Base Language - Feature #3741

implement complete support for the CALL object handle and related attributes and methods

10/04/2018 01:10 PM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

History

#1 - 10/04/2018 01:11 PM - Greg Shah

The application POC which we are working on uses CREATE CALL and a pretty complete set of the attributes and methods to dynamically invoke code. Please implement complete support for this feature set.

#2 - 10/04/2018 03:17 PM - Constantin Asofiei

- Status changed from New to WIP

#3 - 10/04/2018 03:44 PM - Constantin Asofiei

I think parameter passing/validation will be one of the complex parts. 4GL has a validation done at the set-parameter method which checks that we have a valid store (variable/temp-table field only) for output/input-output parameters. It gives a runtime message when this constraint is not met: SET-PARAMETER output parameter <p-num> must be a program variable or temp-table field (subscript expressions are ok). (10060).

As the set-parameter's mode argument can be an expression, we can't just decide to wrap at conversion time the param's value in something to mark it as a variable. We will need to distinguish at runtime between a variable (or func/proc parameter) and a random value - this can be done at UndoableFactory and TypeFactory, as these are used to define variables and parameters. For fields, we can wrap any field reference passed as the param's value using FWD's FieldReference, which will give us access to underlying field info (buffer, etc).

#4 - 10/04/2018 04:30 PM - Constantin Asofiei

For handle attributes and methods, there's interesting stuff, too:

- there is a kind of runtime-validation of the given name: if this is not a valid attribute/method (for any handle), we will have a CALL:INVOKE finds an invalid CALL:CALL-NAME for GET/SET-ATTR-CALL-TYPE. It must be a valid 4gl widget attribute or method. (10092) error. This is does not look easy to solve, but see bellow for an idea.
- after that, 4GL just attempts to access the attribute/method for that handle; this should be easy, just determine the FWD API associated with this legacy method/attribute and invoke that dynamically, using the handle's resource.

The part we really need to implement is the method/attribute prefix alias: for example, move-after-tab-item has as alias move-after and any other 10 or more char prefix for the original move-after-tab-item; we can either pass this at the LegacyMethod/LegacyAttribute annotations, or rely on ProgressLexer.keywords to compute the prefixes - this should be OK as this relies on the full name (same as the legacy annotation). We can always validate this assumption by checking all the legacy interfaces (if all legacy annotations have the same name as the configured ProgressLexer.keywords which are also attributes/methods).

If we have this info from the parser (a mapping for all possible prefixes for any attribute/method to its full name), then we can use this to check if the call-name is a 4GL attribute/method name. If not, raise the 10092 error.

#5 - 10/04/2018 04:38 PM - Greg Shah

Please take a look at ProgressAst.keyword() and ProgressAst.keywordAll(). Is this what you are looking for?

#6 - 10/04/2018 05:07 PM - Constantin Asofiei

Greg Shah wrote:

> Please take a look at ProgressAst.keyword() and ProgressAst.keywordAll(). Is this what you are looking for?

Thanks, this helps. But I'll need to know if a keyword is associated with a method/attribute, too - what we have in initializeAttributesDictionary in progress.g will need some refactoring.

#7 - 10/04/2018 05:37 PM - Constantin Asofiei

Created task branch 3741a from trunk rev 11285.

#8 - 10/04/2018 05:46 PM - Constantin Asofiei

These attributes are not supported at parse time in FWD; they are all related to 'Invoking a Windows DLL routine or a Unix shared library routine' - do we add this now?

```
LIBRARY
LIBRARY-CALLING-CONVENTION
ORDINAL
RETURN-VALUE-DLL-TYPE
```

#9 - 10/04/2018 05:52 PM - Greg Shah

Yes, please add them now.

#10 - 10/04/2018 05:54 PM - Greg Shah

For this POC we also need the built-in functions LIST-QUERY-ATTRS() and LIST-SET-ATTRS(). It seems pretty close to the area in which you are working. Please add these as part of this task.

#11 - 10/04/2018 05:56 PM - Constantin Asofiei

Greg Shah wrote:

> For this POC we also need the built-in functions LIST-QUERY-ATTRS() and LIST-SET-ATTRS(). It seems pretty close to the area in which you are working. Please add these as part of this task.

The problem here is that we will be able to list only attributes/methods which are supported by FWD's conversion and runtime (at least at stub level). If something is supported only at parse level (or not supported at all), it will not be included.

