Base Language - Feature #2181

implement SSL support for sockets

09/16/2013 12:16 PM - Constantin Asofiei

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	GUI Support for a Complex ADM2 App		
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Feature #1639: implement socket support	**Closed**	**01/15/2013**	**06/21/2013**
Related to Bugs - Bug #3057: bad synchronization in AsyncRequestImpl	**Closed**		
Related to Base Language - Feature #4366: support TLS 1.2 and other updated s...	**Closed**		

History

#1 - 09/16/2013 12:20 PM - Constantin Asofiei

@LowLevelSocketImpl.connectSSL@ is not yet implemented. Following are some findings/implementation notes:

- nosessionreuse: "If specified, the connection does not reuse the SSL session ID when reconnecting to the same SSL-enabled server socket." - couldn't find anything in JSSE which would allow control over the SSLSession attached to the SSLSocket.
- The certificate sent by the server socket needs to be validated. Need to determine if nohostverify option will disable validation completely (including expiration date) or it will just exclude the server host name from the validation.
- Documentation states that OpenEdge keeps the public keys (certificates) in a global store. We can use the same storage facilities offered by the TransportSecurity and SecurityManager.
- Need to determine if the global certificate store is per progress installation or per progress client.

LowLevelSocketImpl.enableSSLConnections is not yet implemented. Following are some findings/implementation notes:

- nosessioncache - this disables the SSL session cache. The SSLContext or SSLSessionContext classes don't seem to offer APIs to disable the SSL session cache.
- sessiontimeout represents "The maximum amount of time, in seconds, that the server waits before it rejects a SSL client’s request to resume a session. The default value is 180 seconds." This parameter looks like is something which should be set by the client and not by the server, as (CA thinkgs) it means "if client is idle more than x seconds, than terminate connection". SSLSocket, SSLContext or SSLSessionContext don't have APIs which would suggest that would set a "disconnect idle client after" value.
- Documentation states that OpenEdge keeps the private keys in a global store. We can use the same storage facilities offered by the TransportSecurity and SecurityManager, but we will need to provide a facility to encrypt the key's password.
- Need to determine if the global store is per progress installation or per progress client.
- there is a default_server key with the default password set to password. This is used when connecting in SSL mode, but no key alias or password is specified. Confirmation is needed if these defaults can be configurable in OpenEdge.
- the password specified via keyaliaspasswd option in ENABLE-CONNECTIONS is encrypted. 4GL has a genpassword utility to encrypt a password; the assumption is that it uses a master password (as salt) and the given password to generate some hash value. In our implementation, we will need to encrypt the password too, and I think we can use a similar approach: we can keep the (keyalias, keypassword) in some separate directory node/configuration file and just check the hash sent by the client against the hash obtained from the master password and the associated alias key; if these match, use the real keypassword to decript the private key.

#2 - 10/29/2015 04:42 PM - Greg Shah

- Assignee set to Ovidiu Maxiniuc

- Target version set to Milestone 12

Please start by investigating the open questions listed above. Try to find ways to control the nosessionreuse, nosessioncache and sessiontimeout from J2SE. If there are no options available from pure Java, we will discuss alternative approaches.

#3 - 11/03/2015 05:12 PM - Greg Shah

-------- Forwarded Message --------
Subject: Re: status report
Date: Tue, 03 Nov 2015 23:09:31 +0200
From: Ovidiu Maxiniuc <om@goldencode.com>
To: ges@goldencode.com, Eric Faulhaber <ecf@goldencode.com>

Greg,

I tried to create the helper tools.
Creating a server in P4GL seems quite easy. Even if I could not find any examples, I was able to create a server similar to 'echo'. I could not switch it to use ssl.
The server fails to open sockets with errors similar to these:
Unknown transport error received -51. (5490)
SSL Server initialisation error SSL initialization error (12063)

I also tried to trick the client to connect to a ssl-enabled http server, but this fails because
Secure Socket Layer (SSL) failure. error code -54: unable to get local issuer certificate: for facacbc6.0 in d:\oe102b\certs (9318)
Connection failure for host www.verisign.com port 443 transport TCP. (9407)

I updated the java sources somehow blindly. At this moment the ui is not working for some unknown reason. I get an exception like this:

Caused by: java.lang.NullPointerException
at com.goldencode.p2j.ui.client.Frame.doLayout(Frame.java:1335)
at com.goldencode.p2j.ui.client.Frame.postprocess(Frame.java:3189)
at com.goldencode.p2j.ui.client.WidgetRegistry.pushDefinition(WidgetRegistry.java:400)
at com.goldencode.p2j.ui.chui.ThinClient$18.run(ThinClient.java:7343)
at com.goldencode.p2j.ui.chui.ThinClient.eventBracket(ThinClient.java:13705)
at com.goldencode.p2j.ui.chui.ThinClient.eventDrawingBracket(ThinClient.java:13646)
at com.goldencode.p2j.ui.chui.ThinClient.pushOneDef(ThinClient.java:7331)
at com.goldencode.p2j.ui.chui.ThinClient.pushScreenDefinition(ThinClient.java:7279)

when the (default) frame is first open.

In debugger I was able to inspect internal states ssl data-structures and they seem fine.

Tomorrow, I will go back to researching the ABL References, Programming Interfaces and Core Business Services pdf-s.
I also intend to approach Constantin. I hope he is less busy than the previous days.

Thanks,
Ovidiu

On 11/02/2015 09:46 PM, Greg Shah wrote:

> Ovidiu,
>
> > One of big issue is that I don't have (or I am not aware) of a server that
> > provides the service to connect to (including from P4GL), to see the difference in behavior of the options to be implemented.
>
> I think it is best to create a simple Java SSL sockets server. Use code from P2J to make it easier. The server can be a simple "echo" server that just sends back the data that was read. If you limit the dependencies, it will be easy to setup on lincon01 or windev01 so that you can test from the 4GL too.
>
> OR: write a simple echo server in the 4GL that supports SSL. This one has the advantage that we can test both server and client socket support for SSL.
>
> Greg

#4 - 11/03/2015 05:18 PM - Greg Shah

> Creating a server in P4GL seems quite easy. Even if I could not find any examples, I was able to create a server similar to 'echo'. I could not switch it to use ssl.
> The server fails to open sockets with errors similar to these:
> Unknown transport error received -51. (5490)
> SSL Server initialisation error SSL initialization error (12063)

Please see the "OpenEdge Development: Programming Interfaces" document. There is a short section on sockets.

