Base Language - Feature #1646

implement BASE64-ENCODE/BASE64-DECODE built-in functions

10/21/2012 01:06 PM - Greg Shah

Status:	Closed	**Start date:**	09/27/2013
Priority:	Normal	**Due date:**	10/04/2013
Assignee:	Vadim Gindin	**% Done:**	100%
Category:		**Estimated time:**	32.00 hours
Target version:	Runtime Support for Server Features		
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Blocked by Base Language - Feature #1884: add some of the v10 data types and ...	**Closed**	**08/19/2013**

History

#1 - 10/31/2012 01:35 PM - Greg Shah

- Target version set to Milestone 7

#2 - 10/31/2012 05:34 PM - Greg Shah

- Subject changed from implement BASE64-ENCODE built-in function to implement BASE64-ENCODE/BASE64-DECODE built-in functions

- Estimated time changed from 8.00 to 16.00

- Assignee set to Adrian Lungu

Review the documentation for these functions in the Progress 4GL Language Reference. Write some testcases to ensure that you understand what it does and how it works. Run them to confirm your understanding. Using this understanding, design and document the approach that will properly duplicate it. Write the backing code. It can leverage the Apache commons code that is already in p2j/lib/ and which does already support base-64 encode/decode. Create the result as static methods in com/goldencode/p2j/util/SecurityOps. Make sure you handle all the different forms (any optional parameters...). Write the conversion rule changes (see p2j/rules/convert/builtin_functions.rules) to handle the proper conversion of the code. Then run your testcases in P2J and ensure compatibility. Finally, add this to the list of supported builtin functions in the conversion reference book.

#3 - 11/01/2012 09:28 AM - Adrian Lungu

- Start date set to 11/01/2012

- Status changed from New to WIP

#4 - 11/01/2012 03:40 PM - Adrian Lungu

The BASE64-ENCODE function:
Converts binary data into a Base64 character string, and returns a LONGCHAR containing the character data.

I have a problem with the LONGCHAR data type. I think it's not supported. Conversion of:

```
DEFINE VARIABLE encdlngc AS LONGCHAR NO-UNDO.
```

will raise:

```
EXPRESSION EXECUTION ERROR:
---------------------------
throwException(errmsg)
^  { Unrecognized data type VAR_LONGCHAR. [DEFINE_VARIABLE id <2735894167567> 2:1] }
---------------------------
```

#5 - 11/01/2012 04:17 PM - Greg Shah

Yes, you're right. I forgot about that dependency. The task #1884 is where we are putting some new data type support into our runtime (and conversion).

Go ahead and work on the longchar support first. Get that going (you should see a new email right now on this). Then you will be able to come back and keep going here.

#6 - 04/26/2013 07:52 AM - Greg Shah

- Assignee deleted (Adrian Lungu)

An initial version exists in SecurityOps. This needs to be tested to confirm its behavior matches the 4GL. For example, does the Base64 class generate the same output as the 4GL? What happens when unusual conditions are encountered (e.g. unknown value on input or text input in different code pages)? Make any fixes that are needed as part of this task.

#7 - 04/26/2013 09:53 AM - Greg Shah

- Due date set to 07/30/2013

- Start date changed from 11/01/2012 to 07/27/2013

- Assignee set to Eugenie Lyzenko

#8 - 09/18/2013 08:54 AM - Greg Shah

- Estimated time changed from 16.00 to 32.00

- Due date changed from 07/30/2013 to 10/04/2013

- Assignee changed from Eugenie Lyzenko to Vadim Gindin

- Start date changed from 07/27/2013 to 09/27/2013

#9 - 09/24/2013 04:08 PM - Vadim Gindin

base64-decode(expr) - where expr - character or longchar. Resulting MEMPTR contains binary data
base64-encode(expr) - where expr - memptr or raw. Resulting LONGCHAR is in cp specified by -cpinternal

I've made a some testing of functions in progress. Here are first results:
1. String args

```
def var  tempVar as raw.
/* base64 encoded representation of some binary data */
def var  base64str as char init "w+eBR5Ip".
def var  resStr as char.

tempVar = base64-decode(base64str).
resStr = base64-encode(tempVar).

if compare(base64str, "EQ", resStr, "CASE-INSENSITIVE")  then
  display "yes, source and target are equal".
else
  display "no, source and target aren't equal".
```

We have base64 representation of some binary data. Decode it, encode it and compare to the source. Should be equal. Executes correctly and returns "yes, ... equal".

2. The same test as first but with wrong base64str (for example "abracadabra") gives interesting results. Procedure executes and outputs result "no, .. aren't equal" with error message "Error converting Base64 to RAW (12119)".

3. Encode binary data from file

```
def var  encdmptr as memptr.
def var  encdlngc as longchar.
```

```
copy-lob from file "/home/vig/java.ico" to encdmptr.
encdlngc = base64-encode(encdmptr).
copy-lob from encdlngc to file "/home/vig/testencode".
```

Reads binary data from file, encodes it and writes result to a file. Works correctly. testencode file containts base64 string.

4. Decode base64 data from file

```
def var  decdmptr as memptr.
def var  decdlngc as longchar.
copy-lob from file "/home/vig/testencode" to decdlngc.
decdmptr = base64-decode(decdlngc).
copy-lob from decdmptr to file "/home/vig/java_out.ico".
```

Reads base64 data from prepared file testencode, decodes it and writes result binary data to output file. Works wrong. Can't read data from source file and shows error: "Error -26 in file to longchar". Strange error. It looks like a bug. Do you know something about it?

5. Combined read-write from-to files

```
def var encdmptr as memptr.
def var encdlngc as longchar.
copy-lob from file "/home/vig/java.ico" to encdmptr.
encdlngc = base64-encode(encdmptr).
encdmptr = base64-decode(encdlngc).
copy-lob from encdmptr to file "/home/vig/ico.java".
```

Reads binary data from image-file, encodes it, decodes result and writess to output binary file. Works fine. Source and target files are equal.

Where to place wroted tests?

#10 - 09/24/2013 04:43 PM - Greg Shah

> Can't read data from source file and shows error: "Error -26 in file to longchar". Strange error. It looks like a bug. Do you know something about it?

No, I don't know about that.

> Where to place wroted tests?

testcases/uast/base64/

Make sure to pass bad input to these functions to see what they do. For example, a var set to unknown value for the parameters.

Create a test like number 1 above, but first do the encode on a text string and then decode that encoded version, the result should be identical.

#11 - 09/25/2013 09:48 AM - Vadim Gindin

Greg Shah wrote:

> ..
> Create a test like number 1 above, but first do the encode on a text string and then decode that encoded version, the result should be identical.

base64-encode takes the only arguments of types: memptr or raw. How to compare values of these types?

#12 - 09/25/2013 09:51 AM - Greg Shah

See BinaryData.compareTo() and CompareOps.equals(BDT, BDT).

#13 - 09/25/2013 09:52 AM - Greg Shah

To be clear: memptr and raw are BaseDataTypes that can be compared for equality, inequality, less than...