#12 - 10/04/2018 06:00 PM - Constantin Asofiei

OTOH, I think we should hard-code the values returned by these list-query-attrs and list-set-attrs builtin functions. The result for a CALL handle's list-query-attrs is:

```
ADM-DATA,TYPE,PRIVATE-DATA,HANDLE,ORDINAL,SERVER,UNIQUE-ID,EVENT-PROCEDURE-CONTEXT,CALL-NAME,CALL-TYPE,INVOKE,
IN-HANDLE,IS-PARAMETER-SET,NUM-PARAMETERS,RETURN-VALUE-DATA-TYPE,SET-PARAMETER,ASYNC-REQUEST-HANDLE,INSTANTIAT
ING-PROCEDURE,RETURN-VALUE-DLL-TYPE,LIBRARY-CALLING-CONVENTION,ASYNCHRONOUS,PERSISTENT,CLEAR,LIBRARY,RETURN-VA
LUE,EVENT-PROCEDURE
```

and the order in which these are returned makes no sense.

Also, list-set-attrs for a call handle gives a memory violation error in 4GL.

#13 - 10/04/2018 06:05 PM - Greg Shah

> Also, list-set-attrs for a call handle gives a memory violation error in 4GL.

Of course it does. :)

We won't duplicate this part.

#14 - 10/04/2018 06:08 PM - Greg Shah

I'm OK with hard coding the return values, however that does mean we'll have extra maintenance over time when Progress makes changes.

This has the advantage of being an exact match to the return, though it seems unlikely that the order will matter. On the other hand, we've seen stranger things.

If we based things off of the supported attributes, we won't match the order and would be missing unknown attrs/methods but it would not require extra maint.

I can accept either solution (until we hit an app that depends on the exact return value).

#15 - 10/15/2018 05:08 PM - Constantin Asofiei

Open issues which will be left after everything else is working or postponed:

- runtime for PROCEDURE-TYPE and THREAD-SAFE attributes can be postponed until we have the RUN equivalent (parsing and conversion is working).
- SET-PARAMETER - the argument set as an extent variable (with or without subscript) will be done last.

#16 - 10/15/2018 06:10 PM - Greg Shah

Constantin Asofiei wrote:

> Open issues which will be left after everything else is working or postponed:
>
> - runtime for PROCEDURE-TYPE and THREAD-SAFE attributes (parsing and conversion is working)
> - SET-PARAMETER - the argument set as an extent variable (with or without subscript)

Are you saying that we should not implement these now? Or are you saying they will be implemented last?

#17 - 10/15/2018 06:13 PM - Constantin Asofiei

Greg Shah wrote:

> Constantin Asofiei wrote:
>
> > Open issues which will be left after everything else is working or postponed:
> >
> > - runtime for PROCEDURE-TYPE and THREAD-SAFE attributes (parsing and conversion is working)
> > - SET-PARAMETER - the argument set as an extent variable (with or without subscript)
>
> Are you saying that we should not implement these now?

PROCEDURE-TYPE and THREAD-SAFE can be postponed until we have the equivalent from the RUN statement (as this is specific to latest OE version, 11.6 or 11.7).

> Or are you saying they will be implemented last?

The extent argument issue is delayed until everything else is working.

I've updated #3741-15 to reflect this.

#18 - 10/16/2018 12:07 PM - Constantin Asofiei

Greg, something related to ControlFlowOps: some time ago I had a thought of using a builder approach for invoking things; instead of using e.g. ControlFlowOps.invokeRemoteAsync use something like new Invocation(<target>).setServer(serverHandle).setAsync(true).set<other options>.invoke(). I don't want to make the conversion changes or refactor all ControlFlowOps now, but it may make sense to have this started for the CALL handle, to hide some stuff in the builder's invoke(), instead of using lots of IF's to test if we are async, remote, function or procedure, etc. This will make things easier to add new options for RUN, like SINGLE-RUN, SINGLETON.

The DLL-CALL-TYPE in 4GL is performed outside of the RUN statement emulation, it just invokes the library directly (without any validation of the arg modes, types, etc) - which can cause crashes if everything is not setup exactly. When using a PROCEDURE ... EXTERNAL, assuming that is defined correctly, RUN will validate against this signature - but if the signature is bad, it will too cause crashes in 4GL. This will require some refactoring of the NativeInvoker APIs, to access it directly.

#19 - 10/16/2018 12:13 PM - Greg Shah

> it may make sense to have this started for the CALL handle

Agreed, go with this.

> This will require some refactoring of the NativeInvoker APIs, to access it directly.

I assume you will only do this for DLL-CALL-TYPE and not for the RUN that targets a PROCEDURE ... EXTERNAL since the error handling would be incorrect.

#20 - 10/16/2018 12:41 PM - Constantin Asofiei

Greg Shah wrote:

> > This will require some refactoring of the NativeInvoker APIs, to access it directly.
>
> I assume you will only do this for DLL-CALL-TYPE and not for the RUN that targets a PROCEDURE ... EXTERNAL since the error handling would be incorrect.

Exactly.

If there is a CALL targeting a PROCEDURE ... EXTERNAL then that will go the normal path, as if RUN was used (actually, CALL doesn't know what

the target will be for PROCEDURE-CALL-TYPE, so everything else - including OS library calls - will work as if RUN was used).