To set the server socket into SSL mode, I believe that you have to pass some options in the connection parameters argument to the ENABLE-CONNECTIONS() method.

> I also tried to trick the client to connect to a ssl-enabled http server, but this fails because Secure Socket Layer (SSL) failure. error code -54: unable to get local issuer certificate: for facacbc6.0 in d:\oe102b\certs (9318)
> Connection failure for host www.verisign.com port 443 transport TCP. (9407)

Did you use the SSL options in the connection parameters argument to the CONNECT() method?

If you did then perhaps the issue is with the currently configured well known certificates provided by OpenEdge. I suspect they have a way to update their equivalent to the Java "cacerts" list.

> I updated the java sources somehow blindly. At this moment the ui is not working for some unknown reason. I get an exception like this:
>
> Caused by: java.lang.NullPointerException

You'll have to post your testcase so I can get an idea of what the UI code is trying to do. Are you testing this in the trunk?

#5 - 11/04/2015 06:09 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> Creating a server in P4GL seems quite easy. Even if I could not find any examples, I was able to create a server similar to 'echo'. I could not switch it to use ssl.

Please see the "OpenEdge Development: Programming Interfaces" document. There is a short section on sockets.

That was my primary source of information. However, I chosen to write some parts slightly different.

> To set the server socket into SSL mode, I believe that you have to pass some options in the connection parameters argument to the ENABLE-CONNECTIONS() method.

The main switch is -ssl, the -keyalias and the rest are optional.

> I also tried to trick the client to connect to a ssl-enabled http server, but this fails because Secure Socket Layer (SSL) failure. error code -54: unable to get local issuer certificate: for facacbc6.0 in d:\oe102b\certs (9318)
> Connection failure for host www.verisign.com port 443 transport TCP. (9407)

Did you use the SSL options in the connection parameters argument to the CONNECT() method?

Yes, I did.

> If you did then perhaps the issue is with the currently configured well known certificates provided by OpenEdge. I suspect they have a way to update their equivalent to the Java "cacerts" list.

This is why I tried to access the verisign.com server. I noticed in the d:\oe102b\certs some certificates related to this root authority. I hope that the certificates are already there. I read that there is a certutil command-line tool provided by OpenEdge for adding a new certificate. I could not find it on windev01 in the bin directory of the OE installation folder where I expected it to be located.

> I updated the java sources somehow blindly. At this moment the ui is not working for some unknown reason. I get an exception like this: Caused by: java.lang.NullPointerException

You'll have to post your testcase so I can get an idea of what the UI code is trying to do. Are you testing this in the trunk?

I committed my 2 main files to testcases, in uast/sockets/ssl. I used the trunk as base sources for investigations/fixing implementation. My (temporary) changes are committed to 2181a.

#6 - 11/04/2015 07:03 AM - Ovidiu Maxiniuc

Sorry, I found the certutil and pkiutil in the D:\oe102b\bin. I was expecting for executables, but they are windows BAT scripts that invoke perl scripts.

Now I can see that the default default_server server certificate is installed. According to docs, *If you use the default_server server certificate, it also has a default password that you do not need to specify.* Yet, when I start my ssl server, it fails:

```
Unknown transport error received -51. (5490)
SSL Server initialisation error SSL initialization error (12063)
```

This is not very useful, looking at this error I found the following description: *An internal consistency error occurred within PROGRESS AppServer networking. Please report this error to Progress Software Corporation.*

#7 - 11/04/2015 01:50 PM - Ovidiu Maxiniuc

Now I have some answers to questions from 1st note:
Constantin Asofiei wrote:

> LowLevelSocketImpl.connectSSL is not yet implemented. Following are some findings/implementation notes:
>
> - nosessionreuse: "If specified, the connection does not reuse the SSL session ID when reconnecting to the same SSL-enabled server socket." - couldn't find anything in JSSE which would allow control over the SSLSession attached to the SSLSocket.

The javax.net.ssl.SSLContext class holds all of the state information shared across all objects created under that context, including the session state. I understand that creating a socket in a new context (as opposed to the default one) will create another SSL session.

- The certificate sent by the server socket needs to be validated. Need to determine if nohostverify option will disable validation completely (including expiration date) or it will just exclude the server host name from the validation.

I believe that only the host name is checked. This is what the Core Business Services say: *Verify the SSL server host name*. Also, If the date interval is wrong, the server won't even start. This was the reason I got the error in note 6 above, the default certificate on windev01 is expired from Feb 22 22:04:12 2014 GMT.

- Documentation states that OpenEdge keeps the public keys (certificates) in a global store. We can use the same storage facilities offered by the TransportSecurity and SecurityManager.
- Need to determine if the global certificate store is per progress installation or per progress client.

In OpenEdge 11.2B, the private keys are stored in %DLC%\keys and public certificates in %DLC%\certs. I don't understand, can the Progress client be installed standalone? To install the certificate, you need the utilities in %DLC%\bin. I take as granted that connecting to a socket on another machine means connection to another installation. So in the end, the store is per progress installation.

> LowLevelSocketImpl.enableSSLConnections is not yet implemented. Following are some findings/implementation notes:
>
> - nosessioncache - this disables the SSL session cache. The SSLContext or SSLSessionContext classes don't seem to offer APIs to disable the SSL session cache.

The SSLSessionContext has a sessionCacheSize that looks similar. Setting it to 0 means unlimited, but if set to 1, the engine should not cache client sessions. (This code was already there, is not the needed behavior ?)

- sessiontimeout represents "The maximum amount of time, in seconds, that the server waits before it rejects a SSL client’s request to

resume a session. The default value is 180 seconds." This parameter looks like is something which should be set by the client and not by the server, as (CA thinkgs) it means "if client is idle more than x seconds, than terminate connection". SSLSocket, SSLContext or SSLSessionContext don't have APIs which would suggest that would set a "disconnect idle client after" value.