#14 - 09/25/2013 02:15 PM - Vadim Gindin

Here is additional testing.
6. Empty input.

```
def var  m as memptr.
def var  r as raw.
def var  l as longchar.
def var  c as char.
def var  l0 as longchar.

l = base64-encode(m).
copy-lob from l to file "/home/vig/err_enc_mptr".
```

```
l = base64-encode(r).
copy-lob from l to file "/home/vig/err_enc_raw".

m = base64-decode(l0).
copy-lob from m to file "/home/vig/err_dec_lngchr".

m = base64-decode(c).
copy-lob from m to file "/home/vig/err_dec_char".
```

Here are the calls of base64 functions with empty argument. For base64-encode it works and empty file created. For base64-decode it shows the error: "Error converting Base64 to RAW" and creates empty file.

7. encode-decode from-to memptr

```
def var  c1 as memptr.
def var  c2 as memptr.

def var  l  as longchar.

copy-lob from file "/home/vig/java.ico" to c1.
/* copy-lob from file "/home/vig/java.ico2" to c2. */

/* byte-to-byte compare of memptrs  */
function memptrcmp returns logical (a as memptr, b as memptr):
  def var i as int no-undo.
  if get-size(a) <> get-size(b) then
    return no.
  do i = 1 to get-size(a):
    if get-byte(a,i) <> get-byte(b,i) then
     return no.
  end.
  return yes.
end function.

/*
display "memptr check..".
if memptrcmp(c1,c2) then
  display "equal - f works".
else
  display "not equal - f don't works".
*/

l  = base64-encode(c1).
c2 = base64-decode(l).

if memptrcmp(c1,c2) then
  display "source memptr equals target memptr".
else
  display "not equals".
```

Works fine. Result: equals.

8. encode-decode from raw

```
def var  c1 as raw.
def var  c2 as raw.

def var  l  as longchar.

c1 = generate-uuid.

/* byte-to-byte compare of raws  */
function rawcmp returns logical (a as raw, b as raw):
  def var i as int no-undo.
  if length(a) <> length(b) then
    return no.
  do i = 1 to length(a):
    if get-byte(a,i) <> get-byte(b,i) then
     return no.
  end.
  return yes.
```

```
end function.

l  = base64-encode(c1).
c2 = base64-decode(l).

if rawcmp(c1,c2) then
  display "source raw equals target raw".
else
  display "not equals".
```

Works fine. Result: equals.

I've found that SecurityOps already has base64 methods and buildin_functions.rules already contains corresponding rules.. Should I only check and probably test them?

#15 - 09/25/2013 02:23 PM - Greg Shah

> Should I only check and probably test them?

You can check them, but I can guarantee that they don't properly implement the full behavior (especially the error handling). That stuff will need to be fixed at a minimum.

Also: please add a testcase showing how these functions respond to the following:

1. unknown value memptr
2. unknown value raw

#16 - 09/25/2013 04:53 PM - Vadim Gindin

Greg Shah wrote:

> Also: please add a testcase showing how these functions respond to the following:
>
> 1. unknown value memptr
> 2. unknown value raw

```
l = base64-encode(?)
copy-lob l to file ...
```

returns empty file

```
m = base64-decode(?)
copy-lob m to file ...
```

returns empty file

I can't initialize var with ? and I can't assing ? to var. How can I check ? for the memptr and raw separately?

#17 - 09/25/2013 07:12 PM - Greg Shah

> returns empty file

Instead of using copy-lob in these cases, it is cleaner to just check if the length/size of the raw/memptr is 0.

> I can't initialize var with ? and I can't assing ? to var.

Why not? This should work fine:

```
def var ptr as memptr.
def var rrr as raw.
ptr = ?.
rrr = ?.
```

#18 - 09/26/2013 10:35 AM - Vadim Gindin

You're right. My mistake.
Here is updated error test.

```
def var  m as memptr.
def var  r as raw.
def var  l as longchar.
def var  c as char.

m = ?.
l = base64-encode(m).
if l <> ? then
  display "memptr res is not ?".

r = ?.
l = base64-encode(r).
if l <> ? then
  display "raw res is not ?".

l = ?.
m = base64-decode(l).
if m <> ? then
  display "longchar res is not ?".

c = ?.
m = base64-decode(c).
if m <> ? then
  display "char res is not ? ".
```

This test outputs nothing. So base64 functions work correctly. If the argument is ? than the result will be ?.

#19 - 09/26/2013 10:41 AM - Greg Shah

OK, good. One thing to change: don't use DISPLAY for just printing a line of text. It creates an entire frame which is expensive and generates too much Java code. And worse, since you did not explicitly define a frame name, the resulting frame (if ever actually shown) would contain all of the strings from the program, not just the one you want to show.

Use the MESSAGE stmt instead.

#20 - 09/26/2013 10:43 AM - Vadim Gindin

Thank you, but will the MESSAGE command override it's previous output?

#21 - 09/26/2013 10:46 AM - Greg Shah

I'm not exactly sure what you mean by this question. But I suggest you try this code to see how multiple message stmts work:

```
message 1.
message 2.
message 3.
message 4.
```

#22 - 09/26/2013 12:13 PM - Vadim Gindin

I have a question. Lets see at the method com.goldencode.p2j.util.SecurityOps.base64Encode(BinaryData). There is a null check inside it. I want to understand when the argument can be null? I just don't know what to do in the case when param is null: return new empty object of certain type or generate unknown value. Thanks

#23 - 09/26/2013 12:21 PM - Greg Shah

The null check is just for safety purposes. It should never get used by converted code that was automatically generated but anyone that calls the API directly may cause this failure. Usually, we treat null input the same as receiving unknown value on input. This seems appropriate here.

#24 - 09/26/2013 01:05 PM - Vadim Gindin

Question 2. I converted and ran test 2. Recalling, that it is the same test as 1 but with wrong "base64" string "abracadabra". Progress procedure outputs an error: "Error converting Base64 to RAW (12119)".
Converted application (using apache) normally decodes this string to some binary data. After that it encodes this binary data again and resulting string is "abracadaAAAA". So converted app compares these strings and outputs "no, not equal".

I need to detect when the source string is wrong, i.e. I need some criteria to check the source string to produce the same behaviour as the Progress procedure. Probably I should read more about base64 format to reproduce the correspondance of source string to it. But I don't know how long it may take time. What do you think?

#25 - 09/26/2013 01:22 PM - Greg Shah

I assume the error is occurring in the decode function but it is important to confirm that. Just put a "pause." into the program after the decode to see if the error comes first.

Normally an error message like that is also associated with raising an error condition (which changes control flow like throwing an exception). But in this case, it seems like it is just an error message and not raising the error condition. You need to check if it changes the ERROR-STATUS data to "record" the error. It probably doesn't do this unless NO-ERROR is used.

Please try the same thing but with NO-ERROR added to the end of the decode line. This normally "suppresses" any visual sign of the error but forces it to be recorded in the ERROR-STATUS handle.