#21 - 10/16/2018 03:24 PM - Constantin Asofiei

There is some weird stuff with OUTPUT/INPUT-OUTPUT parameters, when the data-type specified at SET-PARAMETER and the var's data type differs.

A simple example: var-type is "integer", set-parameter uses "char" and the target proc has "char". In this example, if the value received from the proc call is e.g. "00" string, it will convert to int value 12336, which is hex for 3030, which in turn stands for the ASCII code for the two chars in this string (48 is the ASCII code for the 0 char, and concatenating this in hex gives 0x3030).

Another example is this: var-type is "integer", set-parameter uses "char" and the target proc has "date", returning the today value. For a 10/16/18 date it will set the var to 0x31302f31362f3138 which is the hex ASCII codes for each char in the string.

The rule of converting the value I think is this:

- get the value as set by the procedure/function call
- convert it to the parameter data type set via SET-PARAMETER
- convert it to the actual data type of the argument (following some weird rules)

This might show that the variable/field reference is not directly passed to the emulated RUN call, but in turn is copied into a different storage, and that one is passed to the RUN call; and the value will be copied back to the set-parameter argument (Variable or temp-table field). This would explain the Buffer for output parameter 1 not available during SET-PARAMETER. (10062) errors when trying to use a permanent buffer field for output/input-output modes.

A small snippet of this theory is this:

```
procedure proc0.
def output param p1 as date.
p1 = today.
end.

def var h as handle.
def var i1 as int.

create call h.
h:call-type = procedure-call-type.
h:call-name = "proc0".
h:num-parameters = 1.

h:set-parameter(1, "char", "output", i1).
h:invoke().
message i1.
```

which gives 3544384795149545784 value (hex for 0x31302f31362f3138, the 10/16/18 date) - this might be because it interprets whatever value is at the received address as an integer value, and not as a string?

Also, in this test, if we replace "char" with "int" in the set-parameter call, it will give 2458409, which is the equivalent of integer(today) in 4GL.

This is just a first look into the edge cases; for full support, I need to test each combination of data type.

#22 - 10/16/2018 06:42 PM - Greg Shah

My first worry was that this was some kind of direct memory issue. For example, the value might be passed as a memory buffer and then read/interpreted as a different type. This is what happens with mis-matches in the native API calls.

But when I looked deeper into your examples, I see that this is really like the automatic conversion of polymorphic BDT values to the correct type which we see from features like DYNAMIC-FUNCTION(). That should be somewhat predictable, though I agree that the various conversions each need to be tested.

#23 - 10/17/2018 03:08 AM - Constantin Asofiei

Greg Shah wrote:

> My first worry was that this was some kind of direct memory issue. For example, the value might be passed as a memory buffer and then read/interpreted as a different type. This is what happens with mis-matches in the native API calls.

Actually, this is what I think is happening. They are building (using the set-parameter's data-type) temporary variables which are passed during invoke to the actual call (this is easily proved by using input-output and changing the var value after it was used in set-parameter - invoke will see only the value as used at set-parameter, will not see any changes after set-parameter was performed for that var). After the emulated RUN call, they are copying the memory from the temporary variable to the actual variable (how they actually know how many bytes to copy over is still a mystery... maybe they keep a common data structure for all variables and know the number of allocated bytes, something else). See bellow for an example:

```
procedure proc0.
def output param p1 as int.
def output param p2 as char.
p1 = 0x5051.  // this is PQ in text
p2 = "51". // ASCII char for 51 is char '3'
end.

def var h as handle.
def var i1 as char.
def var i2 as char.

create call h.
h:call-type = procedure-call-type.
h:call-name = "proc0".
h:num-parameters = 2.

h:set-parameter(1, "int", "output", i1).
h:set-parameter(2, "int", "output", i2).
h:invoke().
message i1 i2. // this shows PQ 3
```

When decimal is involved, ABL even crashes for certain values:

```
p2 = "123". // this value will crash ABL
...
def var i2 as dec. // if the caller's expected type is dec
...
```

I thought they were using the builtin functions to automatically convert from a data type to another type, something like this:

- build a variable of the set-parameter data type to pass it to emulated RUN
- depending on the actual data type of the set-parameter's 4th argument (the variable where to copy back the output), use i.e. integer, string, date, etc, to convert the intermediate/temporary variable to the expected caller's type. But it doesn't look this is happening, they are copying memory buffers instead...

I've been trying to find a rule how the memory buffer is interpreted for date types, but I can't find any - at least the results seems consistent. Things will complicate further if the results are dependent on the platform byte order, or specific OS implementation of openedge (if they in-memory structure for data types differs across OS).

#24 - 10/17/2018 05:18 AM - Constantin Asofiei

My plan at this time is this:

- implement the infrastructure to make everything working
- if there is difference between data type at set-parameter and the var type, log a warning and use the 4gl functions to convert. Copying memory buffers is unusable in some cases, and I'm not sure we should go down this road until we find a real life usage of this.