I tries to duplicate the client idling after connection but my client was not disconnected. I think that this option means how long the server will keep the session 'warm' after the client is disconnected. I have no idea how to test this.
The 4GL docs specifies that this value is the amount of tine the server *waits before it rejects a SSL client's request to resume a session*. I am a little puzzled here. As I understand the session is kept alive after the last SSL socket disconnects for a number of seconds, and only after this timeout the session is destroyed. If a client opens a SSL socket meanwhile, the connection is reused and probably the negotiations are omitted, increasing networking performance and lowering CPU usage. Otherwise the SSL handshake is required and negotiations start all over again. I think that this is exactly what the SSLSessionContext.setSessionTimeout does.

- Documentation states that OpenEdge keeps the private keys in a global store. We can use the same storage facilities offered by the TransportSecurity and SecurityManager, but we will need to provide a facility to encrypt the key's password.

I understand that this global store is the %DLC%\keys and each pair key/certificate is stored in a single file. The password must be encrypted, and the genpassword utility is used for that purpose. The encrypted string is already in source code (or maybe read from any other place). Nothing else to do here.

- Need to determine if the global store is per progress installation or per progress client.

See above.

- there is a default_server key with the default password set to password. This is used when connecting in SSL mode, but no key alias or password is specified. Confirmation is needed if these defaults can be configurable in OpenEdge.

On windev01, the default_server alias has a expired datestamp. I managed to add a self-signed key/certificate. The pkiutil allows an alias to be removed and another key/cert to be installed. I believe this is just a convention and the certificate can be refreshen. But, since the P4GL installation is accessible for all developers, I did not dare to do the replace with a self-signed key.

- the password specified via keyaliaspasswd option in ENABLE-CONNECTIONS is encrypted. 4GL has a genpassword utility to encrypt a password; the assumption is that it uses a master password (as salt) and the given password to generate some hash value. In our implementation, we will need to encrypt the password too, and I think we can use a similar approach: we can keep the (keyalias, keypassword) in some separate directory node/configuration file and just check the hash sent by the client against the hash obtained from the master password and the associated alias key; if these match, use the real keypassword to decript the private key.

At the time we have access to the password, it should be already encrypted. We need a full-compatibility here. I don't think that the password is encrypted with a reversible algorithm. So the original is most likely not accessible. Because the output is very different from P4GL encrypt, I looked into genpassword source and they are using a java program to do the encryption (com.progress.common.util.genPassword). Probably a hashing function from JSSE.

#8 - 11/17/2015 04:42 PM - Ovidiu Maxiniuc

Task branch 2181a was updated and I rebased it to 2677a/11056. It is ready for runtime testing and reviewing.
Important notes:

- the location where 4GL stores private keys is <OE-Install-Dir>/keys. For now, P2J searches for key stores in current folder. I tried to use independent files for each imported private key, as P4GI does. I don't think the folder is configurable in P4GL, but we can add a directory entry for this location;
- to have a default key (with default_server alias) I generated one, self-signed and imported with following commands:

```
 $ keytool -genkey -keyalg RSA -alias default_server -keystore default_server.store -storepass <password>
-validity 48 -keysize 1024
 $ keytool -exportcert -keystore default_server.store -alias default_server > default_server.cer
 $ keytool -import -alias default_server -file default_server.cer -keystore trusted-cert.store
```

- the location for certificates is also in current location (in P4GL is <OE-Install-Dir>/certs). They store certificates as independent PEM files, named using some kind of hashing (similar to CRC32) I could not identify. I didn't find other uses of this alias. The only occurrences is when printing the host name validation failure. Since the string is specific to each certificate, I guess the value is not critical. Because of these and also to simplify the java code, all available certificates are saved in a single storage, hardcoded as trusted-cert.store;
- I was able to reverse engineer the password encryption. The algorithm is quite naive, using a symmetric bit operation. I implemented a utility class (SymmetricEncryption) to decrypt the password provided in P4GL source code that also exposes a command-line utility very similar to genpassword of Progress.

LE: GES removed the hard coded password from this entry.

#9 - 11/18/2015 12:31 PM - Greg Shah

Some questions about the SymmetricEncryption class:

1. Is it truly compatible with Progress?

2. The "key" used is really the text "PROGRESS"?

3. How did you determine the core algorithm and key? Remember that we are not allowed to disassemble the Progress' binaries.

#10 - 11/18/2015 12:49 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> Some questions about the SymmetricEncryption class:
>
> 1. Is it truly compatible with Progress?

From my testcases, yes, it should be. I did not do a very extensive testing but I am pretty sure I identified the algorithm used for password encryption.

2. The "key" used is really the text "PROGRESS"?

Yes. Check the following on windev01:

```
C:\Users\om>D:\oe102b\bin\genpassword -password PROGRESS
```

It will XOR itself and print "0000000000000000".

3. How did you determine the core algorithm and key? Remember that we are not allowed to disassemble the Progress' binaries.

Well, using classic attacks. This is an encoding system, not a real encryption one.
We are not allowed to disassemble it but we can run it as many times and using any parameter we need.
This is a (very) basic encryption method - substitution. This was clear to me when I realized that the length of coded password was twice the length of the plain input string. Because the output was composed only from hex digits, I assumed they are probably grouped in blocks of 2 to obtain a coded byte - this is usual for representing a set of possible resulting non-printable ASCII characters. Another option would have been BASE64, but this was evidently not the case.

Using normal passwords like: "a^23lO3*=kd" will not reveal much of the inner black-box. However, using some simpler patterns will show the internals.

```
genpassword -password aaa    ->    31332e    ->    49 51 46
genpassword -password bbb    ->    32302d    ->    50 48 45
genpassword -password ccc    ->    33312c    ->    51 49 44
..
genpassword -password AAA    ->    11130e    ->    17 19 14
genpassword -password BBB    ->    12100d    ->    18 16 13
genpassword -password CCC    ->    13110c    ->    19 17 12
```

Notice the digits, they are consecutive (as the input string). I guessed it was a ROR or XOR or a similar symmetrical coding. To identify the length of the key I used a long password to see the point where the generated pattern start repeating itself:

```
genpassword -password PPPPPPPPPPPPPPPPPPPPPPPP    ->    00021f1702150303 00021f1702150303 00021f1702150303 00
```

Then I tried to obtain as many 0 -s using different letters. It was quite a surprise to obtain the PROGRESS key.

