Base Language - Feature #1921

Feature # 1606 (Closed): implement persistent procedures/super procedures

implement super-procedures

11/12/2012 03:53 AM - Constantin Asofiei

Status:	Closed	**Start date:**	11/09/2012
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	60.00 hours
Target version:	Runtime Support for Server Features		
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Base Language - Bug #1971: emit worker versions for functions to f...	**Closed**	**08/07/2013**	**08/08/2013**
Related to Base Language - Feature #2138: finish SEARCH-TARGET support	**Closed**	**08/09/2013**	**08/16/2013**

History

#1 - 11/12/2012 03:54 AM - Constantin Asofiei

- Parent task set to #1606

Implement the following 4GL features:
FUNCTION ... IN SUPER
PROCEDURE ... IN SUPER
RUN SUPER statement
SUPER function

SESSION handle:
super-procedures
add-super-procedure()
remove-super-procedure()

THIS-PROCEDURE/TARGET-PROCEDURE/SOURCE-PROCEDURE handle:
super-procedures
add-super-procedure()
remove-super-procedure()

#2 - 11/15/2012 08:26 AM - Constantin Asofiei

Hi Greg,

Thanks for adding me in- that is interesting, but what a nightmare!

Why did Progress have to come up with such a bizarre implementation for SUPER functionality? If they had made the stack static for each procedure - which I guess most people would have been happy to use - then you wouldn't have to worry about this.

As it is, because we can mess about with the stack whenever we want, you have to write a complete generic implementation.

I know it's unlikely, but would it possible to build some constraints into the conversion - whereby we could somehow declare that we are using a static stack, and what it is (e.g. via comments), and then the conversion could use the simpler, more intuitive java inheritance?

Regards,
Guy

#3 - 11/15/2012 08:36 AM - Constantin Asofiei

If we can somehow use hints to hard code the super procedure hierarchy and, during conversion, extract the entire hierarchy into interfaces, I think something could be done. But, with this, we will need to be sure that:

- all super-function/procedure calls are static calls
- the super-procedure name is not dynamic determined
- all parameters are well defined, i.e. the code will not send a character for an int param (even if the char converts to a valid int value, as 4GL allows these kind of cases).
- plus other issues I can't think of at this time

Greg: I suggest to let this sub-task handle the generic super-procedure implementation and have another sub-task to which we can identify if and how we can convert the super procedure chain into interface hierarchy.

#4 - 11/15/2012 08:59 AM - Greg Shah

Agreed. See #1922.

#5 - 11/19/2012 05:30 AM - Guy M

Just a note, I think we do probably have dynamic calls, even though the SUPER-PROCEDURE stack is effectively static. For instance, in ADM2, we make use of {fn} {fnarg} - which resolve to DYNAMIC-FUNCTION calls.

In our API framework, there's probably little in the way of dynamic calls, but I've already found at least one example (abstnas0.p), although that particular case could easily be swapped out to static call.

#6 - 11/19/2012 10:01 AM - Constantin Asofiei

SUPER-PROCEDURES attribute:
- contains the string representation of the super procedure stack
- the top of the stack is on the left-side
- the string representation is the ID of each handle, separated by commas

ADD-SUPER-PROCEDURE method
- returns false if the target is a Web service procedure or is not a valid handle
- has two parameters: super-proc-handle and proc-search
- if the super-proc-handle already exists in the super-procedures stack, it will move it to the top of the stack.
- applies to session and procedure handles

SEARCH-SELF and SEARCH-TARGET modes (the following applies for both super-functions and super-procedures):
- proc-search parameter for ADD-SUPER-PROCEDURE can be: SEARCH-SELF (default) or SEARCH-TARGET
- proc-search is OPTIONAL. If is not set, it defaults to SEARCH-SELF.
- the proc-search mode is kept at the super-proc-handle parameter, and not at the caller. Thus, it doesn't matter which handle is used to add super-proc-handle as a super-procedure, this will change the proc-search mode for super-proc-handle.
- if super-proc-handle doesn't have the proc-search mode set and it is added at some point as a super-procedure with the proc-search mode explicitly set, no warning will be shown and the proc-search mode will be in an "explicitly set" state.
- if super-proc-handle has its proc-search mode "explicitly set" and the ADD-SUPER-PROCEDURE tries to change this mode, it will get this warning:
"**Changing proc-search for procedure <extproc.p> from '<old-mode>' to '<new-mode>'. (8914)"
where old-mode and new-mode are one of "SEARCH-SELF" or "SEARCH-TARGET".