#25 - 10/17/2018 09:46 AM - Greg Shah

I think your examples are **not** showing direct memory usage.

From #3741-21:

> Another example is this: var-type is "integer", set-parameter uses "char" and the target proc has "date", returning the today value. For a 10/16/18 date it will set the var to 0x31302f31362f3138 which is the hex ASCII codes for each char in the string.

I think the presence of 0x2f in the ASCII output is the hint that there is no direct memory access. In the native API calls, they only honor data types that directly match C language types. This is done for a practical reason: all commercially popular operating systems (UNIX, Linux, Windows, MacOS...) currently define all APIs using C language types. Every language which has API bindings will map to these types. Some like C++ or Java map quite directly. Others such as the 4GL map more indirectly.

The C language does not have a primitive date type. For this reason, APIs which operate with dates or timestamps each have their own way to define their inputs. Almost always, these inputs will be comprised of one or more integer values. Milliseconds from epoch (01/01/1970) is a common example (a 64-bit integer value). Other cases might have a structure defined with multiple integer values (day, month, year, hours, minutes, seconds, mills).

I can't think of a common API usage that uses a string representation. In other words, it would be a very weird approach to directly store the date component separators (0x2f or / in ASCII) in memory. When we see those appear, it strongly suggests that a string conversion is going on.

It seems to me that this date case can be explained if there are 2 conversions happening. And this is possible since there are 3 lvalues involved and the data is transferred (and possibly converted) twice.

In your example:

lvalue1 = target proc output parameter of type date = assigned today which is the julian day value for 10/16/18
lvalue2 = set-parameter type specification of type char which when assigned will have the string "10/16/18"
lvalue3 = variable in caller to receive output from the procedure of type integer which when assigned will have the binary value of the character string 0x31302f31362f3138

Then the conversion is something like this:

lvalue1 --date to char type conversion-> lvalue2 --char to int type conversion-> lvalue3

I think your "51" to "3" conversion in the example in #3741-23 can also be explained by this "double conversion" approach:

lvalue1 = target proc output parameter of type char = assigned "51"
lvalue2 = set-parameter type specification of type int which when assigned from a string will have the binary value 51 of 0x33
lvalue3 = variable in caller to receive output from the procedure of type char which when assigned will have the value of ASCII 51 which is "3"

Then the conversion is something like this:

lvalue1 --char to int type conversion-> lvalue2 --int to char type conversion-> lvalue3

The only funny part here is that the char to int conversion in the second case is not done with the copying of the ASCII value but instead is done by converting the decimal encoded text as a number. In the first example the encoded date cannot be handled in that same way and the binary values of the ASCII text itself were copied. SO the char to int conversion must have some "intelligence" depending on the contents of the data.

There must also be some limits or behavior based on sizes of data and types that are not compatible with conversion.

I think it is important to get these rules figured out now, rather than rushing an implementation and having to tweak it forever to get it right.

On a positive side, if there is no direct memory access it is easier to implement. I hope I'm right.

#26 - 10/25/2018 05:23 PM - Constantin Asofiei

- File call-dec-char-char.txt added

A dec-char-char case (var data type, set-parameter data type, procedure parameter data type), shows some interesting stuff; the rules I think are these:

- first char is ignored
- always negative
- if the length of the string (without first char) is more than 36 chars (288 bits), then set it to 0
- read the char bytes, and first push them to the fractional digits (max 49 digits). After the fractional digits are filled, the left-side is filled with the 'overflow' (from the right side) of the fractional digits, something like this, for a 12345678911122233344498765432101234 56 string:

```
  2 3 4 5 6 7 8 9 1 1 1  2 2 2 3 3 3 4 4 4 9 8 7 6 5 4 3 2 1 0 1 2 3 4 5 6
-32333435363738393131313.232323333333434343938373635343332313031323334353 6
```

If the length of the string is less than 25 chars (not enough digits to fill the fractional part), 4GL fills these with some random values, as two runs of the same program produces different output - see attached for a sample output. The lines are in pairs, first is the string passed as argument, and second line contains the:

- string output of the decimal var as reported by 4GL
- hex dump of the decimal via the put-float and dumping this to hex
- hex dump of the decimal by using the string(dec) function and dumping the string content to hex

In FWD, I'm just filling the missing digits with 0, and not random values.

Also, even if 4GL assumes the decimal can have at most 10 fractional digits, this shows something different; and trying to do arithmetic operations with this 288 bit value (23 non-fractional and 39 fractional digits), 4GL will abend - it will not be able to compute.

Same for integer, it can end up holding a 64 bit value, while this can hold only 32-bit values normally.

Now I'm trying to figure out how to convert the date-char-char case (i.e. how date is read from the memory buffer). I have some ideas to produce a large enough value set, but I'm not sure how long will take to interpret and find the rules. I hope it's something like converting an integer value to a date (i.e. adding days to some base offset and producing the date from that).