To have the confirmation I run:
genpassword -password " " -> 70726f677265737370726f which, if we convert hex to decimals, is:

```
112 p
114 r
111 o
103 g
114 r
101 e
115 s
115 s
112 p
114 r
111 o
```

It didn't take me an half an hour like writing this note, but finally I was happy with the result.

#11 - 11/18/2015 01:01 PM - Greg Shah

Very good work! Once again, Progress' developers have displayed a stunning lack of any understanding about security.

From windev01:

```
PS C:\Users\ges> genpassword -password P
00
PS C:\Users\ges> genpassword -password PR
0000
PS C:\Users\ges> genpassword -password PRO
000000
PS C:\Users\ges> genpassword -password PROG
00000000
PS C:\Users\ges> genpassword -password PROGR
0000000000
PS C:\Users\ges> genpassword -password PROGRE
000000000000
PS C:\Users\ges> genpassword -password PROGRES
00000000000000
PS C:\Users\ges> genpassword -password PROGRESS
0000000000000000
PS C:\Users\ges> genpassword -password PROGRESSP
00000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPR
000000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPRO
0000000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPROG
00000000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPROGR
000000000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPROGRE
0000000000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPROGRES
00000000000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPROGRESS
000000000000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPROGRESSP
0000000000000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPROGRESSPR
00000000000000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPROGRESSPRO
000000000000000000000000000
PS C:\Users\ges> genpassword -password PROGRESSPROGRESSPROG
0000000000000000000000000000
```

#12 - 11/18/2015 05:00 PM - Greg Shah

Code Review Task Branch 2181a Revision 11059

1. We need to discuss the location and approach for the certificate/key store as used in LowLevelSocketImpl.connectSsl() and/or LowLevelSocketListenerImpl.enableSSLConnections(). Some questions:

- Should we store this data in the P2J server's directory instead of in a client-side directory? In particular, what behavior would we be missing if we did this? How reasonable is it for specific users to be directly manipulating their certificate/key stores?
- If we do store it in the client-side filesystem, then we definitely would want control over the location using a value in the server's directory. It would be OK to default it to the current dir.
- If we do store it in the client-side filesystem, then we definitely would want control over the keystore filename. It can be defaulted to "trusted-cert.store" but it should be customizable for each installation.

2. We should not have a hard coded password in LowLevelSocketImpl.connectSsl() or LowLevelSocketListenerImpl.enableSSLConnections(). Let's discuss the problem we are trying to solve and come up with an approach.

3. The certId = "a4b3c2d1.0" in LowLevelSocketImpl.connectSsl() seems to only be used in the error message. Where does Progress come up with this?

4. In LowLevelSocketImpl.connectSsl(), you can remove the TODO: this is not yet implemented. notice in the javadoc. Also the item "Need to determine if the global certificate store is per progress installation or per progress client." has already been addressed, right?

#13 - 11/19/2015 07:08 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> Very good work! Once again, Progress' developers have displayed a stunning lack of any understanding about security.

The the encoding schema used for password is, indeed, a low-security one. I guess it is meant only to hide the real password in the source code implementing some security by obscureness. I guess the passwd should be stored to some files. On the other hand, it had to be symmetrical to be able to extract it back faster, which reduces a lot the alternatives.

I am sorry I could not identify the hashing algorithm used by the certutil -import tool when creating the alias. It looks like a crc32 computed on certificate itself (only the digits between the PEM headers) but my attacks failed. The best guess is a hash similar to openssl x509 -hash -noout, but the values were not a match (I also tried the java.util.zip.CRC32, too). To note that some minor tampers (changing a single char) of the certificate will not affect the result in both P4GL and openssl tools, so it is not a crc32 (which would diffuse and alter completely the output).
Anyway, as documented in code, this is used only for display as I can say, and chose to hardcode it to a constant value in that case.

#14 - 11/19/2015 08:31 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> Code Review Task Branch 2181a Revision 11059
>
> 1. We need to discuss the location and approach for the certificate/key store as used in LowLevelSocketImpl.connectSsl() and/or LowLevelSocketListenerImpl.enableSSLConnections(). Some questions:
>
> - Should we store this data in the P2J server's directory instead of in a client-side directory? In particular, what behavior would we be missing if we did this? How reasonable is it for specific users to be directly manipulating their certificate/key stores?
> - If we do store it in the client-side filesystem, then we definitely would want control over the location using a value in the server's directory. It would be OK to default it to the current dir.
> - If we do store it in the client-side filesystem, then we definitely would want control over the keystore filename. It can be defaulted to "trusted-cert.store" but it should be customizable for each installation.

Both <OE-Install>/Certs and <OE-Install/keys> are fixed/global. This means their content is shared by all users that access P4GL installation. It is difficult to say whether this is good or bad: if one user adds a private key, the other can access it, but they lack the password. OTOH, if one user adds a trusted certificate, it will be trusted for all users in the system. These suggest that we should store the key/certs in same common place. I don't think we loose current capability storing them in directory as BASE64 or HEX encoding of the binary stores. On the other hand, storing only the location of the binary stores from HDD would give us greater flexibility and less steps when doing the install and update procedures (not required to encode the store and insert the ASCII string into directory.xml).

> 2. We should not have a hard coded password in LowLevelSocketImpl.connectSsl() or LowLevelSocketListenerImpl.enableSSLConnections(). Let's discuss the problem we are trying to solve and come up with an approach.

The keystore password is not required/existent in P4GL. We can drop it completely. P4GL uses only key-level passwords. The keystore password is Java-specific.

> 3. The certId = "a4b3c2d1.0" in LowLevelSocketImpl.connectSsl() seems to only be used in the error message. Where does Progress come up with this?