In any case where the ERROR-STATUS handle is updated should be tested in your testcase. Code like this can be helpful:

```
rvar = base64-decode(somebadtext) no-error.
if not error-status:error or not error-status:get-number(1) eq 12119 then message "Missing error 12119 from ba
se64-decode of trash.".
```

> I need to detect when the source string is wrong, i.e. I need some criteria to check the source string to produce the same behaviour as the Progress procedure. Probably I should read more about base64 format to reproduce the correspondance of source string to it.

I'm not sure it will work, but a later version of the Apache commons codec support (starting at version 1.4; we currently have 1.3) provides a Base64.isBase64() method which may be able to detect the problem. Check that first. You can always update our apache codec jar as part of your update.

#26 - 09/26/2013 04:04 PM - Vadim Gindin

Greg Shah wrote:

> I assume the error is occurring in the decode function but it is important to confirm that. Just put a "pause." into the program after the decode to see if the error comes first.

I did it. The error is occurring in the decode function as you expected.

> Normally an error message like that is also associated with raising an error condition (which changes control flow like throwing an exception). But in this case, it seems like it is just an error message and not raising the error condition. You need to check if it changes the ERROR-STATUS data to "record" the error. It probably doesn't do this unless NO-ERROR is used.
>
> Please try the same thing but with NO-ERROR added to the end of the decode line. This normally "suppresses" any visual sign of the error but forces it to be recorded in the ERROR-STATUS handle.

I did it. It don't change the ERROR-STATUS:ERROR handle. It remains false. However it add the error number 12119. I.e.
ERROR-STATUS:GET-NUMBER = 12119

> I'm not sure it will work, but a later version of the Apache commons codec support (starting at version 1.4; we currently have 1.3) provides a Base64.isBase64() method which may be able to detect the problem. Check that first. You can always update our apache codec jar as part of your update.

I ran some tests with a string "abracadabre" and I found that Base64 has a method isArrayByteBase64 to check but it returns true for our string. By the way this method is defined as deprecated in later versions and marked to be replaced by isBase64. I also tried this method too (using latest version of common-codec). But all results are the same: "abracadabre" is a base64 string. I read that it check every symbol in a string whether it is a base64 symbol. So in our string it really is. As a result It stays unclear why the Progress shows an error.

#27 - 09/26/2013 04:29 PM - Greg Shah

> I did it. It don't change the ERROR-STATUS:ERROR handle. It remains false. However it add the error number 12119. I.e.

ERROR-STATUS:GET-NUMBER = 12119

Does the error get displayed on the screen when NO-ERROR is used?

See ErrorManager.recordOrShowError(), ErrorManager.warningModeEnable() and ErrorManager.recordOrThrowError() for tools to use to duplicate this behavior.

As a result It stays unclear why the Progress shows an error.

See if this http://en.wikipedia.org/wiki/Base64 helps. There are a couple of things here that may explain it:

Base64 encoding converts three octets into four encoded characters.

When the number of bytes to encode is not divisible by three (that is, if there are only one or two bytes of input for the last 24-bit block), then the following action is performed: Add extra bytes with value zero so there are three bytes, and perform the conversion to base64. If there was only one significant input byte, only the first two base64 digits are picked(12 bits), and if there were two significant input bytes, the first three base64 digits are picked(18 bits). '=' characters might be added to make the last block contain four base64 characters.

As a result: When the last group contains one octet, the four least significant bits of the final 6-bit block are set to zero; and when the last group contains two octets, the two least significant bits of the final 6-bit block are set to zero.

Work to duplicate their results based on this information.

#28 - 09/27/2013 04:30 PM - Vadim Gindin

The error is not displayed when NO-ERROR is specified. error-status:error is not set. only error-status:get-number(1) is equal to 12119.

I wroted another one test:

```
function b64dec returns char (s as char):
  def var tempVar as raw.
  def var resStr as char.

  tempVar = base64-decode(s) no-error.

  if error-status:get-number(1) eq 12119 then
    resStr = "12119".
  else
    resStr = base64-encode(tempVar).

  return resStr.
end function.

output to b64dec.

message "abcd/= : "       b64dec("abcd/=")  skip
        "aabc : "         b64dec("aabc") skip
        "aaasdgsdgsdf : " b64dec("aaasdgsdgsdf") skip
        "aaaa : "         b64dec("aaaa") skip
        "aaaa= : "        b64dec("aaaa=") skip
        "aaaaa= : "       b64dec("aaaaa=") skip
        "aaaaaa= : "      b64dec("aaaaa=") skip
        "aaaaaaa= : "     b64dec("aaaaaaa=") skip
        "aaaaaaaa= : "    b64dec("aaaaaaaa=") skip
        "abracadabre : "  b64dec("abracadabre") skip
        "+dfgrte= : "     b64dec("+dfgrte=") skip
        "+\/+\/ : "         b64dec("+\/+\/") skip.
...
```

This test tries various inputs for base64-decode function. It runs decode-encode functions for various base64 strings. Here are results:

```
abcd/= :  12119
aabc :  aabc
aaasdgsdgsdf :  aaasdgsdgsdf
aaaa :  aaaa
aaaa= :  12119
aaaaa= :  12119
aaaaaa= :  12119
aaaaaaa= :  aaaaaaY=
aaaaaaaa= :  12119
abracadabre :  12119
+dfgrte= :  +dfgrtc=
+/+/ :  +/+/
= : 12119
a : 12119
a+ : 12119
as= : ag==
a=a : 12119
awq : awo=
```

Here is a result of common-codec decode-encode for the same strings:

```
abcd/= : abcd
aabc : aabc
aaasdgsdgsdf : aaasdgsdgsdf
aaaa : aaaa
aaaa= : aaaa
aaaaa= : aaaa
aaaaaa= : aaaaaQ==
aaaaaaa= : aaaaaaY=
aaaaaaaa= : aaaaaaaa
abracadabre : abracadabrc=
+dfgrte= : +dfgrtc=
+/+/ : +/+/
= : 
a+ : aw==
as= : ag==
a=a : 
awq= : awo=
```

1. As it follows form description of base64 algorithm the resulting string will have a lenght which is aliquot to 4 and should consists of characters from this set: «ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/» and probably '=' at the end.
2. I suspect that Progress uses some simple rule to define: show or not to show an error. This simple rule could be: length is aliquot to 4 and right characters from specified set and symbol '=' at the end. This rule is applicable for source strings of length >= 4 and checked by executed tests. What do you think?
3. When the length < 4 there is a strange test result there:
as= : ag==
Our simple rule is not suited here probably because of correspondent special case in base64 algorithm.
Should I write some implement of this algorithm to understand it indeed?
4. When the result of decode-encode run is not equal to a source string but the Progress don't show an error. It is correct result too. common-codec behaves in the same way.

Here is my questions:
1. Am I on the right way or I didn't understood your last comment?
2. Is the simple rule suits to the case when length >= 4, What do you think?
3. It's unclear what the Progress do when length < 4. Could you advice me something?

#29 - 09/27/2013 05:15 PM - Greg Shah

> 1. Am I on the right way or I didn't understood your last comment?