#27 - 10/26/2018 04:38 PM - Constantin Asofiei

Ovidiu, I think there is a bug in date.assign(BaseDataType) - if the current date instance is unknown and the argument is a i.e. NumberType non-unknown value, the date.unknown flag is not cleared - I assume not clearing this was not by and this is a bug, correct?

#28 - 10/29/2018 04:07 AM - Constantin Asofiei

An ETA for what is left:

- the dll-call-type - I'll finish this and have a cleaned up version by tomorrow, to put it in regression testing (full ETF and MAJIC) and for intermediate review; we can do this because what is left will not affect our automated testing runtime, as will be specific to the CALL handle.
- what is left is writing more testcases to cover more combinations of data types. SET-PARAMETER allows all data types, so configuring all 15 combinations for 3 possible places gives 3375 combinations and this is too much, so I'll combine only var and set-parameter first and going deeper only for what seems more complicated. I think ~2-3 days to have these and interpret them. I have some ideas to automate these tests so not rely on manual comparison of output files.

So the ETA for what is left is end of this week/early next week.

#29 - 10/29/2018 06:35 AM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> Ovidiu, I think there is a bug in date.assign(BaseDataType) - if the current date instance is unknown and the argument is a i.e. NumberType non-unknown value, the date.unknown flag is not cleared - I assume not clearing this was not by and this is a bug, correct?

Yes, you are correct. It seems that I forgot to reset the unknown flag. Please add the necessary code for all types that need it.

#30 - 10/29/2018 02:28 PM - Constantin Asofiei

Forgot about runtime/conversion of:

- extent arguments
- field reference arguments
- by-reference and append argument modes

I'll try not to affect the estimate.

#31 - 10/29/2018 03:39 PM - Constantin Asofiei

Some other weird things related to DLL-CALL-TYPE: if the LIBRARY attribute is not set BEFORE the SET-PARAMETER, then the invoke I think fails (is not performed?).

This might be explained if they do some preparation of the parameters during SET-PARAMETER, if LIBRARY attribute is set; and why 4GL abends in some cases, when LIBRARY is set some time between the first and the last SET-PARAMETER call (when setting the parameters).

#32 - 10/29/2018 04:39 PM - Constantin Asofiei

Ovidiu, please confirm another possible bug: BaseDataType.generateDefault now generates an unknown value, as the BDT default c'tors (in

sub-classes) have been changed to create an unknown instance, correct?

#33 - 10/29/2018 07:43 PM - Constantin Asofiei

Constantin Asofiei wrote:

> Ovidiu, please confirm another possible bug: BaseDataType.generateDefault now generates an unknown value, as the BDT default c'tors (in sub-classes) have been changed to create an unknown instance, correct?

My bad, I missed the return bdt.instantiateDefault(); call; so the API is OK.

#34 - 10/30/2018 01:01 PM - Constantin Asofiei

Greg, documentation states that if the OUTPUT/INPUT-OUTPUT variable is out of scope, then terminate with an error condition. Actually, 4GL abends. We can either:

- add the overhead of tracking the scope of each defined variable
- just let it assign - the user will not be able to access the var (as it was out of scope), but the call will not stop because of this.

#35 - 10/30/2018 01:14 PM - Greg Shah

Applications can't depend upon behavior that abends in the 4GL. It is safe to let it assign. Put notes in the Javadoc that we are deliberately NOT tracking the variable scope to reduce overhead and that any assignment to something out of scope thus will not generate an error. Also, do mention that the reason was that the 4GL abends so there was no feature to match.

#36 - 10/30/2018 03:37 PM - Constantin Asofiei

Ovidiu, please help me with something: shouldn't TM.deregisterOutputParameterAssigner be called in case the i.e. RUN is not performed at all? I think we currently have a leak, if the RUN is not performed (i.e. argument modes not OK, etc), then all the AbstractParameter instances remain registered (as they register themselves in the current scope when they are created). These need to be cleaned up if the call is not even performed.

#37 - 10/30/2018 04:34 PM - Ovidiu Maxiniuc

The TM.deregisterOutputParameterAssigner() is called before the call of an procedure/function and native API routine, even before the init() of the respective block (but before its pre() code). Is it possible that somewhere is a decision that the block will not be called after the wrap -s were called?

OTOH, if an assigner is left on a block scope, it will be dropped as the scope is removed from stack. If the assigner is for current scope (a new scope is pushed to stack after TM.deregisterOutputParameterAssigner() is called) and somehow not deregistered, then any attempt to re-register will fail because registerOutputParameterAssigner() will check whether the outputParmAssigner of the top block is null. Is that your case? Are you getting an IllegalStateException("Illegal attempt to register output parameter assigner") ?

