Appendix A

Updates for VisualWorks 7.4.1

Overview

VisualWorks 7.4.1 is a “patch” release, consisting mainly of fixes to problems we have found or have been reported with 7.4. There are also a couple of features being moved from preview into full supported product.

With only a few exceptions, documentation has not been updated for this release. These release notes cover the essential changes and additions.

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

ARs Resolved in this Release

The Action Requests (ARs) resolved in this release are listed in: fixed_ars.txt in the **doc/** directory.

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

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

VisualWorks Base

ExternalInterface Symbol Lookup

(AR50458) A problem has been noticed that UnixSystemSupport could not find libc on some Linux distributions. The problem has been resolved with the following change.

It is now possible to specify that ExternalInterfaces look-up symbols within the VM and its associated libraries. To specify that an ExternalInterface do this, use the symbol #linkedIn in the same place one uses a library name. Here is an actual example from UnixSystemSupport:

```
Smalltalk.OS defineClass: #UnixSystemSupport
    superclass: #{OS.OSSystemSupport}
    indexedType: #none
    private: false
    instanceVariableNames: ''
    classInstanceVariableNames: ''
    imports: '
        OS.UnixSystemSupportDictionary.*
        '
    category: 'OS-Unix'
    attributes: #(
        #(#includeFiles #())
        #(#includeDirectories #())
        #(#libraryFiles #(#linkedIn))
        #(#libraryDirectories #())
        #(#beVirtual false)
        #(#optimizationLevel #full))
```

Note that you can mix libraries in the file-system with #linkedIn. Library lists are searched in order. So to search the VM and its libraries first, put #linedIn first in the list of libraries.

MustBeBoolean Exception

When a sender sends a message to an object that does not understand the message, doesNotUnderstand: is sent to the receiver, which, by default, raises a MessageNotUnderstood exception. This is a subclass of Error, thus

```
[a not understood message] on: Error do:[:ex| do something]
```

catches the situation.

This behavior does not apply when sending ifTrue:ifFalse:. In this case the message mustBeBoolean is sent to the receiver. This message has been raising a Notification which is not a subclass of error. Thus

```
[send ifTrue:ifFalse:: to nil] on: Error do:[:ex| do something]
```

does not catch the situation.

In 7.4.1 we have introduced a new Error subclass, MustBeBoolean. This class is marked resumable to maintain the current behavior of "proceed with true," which is raised as the default behavior of the mustBeBoolean method. In previous releases, a Notification was raised.

Note that any code that relies on mustBeBoolean raising a Notification, rather than an Error, must be updated. If you have used a proprietary fix for the original condition, you are advised to remove the proprietary code in favor of this solution.

Security

Private Key Encryption

This package implements PKCS #8: Private-Key Information Syntax Standard (http://www.rsasecurity.com/rsalabs/node.asp?id=2130). This standard describes a binary encoding for transfer and storage of private-keys. Private-key information includes a private key for some public-key algorithm and a set of attributes.

The standard also describes encoding for encryption of private keys to protect their confidentiality. It proposes few specific methods of encryption based on password-based encryption algorithms from PKCS#5 (http://www.rsasecurity.com/rsalabs/node.asp?id=2127). See also package PKCS5. Given the highly sensitive nature of private keys, there is rarely a good reason to store or transport them unencrypted.

The standard does not cover the actual format for specific kinds of private keys, and it is apparently non-trivial to find authoritative descriptions of those. The openssl man page for the pkcs8 command points in the direction of PKCS#11 (http://www.rsasecurity.com/rsalabs/node.asp?id=2133). We have used this source for our defintitions. The relevant section from PKCS#11 v2.01 is section 11.9. The latest version, PKCS#11v2.20, covers it in section 12.6, which also adds few more formats for eliptic curve algorithms. This section is quoted in the PKCS8 class comment.

