Base Language - Feature #1792

implement widget-pool support

10/30/2012 09:44 AM - Greg Shah

Status:	Closed	**Start date:**	01/30/2013
Priority:	Normal	**Due date:**	07/04/2013
Assignee:		**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Runtime Support for Server Features		
billable:	No	**version:**	
vendor_id:	GCD		

Description

Subtasks:

Feature # 1982: conversion support for widget pools	**Closed**
Feature # 1983: runtime support for widget pools	**Closed**

History

#1 - 10/30/2012 09:45 AM - Greg Shah

CREATE WIDGET-POOL, DELETE WIDGET-POOL and any scoping of resources

#2 - 10/31/2012 02:56 PM - Greg Shah

- Target version set to Milestone 12

#3 - 01/30/2013 04:19 PM - Greg Shah

- Target version changed from Milestone 12 to Milestone 7

This is heavily used, even on the server since it seems to affect non-UI resources too.

#4 - 01/30/2013 04:20 PM - Greg Shah

- Project changed from User Interface to Base Language

#5 - 01/31/2013 08:41 AM - Constantin Asofiei

From the documentation and some testing, it seems that scoping is done on the nearest executing block; we should be able to implement the scoping using a Scopeable instance registered with the TransactionManager for scope notifications, without any other scope-related code required in the generated sources.

There is already a WidgetPool class added in the p2j.ui package - this will be moved to p2j.util. More, anyone who implements CREATE statements which accept a IN WIDGET-POOL clause, should provide support for this clause.

#6 - 01/31/2013 08:49 AM - Constantin Asofiei

- File ca_upd20130131b.zip added

Added update for review (on top of bzr revision 10158).

#7 - 01/31/2013 09:56 AM - Greg Shah

It seems like there are 4 kinds of pools:

- un-named session pool (created automatically by the 4GL and used as the default pool)
- un-named scoped pool that attaches to the nearest enclosing procedure or method
- named global pool (whose lifetime is scoped to the creating procedure)
- named global persistent pool (must be deleted explicitly, it outlives its creating procedure)

I have these general questions about widget-pools:

1. Do we need to implement a global unnamed widget-pool like Progress does? This would ensure that "proper" deletes occur on every resource that is not created in a scoped or named pool.

2. Which dynamic/handle resources are created in widget-pools?

3. What changes do we need to make to those resources to properly "register" with their pool? I assume we need a scoped dictionary to find the correct pool. Plus we will have to support using a named pool.

4. What happens to a resource when it is in use and gets deleted. For example, a dynamic button can have a CHOOSE trigger that deletes the button (or even the entire pool it resides in). What happens to the UI in that case?

5. Is the only real behavior here all about doing an "implicit" DELETE OBJECT based on leaving a particular scope? I see no other functional behavior based on the documentation.

6. Do unnamed widget pools scope to all top-level blocks (e.g. triggers, functions)?

7. Is it correct that a locally scoped unnamed pool superseded within the same procedure by calling another internal procedure that creates its own unnamed pool?

8. Is it correct that an unnamed scoped pool that is associated with a persistent procedure will be used for internal procedures called on that proc handle, even if other widget pools are on the call stack (e.g. the caller is outside of the persistent proc and has its own widget pool)?

9. Does a named pool really get deleted when the procedure that created it exits? If so, what is the value of using this?

I'll look at your code next.

#8 - 01/31/2013 10:16 AM - Greg Shah

Code feedback:

1. The WidgetPool.create/delete methods need variants for character expressions.
2. Perhaps I missed it, but I don't see the support for converting PERSISTENT into a boolean true.

Otherwise it looks good.

One other thought: in regard to issues 6 and 8 above, they imply that each new top-level scope needs to have a "registration target" which is the widget-pool to which newly created resources get added.

#9 - 01/31/2013 11:46 AM - Constantin Asofiei

- File ca_upd20130131e.zip added

1. The WidgetPool.create/delete methods need variants for character expressions.

OK, I've added them.

2. Perhaps I missed it, but I don't see the support for converting PERSISTENT into a boolean true.

It was already added in convert/ui_statements.rules, I've moved it to convert/language_statements.rules now.

I'll take a deeper look at the widget-pool scoping tomorrow, when I have a clearer mind. At this time, my goal would be to prove that we can provide the scoping support at runtime, with no need of addition conversion changes <g>

#10 - 02/01/2013 01:27 PM - Constantin Asofiei

1. Do we need to implement a global unnamed widget-pool like Progress does? This would ensure that "proper" deletes occur on every resource that is not created in a scoped or named pool.