Progress uses this hash as filename for trusted certificates. They are computed when the certificate is imported with certutil. On connection/handshake errors, they are just printed. However, if there is no matching certificate installed to match the server's, it will still print some 'alias'. Another strange thing is, what happens when a collision occur? It is quite possible to have two identical hashes for different CN-s on the 32bit space.

> 4. In LowLevelSocketImpl.connectSsl(), you can remove the TODO: this is not yet implemented. notice in the javadoc. Also the item "Need to determine if the global certificate store is per progress installation or per progress client." has already been addressed, right?

Right. Done. Changes will appear in task branch with the next commit. I will update the javadoc as we clarify the certificate managements from point 1.

> Both <OE-Install>/Certs and <OE-Install/keys> are fixed/global. This means their content is shared by all users that access P4GL installation. It is difficult to say whether this is good or bad: if one user adds a private key, the other can access it, but they lack the password. OTOH, if one user adds a trusted certificate, it will be trusted for all users in the system.

My only concern here is that in GUI, each user may have their own personal installation of Progress, since it is usually installed right onto their desktop systems. For this reason, I suspect we must allow per-user key/cert stores.

On the other hand, having a centralized store (in the directory) for these will often be much easier for deployment purposes since it can eliminate the need for a file-system dependency.

> These suggest that we should store the key/certs in same common place.

I agree with this. We probably should support our normal idea of a precedence of lookups (e.g. account, then group, then server then default). This would allow certificates to be installed in a single entry for all users, while still letting individual keys be possible.

> On the other hand, storing only the location of the binary stores from HDD would give us greater flexibility and less steps when doing the install and update procedures (not required to encode the store and insert the ASCII string into directory.xml).

Yes, I agree we should allow this too. I guess both directory encoded and the option to load from a configured location in the client file system gives us the full capability.

> The keystore password is not required/existent in P4GL. We can drop it completely. P4GL uses only key-level passwords.

Good.

> > 3. The certId = "a4b3c2d1.0" in LowLevelSocketImpl.connectSsl() seems to only be used in the error message. Where does Progress come up with this?
>
> Progress uses this hash as filename for trusted certificates. They are computed when the certificate is imported with certutil. On connection/handshake errors, they are just printed. However, if there is no matching certificate installed to match the server's, it will still print some 'alias'. Another strange thing is, what happens when a collision occur? It is quite possible to have two identical hashes for different CN-s on the 32bit space.

Strange. Is "a4b3c2d1.0" an actual hash that was generated? Please show some example mappings between CN and the hash created by Progress.

#16 - 11/20/2015 09:16 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> Both <OE-Install>/Certs and <OE-Install/keys> are fixed/global. This means their content is shared by all users that access P4GL installation. It is difficult to say whether this is good or bad: if one user adds a private key, the other can access it, but they lack the password. OTOH, if one user adds a trusted certificate, it will be trusted for all users in the system.

My only concern here is that in GUI, each user may have their own personal installation of Progress, since it is usually installed right onto their desktop systems. For this reason, I suspect we must allow per-user key/cert stores.
On the other hand, having a centralized store (in the directory) for these will often be much easier for deployment purposes since it can eliminate the need for a file-system dependency.

> These suggest that we should store the key/certs in same common place.

I agree with this. We probably should support our normal idea of a precedence of lookups (e.g. account, then group, then server then default). This would allow certificates to be installed in a single entry for all users, while still letting individual keys be possible.

> On the other hand, storing only the location of the binary stores from HDD would give us greater flexibility and less steps when doing the install and update procedures (not required to encode the store and insert the ASCII string into directory.xml).

Yes, I agree we should allow this too. I guess both directory encoded and the option to load from a configured location in the client file system gives us the full capability.

As I understand, P2J server will run on a dedicated box and and the GUI clients will connect 'remote' using installations on individual workstations. In this case, we should keep the stores in directory.xml because the filepath for each user is different. As a bonus, if the user connects from other workstation, (s)he will obtain the store key/cert when needed. What seems clear is that, if the stores are kept on filesystem, it must be client-local path.
However, in the case of the directory-embedded stores, we need to decide, if the keys/certs are transferred to client with open API or it should be obtained on request, on a separate message between P2J components.

> The keystore password is not required/existent in P4GL. We can drop it completely. P4GL uses only key-level passwords.

Good.

I did some more research here. Apparently, although a keystore can be programmatically created without password (better said with empty password), since JSDK 1.3, you cannot create a keystore with a blank password with keytool utility (also it *"must be at least 6 characters long"*). So, unless we create a separate utility for importing certs, we are stuck with a minimal password.

> 3. The certId = "a4b3c2d1.0" in LowLevelSocketImpl.connectSsl() seems to only be used in the error message. Where does Progress come up with this?

> Progress uses this hash as filename for trusted certificates. They are computed when the certificate is imported with certutil. On connection/handshake errors, they are just printed. However, if there is no matching certificate installed to match the server's, it will still print some 'alias'. Another strange thing is, what happens when a collision occur? It is quite possible to have two identical hashes for different CN-s on the 32bit space.

Strange. Is "a4b3c2d1.0" an actual hash that was generated? Please show some example mappings between CN and the hash created by Progress.

I'm not sure the hash is generated from CN. Could be, I must check this out. The "a4b3c2d1.0" is just a string I used, not an actual value obtained from Progress. Here are some true hashes from windev01:

```
412bea73    CN=Thawte Personal Premium CA
c33a80d4    CN=Thawte Premium Server CA
ddc328ff    CN=Thawte Server CA
d9855a82    CN=Progress Server Certificate Authority (expired since Feb 22 21:18:04 2014 GMT)
7fbe1bee    CN=www.google.co.uk (imported by me)
dcdd9741    CN=www.google.com (imported by me)
e5592c6c    CN=localhost (my self-signed)
```

Other, like 18d46017, do not have a CN declared but the organization: O=VeriSign.

One of the hashes I could obtain is 594f1775.0. This is not listed in imported certs. The server I intended to connect is www.google.co.uk, whose certificate I previously imported using certutils and it is listed as 7fbe1bee alias.

#17 - 11/23/2015 03:44 PM - Greg Shah

> As I understand, P2J server will run on a dedicated box and and the GUI clients will connect 'remote' using installations on individual workstations.