Limitation

We support 2 types of keys

- RSAPrivateKey
- DSAPrivateKey

Encryption algorithm support is limited by the capabilities of our PKCS#5 implementation, which currently means that we support the following:

- DES for encryption and MD5 for key derivation (OID pbeWithMD5AndDES-CBC)
- DES for encryption and SHA1 for key derivation (OID pbeWithSHA1AndDES- CBC)

These are among the most widely supported algorithms. Our plan is to add more secure versions based on RC4 and tripple DES (defined by PKCS#12, http://www.rsasecurity.com/rsalabs/node.asp?id=2138) in the near future. The default algorithm used for encryption is DES with SHA1.

Usage

The API is fairly straightforward. To encode a key on a stream use message writeOn:, for example

```
key := (RSAKeyGenerator bitSize: 512) privateKey.
stream := ByteArray new readWriteStream.
key writeOn: stream.
```

Note, that the stream must be binary and positionable. If you are writing the key out into a file, use an ExternalReadWriteStream, not ExternalWriteStream which is not positionable.

To read the key back from the stream use class side readFrom: message.

```
stream reset.
RSAPrivateKey readFrom: stream.
```

There is an alternative API for reading keys on class side of PKCS8 class, so you can also read the key back as follows.

```
stream reset.
PKCS8 readKeyFrom: stream.
```

This is useful if you are not sure which type of key is next in the stream (DSA or RSA). The PKCS8 API will return whichever key type is there. The reading methods on the key classes (RSAPrivateKey or DSAPrivateKey) will raise a PKCS8Error if the key type on the stream does not match the key class, to prevent inadverent use of wrong key type. Therefore, use PKCS8 class if you do not care what type of key you'll get, and use specific key class if you want specific key type.

The PKCS8 encoded keys are sometimes distributed in additional ASCII “armoring,” called PEM format. This is basically the PKCS#8 bytes encoded in Base-64 encoding with some identification header and footer lines attached. For convenience we also provide equivalent reading methods for this PEM format readPEMFrom: and readKeyPEMFrom:. So you could read a key from a .pem file like this:

```
PKCS8 readKeyPEMFrom: 'key.pem' asFilename readStream.
```

Encrypting Keys

Private keys are highly sensitive information, therefore most of the time it is preferrable to keep the key encrypted while it is not used. The API to store a key encrypted is very much like the one shown earlier, it just acquires additional password: keyword and parameter. The password parameter must be a ByteArray. We purposely choose not to provide a String based equivalent to avoid any ambiguities. It is up to the application to take care of the character encoding issues. For example, to write the key:

```
stream reset.
password := 'very secret password' asByteArrayEncoding: #utf_16.
key writeOn: stream password: password.
```

and to read the key back in

```
stream reset.
RSAPrivateKey readFrom: stream password: password.
```

or

```
stream reset.
PKCS8 readKeyFrom: stream password: password.
```

The default algorithm used for key encryption is the PKCS#5, password-based encrytption using DES and SHA1 used to derive the encryption key from the provided password (identified by the PKCS#5 specification as pbeWithSHA1AndDES-CBC). It is possible to use other algorithms as well. See the documentation of the PKCS5 package for more details about these algorithms. Here we will discuss only how to choose and tune an algorithm for key encryption. Let's take the following example.

```
stream reset.
info := EncryptedPrivateKeyInfo PBE_MD5_DES.
info algorithmIdentifier parameters count: 2048.
info key: key password: password.
info writeOn: stream.
```

We have to work directly with the EnryptedPrivateKeyInfo objects to tune encryption parameters. Choose the desired encryption algorithm by using the corresponding instance creation method, in this case PBE_MD5_DES (as opposed to the default which uses SHA1 instead of MD5). With the information instance in hand, we can tweak specific parameters of the chosen algorithm. In this example we are explicitly increasing the iteration count to 2048. Again, details about these algorithms and their parameters belong to the PKCS#5 realm. Once the we have the algorithm set up the way we want we can store a key in the info instance using the key:password: message. Note that the key is encrypted immediately, so any parameters must be tuned before this step. Finally we need to write the info instance out on the stream.

Reading this is much easier because all the information about the encryption altgorithm and specific parameters is stored along with the encrypted key. Therefore reading this is no different than reading a key encrypted with the default setup.

```
stream reset.
PKCS8 readKeyFrom: stream password: password.
```

COM

COM Automation Wizard

The COM Automation Wizard can now save and restore settings to create a VW COM server image.