At this time, I can't find any scoping-related differences between unnamed and named widget-pools. Both seem to behave the same. See question 7 for more details.

2. Which dynamic/handle resources are created in widget-pools?

The following CREATE statements accept an IN WIDGET-POOL clause. Thus, I think these are the only statements which can create resources and add them to widget-pools. Testing other statements (like CREATE SERVER and CREATE SOCKET) shows that the resource is not added to the pool.

```
CREATE BROWSE
CREATE BUFFER
CREATE CALL
CREATE DATASET
CREATE DATA-SOURCE
CREATE QUERY
CREATE SAX-ATTRIBUTES
CREATE SAX-READER
CREATE SAX-WRITER
CREATE SOAP-HEADER
CREATE SOAP-HEADER-ENTRYREF
```

```
CREATE TEMP-TABLE handle
CREATE { BUTTON     | COMBO-BOX | CONTROL-FRAME | DIALOG-BOX     |
         EDITOR     | FILL-IN   | FRAME         | IMAGE          |
         MENU-ITEM  | RADIO-SET | RECTANGLE     | SELECTION-LIST |
         MENU       | SLIDER    | SUB-MENU      | TEXT           |
         TOGGLE-BOX | WINDOW    | VALUE ( string-expression )
       }
CREATE X-DOCUMENT
CREATE X-NODEREF
```

3. What changes do we need to make to those resources to properly "register" with their pool? I assume we need a scoped dictionary to find the correct pool. Plus we will have to support using a named pool.

If an unnamed pool is in effect, the CREATE statements which accept the IN WIDGET-POOL clause (and don't have the clause) will register the resource with the unnamed pool. Thus, conversion rules for the CREATE statements with an IN WIDGET-POOL clause will have to emit the widget-pool's name, if available. Also, the runtime will need to register the resource with the appropriate pool (named or unnamed).

4. What happens to a resource when it is in use and gets deleted. For example, a dynamic button can have a CHOOSE trigger that deletes the button (or even the entire pool it resides in). What happens to the UI in that case?

The resource can be deleted using DELETE OBJECT (with no reported errors) and in case of a UI resource, the UI gets updated by removing the widget from the screen (immediately in case of triggers). Same for deleting a widget-pool - all contained resources get deleted (same for UI as with delete object).

5. Is the only real behavior here all about doing an "implicit" DELETE OBJECT based on leaving a particular scope? I see no other functional behavior based on the documentation.

Yes, this is what I found too - DELETE OBJECT is used to delete the resources when the widget-pool gets deleted.

6. Do unnamed widget pools scope to all top-level blocks (e.g. triggers, functions)?

See answer for question 9.

7. Is it correct that a locally scoped unnamed pool superseded within the same procedure by calling another internal procedure that creates its own unnamed pool?

If the external procedure, internal procedure proc0 and internal procedure proc1 create their own unnamed pools, like:

```
def var h as handle.
procedure proc0.
   create widget-pool.
   create button h.
   message valid-hande(h). /* shows yes */
end.

procedure proc1.
   create widget-pool.
   run proc0.
   message valid-hande(h). /* shows no */
end.

run proc1.
message valid-handle(h). /* shows no */
```

each top-level block will have their own unnamed pool. But, if one of them creates an unnamed persistent pool, like in:

```
def var h as handle.
def var h2 as handle.
def var h3 as handle.

procedure proc0.
   create widget-pool persistent.
   create button h.
   message valid-hande(h). /* shows yes */
end.

procedure proc1.
   create widget-pool.
   create button h2.
   run proc0.
   create button h3.
   message valid-hande(h) valid-handle(h2) valid-handle(h3). /* shows yes yes yes */
end.

run proc1.
message valid-hande(h) valid-handle(h2) valid-handle(h3). /* shows yes no yes */
```

the "persistent" mode does not propage entirely down the stack. Instead, it seems like when a CREATE WIDGET-POOL PERSISTENT call is encountered, the "global unnamed persistent" pool is moved on the top of the stack, as subsequent created resources are added to the "global" pool, and not to the local pool.

> 8. Is it correct that an unnamed scoped pool that is associated with a persistent procedure will be used for internal procedures called on that proc handle, even if other widget pools are on the call stack (e.g. the caller is outside of the persistent proc and has its own widget pool)?

When RUN ... PERSISTENT is used, the named/unnamed pools are not deleted (I guess they will be alive for the duration of the persistent procedure). For your specific case, I think you are correct, as when the persistent procedure is deleted, the resource is gone too (thus it must have been created in the persistent proc's unnamed pool).