- setting SESSION:SUPRESS-WARNINGS avoids the above warning message.
- if super-proc-handle has its proc-search mode "explicitly set" and the ADD-SUPER-PROCEDURE call has no explicit proc-search mode, then the proc-search mode for super-proc-handle is set back to default mode.
- if remove-super-procedure(super-proc-handle) is called, it does not change the search mode for super-proc-handle
- when RUN SUPER and SUPER is called, the search mode is checked for THIS-PROCEDURE
- when the search mode for THIS-PROCEDURE is SEARCH-SELF, RUN SUPER and SUPER start searching for the super implementation using the THIS-PROCEDURE:super-procedures stack.
- when the search mode for THIS-PROCEDURE is SEARCH-TARGET, RUN SUPER and SUPER start searching using the TARGET-PROCEDURE:super-procedures stack. This creates an iterator from the top of the TARGET-PROCEDURE:super-procedures stack and each time a RUN SUPER/SUPER call is executed in these conditions the current iterator's position is used to start searching down the stack.

REMOVE-SUPER-PROCEDURE method
- returns false if the target is a Web service procedure or the handle is not valid
- removes the super-proc-handle from the caller's list of super-procedures

SUPER and RUN SUPER
- the passed parameters must have the same order and type
- parameter values or references can be adjusted
- the target implementation is found following these rules:

1. if the search mode for THIS-PROCEDURE is default or SEARCH-SELF, it searches the THIS-PROCEDURE:super-procedures stack
2. if the search mode for THIS-PROCEDURE is SEARCH-TARGET, it goes down the TARGET-PROCEURE:super-procedures stack
3. search distinguishes between internal procedures and functions. Thus, SUPER will always search for functions and RUN SUPER will always search for internal procedures.
4. search ends on the first name match. All "IN SUPER" procedures/functions are excluded from the search ("IN handle" functions are used).
   - each time a RUN SUPER or SUPER invokes a super-procedure, it will go at least one step down (i.e. the super implementation might not be in the next available super-procedure, but in a subsequent one).

The target of a "RUN proc" stmt or DYNAMIC-FUNCTION call ends on first find in:
- THIS-PROCEDURE
- THIS-PROCEDURE:super-procedures
- SESSION:super-procedures
- if an external program with the given name is found (for the "RUN proc" statement without "IN handle" clause).
- search distinguishes between procedures and functions with the same name. On first name match, will attempt to invoke it.
- is not necessarily for the target procedure to have its header defined in the current procedure. Applies only to "RUN proc" statement.
- "RUN proc0." and "proc0()" will end up calling different code (one is a super-proc, the other is a super-function). The body for these two may be in different super-procedures.

Implementation notes:
- RUN SUPER, SUPER, ADD-SUPER-PROCEDURE and REMOVE-SUPER-PROCEDURE will be APIs in ProcedureManager
- the super-procedure chain will be kept in ProcedureManager, but not by the handle, but by the associated external procedure.
- handle:SUPER-PROCEDURES attribute will be retrieved using ProcedureManager.superProcedures(handle), which in turn will get the super-procedures chain for the external procedure referenced by the handle.
- session:super-procedures attribute will be retrieved using ProcedureManager.superProcedures(), which in turn will get the super-procedures chain for the global scope (i.e. the session).

#7 - 11/19/2012 11:36 AM - Greg Shah

Would you please show an example of both SEARCH-SELF and SEARCH-TARGET?