GUI Development

FontPolicy No Matching Font Default Action

A block or evaluable action may now be assigned to a FontPolicy to answer the default font to be used in case no font matches a request. If defined, the NoMatchingFontError exception is not raised. Users will define this block if it is preferable for the image to continue operating with their best font guess available rather than halt with an unhandled exception.

The following example defines a noFontBlock that uses the first available font if no font otherwise matches the requested font:

```
policy := Screen default defaultFontPolicy.
policy noFontBlock:[:fontRequested| policy availableFonts first].
```

Pollock Namespace Change

Of special note is that the Pollock namespace has been renamed to Panda, and all Pollock classes have been moved to that namespace. This was done in recognition of the fact that Pollock is only one phase of a much larger GUI framework redesign, which is being called “Panda.”

Web Services

WebServices.Struct Moved

Starting in 7.4.1, WebServices.Struct is a subclass of Protocols.Struct, rather than of Dictionary. The 7.4 release notes mentioned this pending change, though anticipated it for 7.5; the change over has been advanced to this release.

This change may well affect your application code, so careful testing and revision is required.

For further information, refer to the comments on Protocols.Struct in the 7.4 release notes, especially in the Security and Opentalk sections.

Setting SOAPClient Retries and TImeout

WsdlClient now allows you to set a number of retries and timeout. Default values are configurable using Net Client settings (load the NetConfigTool parcel, then use the **Net Client** page). The values are initially set to 5 (not 2, as it had been in SoapHttpBindingDescriptor>> sendAndWaitForReply:.

SoapHttpBindingDescriptor allows setting the exception raised when the retry limit is exceeded. The default exceptions are set in SoapHttpBindingDescriptor class>>defaultRetryExceptionsValue:

```
^(ExceptionSet new: 4)
    add: HttpTimeout;
    add: OSErrorHolder unpreparedOperationSignal;
    add: OSErrorHolder needRetrySignal;
    add: HttpStatusLineError;
    yourself
```

The exception set can be customized, by sending retryExceptions: to the class with a new ExceptionSet.

Resetting the retry exception to the default can be done using a wsdl client:

```
client := MyWsdlClient new.
client resetRetryExceptions.
```

or

```
b := WebServices.WsdlBinding bindingAt: 'bindingName' ifAbsent: [].
b transport retryExceptions: nil.
```

Net Clients

Digest and NTLM Authentication HttpClient

The new class AuthenticationPolicy provides different types of authentication for HTTP messages. An HttpClient creates an instance of the authentication policy when it receives 407/401 messages.

The policy selects a supported authentication scheme from the server challenge, creates an instance of the specific authentication, and adds an authorization field to a request.

The policy will try to handle the server challenge if:

- a user name and password is provided
- the server challenge includes an authentication scheme that is supported by HttpClient

Currently supported authentication schemes are: Basic, Digest, and NTLM. The client side preferences for authentication mechanism are controlled by the authentication order (#authOrder), which can be specified either at the global level (class side) or at the individual instance level.

How to Use AuthenticationPolicy

An HTTP client always sends an unauthorized message first, which results in a challenge response if the site requires authentication/authorization. How the challenge is answered depends on the information available to the HttpClient. Here are a few cases.

1. The user name and password provided before the request, by being set in the HttpClient instance.

   ```
   cl := HttpClient new.
   cl username: 'winUsername' password: 'winPass'.
   reply := cl get: aURI.
   ```

   The authentication scheme will be selected from the server 401/407 reply. The user name and password will then be encoded based on this scheme and the request will be sent to the server.

2. The HttpClient may be configured to respond with a specific authorization scheme:

   ```
   cl := HttpClient new.
   cl username: 'winUsername' password: 'winPass'.
   cl useBasicAuth.
   reply := cl get: aURI
   ```

If the specified scheme is not acceptable to the server, the request will fail. In addition to useBasicAuth, used above, there is also useNTLMAuth to specify the policy.