This is one case. For web clients, the client processes will be spawned on a server. That server will often be the same system on which the P2J server runs. But it may be a separate box too.

On the other hand, customers may have some reason that the centralized model doesn't work. I think we should implement both.

> What seems clear is that, if the stores are kept on filesystem, it must be client-local path.

Yes.

This path could be "well known" so that each client needs the same path.

> However, in the case of the directory-embedded stores, we need to decide, if the keys/certs are transferred to client with open API or it should be obtained on request, on a separate message between P2J components.

I'm not sure we care too much. The 4GL code doesn't really access the keys directly, right? If I understand correctly, it is more like something the runtime looks up and uses when needing to establish a connection or listen on a socket. Keep the solution as simple as possible.

> So, unless we create a separate utility for importing certs, we are stuck with a minimal password.

When we store keys in the directory, we aren't using the Java keystore. We already have code to turn those into key/truststores.

When we store in the filesystem, we may want to keep that as close to the Progress approach as possible, so that it is possible for customers to easily just copy their current files to the new installation. Do they just keep each key in a separate file that just has the key as binary contents?

#18 - 11/24/2015 03:21 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> So, unless we create a separate utility for importing certs, we are stuck with a minimal password.

> When we store keys in the directory, we aren't using the Java keystore. We already have code to turn those into key/truststores.

I see. Anyway, Java keytool is not able to import directly PEM keys into keystore. To get a private key from P4GL into a Java keystore it's a 2-step process. The key must be exported using openssl pkcs12 -export and then the intermediary store converted to java using keytool -importkeystore.

> When we store in the filesystem, we may want to keep that as close to the Progress approach as possible, so that it is possible for customers to easily just copy their current files to the new installation.

I am not sure if the key/certificate import is just a matter of copying files. I wonder if they keep some kind of database table of the available keys.

> Do they just keep each key in a separate file that just has the key as binary contents?

P4GL keeps each key in a separate file. A copy of the public certificate is also stored there, but the SSL client will not access it from there. All their stores files use PEM format / ASCII text encoding.

#19 - 11/24/2015 02:42 PM - Ovidiu Maxiniuc

After further investigations, I found a solution to have the trusted certificates AND private keys exactly as P4GL. This means we can copy the exact files form OE keys/certs folders to our location designed in directory.xml and P2J should handle them (at least the certs because the private keys are

machine-dependent). Implementation is in progress.

Meanwhile, I also managed to invoke a WEB Service (#2687) over SSL. The only issue is choosing the cert location wisely because the normal secure socket connections are open from p2j client and the web services are invoked from P2J server side. The connection/ssl-handshaking code is common.

#20 - 11/24/2015 02:44 PM - Greg Shah

> The only issue is choosing the cert location wisely because the normal secure socket connections are open from p2j client and the web services are invoked from P2J server side.

Web services should not be invoked on the server side. It should be client based, just like the sockets support.

#21 - 11/24/2015 02:56 PM - Ovidiu Maxiniuc

Are you sure?
I don't remember to encounter anything specified in docs. I must think of a solution for detecting this.
Calling a web-service in P4GL is like calling a (remote) procedure. To invoke the web service from client side, we need to 'move' all input parameters there and then back to P2J server after the operation completes.

#22 - 11/24/2015 03:16 PM - Greg Shah

> Are you sure?

Sadly, yes.

> I don't remember to encounter anything specified in docs. I must think of a solution for detecting this.

In the 4GL there is only the client side. They have no server side. All such code is implicitly executed in the client. We move anything possible to the server, but the problem here is that any given application may be dependent upon that client-side behavior in order to work. For example, the application may depend on a specific firewall/routing setup that makes the destination web service visible only from the client.

> Calling a web-service in P4GL is like calling a (remote) procedure. To invoke the web service from client side, we need to 'move' all input parameters there and then back to P2J server after the operation completes.

I understand.

#23 - 11/24/2015 03:21 PM - Constantin Asofiei

Greg Shah wrote:

> The only issue is choosing the cert location wisely because the normal secure socket connections are open from p2j client and the web services are invoked from P2J server side.

Web services should not be invoked on the server side. It should be client based, just like the sockets support.

Greg, currently P2J implementation for web services is server-side, not client-side. If we really need for the web-services to be invoked from the client-side (like you mentioned in note 22), then some of the APIs in WebServiceHelper (like invoke) need to be moved to the client-side.

#24 - 11/24/2015 04:41 PM - Greg Shah

> The only issue is choosing the cert location wisely because the normal secure socket connections are open from p2j client and the web services are invoked from P2J server side.

Web services should not be invoked on the server side. It should be client based, just like the sockets support.

> Greg, currently P2J implementation for web services is server-side, not client-side. If we really need for the web-services to be invoked from the client-side (like you mentioned in note 22), then some of the APIs in WebServiceHelper (like invoke) need to be moved to the client-side.

I understand. While many use cases might work fine, we cannot guarantee that all use cases will work properly unless we resolve this.

#25 - 01/25/2016 02:47 PM - Greg Shah

- Status changed from New to WIP

#26 - 03/29/2016 05:19 PM - Greg Shah

Code Review Task Branch 2181a Revision 10995

This is really good work.

1. LowLevelSocketImpl.initClientSideSSL() has a ProtocolSocketFactory instance that creates sockets using a custom port number, but then there is this code which hard codes the port to 443:

```
Protocol.registerProtocol("https", new Protocol("https", factory, 443))
```

Is that a conflict or problem?

2. In LowLevelSocketImpl.createSslSocket() we need to use customer-specific configuration to find the keyStoreName and the keyStorePass[].

It is OK to default the keyStoreName BUT we should allow the customer to override this file name with a bootstrap config value or override from the command line.

The keyStorePass[] must not be hard coded. If not specified in the bootstrap config (or override from the command line), we could assume no password at all, which is really not much different to hard coding <password>, except that it doesn't look as bad. Or we can just say: you must specify a password in the bootstrap config.

We already are forcing the customer to manually migrate their certs to the Java keystore. Having these be forced to have a specific password is not too much of an expectation.