#8 - 11/20/2012 07:23 AM - Constantin Asofiei

To demonstrate how SEARCH-SELF and SEARCH-TARGET behave, let's assume we have this super-procedure hierarchy:

```
      t1
       ^
       |
      t2
       ^
       |
t6 t5 t3
^  ^   ^
 \ |   |
  \ \ /
   -t4
```

which means that:

- t2 has t1 as super-procedure
- t3 has t2 as super-procedure
- t4 has t6,t5 and t3 as super-procedures (added in this order).

Each of the t1,t2,t3,t5 and t6 programs (all except t4) have a single method, "proc0", with this body:

```
procedure proc0.
  message "test#" target-procedure:name.
  run super.
end.
```

where "test#" is the test's number, i.e. "test1" for t1.p, "test2" for t2.p etc.

Beside these programs, there is another runner program (named t0.p), which sets up the hiearchy and runs proc0 in t4:

```
def var h1 as handle.
def var h2 as handle.
def var h3 as handle.
def var h4 as handle.
def var h5 as handle.
def var h6 as handle.

run t1.p persist set h1.
run t2.p persist set h2.
run t3.p persist set h3.
run t4.p persist set h4.
run t5.p persist set h5.
run t6.p persist set h6.

h2:add-super-procedure(h1)./* default mode is SEARCH-SELF */
h3:add-super-procedure(h2).
h4:add-super-procedure(h6).
h4:add-super-procedure(h5).
h4:add-super-procedure(h3).

message "test search-target?" update l as logical.
if l then do:
  /* these are added to enforce the SEARCH-TARGET mode for t3, t5 and t6 programs. */
  h3:add-super-procedure(h3, search-target).
  h3:remove-super-procedure(h3).
  h5:add-super-procedure(h5, search-target).
  h5:remove-super-procedure(h5).
  h6:add-super-procedure(h6, search-target).
  h6:remove-super-procedure(h6).
end.

run proc0 in h4.
```

After executing this program with the "test search-target" set to no, the output will be:

```
test3 t4.p
test2 t4.p
test1 t4.p
Procedure proc0 t1.p has no SUPER procedure with internal procedure proc0. (6439)
```

As the search mode for t3, t2 and t1 is SEARCH-SELF, it will search the THIS-PROCEDURE:super-procedures list for a proc0 match.

After executing this program with the "test search-target" set to yes, the output will be:

```
test3 t4.p
test5 t4.p
test6 t4.p
Procedure t4.p has no SUPER procedure with internal procedure proc0. (6439)
```

As the search-mode for t3, t5 and t6 is SEARCH-TARGET, it will use h4:super-procedures list to search a candidate. Note that if RUN SUPER runs on a program with SEARCH-SELF mode, then it will search the THIS-PROCEDURE:super-procedures for a candidate. The pecualiarity here is when THIS-PROCEDURE is the same as TARGET-PROCEDURE. In this case, even if THIS-PROCEDURE's mode is SEARCH-TARGET, it will act as like the procedure can not be found. For this, the changes are:

1. add proc0 with the same body to t4.p
2. modify t0.p this way:

   ```
   ...
   if l then do:
     /* set t4's mode to SEARCH-TARGET*/
     h4:add-super-procedure(h4, search-target).
     h4:remove-super-procedure(h4).
     ...
   end.
   ...
   ```

   After executing it, the result will be (when setting "test search-target" to yes):

   ```
   test4 t4.p
   Procedure t4.p has no SUPER procedure with internal procedure proc0. (6439)
   ```

   If "test search-target" is set to no, the result is:

   ```
   test4 t4.p
   test3 t4.p
   test2 t4.p
   test1 t4.p
   Procedure proc0 t1.p has no SUPER procedure with internal procedure proc0. (6439)
   ```

#9 - 11/20/2012 08:43 AM - Greg Shah

This is ridiculous. Some thoughts in general:

1. The complexity of this behavior is high.
2. There is a mismatch with the natural inheritance behavior of virtually all OO languages, including Java.
3. The fact that the hierarchy can and often is setup outside of the code that it affects makes it difficult to see and impossible to enforce specific behavior.
4. The hierarchy can change at runtime and those changes can be made without any cooperation by the code it affects. This makes the result very fragile.