#38 - 10/30/2018 04:43 PM - Constantin Asofiei

I'm getting this:

```
Caused by: java.lang.IllegalStateException: The extent parameter reference was not set, before the function/pr
```

```
ocedure end !
        at com.goldencode.p2j.util.AbstractExtentParameter.assign(AbstractExtentParameter.java:204)
        at com.goldencode.p2j.util.AbstractParameter$Scope.processAssignments(AbstractParameter.java:316)
        at com.goldencode.p2j.util.BlockManager.topLevelBlock(BlockManager.java:7336)
        at com.goldencode.p2j.util.BlockManager.internalProcedure(BlockManager.java:394)
        at com.goldencode.p2j.util.BlockManager.internalProcedure(BlockManager.java:380)
        at com.goldencode.testcases.CallRun2.proc0(CallRun2.java:42)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at com.goldencode.p2j.util.ControlFlowOps$InternalEntryCaller.invokeImpl(ControlFlowOps.java:5540)
        at com.goldencode.p2j.util.ControlFlowOps$InternalEntryCaller.invoke(ControlFlowOps.java:5513)
        at com.goldencode.p2j.util.ControlFlowOps.invokeImpl(ControlFlowOps.java:4387)
        at com.goldencode.p2j.util.ControlFlowOps.invoke(ControlFlowOps.java:3408)
        at com.goldencode.p2j.util.ControlFlowOps.invokeImpl(ControlFlowOps.java:4062)
        at com.goldencode.p2j.util.ControlFlowOps.invokeImpl(ControlFlowOps.java:3991)
        at com.goldencode.p2j.util.ControlFlowOps.invokeWithMode(ControlFlowOps.java:415)
        at com.goldencode.p2j.util.ControlFlowOps.invokeWithMode(ControlFlowOps.java:397)
        at com.goldencode.testcases.CallRun2.lambda$0(CallRun2.java:29)
```

For a test like this:

```
def var i as int extent 2.

procedure proc0.
def output param i1 as int extent 2.
i1 = 5.
end.

i = 4.

run proc0 ("abc", output i) no-error.

run proc0 (output i).

message i[1] i[2].
```

See how the first call is not being performed because of an argument mismatch, but in FWD the ExtentExpr instance remains registered and is leaked to the second call.

#39 - 10/30/2018 05:28 PM - Ovidiu Maxiniuc

Indeed, there is one way out before the new block is processed: the ControlFlowOps.invokeImpl() ~ line 3350. Here, there are multiple exit points before invoking actual code being invoked. In this case the execution flow returns because validArguments() returns with mismatched parameters exception - as expected.

I think in all these cases we need to do the proper cleanup, in this case to call deregisterOutputParameterAssigner(). Probably the easiest way is to put the method at invokeImpl() in a big finally. Calling multiple times deregisterOutputParameterAssigner() has no side effects. If the subroutine was successfully called, then the output parameters are already assigned back.

#40 - 10/30/2018 07:39 PM - Constantin Asofiei

3741a rev 11293 is under MAJIC testing now.

What is left are to finish the conversion of output parameters and javadoc for some of Call.java code (at the end of the file). Note #3741-38 issue is not fixed yet.

Greg, if you want, please do an initial review of 3741a.

Eric: please let me know if you can help me with ETF testing; you will need to reconvert - some changes are expected, depending on which handle attributes are in use.

#41 - 10/30/2018 08:02 PM - Eric Faulhaber

Constantin Asofiei wrote:

> Eric: please let me know if you can help me with ETF testing; you will need to reconvert - some changes are expected, depending on which handle attributes are in use.

Sure, just let me know when the final conversion changes are ready.

#42 - 10/31/2018 11:46 AM - Constantin Asofiei

3741a rev 11295 fixes some regressions and it passed MAJIC main and conversion testing.

Eric: please go ahead with ETF testing (runtime and conversion) using 3741a rev 11295; also, please send me the diffs (or the baseline and converted sources) once you have the converted code.

#43 - 10/31/2018 12:41 PM - Eric Faulhaber

OK, converting now.

#44 - 10/31/2018 12:51 PM - Greg Shah

Code Review Task Branch 3741a Revision 11295

Wow! That is a massive code drop. It took me all morning to review it. It is very good, though I can't be sure that all the invocation logic and error handling is perfect. Some minor feedback:

1. The assignDirect and variable members of BDT should probably be marked transient so it is clear they don't get serialized. Up until now we've deliberately kept instance data out of that class.

2. When HandleOps.listQueryAttributes() inspects annotations to make the list of query attributes, shouldn't it check !la.setter()?

3. In Call.java, the setCallName(String callName) method references the ordinal member in its error checking instead of teh callName parameter.

4. I think the support level for THREAD-SAFE should be runtime stubs instead of none.

5. The Call inner classes CallParameter, ExtentCallParameter and NativeCallParameter and their methods need javadoc. You may be waiting on that until the functionality is complete.