Yes.

> 2. Is the simple rule suits to the case when length >= 4, What do you think?

This seems reasonable. Add more input tests to find any difference from your proposed rule. If you can't find any difference, go with it.

3. It's unclear what the Progress do when length < 4. Could you advice me something?

Add more input tests to try to determine a pattern. One approach: look inside the Apache code to see what they do and then try to test out if Progress has a similar implementation.

#30 - 09/30/2013 03:22 PM - Vadim Gindin

I added more tests, as you adviced. Here is a results.

```
= :  12119
a :  12119
/ :  12119
a+ :  12119
a= :  12119
== :  12119
/= :  12119
=a :  12119
++ :  12119
as :  12119
as= :  ag==
a=a :  12119
=== :  12119
aa= :  aQ==
as= :  ag==
aa= :  aQ==
+/= :  +w==
+dd :  12119
P/q :  12119
+d= :  +Q==
P/= :  Pw==
//= :  /w==
/== :  12119
a== :  12119
aabc :  aabc
ab/+ :  ab/+
/++= :  /+8=
a=a= :  12119
//// :  ////
!ddd :  12119
AAA= :  AAA=
aaaa :  aaaa
awq= :  awo=
aw== :  aw==
a=== :  12119
aaaa+ :  12119
aaaa= :  12119
aaa== :  aaY=
aa=== :  aQ==
abcd/= :  12119
aaaa+= :  12119
aaaa+/ :  12119
aaaaa= :  12119
+/+/ :  +/+/
aaaa+/= :  aaaa+w==
abcde+/ :  12119
aaaaaa= :  12119
aaaa+/+= :  aaaa+/8=
aaaaaaa= :  aaaaaaY=
+dfgrte= :  +dfgrtc=
aaaaaaaa= :  12119
abracadabre :  12119
abracadabren :  abracadabren
abracadabre= :  abracadabrc=
abracadabr/= :  abracadabr8=
abracadabr// :  abracadabr//
```

```
aaasdgsdgsdf :  aaasdgsdgsdf
=abracadabr= :  12119
```

I can formulate the common (Progress) algorithm as follows (at this moment):
1. Characters must be from «ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/» and at the end of the source string the '=' or '==' could be.
2. If len >= 4 then l % 4 must be 0.
3. If l % 4 <> 0 then src.substring((l/4)*4) must be correct base64 string (see rule 3).
4. If len < 4 then if len 3 && src[len-1] '=' && src[len-2] <> '=' then source string is correct
('+d=', 'P/=', 'as=' - correct, 'a==' - incorrect).

Let's look at example string src = "aaaa+/=".
src.len = 7
src.substr(0,(len/4)*4) - correct string by rule 2.
src.len % 4 = 3
src.substr((len/4)*4) = "+/=" wich is correct by rule 4.
=> src string is correct (Progress won't show an error)

In other words, I suppose that Progress divides source string on two parts. The first is the substring with the length aliquot to 4 and the second is the string with the length - remainder l%4. After that it checks first part with simple rule (rule 1) and checks the second part with rule 4.

This is the currect result, but there is a test case which is not suit the formulated algorithm: "aaa=="
Progress don't show the error for this string. Probably this is some boundary case. I'm going to make more testing cases to find out.

#31 - 09/30/2013 03:48 PM - Greg Shah

Great work! Hopefully you will find the final part of the rule soon.

#32 - 10/01/2013 02:32 PM - Vadim Gindin

I added more tests and find out several things:
1. Source string can contain any number of pad symbols ('=') at the end.
2. It seems that Progress ignores all pad symbols except the first. Here are examples.

```
"aaa="
"aaa=="        have the same decode-encoded result string "aaY="
"aaa==="
"aaa======="

"aa="
"aa==="        have the same decode-encoded result string "aQ=="
"aa=="
"aa========"

"aaaaaaa===" ->  "aaaaaaY="
"aaaaaa====" ->  "aaaaaQ=="
```

As a result here is modified rules, that, I suspect, the Progress uses to make a decision: whether to show an error or not.
1. The source string can ends with some number of PAD symbols ('='), but the Progress ignores all of them except the last one. So first remove them from the source string.
2. All characters must be from «ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/» set.
3. Lets name l - length of the source string. If l % 4 == 0 than we only check rule 2. The Last char can be PAD.
4. Lets name r = l % 4. If r != 0 than we get the last r chars to check. R must be only 3. The last char must be PAD. The first and second chars of the remainder part must follow rule 2. At the end if remainder is good check the main part (of the length aliquot to 4) by the rule 2.

Remainders or self source strings: '+d=', 'P/=', 'as=' - correct, 'a==' - incorrect.

Lets look at example:
"aaaaaa===="
1. removing trailing PAD chars. and getting "aaaaaa="
2. l = 7, r = l % 4 = 3.
Remainder: "aa=" and the main part is "aaaa".
The main part follows the rule 2: all chars from the base64 alphabet and the length is aliquot to 4 (by the algorithm).
The remainder is also correct. It follows the rules 3 and 4: remainder length = 3, last char is PAD and first and second chars are from the base64 alphabet.

As remainder and main part are correct so there's no reason to show an error.

#33 - 10/01/2013 02:40 PM - Vadim Gindin

Here is an updated full set of tests for base64-decode function:

```
= :  12119
a :  12119
/ :  12119
a+ :  12119
a= :  12119
== :  12119
/= :  12119
=a :  12119
++ :  12119
as :  12119
as= :  ag==
a=a :  12119
=== :  12119
aa= :  aQ==
as= :  ag==
+/= :  +w==
+dd :  12119
P/q :  12119
+d= :  +Q==
P/= :  Pw==
//= :  /w==
/== :  12119
```

```
a== :  12119
aabc :  aabc
ab/+ :  ab/+
/++= :  /+8=
a=a= :  12119
//// :  ////
!ddd :  12119
AAA= :  AAA=
aaaa :  aaaa
aaa= :  aaY=
aa== :  aQ==
awq= :  awo=
aw== :  aw==
a=== :  12119
aaaa+ :  12119
aaaa= :  12119
aaa== :  aaY=
aa=== :  aQ==
a==== :  12119
//=== :  /w==
abcd/= :  12119
aaaa+= :  12119
aaaa+/ :  12119
aaaaa= :  12119
aaaa== :  12119
aaa=== :  aaY=
+/+/ :  +/+/
aaaa+/= :  aaaa+w==
abcde+/ :  12119
aaaaaa= :  aaaaaQ==
aaaaa== :  12119
aaaa=== :  12119
aaaa+/+= :  aaaa+/8=
aaaaaaa= :  aaaaaaY=
aaaa==== :  12119
+dfgrte= :  +dfgrtc=
aaaaaaaa= :  12119
aaaaaaa== :  aaaaaaY=
aaaaaa=== :  aaaaaQ==
aaaaa==== :  12119
aaaaaaaa== :  12119
aaaaaaa=== :  aaaaaaY=
aaaaaa==== :  aaaaaQ==
aaaaa===== :  12119
aaaa====== :  12119
aaa======= :  aaY=
aa======== :  aQ==
a========= :  12119
abracadabre :  12119
abracadabren :  abracadabren
abracadabre= :  abracadabrc=
abracadabr/= :  abracadabr8=
abracadabr// :  abracadabr//
aaasdgsdgsdf :  aaasdgsdgsdf
=abra=adabr= :  12119
```

#34 - 10/01/2013 03:10 PM - Greg Shah

Good work! Make sure that the javadoc for the method has this algorithm included.