3. If the user name and password are not provided, the HttpClient will raise an HttpUnauthorizedError exception. Your error handling code will specify the handling.

```
cl := HttpClient new.
[reply := cl get: aURI.
] on: Net.HttpUnauthorizedError
do: [ :ex |
    cl username: 'winUsername' password: 'winPass'.
    ex retry]
```

4. You can also specifying authorization information for a proxy server.

```
proxy := (HostSpec new
            name: 'ntlmAuthProxyServer';
            type: 'http';
            yourself).
proxy netUser:
    (NetUser username: 'winUsername' password: 'winPass').
cl := HttpClient new.
cl
    proxyHost: proxy;
    useProxy: true.
reply := cl get: html.
```

This next sample demonstrates how an HTTPClient uses the Authentication policy.

The HttpClient received the 401 reply from the server. The server can accept the NTLM and Basic authentication.

```
request := HttpRequest readFrom:
'GET http://www.cincomx.com/en/index.asp HTTP/1.1
Host: www.cincom.com:4545
Connection: Keep-Alive' readStream.

reply := HttpResponse readFrom:
'HTTP/1.1 401 Unauthorized
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
WWW-Authenticate: Basic realm="testrealm@host.com"' readStream.
```

To process the 401 message the HttpClient creates an instance of the authentication policy.

```
polBuilder := AuthenticationPolicy new.
```

The default policy order is set as

```
AuthenticationPolicy class>>defaultAuthOrder
    ^Array
        with: NTLMAuthentication
        with: DigestAuthentication
        with: BasicAuthentication
```

This order can be changed at the instance level, if desired:

```
polBuilder policiesOrder: (Array
        with: BasicAuthentication
        with: DigestAuthentication
        with: NTLMAuthentication).
```

Set the authentication information:

```
polBuilder username: 'aUser' password:  'password'.
```

Check if the policy can process the reply challenge.

```
polBuilder acceptChallenge: reply request: request.
```

Based on the authentication policy order the Basic scheme will be selected to authorize the request.

```
polBuilder addAuthorizationTo: request.
```

Application Server

The Application Server now has the ability to pass arbitrary data to an application through either the global or a site specific INI file. Simply add name/value entries to the **[configuration]** section of the appropriate file.

Two methods have been added on the class side of WebConfigurationManager: to obtain the data once server configuration is complete:

configParameterNamed:

configParameterNamed:forSite:

These answer the value of the named parameter from the global INI configuration file or the INI configuration file for the named site respectively, or the empty string if the parameter does not exist.

Note that in a deployment environment, the application server does a lazy configuration when the first request is served. Therefore, if you want your application to start its own servers, you need a postLoad action on your application parcel, or some other mechanism which you can use to invoke:

WebConfigurationManager configureServer

which will initiate the configuration process before a request is received.

In order to be sure all the relevant configuration data is available to your application, if you are planning to provide information with which to configure server listeners, for example, you can register to receive notification of the finishedServerConfiguration event triggered in WebConfigurationManager class >> installConfigurationUsing:.

WinCE Device VM

The problem is that VisualWorks will just crash on some CE device because the device does not like the way we flush the processor cache. The problem does not depend on the processor but on the device manufacturers implementation of the operating system. Therefore we have not found a reliable way to find out, which of our cache flushing routines will work.

Instead the user can configure the VM to use the correct one by adding (or not adding) a resource to the VM.

To add the resource you can use the program ResHacker from $(VISUALWORKS)\packaging\win.

1. Open the executable with **File->Open…**
2. Add the resource with **Action->Add a new resource**, in the dialog, press **Open file with new resource…** and select file **flush.res** (in the codeFixes directory).
3. In the tree view select **1033** and press **Add resource.**
4. Save the modified executable.

Devices that require the resource include most of the machines from Psion and Demolux.

Devices that *must not* have the resource include HP's iPAQ.

DotNET Connect

This version of DotNET Connect fixes several minor bugs and incompatibilities with VW 7.4/7.4.1. It also now works with Microsoft's new compiler.