3. The SSL socket creation is noted as using TLSv1. Which specific version is it? v1.0 is known to be insecure in many conditions and v1.1 is insecure in some configurations. Only v1.2 is known to be reasonable secure at this time. Does the 4GL allow control over this? Can you pick your cipher suite in the 4GL? How does one mitigate security issues with poor/insecure cipher suites in the 4GL?

4. Same issue as item 1 but for LowLevelSocketListenerImpl. The keyStorePass must not be hard coded. At least the path for the keyStoreName needs to be configurable by bootstrap config.

5. Same issue as item 3 but for LowLevelSocketListenerImpl. What protocol suite is really in use there and what ciphers are possible? How does this match the 4GL?

6. SymmetricEncryption.asBytes() should have the open curly brace on the next line.

7. In SocketListenerImpl.enableConnections(), what is the "20333c34252a2137" default password used for? I'm trying to figure out if this is something we need to match from the 4GL or if we can get rid of this.

8. In WebServiceHelper.invoke() the javadoc has // TBA.

LE: GES removed the hard coded password from this entry.

#27 - 03/29/2016 05:20 PM - Greg Shah

Constantin: please review this.

#28 - 03/30/2016 05:08 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> Code Review Task Branch 2181a Revision 10995
>
> 1. LowLevelSocketImpl.initClientSideSSL() has a ProtocolSocketFactory instance that creates sockets using a custom port number, but then there is this code which hard codes the port to 443:
> [...]
> Is that a conflict or problem?

No. The first parameter, "https" from registerProtocol is just an identifier for Protocol mapping. According to documentation, the Protocol first parameter is the one that identifies the actual scheme *(e.g. http, https)* used by HttpClient 3.x to identify the secured connections. The last parameter 443 is the default post, if none is specified in connection URL.

> 2. In LowLevelSocketImpl.createSslSocket() we need to use customer-specific configuration to find the keyStoreName and the keyStorePass[]. It is OK to default the keyStoreName BUT we should allow the customer to override this file name with a bootstrap config value or override from the command line.
> The keyStorePass[] must not be hard coded. If not specified in the bootstrap config (or override from the command line), we could assume no password at all, which is really not much different to hard coding <password>, except that it doesn't look as bad. Or we can just say: you must specify a password in the bootstrap config.
> We already are forcing the customer to manually migrate their certs to the Java keystore. Having these be forced to have a specific password is not too much of an expectation.

I understand. I will add a client-side bootstrap config for both key store and password. I will continue my attempts to read OE certificates in my free time. With some luck, we will support the original ones just by copying them from OE/certs/ to P2J installation.

> 3. The SSL socket creation is noted as using TLSv1. Which specific version is it? v1.0 is known to be insecure in many conditions and v1.1 is insecure in some configurations. Only v1.2 is known to be reasonable secure at this time. Does the 4GL allow control over this? Can you pick your cipher suite in the 4GL? How does one mitigate security issues with poor/insecure cipher suites in the 4GL?

I will some tests in this area on OE box. I checked the docs from OE10.2B and they list the following SSL standards:

- *The SSL Protocol Version 2.0/3.0*, Transport Layer Security Working Group, Netscape Communications Corporation
- *The TLS Protocol Version 1.0*, Network Working Group RFC2246
- *Public-Key Infrastructure X.509*, PKIX Working Group

> 4. Same issue as item 1 but for LowLevelSocketListenerImpl. The keyStorePass must not be hard coded. At least the path for the keyStoreName needs to be configurable by bootstrap config.
> 5. Same issue as item 3 but for LowLevelSocketListenerImpl. What protocol suite is really in use there and what ciphers are possible? How does this match the 4GL?

OK.

> 6. SymmetricEncryption.asBytes() should have the open curly brace on the next line.

Old habit. Fixed.

> 7. In SocketListenerImpl.enableConnections(), what is the "20333c34252a2137" default password used for? I'm trying to figure out if this is something we need to match from the 4GL or if we can get rid of this.

That is the default password (password symmetrical encrypted) for default_server. It it specified in docs.

> 8. In WebServiceHelper.invoke() the javadoc has // TBA.

I noticed it was missing. Fixed.

LE: GES removed the hard coded password from this entry.

#29 - 03/31/2016 07:10 AM - Constantin Asofiei

Ovidiu, about 2181a rev 10998:

1. LowLevelSocketImpl.createSslSocket - it can return null if host name can't be verified, and NO-ERROR is on. How are the callers of SSLProtocolSocketFactory.createSocket behaving in such cases?
2. LowLevelSocketImpl.enableSSLConnections - I think we might need to provide case-insensitive search of store files, in case of a case-sensitive file-system, considering that the keyalias is passed by the legacy 4GL code, via the -keyalias connect option. I'm thinking of the portability of Windows OS configuration to unix systems.
3. WebServiceHelper.connect - the validation must be performed in the specified order, by the step comments. You can't perform step2a before step1 validation - this is the order 4GL validates/processes the connection arguments.
4. WebServiceHelper.invoke - on line 598, you are calling webService.cleanupInvocation(webServiceId); - this is done in the finally block, too, so is not needed there.
5. WebServiceHelper.connect - why not execute this code:

```
webService.configure(webServiceId,
                             noSessionReuse,
                             noHostVerify,
                             maxConnections,
                             connectionLifetime);
```

   at the same time webService.connect is called, directly on client-side? Is needed to be executed anyway if the connect was successfull, and we will avoid a second trip to the client. Also, you can encapsulate all the parameters of the webService.connect API into a WebServiceConnectOptions externalizable class, and pass this as an argument, avoiding a long argument list.

And a more general issue: Utils.uniqueId (and AppServerManager.uniqueId) are prone to errors, as they don't track used values - when it overflows, if ID 0 is still in use by someone, there will be a collision. Same applies for UniqueIdGenerator class. The chances to collide are small at first, but they will increase in long-running applications. Maybe a separate task is better for this.

#30 - 04/01/2016 04:15 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> Ovidiu, about 2181a rev 10998:
> 1. LowLevelSocketImpl.createSslSocket - it can return null if host name can't be verified, and NO-ERROR is on. How are the callers of SSLProtocolSocketFactory.createSocket behaving in such cases?