#35 - 10/02/2013 01:25 PM - Vadim Gindin

Here are results of testing new base64Decode method with new check.
Behaviour of Base64.decode() method differs from the Progress analogue at following groups of tests:

- Base64.decode() method returns empty byte[2] array for all source strings with length < 4.
- Base64.decode() throws ArrayIndexOutOfBoundsException for source strings aaa======= and aa======== from our tests set because of previous point (It ingores trailing PAD symbols '=' and the remainder length is less then 4
- For the source string +\/+\/ in Progress procedure the converted class contains +\\/+\\/ string and I'm not sure that it's wrong...
- For the strings of length that is not aliquot to 4 behaviour also differs:

```
string             progress decode-encode             Base64 decode-encode
---------------------------------------------------------------------------------
aaaa+/=            aaaa+w==                           aaaaAAA=
aaaaaa= :          aaaaaQ==                           aaaaAAA=
aaa=======         aaY=                               ArrayIndexOutOfBoundsException => 12119
aa========         aQ==                               ArrayIndexOutOfBoundsException => 12119
```

#36 - 10/02/2013 01:36 PM - Greg Shah

Before we go too far with the Apache commons base64 support, perhaps it makes sense to try base64 encode/decode that already exists in P2J. Yes, sadly we have our own implementation that was put in from an older project that we did about 15 years ago.

Anyway, please do try the com/goldencode/p2j/directory/Base64.java and let me know the results.

Worst case, you can implement a custom version of this algorithm that exactly matches the Progress approach.

#37 - 10/02/2013 02:43 PM - Vadim Gindin

I used the old Base64.java class. In company with my check it produces no deviations from Progress behavior on our tests set. I tried to use it also without my check and it produced some correct result for incorrect cases too.
Finally, the old Base64.java - is real godsend :)

#38 - 10/02/2013 02:47 PM - Greg Shah

Interesting! And this is great news!

Please put comments in the code to explain that the Apache commons base64 library is incompatible. The comment doesn't need to be javadoc, just put a // comment right before the use of the Base64 class. I want to avoid developers coming along later and mistakenly switching the library.

#39 - 10/02/2013 03:46 PM - Vadim Gindin

2 remarks:

- How about "For the source string +\/+\/ in Progress procedure the converted class contains +\\/+\\/ string and I'm not sure that it's wrong..."?
- There is a wrong effect of skip expression in the output file:

  ```
  = :   com.goldencode.p2j.ui.SkipEntity@5df3c5 a :
  com.goldencode.p2j.ui.SkipEntity@2f3bf0 / :
  com.goldencode.p2j.ui.SkipEntity@1082661 a+ :  aw==
  com.goldencode.p2j.ui.SkipEntity@fbd45 a= :
  com.goldencode.p2j.ui.SkipEntity@1531aca == :
  com.goldencode.p2j.ui.SkipEntity@103f4aa /= :
  com.goldencode.p2j.ui.SkipEntity@b3ea59 =a :
  ```

  Should I work on it in this task? If it is just SkipEntity.toString() (see LogicalTerminal.message(..) line 1548) why there is a move to the next line in the output? It seems the SkipEntity is just not implemented yet.

#40 - 10/02/2013 04:20 PM - Vadim Gindin

Here are some conversion results of other base64 tests.

1. The command copy-lob from file "/home/vig/testencode" to decdlngc. is not correctly converted:
LargeObjectOps.writeToFile("/home/vig/testencode", decdlngc);
At least LargeObjectOps.writeToFile() method takes only BinaryData as a first parameter.
It happens in different tests for example test 4.

2. Expression c1 = generate-uuid. is also converted incorrectly: c1.assign();. The assign method expects BinaryData parameter. It seems the generate-uuid is just not implemented yet.

What should I do in described cases of this and previous on comments?

#41 - 10/02/2013 04:25 PM - Greg Shah

> How about "For the source string \/\/ in Progress procedure the converted class contains \\/\\/ string and I'm not sure that it's wrong..."?

Is this the only input string that still fails?

My only idea is to test other inputs that have the \ character to see how they handle it.

> There is a wrong effect of skip expression in the output file:

Please provide a simple 4GL testcase that creates the problem when converted.

> Should I work on it in this task?

No. But I need to understand what is going on. Depending on that we may create a new task for future work.

> At least LargeObjectOps.writeToFile() method takes only BinaryData as a first parameter.

Yes, we only have a partial implementation of COPY-LOB. I just put something in there to help with the testcases I was running at the time. It is not even known to be always compatible.

> It seems the generate-uuid is just not implemented yet.

Right, we don't support it yet.

Whenever there are questions like this, do look in the conversion rules and in the runtime to track down the level of support.

> What should I do in described cases of this and previous on comments?

Just do the minimum to get through. If you can change the testcases to avoid usage of the missing features, that is usually best. If you have to add some limited support in the conversion or runtime for a feature, do the minimum.

#42 - 10/03/2013 02:37 PM - Vadim Gindin

- File vig_upd20131003a.zip added

Here are results of additional testing and the first code for review.

1. Slashes.
Progress docs says that '\' is not escape symbol for Progress. It's only escape symbol for Unix and it's a path separator in Windows. The symbol '~' before special character in Progress causes it to read the special character literally. Here is the test procedure.

```
output to b64slash.
message(string(base64-encode(base64-decode("+\/+\/")))) .
message(string(base64-encode(base64-decode("+~\/+~\/")))) .
message(string(base64-encode(base64-decode("~\+~\+~\+~\+")))) .
message(string(base64-encode(base64-decode("~\~\~\~\")))) .
message(string(base64-encode(base64-decode("~\aaa")))) .
message(string(base64-encode(base64-decode("~\~\aa~\~\bb")))) .
```

If I add "no-error" at the end of each line with message statement the Progress will show an error with the string. I only did it to understand how the Progress parses these strings itself. I also ran this procedure. and here is the result table where result string combined with the string from compile error message.

```
Procedure code      compile error msg    result
---------------------------------------------------
+\/+\/              +/+/                 +/+/
+~\/+~\/            +\/+\/               12119
~\+~\+~\+~\+        \+\+\+\+             12119
~\~\~\~\            \\\\                 12119
~\aaa                \aaa                 12119
~\~\aa~\~\bb        \\aa\\bb             12119
```

In the first column there are strings from the procedure source code displayed. In the second column there are same strings but referred in compile error message dialog. In the third column there are the result of each line.

