorks. In this release, it has been removed from the Net Clients class library, and its functionality is superceded by class SMTPClient.

# Opentalk

## SNMP

The SNMP parcels included in prior releases no longer work in this release, and so have been removed. No further work on SNMP is currently planned.

## Opentalk-Remote-Testing

The Opentalk-Remote-Testing package has been removed from this release, and there are no plans to develop it in the future.

# Plugin

## Plugin Obsolesced

The socket-based Plugin has been withdrawn from future consideration and those interested in developing VisualWorks applications for a web browser should look instead at WebVelocity.

# Application Server

## WebServerStartup

The WebServerStartup class has been deprecated. It has been moved to the /obsolete directory (Wave-Base-Obsolete parcel) for this release if you still need it.

Please consider using a subsystem or startup configuration file to configure your servers, if you are currently using this class.

# 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.

## 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.

## DB2 Support

Updated support for Glorp for DB2 is available in the /preview/glorp directory. EXDI support for DB2 is now supported product.

## 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. It has not been finished yet, but may be loaded. 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 7.7 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 of the Smalltalk 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..."].
```

**Warning:** 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 with 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.

## Smalltalk Archives

Even though Smalltalk Archives are a supported feature of ObjectStudio 8, they are supported only in preview for VisualWorks at this time.

A Smalltalk Archive is a file containing a collection of parcels, compressed (using tar). The archive specifies a load order for the parcels, and supports override behavior.

Unlike publishing a bundle as a parcel, a Smalltalk Archive preserves package (and parcel) override behavior and package load order. Accordingly, Smalltalk Archives are a good alternative to publishing a bundle, in some circumstances.

Smalltalk archive files have a .store filename extension.

To load an archive, use the File Browser to locate the file, then right-click on it and select **Load**.

The archive loads with the parcel and bundle structure it had when it was saved. Database links might or might not be preserved, depending on the settings at the time it was saved.

To publish a Smalltalk Archive, select the bundle (or package) in the System Brower, the select **Package** > **Publish as Parcel…** The options in the publish dialog are the standard publish options, except for the Store options section. Because you want to save as a Smalltalk Archive, leave that checkbox selected. If you are using the Smalltalk Archive for deployment, and so do not need the database links, uncheck the **With database links checkbox**; otherwise, leave it checked.

The archive is published, by default, in the current directory, typically the image directory. The file name is the bundle name with a .store filename extension.

## WriteBarriers

The immutability mechanism in VW can be used to detect any attempts to modify an object. All it takes is marking the object as immutable and hooking into the code raising the NoModificationError. The ability to track changes to objects can be useful for number of different purposes, e.g., transparent database updates for persistent objects, change logging, debugging, etc. While the mechanism itself is relatively simple, it is difficult to share it as is between multiple independently developed frameworks.

WriteBarriers (loadable from preview/parcels/WriteBarriers.pcl) allow multiple frameworks to monitor immutability exceptions at the same time. This framework makes object change tracking pluggable through subclasses of Tracker. A Tracker must implement a couple of methods:

**isTracking:** *anObject*

Answer true if the tracker is tracking anObject

**privateTrack:** *anObject*

Register and remember anObject

**privateUntrack:** *anObject*

Forget about the object you were tracking

**applyModificationTo:** *anObject* **selector:** *selector* **index:** *index* **value:** *value*

We've accepted that we are tracking the object and a change has been made to it, what do we want to do? The default behavior is to apply the change to the object.

Note that the framework does not provide a mechanism for keeping track of which objects each tracker is managing. Instead, it leaves the options open. Frameworks may already have a registry of objects that they want to track (e.g., a persistency framework will likely cache all persitent objects to maintain their identity) in which case a separate registry for the corresponding tracker would waste memory unnecessarily.

A deliberate limitation of WriteBarriers is that they will refuse to track any previously immutable objects. Trackers can decide to not apply the modification and emulate the original immutability that way, and refusing to track immutable objects reduces complexity of the solution.

Here's a sketch of a tracker that will announce a Modified announcement for any modification that occurs. Let's assume that this AnnouncingTracker will have its own registry for tracked objects in the form of an IdentitySet (inst var. objects). The first three required methods are fairly obvious:

```
privateTrack: anObject
    objects add: anObject

privateUntrack: anObject
    objects remove: anObject ifAbsent: []

isTracking: anObject
    ^objects includes: anObject
```

The modification callback needs to call super so that the modification is actually applied, but in addition makes the announcement as well. Note that Tracker subclasses Announcer to make Announcement use easy.

```
applyModificationTo: anObject selector: selector index: index value: value

    super applyModificationTo: anObject selector: selector
        index: index value: value.
    self announce: (Modified subject: anObject)
```

To make use of the tracker, it has to be instantiated, which automatically registers it in the global registry, Tracker.Trackers. Any objects to be tracked by it have to be explicitly registered with it using the #track: message.

```
tracker := AnnouncingTracker new.
tracker when: Modified do: [ :ann | Transcript space; print: ann subject ]
string := 'Hello' copy.
tracker track: string.
string at: 1 put: $Y.
```

The last statement will trigger the Transcript logging block. To stop tracking an object use the #untrack: message.

```
tracker untrack: string
```

And to deactivate the tracker altogether use the #release message.