I could go on.

Now for some questions:

```
if l then do:
  /* these are added to enforce the SEARCH-TARGET mode for t3, t5 and t6 programs. */
  h3:add-super-procedure(h3, search-target).
  h3:remove-super-procedure(h3).
```

I don't understand the why h3 is the referent for the add-super-procedure() method. In the original setup, h3 is the last super-procedure for h4.

I also don't understand why there is the remove-super-procedure() right after this. Is this just a strange idiom for changing the search mode without removing it from being an h4 super-procedure? If so, what happens if you don't have the remove-super-procedure() call? Being your own super-procedure doesn't make much sense.

```
The pecualiarity here is when THIS-PROCEDURE is the same as TARGET-PROCEDURE. In this case, even if THIS-PROCE
DURE's mode is SEARCH-TARGET, it will act as like the procedure can not be found.
```

I can't say that I really understand this. It seems like bad behavior because t3, t5 and t6 are all still on the super-procedure stack, right?

Could this just be an artifact of this code:

```
  /* set t4's mode to SEARCH-TARGET*/
  h4:add-super-procedure(h4, search-target).
  h4:remove-super-procedure(h4).
```

?

Does it act differently if you set it up as SEARCH-TARGET from the beginning rather than using this trick?

One final note: I setup SEARCH-SELF and SEARCH-TARGET as "global vars" in progress.g. But it is better to move then into the "literals" rule and match them as a constant. They will get a more natural conversion that way.

#10 - 11/20/2012 09:20 AM - Constantin Asofiei

> This is ridiculous.

I agree, it took me a while to figure it out that the search mode is not for the caller, but for the super-proc-handle parameter. With these findings, unless we can hard-code the super procedure hierarchy using hints, there is no way to build a generic algorithm which determines it at conversion time.

> I don't understand the why h3 is the referent for the add-super-procedure() method. In the original setup, h3 is the last super-procedure for h4.
>
> I also don't understand why there is the remove-super-procedure() right after this. Is this just a strange idiom for changing the search mode without removing it from being an h4 super-procedure?

Yes, this code:

```
  h3:add-super-procedure(h3, search-target).
  h3:remove-super-procedure(h3).
```

which uses the same handle as the caller and as the super-proc-handle parameter is just a trick to enforce the search-target mode for that procedure. This is because the SEARCH-TARGET and SEARCH-SELF modes are not set for the caller, but for the super-proc-handle parameter. The remove-super-procedure was added so that the hierarchy is not affected.

> If so, what happens if you don't have the remove-super-procedure() call? Being your own super-procedure doesn't make much sense.

If the remove-super-procedure() call from the code above is not removed, then the handle becomes a super-procedure of itself... and when RUN SUPER is called, it goes into an infinite recursion.

> I can't say that I really understand this. It seems like bad behavior because t3, t5 and t6 are all still on the super-procedure stack, right?

No, it doesn't matter what other super-procedures t4 has. What matters is that, when the RUN SUPER call is encountered, the search-mode for THIS-PROCEDURE (t4) is SEARCH-TARGET and TARGET-PROCEDURE is also t4.

> Could this just be an artifact of this code:
>
> [...]
>
> ?
> Does it act differently if you set it up as SEARCH-TARGET from the beginning rather than using this trick?

No, because if we replace the body of the IF block with:

```
if l then do:
  h6:add-super-procedure(t4, search-target).
end.
```

the result is the same, as the add-super-procedure() call enforces the SEARCH-TARGET mode for t4.

> One final note: I setup SEARCH-SELF and SEARCH-TARGET as "global vars" in progress.g. But it is better to move then into the "literals" rule and match them as a constant. They will get a more natural conversion that way.

Noted.