2. Skip expression wrong effect.
Testing procedure:

```
output to skip_test.

message "first string " skip
        "second string " skip
        "third string " skip.
```

In the Progress the file skip_test consists of this text:

```
first string
second string
third string
```

The result of converted procedure class is followed:

```
first string  com.goldencode.p2j.ui.SkipEntity@9e7d46 second string
com.goldencode.p2j.ui.SkipEntity@132b038 third string
com.goldencode.p2j.ui.SkipEntity@cf5006
```

As you can see there are additional characters in the output that appeared as a result of SkipEntity.toString() during execution. It seems that skip expression is also not implemented yet. I can modify my base64 test procedures to exclude the skip statement from it. But me be this info will be usefull in the future.

3. Other conversion errors related to not-implemented or partially-implemented functions generate-uuid and copy-lob Progress statements. I also modified my test scripts to exclude these statements. To initialize the raw variable I used the following code:

```
def var r as raw.
put-string(r,1) = "some string".
```

and to initialize the memptr variable I used this code:

```
def var r as raw.
set-size(r) = 20.
put-string(r,1) = "some string".
```

4. longchar unknown instantiation method. I had to add this method for longchar type to cause it return new longchar instead of character (returned from baseclass method). See archive.

5. I added the string ErrorManager.recordOrShowError(12119, ERROR_12119, false); to generate an error when the string could not be decoded by Progress. But this method set error-status:error flag along with error-num. But the case is that I need only to set the error-num to 12119. How can I do this? Should I add new method to ErrorManager?

#43 - 10/04/2013 03:44 PM - Greg Shah

> Progress docs says that '\' is not escape symbol for Progress. It's only escape symbol for Unix and it's a path separator in Windows. The symbol '~' before special character in Progress causes it to read the special character literally.

Yes. When running on UNIX or Linux, both '\' and '~' are escape chars (causing the following char to be read literally instead of whatever normal interpretation may be made).

When running on Windows, only '~' is an escape char.

> If I add "no-error" at the end of each line with message statement the Progress will show an error with the string. I only did it to understand how the Progress parses these strings itself.

You can also save the code in a .p file (let's say "something.p" and then use this 4GL code:

COMPILE something.p PREPROCESS something.p.preproc.txt.

Then look at the resulting text file to see the preprocessed version.

> In the first column there are strings from the procedure source code displayed. In the second column there are same strings but referred in compile error message dialog. In the third column there are the result of each line.

So the results just look like the escape chars are being processed first as one would expect. We already handle this in the conversion. Is there something wrong with how we handle these cases?

> 2. Skip expression wrong effect.

From the Progress docs:

```
SKIP [ ( n ) ]
```

```
   Indicates a number (n) of blank lines to insert into the message. The value of n can
   be 0. If you do not specify n, or if n is 0, a new line is started unless the current
   position is already the start of a new line.
   You can only use this option with the VIEW-AS ALERT-BOX option.
```

The LogicalTerminal.message() is the normal message statement. It is NOT the alert-box version. Using SKIP is not supposed to work there. I do agree that it is not handled properly today. This code:

```
message "1" skip "2".
```

outputs this:

```
1 
```

And this code:

```
message "1" skip "2" view-as alert-box.
```

outputs this:

```
                                                     ┌────── Message ──────┐
                                                     │          1          │
                                                     │          2          │
                                                     │ ─────────────────── │
                                                     │         <OK>        │
                                                     └─────────────────────┘
```

Please go ahead and make a suggested fix for LogicalTerminal.message(). But it needs to be 100% compatible with how Progress works. Make sure to truncate the output of the statement at the first SkipEntity. Please note that it looks like Progress adds a space char to the end, before it truncates the output.

> 5. I added the string ErrorManager.recordOrShowError(12119, ERROR_12119, false); to generate an error when the string could not be decoded by Progress. But this method set error-status:error flag along with error-num. But the case is that I need only to set the error-num to 12119. How can I do this? Should I add new method to ErrorManager?

Have you tried the version as ErrorManager.recordOrShowError(12119, ERROR_12119, false, false, false);?

#44 - 10/04/2013 03:53 PM - Greg Shah

Code Review 1003a

1. The SecurityOps.ERROR_12119 and SecurityOps.PAD members need javadoc.

2. There should be a blank line after SecurityOps.ERROR_12119.

3. As far as I know, new longchar() does already yield an unknown longchar instance. Why did you switch to using BDT.generateUnknown() in SecurityOps?

#45 - 10/07/2013 09:14 AM - Vadim Gindin

Greg Shah wrote:

> Code Review 1003a

..

> 3. As far as I know, new longchar() does already yield an unknown longchar instance. Why did you switch to using BDT.generateUnknown() in SecurityOps?

I just missed it. You're right, but from the other point of view what the method BDT.generateUnknown() needed for?
If we use this method with longchar - it will leed to class cast error because it will return new character() from the base class (Text).

#46 - 10/07/2013 09:17 AM - Greg Shah

> what the method BDT.generateUnknown() needed for?

It is used when you have an instance of a BDT and it can be any of the subclasses. This method allows you to easily create an unknown instance of the same type, without having to put more complex reflection logic everywhere.

> If we use this method with longchar - it will leed to class cast error because it will return new character() from the base class (Text).

I'm OK with your fix for BDT.generateUnknown(). We will keep that change.

#47 - 10/07/2013 03:30 PM - Vadim Gindin

Greg Shah wrote:
..

> Have you tried the version as ErrorManager.recordOrShowError(12119, ERROR_12119, false, false, false);?

There is no such method..

#48 - 10/07/2013 03:44 PM - Greg Shah

Please make sure your p2j project is updated to the latest version in bzr. In my version of ErrorManager, the method is found at line 578 and the latest history entry is this:

```
** 049 CA  20130927          Static proxies need the unknown() API. Resource type is determined
**                           from LegacyResource constants.
```

#49 - 10/07/2013 06:08 PM - Vadim Gindin

I got the compile error after update: Category cannot be resolved or is not a field

```
decimal.java
636                       new DecimalFormatSymbols(Locale.getDefault(Locale.Category.FORMAT));
```

I installed jdk-7 but the error still exists..

#50 - 10/08/2013 05:36 AM - Greg Shah

This command can be used to display and change the default JDK:

```
sudo update-alternatives --config java
```

#51 - 10/10/2013 01:37 PM - Vadim Gindin

- File vig_upd20131010a.zip added

I switched to java7, implemented "skip" operator support for a message statement including message truncating. By the way in this case there is a message impl effect. Exactly the message impl in the Progress is responsible for input message truncation. It happens when the input message contains line break character sequence: "\n", "\n\r", "\r\n", "\r". For example, the statement message "One \ntwo \r\ntree\r!" will output One . See the next changes archive.

#52 - 10/10/2013 02:09 PM - Greg Shah

Code Review 1010a

1. Is this todo in base64Encode() still valid?

```
      // TODO: test the results to see if empty string and other data inputs get the
      // proper compatible result
```

2. The private methods checkFormat() and isBase64() need to be placed in the proper location for private methods (i.e. it should be below the public ones). The same issue must be fixed with ThinClient.truncate().