This release includes two new files:

- DotNETConnect\libraries\ReflectionProxy.dll
- DotNETConnect\Sources-Dll\standard.mainfest

Two DLLs are no longer needed, and have can been removed:

- DotNETConnect\libraries\DotNETProxy.dll
- DotNETConnect\libraries\DotNETReflection.dll

Opentalk

Realigning configuration and component names

Since configurations parallel the broker component hierarchy, it is desirable for the two to follow the same naming conventions. Therefore we have renamed ConnectionOrientedAdaptor to just ConnectionAdaptor, to match the ConnectionAdaptorConfiguration. We also added a backward compatibility alias for the old name to keep old code working but that will be removed in future releases.

Controlling pass mode of indexed instance variables in STST

In previous releases there was no way for an object to control the pass mode of its indexed instance variables. Users would have to write custom marshaling code for their classes to achieve that. In this release we have added a mechanism modeled after the one used for named instance variables (passInstVars). The new method, passModeForIndexedSlot:value:, is invoked for each indexed variable of an object being marshaled with the index and the value of that slot, and is expected to return one of the pass mode symbols, #default, #skip, #value, #reference, etc. These are the same as with the passInstVars method.

Here is an example of how CompiledCode uses this feature to make sure any "embedded" block closures are passed by value and not by reference:

```
CompiledCode>>passModeForIndexedSlot: index value: anObject

    ^(anObject isKindOf: BlockClosure)
        ifTrue: [#value]
        ifFalse: [#default]
```

Graphics

GIF and PNG Image Readers

(ARs 50543 and 50247) The ImageReader framework has been improved to correctly read some GIF and PNG files. Prior to this release, GIF files with local color table definitions could not be read without an exception. PNG files of Color Type 2 that define a color in its palette as transparent now create a correct mask. These errors have been fixed.

ActiveX Browser Plugin

Deployment of the VisualWorks ActiveX Control Plugin has changed to use two CAB files: one for the ActiveX Control DLL and one for the remaining Smalltalk components. This configuration allows you to update the Control (DLL) and the supporting Smalltalk code (your Applet) independently.

Please see plugin/deploy/readme.txt for more information and specific instructions for using the supplied tools to build your plugin deployment files.

To ensure the most recent version of your plugin is used, you must specify a version on the CODEBASE= URL in the OBJECT tag in your HTML, is in this example for the HelloWorld applet.

```
<!-- VisualWorks ActiveX PlugIn -->
<OBJECT ID="VWHelloWorld"
    CLASSID="CLSID:FF48278C-094A-4188-95AA-4B1E03F3163C"
    CODEBASE="http://localhost/plugin-install/vwpluginax.cab#version=
        -1,-1,-1,-1"
    WIDTH="200" HEIGHT="200" ALIGN="BOTTOM">
        <PARAM NAME="PARCEL" VALUE="pcl/HelloWorld.pcl">
        <PARAM NAME="VWOPEN" VALUE="HelloWorldExample">
        <B>The ActiveX PlugIn was not installed!</B>
    </OBJECT>
```

When you update your plugin to a new version, you only recompile the DLL if you need to change the C/C++ code. For updates to anything Smalltalk, you simply rebuild your image and then rebuild/redeploy your CAB files.

To update only Smalltalk bits, rebuild/redeploy both CAB files, including the existing DLL referencing its version in vwpluginax.inf, namely run the following tools in this order:

- mkcabplugin.bat
- mkcatplugin.bat
- mkcabpluginax.bat

(This is the function of the mkaxdeploy.bat tool.)

To update only the DLL, rebuild/redeploy vwpluginax.cab, including the new DLL referencing its new version in vwpluginax.inf, namely run the following tools in this order

- mkcatplugin.bat
- mkcabpluginax.bat

To update both, rebuild/redeploy both CAB files, including the new DLL referencing its new version in vwpluginax.inf, namely run the following tools in this order

- mkcabplugin.bat
- mkcatplugin.bat
- mkcabpluginax.bat

As long as the version is -1,-1,-1,-1 on the object, IE will pull down and install any new CAB file(s). However, the ActiveX Control DLL will not be installed again until its version changes. When it does change, both the version in the DLL and the version in the INF file must match.