#11 - 12/07/2012 05:36 AM - Constantin Asofiei

Entry 43 for #1920 was added a test update which covers the conversion rules for both persistent and super-procedures. A relevant extract from those notes is:

- ProcedureManager - API definitions for THIS-PROCEDURE, TARGET-PROCEDURE, SOURCE-PROCEDURE and SESSION's ADD-PERSISTENT-PROCEDURE, REMOVE-PERSISTENT-PROCEDURE, FIRST-PROCEDURE, LAST-PROCEDURE, ADD-SUPER-PROCEDURE, REMOVE-SUPER-PROCEDURE, SUPER-PROCEDURES plus the RUN SUPER and SUPER statements (both RUN SUPER and SUPER statements convert to ProcedureManager.runSuper, I don't think there is a need to differentiate between them - if I find there is a need during runtime testing, I will change the conversion). I've just realized I miss tests for RUN SUPER and SUPER cases when the procedure/function has parameters, I will test this too.

#12 - 12/07/2012 05:39 AM - Constantin Asofiei

- % Done changed from 0 to 30

#13 - 12/11/2012 07:51 AM - Constantin Asofiei

4GL allows procedure (and function) definitions with the same name, under some circumstances.
If two or more procedures have the same name, then:

1. if defined IN SUPER and local, then local procedure definition is invoked (IN SUPER is removed)

```
procedure whatami in super.
end.
procedure whatami. /* this one survives */
end.
```

2. if all are IN SUPER, then only one survives

```
procedure whatami in super.  /* only first one survives */
end.
procedure whatami in super.
end.
```

3. if all are local, then compile error

If two or more functions have the same name, then:

1. if defined in super and local, then in super is "ignored" and the local user-defined function is executed.

```
function whatami returns int in super.
function whatami returns int. /* this one survives */
end.
```

2. if defined in super and in handle, then in super is "ignored" and the IN handle function is executed.

```
function whatami returns int in super.
function whatami returns int in h. /* this one survives */
```

3. if defined in super, in handle and local, then in super and local are "ignored" and the IN handle function is executed..

```
function whatami returns int in super.
function whatami returns int in h. /* this one survives */
function whatami returns int.
end.
```

4. if defined in handle and local, then compile error
5. if all are defined in handle, then only one survives

```
function whatami returns int in h.  /* this one survives */
function whatami returns int in h.
function whatami returns int in h.
```

6. if all are defined in super, then only one survives

```
function whatami returns int in super.  /* this one survives */
function whatami returns int in super.
function whatami returns int in super.
```

7. if all are local, then compile error

As long as the IN SUPER or IN handle clause is present, two procedures and functions can have the same name. The following code compiles OK:

```
procedure whatami in super.
end.
function whatami returns int in super.
function whatami returns int in h.
```

These cases are fixed in post_parse_fixups.xml.

#14 - 12/11/2012 08:28 AM - Greg Shah

Good findings. I find it hard to understand the value of this complexity. Much of the behavior seems arbitrary.

I only have 1 question: what happens when you change the order of the definitions? Is this:

```
procedure whatami in super.
end.
procedure whatami. /* this one survives */
end.
```

different that this:

```
procedure whatami.
end.
procedure whatami in super. /* does this one survive? */
end.
```

#15 - 12/11/2012 09:02 AM - Constantin Asofiei

Greg, the order of the procedure/function is not important. No matter how you mix the procedures/functions in the above examples, the same rules will apply.

#16 - 12/11/2012 09:21 AM - Greg Shah

What is your approach to handling this behavior? Do you just hide or drop the sub-trees?

One thing to consider: any dropped local procedure (or function) may have some effect in the 4GL. For example, it may modify a frame definition even though it can never be called. Please check on this and let's at least know if there is any issue.

#17 - 12/11/2012 09:29 AM - Constantin Asofiei

My first approach is to just drop the sub-trees.
Thanks for the frame definition tip, I'll check also how these cases behave when the function/procedure has a different signature and let you know.