3. I don't understand the ThinClient.truncate() implementation.

```
   private String truncate(String message)
   {
      String result = message;

      int nIdx = result.indexOf("\n");
      if (nIdx != -1)
      {
         result = result.substring(0, nIdx);
      }

      int nrIdx = result.indexOf("\n\r");
      if (nrIdx != -1)
      {
         result = result.substring(0, nrIdx);
      }

      int rnIdx = result.indexOf("\r\n");
      if (rnIdx != -1)
      {
         result = result.substring(0, rnIdx);
      }

      int rIdx = result.indexOf("\\r");
      if (rIdx != -1)
      {
         result = result.substring(0, rIdx);
      }
      return result;
   }
```

- How can nrIdx ever be found to be >= 0 when we have already used substring up to the first "\n"?
- What kind of newline is "\n\r"? Are you sure that Progress does something with that specifically or is it just processing the "\n"?
- Why process "\r\n" instead of just processing "\r"?
- Why is there logic for "\\r"? It isn't a normal line end.

4. In LogicalTerminal, why do this:

```
         else if (list[i] instanceof SkipEntity)
         {
            // if skip stmt is the last - don't insert the line break
            if (i < list.length - 1)
            {
               sb.append(System.getProperty("line.separator"));
            }
         }
```

As far as I know, the text should just be truncated here. The VIEW-AS ALERT-BOX case does not use LogicalTerminal.message() but instead it uses LogicalTerminal.messageBox(). So it seems to me that we should just truncate the string as soon as the SkipEntity is encountered.

#53 - 10/10/2013 04:02 PM - Vadim Gindin

I will make additional testing to answer these questions.

#54 - 10/11/2013 09:30 AM - Vadim Gindin

How to create memptr with negative size? I tried to call set-size(..)=-100 but it didn't work. After this call the following call get-size(..) returned 0.

#55 - 10/11/2013 10:12 AM - Greg Shah

> How to create memptr with negative size?

Search for REAL_LOWER_BOUND in memptr.java. The short story: in Progress you can define a memptr with any value from -7 through -1 as well as positive values. It is undocumented and it doesn't make much sense.

In other words: please try testcases that have memptr sizes in this -1 through -7 range and handle the behaviors that occur.

#56 - 10/11/2013 11:42 AM - Vadim Gindin

There is the old comment in the SecurityOps.base64Encode method:

```
      // TODO: negative memptr sizes support
      // larger negative reported sizes for a memptr can cause this error:
      // Unable to allocate memory for result from BASE64-ENCODE function (12118)
      // a size of -1 can cause a 0-length string as a result
      // neither of these results are properly supported here
```

I've tested these cases. Here are results:
1) For the sizes from -7 to -1 (as you wroted) resulting memptr has that size "allocated". But there is not possibility to write something there. Never mind. For such memptr the base64Encode function returns 0-length string.
2) For the sizes from set 0, -8, -9 or less resulting memptr has size = 0 and the base64Encode function returns unknown value for such memptrs. There is no error, mentioned in the old comment.

#57 - 10/11/2013 12:34 PM - Greg Shah

OK, do implement the support for your current findings.

> There is no error, mentioned in the old comment.

I think it can be found with something like this:

```
def var ptr1 as memptr.
def var ptr2 as memptr.
def var addr as int64.

/* this is a technique to initialize a memptr from an address */
/* when you do this, the 4GL doesn't know the size of the memory */
/* region, so you must set it using set-size(); but this size can */
/* be completely invalid AND the size is only used for some bounds */
/* checking */
set-size(ptr1) = 16.
addr = get-pointer-value(ptr1).
set-pointer-value(ptr2) = addr.
set-size(ptr2) = -128.

/* use the ptr2 memptr with base64-encode here */
/* I think you may get the error now, since get-size(ptr2) will return -128 */

/* cleanup */
set-size(ptr1) = 0.
```

#58 - 10/11/2013 12:41 PM - Vadim Gindin

Interesting... I'll try.

Here is the progress procedure I wroted about.

```
def var str as longchar no-undo.
def var ptr as memptr no-undo.

def var str1 as longchar no-undo.
def var ptr1 as memptr no-undo.

set-size(ptr) = 90000.

put-string(ptr, 1, 30000) = fill("1", 30000).
put-string(ptr, 30001, 30000) = fill("2", 30000).
put-string(ptr, 60001, 30000) = fill("3", 30000).

str = base64-encode(ptr).

message length(str) view-as alert-box.

set-size(ptr) = 0.

set-size(ptr1) = -6.
/* don't work: put-string(ptr1, 1) = "1". */

message 'ptr1 size: ' get-size(ptr1) view-as alert-box.

str1 = base64-encode(ptr1).
```

```
message 'strlen: ' length(str1) view-as alert-box.
```

Converted java-class works a little differently because of implementation of length function for the longchar argument. For this function the converted code is DynamicOps.length(str) and during execution it goes to TextOps.length(Text, character) where the second argument is null. It seems, it is not implemented yet for the longchar. By the way it's really not obvious what to return in that case: length of string or length of byte array. I think that the second choice is close to truth.

#59 - 10/11/2013 03:50 PM - Vadim Gindin

- File second.png added

- File first.png added

- File third.png added

Greg Shah wrote:

> Code Review 1010a
> [...]
> 4. In LogicalTerminal, why do this:
>
> [...]
>
> As far as I know, the text should just be truncated here. The VIEW-AS ALERT-BOX case does not use LogicalTerminal.message() but instead it uses LogicalTerminal.messageBox(). So it seems to me that we should just truncate the string as soon as the SkipEntity is encountered.

The key is that the truncation is going in the message impl but not in the skip impl. As I understood the skip in my testcases only adds a \n symbol to message string. See following test:

```
message "first" skip
        "second" skip
        "third" skip view-as alert-box.
```

Works correctly. The alert-box contains all three strings one in each row. This code is almost equal to following:

```
message "first\nsecond\nthird" view-as alert-box.
```

It also works correctly, showing all three strings. The only difference is that in the first case the alignment in the alert box is by right-boundary and in the second case - is by left-boundary.

Conclusion: truncation goes in the message impl, not in the skip impl. Therefore I corrected only LogicalTerminal.message().

I wrote another one test:

```
 message "one \ntwo \nthree".
 message "1 \n\r2 \n\r3".
 message "11 \r\n12 \r\n13".
 message "21 \r22 \r23".
```

In all cases result is truncated to first special character. You are right, my implementation was incorrect and sure in this situation is sufficient to process only \n and \r to cover this cases. I also was wrong, when I thought that view-as alert-box works correctly! It does (except alignment) only for \n as I wrote above. But if the message string contains \r there are real visual defects. Here is the test:

```
 message "1 \n\r2 \n\r3" view-as alert-box.
 message "11 \r\n12 \r\n13" view-as alert-box.
 message "21 \r22 \r23" view-as alert-box.
```

I've attached 3 screen shots to show sequenced screens (see png's).
When you'll see these images you'll find that there are several errors there:
1) Probably \r as a carriage return character really moves carriage to beginning of new line and ignores alert-box boundaries. 2 and 3 are in the start of the lines.
2) The second alert-box is almost correct except on space before 13 and of course it didn't erase 2 and 3 from the first screen
3) The third screen is the most cryptic. 22 is gone. and 23 overwrites 2 from the first screen.