6. The Clearable.java module comment is listed as EventProcedure.java.

7. AbstractParameters needs a history entry when you address the TODO.

#45 - 10/31/2018 03:10 PM - Constantin Asofiei

Greg Shah wrote:

> 1. The assignDirect and variable members of BDT should probably be marked transient so it is clear they don't get serialized. Up until now we've deliberately kept instance data out of that class.

Thanks, fixed.

> 2. When HandleOps.listQueryAttributes() inspects annotations to make the list of query attributes, shouldn't it check !la.setter()?

This API is used with another intent, to populate all the cache 'legacy name to java method name' maps. And the use is OK, as it will not collect the same name twice - and I don't think there is a case where an attribute can be written, but not read. I'll fix the javadoc.

> 3. In Call.java, the setCallName(String callName) method references the ordinal member in its error checking instead of teh callName parameter.

That's expected, you can't have ORDINAL set when trying to set CALL-NAME (and vice-versa).

> 4. I think the support level for THREAD-SAFE should be runtime stubs instead of none.

Fixed.

> 5. The Call inner classes CallParameter, ExtentCallParameter and NativeCallParameter and their methods need javadoc. You may be waiting on that until the functionality is complete.

Correct.

> 6. The Clearable.java module comment is listed as EventProcedure.java.

Fixed.

> 7. AbstractParameters needs a history entry when you address the TODO.

Yes, I know. I think I know how to fix it.

#46 - 11/01/2018 02:42 AM - Eric Faulhaber

Constantin Asofiei wrote:

> please send me the diffs (or the baseline and converted sources) once you have the converted code.

See devsrv01:/tmp/srcs_full_201810*.zip.

#47 - 11/01/2018 02:39 PM - Constantin Asofiei

Eric, the diffs are OK, please go ahead with runtime testing (if you haven't already).

#48 - 11/01/2018 02:59 PM - Eric Faulhaber

I'm running the last round (batch) now. The other two have passed.

#49 - 11/01/2018 03:05 PM - Eric Faulhaber

Greg Shah wrote:

> The assignDirect and variable members of BDT should probably be marked transient so it is clear they don't get serialized. Up until now we've deliberately kept instance data out of that class.

The addition of instance data to BDT for a feature that AFAICT is not used in many use cases is concerning to me, in that we are enlarging the memory footprint of every BDT instance with this instance data, all the time. This can really add up when a lot of database records are held in memory (e.g., all temp-tables, and any records held in the JDBC driver while processing queries). The number of records in memory at any given time across multiple sessions can be quite large.

#50 - 11/01/2018 03:16 PM - Eric Faulhaber

Eric Faulhaber wrote:

> ...all temp-tables, and any records held in the JDBC driver...

Actually, temp-table records will still be managed as before by the database storage engine and record data in the JDBC driver are stored according to the driver's implementation, but anything in object (DMO) form will suffer from this expansion. This includes all records in buffers, or pinned as undoable in open transactions. Still can add up to a big footprint.

#51 - 11/01/2018 03:27 PM - Constantin Asofiei

Eric Faulhaber wrote:

> Eric Faulhaber wrote:
>
> > ...all temp-tables, and any records held in the JDBC driver...
>
> Actually, temp-table records will still be managed as before by the database storage engine and record data in the JDBC driver are stored according to the driver's implementation, but anything in object (DMO) form will suffer from this expansion. This includes all records in buffers, or pinned as undoable in open transactions. Still can add up to a big footprint.

I understand. Unfortunately, the 4GL's check 'is this a variable or field?' is done at runtime, and not compile time. The only alternative I see (to keep the functionality) is to sub-class everything as integervar, decimalvar, etc, which would in turn implement a BaseDataTypeVar interface (and also extend the associated type) - and these new types will be used by our TypeFactory (which is FWD's way of creating variables).

Should I go with this?

#52 - 11/01/2018 03:32 PM - Greg Shah

No, this will create a huge mess and much confusion.

If this really creates too much overhead to be included in BDT, then can we pass a wrapper for the lvalue given to set-parameter that contains this configuration?

#53 - 11/01/2018 03:39 PM - Constantin Asofiei

Greg Shah wrote:

> If this really creates too much overhead to be included in BDT, then can we pass a wrapper for the lvalue given to set-parameter that contains this configuration?

Hm... yes, conversion time can check if the 4th argument is a standalone variable; and wrapping this into something like BaseDataTypeVariable should work.

#54 - 11/01/2018 04:43 PM - Eric Faulhaber

3741a/11295 has passed all ETF testing.

#55 - 11/01/2018 04:49 PM - Constantin Asofiei

Eric, thanks!

Greg: I have another issue, this time in BinaryData.setString - this assumes the number of bytes in the string is the same as the string's length - but this is not the case, if the string contains multi-byte characters. String.length() gives the number of characters, not the bytes.