#18 - 12/11/2012 10:12 AM - Constantin Asofiei

If there are frame definition inside a "to-be-hidden" local function or procedure, then the frame definition "leaks" outside of the function/procedure's body and is seen by the 4GL code. So the solution is to hide the nodes (to let the frame definition statements be collected by the downstream conversion rules). But unfortunately this is not the only change I need to do. In a code like this:

```
function func0 returns int in super.
function func0 returns int in h. /* only this one survives */
function func0 returns int.
   display ch i with frame f1.
   return ?.
end.
func0().
```

the parser links the func0() call (via tempidx which later gets converted to refid annotation, in post_parse_fixups.xml) with the local function definition, so it will be incorrectly converted to a java method call, "func0()", instead of being converted to a ControlFlowOps.invokeFunction call (as the 4GL "func0()" call is associated with the IN handle version of the function).

#19 - 12/11/2012 10:48 AM - Constantin Asofiei

After looking at progress.g, SymbolResolver.addFunction and Function.annotateOptions, the simplest way I see to solve this is to modify post_parse_fixups.xml so that it re-computes the refid for each func_ref AST, in a rule-set which:
1. in walk-rules, collects all non-hidden function definitions and all non-builtin function references
2. in post-rules, walk the non-builtin function references, and set its refid from the set of non-hidden function definitions

The alternative would be to move/duplicate the rules which disambiguate the function calls from post_parse_fixups.xml to the parser level. I will add comments to the code where tempidx is set by the parser and uast.* code so future readers will know that tempidx annotation for functions is obsolete and refid computation is delayed in post_parse_fixups.xml (or we can remove this code entirely).

#20 - 12/11/2012 11:04 AM - Greg Shah

I'm OK with your approach.

#21 - 12/20/2012 10:05 AM - Constantin Asofiei

I found a case which I haven't thought about until now. Consider we have these procedures:

- t1.p:

```
def var h as handle.
function func0 returns int in h.
```

- t2.p:

```
function func0 returns int in super.
```

- and this tmain.p:

```
def var h1 as handle.
def var h2 as handle.
run t1.p persist set h1.
run t2.p persist set h2.
h2:add-super-procedure(h1).
dynamic-function("func0" in h2).
```

When executed, tmain.p will execute "func0" from t1.p, which is mapped to handle "h". Some additional info needs to be emitted in the t1.p's converted Java class and in its associated name_map.xml entry:
1. in t1.p's converted Java class, will need to have a method named "getHandleExpr#" which will resolve its associated HandleExpr#
2. in the t1.p's entry in name_map.xml, for "func0" we need to save "handleExpr#", which is the name of the expression which resolves the handle for this function.
3. when it is determined that the super version of a function needs to be invoked, if this function is marked as "IN handle", then the handle is obtained via a getter call for the handleExpr# set in name_map.xml.

#22 - 12/20/2012 10:58 AM - Greg Shah

```
1. in t1.p's converted Java class, will need to have a method named "getHandleExpr#" which will resolve its as
sociated HandleExpr#
```

As an alternative, why not just emit as registration call that passes the handle reference to the runtime and registers it as the handle to use with func0? This avoids extra reflection and extra methods and doesn't really "cost" much. We just would need to track a separate list of registered handles for each **instance** of that class.

#23 - 12/20/2012 11:16 AM - Constantin Asofiei

Ok, that makes sense.

#24 - 01/28/2013 02:15 AM - Constantin Asofiei

- % Done changed from 30 to 80

#25 - 04/12/2013 05:45 AM - Constantin Asofiei

A resume on what else needs to be done regarding super-procedure chains:

- make the SEARCH-TARGET and SEARCH-SELF emit as integer constants (was not a good idea to emit them as string literals, as this is not the way these are defined in 4GL) - 2 hours
- ProcedureManager.java - the TODO is related to SEARCH-TARGET mode, if the super-procedures list gets changed while in super calls for this mode, no more super calls are possible. - this is hard to estimate, as I couldn't figure out how 4GL keeps the procedure chain, so that it is invalidated when changed... I want to reserve at least 25 hours for this.
- once #1971 is completed, do a thoroughly check of how the super procedures get invoke, in both SEARCH-TARGET and SEARCH-SELF modes - 10 hours

#26 - 04/26/2013 02:25 PM - Greg Shah

- % Done changed from 80 to 100

- Status changed from WIP to Closed

#27 - 11/16/2016 11:43 AM - Greg Shah

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