```
tracker release
```

## Sparing Scrollbars

Sparing Scrollbars extends VisualWorks widgets to optionally set scrollbars to be dynamic and only appear when necessary. When loaded on top of the UIPainter, the dataset, list, table, tree, text editor, hierarchical view, or view holder widgets, or the window itself may be specified to use this feature in the **Details** page of the UIPainter tool. Sparing Scrollbars includes some test and sample classes which demonstrate the usage of this new behavior.

The Sparing Scrollbars component can be loaded from the parcel preview/SparingScrollbars.pcl.

## Multithreaded COM

Composed of a DLL and three parcels, the multithreaded COM option for COM Connect introduces the ability to perform non-blocking COM calls in VisualWorks. This improves responsiveness of COM servers implemented in VisualWorks using COM Connect. It currently operates for VisualWorks versions 7.3 to 7.6 on Window XP platforms with 32Bit Intel Architecture.

This option changes the threading model of an existing VisualWorks to free-threaded, but synchronization is still performed via VisualWorks DLLCC mechanisms. Threaded COM call-outs and all in-bound calls will be routed through the multithreaded COM interface. Non-threaded callouts will still go through the original COM primitives provided by the VisualWorks virtual machine.

Additional notes and usage instructions appear in /preview/multithreaded com/DLLCOM.pdf.

## COM User Defined Type (UDT) Support

Prior to this release, COM Connect was unable to call COM functions that use user-defined parameter types (structures). The COM UDT Support preview component adds support for such data types for 7.6.

The COM UDT support component consists the parcel files and a PDF file in preview/com udt support/.

## 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

## Store for Access

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

The enhancements include:

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

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

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

## Store for Supra

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

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

- Blob data (i.e. LONGVARCHAR column) is returned as null when accessed through a view.
- INSERT statement may not be combined with a SELECT on the same table (CSWK7025)

- UPDATE statement may not update any portion of the primary KEY (CSWK7042)
- DELETE statement may not have a WHERE... IN (...) clause with lots of values (CSWK1101, CSWK1103)
- When blob data (i.e. LONGVARCHAR column) is retrieved from the data base, the maximum length is returned rather than the actual length
- Supra SQL does not have SEQUENCE

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

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

### StoreForSupra installation instructions

1 Install Supra SQL

2 Create a Supra database

3 Use XPARAM under Windows to set the following

- set Password Case Conversion = Mixed
- set Supra tuning variable SQLLONGVARCHAR = Y
- set Supra tuning variable SQLMAXRESULTSETS = 256

4 Start the Supra database

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

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

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

6 Load the StoreForSupra parcel.

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

Store.DbRegistry goOffLine installDatabaseTables.

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

Store.DbRegistry goOffLine deinstallDatabaseTables.

# Security

## OpenSSL cryptographic function wrapper

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Opentalk

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

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

## Opentalk HTTPS

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

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

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

Here's the full list of all HTTPSTransportConfiguration parameters:

**clientContext**

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

**serverContext**

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

**clientValidator**

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

**serverValidator**

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

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

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

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

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

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

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

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

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

## Distributed Profiler

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

### Installing the Opentalk Profiler in a Target Image

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

- Opentalk-Profiler-Core

### Installing the Opentalk Profiler in a Client Image

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

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

### Opentalk Remote Debugger

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

The remote debugger is contained in two parcels.

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

```
RemoteDebuggerClient startMonitor
```

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

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

```
RemoteDebuggerClient stopMonitor
```

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

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

## 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"
```

## Polycephaly

This package provides simple mechanisms for spawning multiple running copies of the 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.

The spawned images (drones) are represented by instances of VirtualMachine. The primary API are the variations of the #do: message which take an action expressed either as a String containing valid Smalltalk code

```
[ :vm | [ vm do: '3 + 4' ] ensure: [ vm release ] ]
    value: VirtualMachine new
```

or an instance of "clean" BlockClosure (one that does not reference any variables, including globals, outside of its scope)

```
| vm |
    vm := VirtualMachine new.
    [ vm do: [3 + 4]
    ] ensure: [ vm release ].
```

Optional arguments can be attached as well.

```
    | vm |
    vm := VirtualMachine new.
    [   vm do: [ :a :b | a + b ] with: 3 with: 4
    ] ensure: [ vm release ].
```

Note that the arguments will be "BOSS-ed out" for transport, so anything in the scope of objects they reference (transitively) will be included as well. Avoid including things like classes or external resources. For more complex cases where the clean block with arguments is hard to achieve, or when the action needs to hook into the objects in the drone image, the String based action is more suitable. Its advantage is that the code will be compiled in the Drone and can therefore reference classes and globals available in that

image. In this case any arguments are referenced in the code as named variables and has to be passed in as a dictionary mapping the variable names to values. They will be included in the compilation scope as other shared variable bindings.

```
| vm |vm := VirtualMachine new.[ vm do: 'a + b' environment: (Dictionary
new at: #a put: 3;
      at: #b put: 4;
      yourself)
   ] ensure: [ vm release ].
```

Note that a VM instance can be reused multiple times.

```
| vm |
   vm := VirtualMachine new.
   [ (1 to: 5) collect: [ :i | vm do: '3 + 4' ]
   ] ensure: [ vm release ].
```

Consequently it needs to be explicitly shut down with the #release message when no longer needed.For cases when multiple VMs are needed to execute the same action in parallel, VirtualMachines class allows to maintain the whole set of machines as one.

```
| vm |
   vm := VirtualMachines new: 2.
   [ vm do: '3 + 4'
   ] ensure: [ vm release ].
```