I'm changing it to use String.getBytes().length instead of String.length(), on line 1509 which sets the length, len = data.length() + 1;.

#56 - 11/01/2018 05:14 PM - Greg Shah

Using getBytes().length makes sense.

#57 - 11/01/2018 07:12 PM - Constantin Asofiei

I have a heavy duty test which uses lots of PUT statements - and what 4GL does in 1-2 seconds FWD does in 10 to 30. I think some part in this (~5-10% from the difference) is because of usage of readObject and writeObject to handle native array cases, in ClientState and ServerState classes (this was fixed easily with a custom IntArraySerializer, added to our NativeTypeSerializer) and in Protocol$Reader/Writer.run, where the byte array is managed (replaced with a ByteArraySerializer added to NativeTypeSerializer).

There are other usages like this throughout FWD. It might be worth exploring if there some gain by replacing this (especially the object correspondents for Java native types which use i.e. val = (Integer) in.readObject() or val = (String) in.readObject()) in widget config classes and other network-transferred objects, to avoid going the Serializable way instead of Externalizable.

#58 - 11/06/2018 07:34 PM - Constantin Asofiei

Greg, 3741a rev 11298 contains the final changes related to CALL. I've not included the changes I mentioned in #3741-57 - I will create a separate task and put them there, they need to be explored/tested on their own, for performance POV.

There is some other issue related to RAW and GET/PUT-STRING: this must interpret the characters as single-byte, and not multi-byte; I've made changes to use the ISO-8859-1 charset when working with the String's bytes. I'm not sure how correct this is.

#59 - 11/07/2018 10:17 AM - Constantin Asofiei

Rebased 3471a from trunk rev 11289 - new rev 11299.

#60 - 11/07/2018 11:29 AM - Constantin Asofiei

Note that I couldn't find a way to get rid of the BaseDataType.assignDirect flag - this is still in use.

#61 - 11/07/2018 12:29 PM - Greg Shah

Code Review Task Branch 3741a Revision 11299

The changes are good.

I think the only remaining issue is the BDT.assignDirect. It is used for such a limited requirement, that we really don't want to take the overhead in every BDT forever. I think the following would be a better solution:

- Add a public boolean isAssignDirect() { return false; } to BDT.
- In Call.copyOutput(), when int64 or decimal are used, wrap the instance in an int64 or decimal anonymous subclass that overrides public boolean isAssignDirect() { return true; }.
- Modify the int64.setValue() and decimal.setValue() to call isAssignDirect().

I realize that it makes Call.copyOutput() a bit more complex but not too much. The trade off is well worth it to retain efficiency in all other BDT usage.

#62 - 11/07/2018 12:30 PM - Greg Shah

In regard to #3741-57, I agree it is worth looking into this performance case. Please open another task for this.

#63 - 11/07/2018 12:31 PM - Greg Shah

Sorry, I see you already did this with #3789. Nice!

#64 - 11/07/2018 01:29 PM - Constantin Asofiei

Greg Shah wrote:

> I think the only remaining issue is the BDT.assignDirect. It is used for such a limited requirement, that we really don't want to take the overhead in every BDT forever. I think the following would be a better solution:

I don't think this will work. In CallParameter.copyOutput, the CallParameter.value is a direct reference to the variable passed to SET-PARAMETER. So creating another instance will require changing this reference, which is not possible without an approach similar to how we change references for extent variables. And if we are changing references, this will make it complicated to switch this reference in any other FWD part which contains a reference to it - like TransactionManager for undo support.

I can comment this out and document it as a known limitation.

#65 - 11/07/2018 04:23 PM - Greg Shah

Can you use a local int64 or decimal subclass for the conversion and then assign back the known-valid value to the original var reference?

#66 - 11/07/2018 06:29 PM - Constantin Asofiei

Greg Shah wrote:

> Can you use a local int64 or decimal subclass for the conversion and then assign back the known-valid value to the original var reference?

Hmm... I think this is the way to go (the idea is to copy the BDT state directly from this local subclass instance to our var reference). Will fix it tomorrow.

#67 - 11/08/2018 12:04 PM - Constantin Asofiei

Greg, thanks for the idea - I've implemented this in 3741a rev 11300.

#68 - 11/08/2018 04:00 PM - Greg Shah

Code Review Task Branch 3741a Revision 11300

I'm good with the changes. You can merge to trunk when ready.

#69 - 11/08/2018 04:03 PM - Constantin Asofiei

Greg Shah wrote:

> Code Review Task Branch 3741a Revision 11300
>
> I'm good with the changes. You can merge to trunk when ready.

It passed runtime/conversion testing, I'll merge it shortly.

#70 - 11/08/2018 04:14 PM - Constantin Asofiei

- % Done changed from 0 to 100

Branch 3741a was merged to trunk rev 11290 and archived.

#71 - 11/08/2018 04:15 PM - Greg Shah

- Status changed from WIP to Closed

Files