I suspect that the reason can be in a terminal specific features. Anyway I'm going far away from the original task. May be it makes sense to create separate task for this message problem?

#60 - 10/11/2013 05:10 PM - Greg Shah

> For this function the converted code is DynamicOps.length(str) and during execution it goes to TextOps.length(Text, character) where the second argument is null. It seems, it is not implemented yet for the longchar.

Please make the following fixes:

1. In DynamicOps.length(bdt, type), if type null, then set type to "character". This affects 2 methods. That is supposed to be the default measurement unit for the LENGTH() builtin function.
2. In TextOps.length(c, type), if type null, then set type to "character". This affects 2 methods length(Text, character) and length(String, character). That is supposed to be the default measurement unit for the LENGTH builtin function.

If you make these changes, will that be enough for your testcase to work?

> By the way it's really not obvious what to return in that case: length of string or length of byte array. I think that the second choice is close to truth.

The default is supposed to be character, so length of string is OK. The "RAW" measurement type is the one to use for the length of the byte array.

> The key is that the truncation is going in the message impl but not in the skip impl.

Yes, I agree.

> Conclusion: truncation goes in the message impl, not in the skip impl. Therefore I corrected only LogicalTerminal.message().

The problem is that your change in LogicalTerminal.message() was incorrect. That method is not used for VIEW-AS ALERT-BOX. LogicalTerminal.message() is only for the "regular" MESSAGE statement, which must always be on a single line because the SKIP truncates the text that is displayed. So, for LogicalTerminal.message() all we need to do is to immediately truncate the text and break out of that loop that is processing the args. There is no need to actually add newlines or other processing since it can't ever be displayed.

> But if the message string contains \r there are real visual defects.

Please create a new Bug task and put the recreate testcase, the captured images and other information there. Create the task in the P2J -> User Interface sub-project.

Then do finalize the changes I have requested and upload a new drop of the code for review.

#61 - 10/14/2013 06:44 AM - Vadim Gindin

- File vig_upd20131014a.zip added

I've made proposed corrections. It was enough to make my test case working. Here is and archive.

#62 - 10/14/2013 09:41 AM - Greg Shah

Code Review 1014a

1. DynamicOps and TextOps are both missing a history entry.

2. This code in if(type null) in DynamicOps needs to be this: if (type null).

3. This DynamicOps code also needs changing from:

```
   public static integer length(BaseDataType bdt)
   {
      return DynamicOps.length(bdt, (String)null);
   }
```

to this:

```
   public static integer length(BaseDataType bdt)
   {
      return DynamicOps.length(bdt, "character");
   }
```

4. In TextOps:

```
   public static integer length(Text c, character type)
   {
      if (type == null || type.isUnknown())
      {
         return length(c, "character");
      }
```

This seems wrong. If type is unknown, does Progress return the number of characters? I suspect forcing the type is only valid when it is passed as null.

This problem also exists in your change to the length(String, character) version.

5. I think TextOps.length(Text c, String type) also needs to be protected:

```
   public static integer length(Text c, String type)
   {
      if (c == null || c.isUnknown())
      {
         return new integer();
      }
      else
      {
         return TextOps.length(c.getValue(), type == null ? "character" : type);  // <-- my changes are here
      }
   }
```

6. It probably makes sense to use a well known public constant instead of all these hard coded references to "character".

7. I think the change to memptr is unnecessary. Please call BinaryData.lengthOf() which calls memptr.lengthWorker() which returns size.

8. What about the changes to LogicalTerminal and ThinClient?

#63 - 10/14/2013 04:18 PM - Vadim Gindin

- File vig_upd20131014b.zip added

I made a corrections and attached an archive.

About LogicalTerminal and ThinClient I want to say following.

1. First of all here is 2 ways in one direction: message view-as alert-box and simple message, and 2 ways in the other direction: output to terminal and file. message view-as alert-box statement do not truncate the message string. It means there is no need to make modifications in LogicalTerminal.messageBox() method (except the special case for \r). This message statement outputs an empty file. But when using the simple message statement the procedure outputs truncated lines and normal output file - not truncated.

2. Special case for the \r symbol. Is described in a new task #2193 and here above.
By the way I found very mysterious test:

```
message "111 \r222 \r333".
```

It outputs 333!

3. I tested skip test again

```
 output to skip_test.
 message "one \ntwo \nthree".
 message "1 \n\r2 \n\r3".
   message "11 \r\n22 \r\n33".
   message "111 \r222 \r333".
```

Converted class contained:

```
            message("one \\ntwo \\nthree");
            message("1 \\n\\r2 \\n\\r3");
            message("11 \\r\\n22 \\r\\n33");
            message("111 \\r222 \\r333");
```

It escape slashes so truncation didn't work.

May be I just didn't understand you correctly, but I suppose that all that story with "message" statements truncation and visual effects is very cryptic, and I suggested earlier to extract all the work on it in a new created task #2193. What do you think?

Do you insist to include the changes in the LogicalTerminal and ThinClient int the current change?

#64 - 10/15/2013 06:56 PM - Greg Shah

Code Review 1014b

I am OK with these changes. Get them regression tested.

> May be I just didn't understand you correctly, but I suppose that all that story with "message" statements truncation and visual effects is very cryptic, and I suggested earlier to extract all the work on it in a new created task #2193. What do you think?

OK.

> Do you insist to include the changes in the LogicalTerminal and ThinClient int the current change?

No.

Please add all the rest of the problem details to #2193 so that none of your findings are lost.

#65 - 10/17/2013 01:32 PM - Vadim Gindin

committed to bzr. Revision 10399.

#66 - 10/17/2013 02:12 PM - Greg Shah

What is the status of regression testing? (I know the answer but you were supposed to post it here so there is documentation)

#67 - 10/17/2013 05:32 PM - Vadim Gindin

I'm sorry, I forgot..
It passed regression testing.

#68 - 10/18/2013 08:52 AM - Greg Shah

- Status changed from WIP to Closed

- % Done changed from 0 to 100

#69 - 11/16/2016 11:43 AM - Greg Shah

- Target version changed from Milestone 7 to Runtime Support for Server Features

Files

vig_upd20131003a.zip	7.4 KB	10/03/2013	Vadim Gindin
vig_upd20131010a.zip	175 KB	10/10/2013	Vadim Gindin
first.png	5.5 KB	10/11/2013	Vadim Gindin

second.png	4.24 KB	10/11/2013	Vadim Gindin
third.png	4.02 KB	10/11/2013	Vadim Gindin
vig_upd20131014a.zip	35.6 KB	10/14/2013	Vadim Gindin
vig_upd20131014b.zip	27.6 KB	10/14/2013	Vadim Gindin