You're right. The factory always return a non-null object. In case of errors exceptions are thrown. Fixed.

> 2. LowLevelSocketImpl.enableSSLConnections - I think we might need to provide case-insensitive search of store files, in case of a case-sensitive file-system, considering that the keyalias is passed by the legacy 4GL code, via the -keyalias connect option. I'm thinking of the portability of Windows OS configuration to unix systems.

Indeed, this could happen when running application written for other OS. As all accesses to client fs, this need informations from p2j.cfg.xml and then use FileScope.findCaseInsensitively(String filename, Environment env) to identify the correct file. I added a TODO for the moment.

> 3. WebServiceHelper.connect - the validation must be performed in the specified order, by the step comments. You can't perform step2a before step1 validation - this is the order 4GL validates/processes the connection arguments.

I understand now. I wanted to keep as much code on server side.

> 4. WebServiceHelper.invoke - on line 598, you are calling webService.cleanupInvocation(webServiceId); - this is done in the finally block, too, so is not needed there.

Thanks for spotting this. The code evolved a little. I failed to keep track of that.

> 5. WebServiceHelper.connect - why not execute this code:
> [...]
> at the same time webService.connect is called, directly on client-side? Is needed to be executed anyway if the connect was successfull, and we will avoid a second trip to the client. Also, you can encapsulate all the parameters of the webService.connect API into a WebServiceConnectOptions externalizable class, and pass this as an argument, avoiding a long argument list.

Done.

> And a more general issue: Utils.uniqueId (and AppServerManager.uniqueId) are prone to errors, as they don't track used values - when it overflows, if ID 0 is still in use by someone, there will be a collision. Same applies for UniqueIdGenerator class. The chances to collide are small at first, but they will increase in long-running applications. Maybe a separate task is better for this.

All these issues were handled and the last update committed to task branch. After rebase operation to latest trunk, 2181a is at revision 11004.

#31 - 04/01/2016 04:36 PM - Greg Shah

> LowLevelSocketImpl.enableSSLConnections - I think we might need to provide case-insensitive search of store files, in case of a case-sensitive file-system, considering that the keyalias is passed by the legacy 4GL code, via the -keyalias connect option. I'm thinking of the portability of Windows OS configuration to unix systems.

> Indeed, this could happen when running application written for other OS. As all accesses to client fs, this need informations from p2j.cfg.xml and then use FileScope.findCaseInsensitively(String filename, Environment env) to identify the correct file. I added a TODO for the moment.

You are describing conversion-time facilities.

We already have the equivalent facility for runtime. Perhaps you can leverage the FileSystemDaemon for the search?

#32 - 04/02/2016 08:03 AM - Greg Shah

Code Review Task Branch 2181a Revision 11004

The changes look very good.

1. In the future, "simple container" classes like WebServiceConnectOptions do not need getter/setter methods. It is OK to make all the data public and directly access them. You don't need to change the current code for this reason, this is just a note for future implementations.

2. What is the implication of the missing implementation for WebServiceHelper.sendStop()?

3. For the case-insensitive support, please look at the FileSystemDaemon.searchPath(). This already handles the case-insensitivity AND it handles the propath lookup for relative filenames. It is very likely that the 4GL code also has this behavior. If so, then finding the correct file using FileSystemDaemon.searchPath() is the correct way to go.

#33 - 04/04/2016 04:48 PM - Greg Shah

Code Review Task Branch 2181a Revision 11005

The changes are fine.

Please resolve the case-insensitive lookup issue.

#34 - 04/05/2016 12:58 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> Code Review Task Branch 2181a Revision 11005
>
> The changes are fine.
>
> Please resolve the case-insensitive lookup issue.

The issue was fixed and committed to task branch 2181a as revision 11006.

#35 - 04/05/2016 01:29 PM - Greg Shah

Code Review Task Branch 2181a Revision 11006

The changes are good. Resolve the following 2 issues and then put this code into runtime regression testing.

1. Please rebase.

2. Is the old comment from LowLevelSocketListenerImpl.enableSSLConnections() is still accurate?

```
      // we store each alias in separate store, similar to 4GL, however, the format differ and
      // the keys/ certificates must be imported (but keeping same name).
      // 4GL keystore is the <DLC>/key folder. The files are in ASCII PEM format and include both
      // the key and the certificate.
```

If still accurate, then please put it back in as it seems useful.

#36 - 04/05/2016 01:47 PM - Ovidiu Maxiniuc

Yes, the information is correct.
The comment was not completly removed, I moved the 4 lines up to method javadoc. I believe is more visible there, and the information is also available standalone within extracted docs.
If you think otherwise, I will move it back (or copy it for emphasis).

Please let me know your choice so I can rebase afterwards.

#37 - 04/05/2016 01:52 PM - Greg Shah

> If you think otherwise, I will move it back (or copy it for emphasis).

No, don't worry about it.

#38 - 04/05/2016 03:25 PM - Ovidiu Maxiniuc

Task branch 2181a was rebased and pushed up to revision 11010.

#39 - 04/05/2016 03:45 PM - Greg Shah

Great! Please post your regression testing results here as soon as you have them. Branch 1811t is waiting for your merge to trunk for a rebase and then to start testing...

#40 - 04/06/2016 08:30 AM - Ovidiu Maxiniuc

The regression test passed. There were a few failed testcases, but they are known to fail occasionally.

#41 - 04/06/2016 09:40 AM - Greg Shah

Please merge to trunk.

#42 - 04/06/2016 10:12 AM - Ovidiu Maxiniuc

The final update for task branch 2181a was committed to trunk as revision 10995.
The test results were saved and branch was archived. A notification was sent to all team members.

#43 - 04/06/2016 10:17 AM - Greg Shah

- % Done changed from 0 to 100

- Status changed from WIP to Closed

#44 - 11/16/2016 12:12 PM - Greg Shah

- Target version changed from Milestone 12 to GUI Support for a Complex ADM2 App

#45 - 09/01/2021 03:15 PM - Greg Shah

- Related to Feature #4366: support TLS 1.2 and other updated secure socket protocols added