> 9. Does a named pool really get deleted when the procedure that created it exits? If so, what is the value of using this?

Yes, it does get deleted, unless is the pool or the procedure is PERSISTENT.

If we have these programs:

```
/*wp1.p*/
def new shared var h as handle.
def new shared var fh as handle.
def var i as int.

form i with frame f1.
fh = frame f1:handle.
run wp2.p.
message valid-handle(h).


/*wp2.p*/
def shared var h as handle.
def shared var fh as handle.

message "1-named pool, 2-unnamed pool, 3-no pool, 4-named persistent, 5-unnamed persistent " update i as int.

if i = 1 then do:
  create widget-pool "pp".
  create button h in widget-pool "pp" assign label="bla1" visible=true row=3 col=3 frame=fh.
end.
if i = 2 then do:
  create widget-pool.
  create button h assign label="bla1" visible=true row=3 col=3 frame=fh.
end.
if i = 3 then do:
  create button h assign label="bla1" visible=true row=3 col=3 frame=fh.
```

```
end.
if i = 4 then do:
  create widget-pool "pp" persistent.
  create button h in widget-pool "pp" assign label="bla1" visible=true row=3 col=3 frame=fh.
end.
if i = 5 then do:
  create widget-pool persistent.
  create button h  assign label="bla1" visible=true row=3 col=3 frame=fh.
end.

message valid-handle(h).
```

after running wp1.p, the handle will be valid only in cases when there is no widget-pool or the widget-pool is persistent.
If we modify wp2.p to scope the IF blocks to a function, trigger or internal procedure, then the valid-handle(h) call in wp2.p will return the same value as valid-handle(h) in wp1.p. Thus, pools are scoped on any kind of top-level block. Some other notes:

- when working with widget-pools, it is important to run the program from the command line, else the editor will cache the persistentwidget-pools and will lead you on a wrong path.
- when creating the unnamed pool, multiple CREATE WIDGET-POOL calls in a row will have no effect. The following code:

  ```
  def var h as handle.
  procedure proc0.
     create widget-pool persistent.
     create button h.
  end.

  run proc0.
  message valid-handle(h). /* shows yes */
  create widget-pool persistent.
  message valid-handle(h). /* shows yes */
  ```

  will not alter the pooled resources, even if another CREATE WIDGET-POOL statement is executed.
- When attempting to create a named pool, if already exists a pool with the given name, then ERROR condition is raised and this message is shown:

  ```
  Attempt to create widget-pool <name>, but it already exists. (4116)
  ```

- when deleting the unnamed widget pool, no error appears if there is no active unnamed widget pool.
- when deleting the named widget pool, the following error is shown if the widget-pool does not exist:

  ```
  Attempt to delete widget-pool <name>, but it doesn't exist. (4117)
  #
  ```

#11 - 02/07/2013 01:51 AM - Constantin Asofiei

Regression testing for the conversion support update has passed, applied to bzr revision 10162.

#12 - 04/12/2013 07:04 AM - Constantin Asofiei

Should the runtime task include working on the resources which can be added in a pool or add just the pooling support and let the registration of the resources with the pool be added when the resource is implemented?

My approach would be to add pooling support only for the resources which are currently fully-implemented (like widgets) and for the others, add the registration when the resource is implemented. Also, first I would provide basic stubs for resource pooling (for named and unnamed), to let the people implementing the resource be able to add the registration code, even if pooling is not ready yet.

The estimated time seems accurate, assuming not all poolable resources need to be touched/tested.

#13 - 09/19/2013 12:53 PM - Constantin Asofiei

- File widget_pool_tests_20130919a.zip added

- File ca_upd20130919a.zip added

The attached update adds runtime support for widget pools. Still need to perform regression testing on it.

#14 - 09/19/2013 02:55 PM - Greg Shah

Code Review 0919a

Overall, it looks very good.

1. Why throw IllegalStateException in WidgetPool.addResource()?

2. Why does WidgetPool.newPool() have an unnamed boolean flag if name == null can be used to detect if this should be unnamed?

3. WidgetPool.getPool(String name, boolean currentScope) may be better named as findActivePool() to make it clear that there is a scope-based search going on.

4. Do we need code to do a final cleanup/removal of unnamed persistent pools when the user's session exits?

5. Where do we create the initial unnamed widget pool (that is implicitly persistent)?

6. Please enhance the class javadoc to address the following:

   1. What exactly is the purpose of a widget-pool? If I understand correctly, it is a mechanism to explicitly define the scope at which resources get an "implicit" DELETE OBJECT when the control flow causes the program to exit that particular scope. At its most basic, a widget pool is just a list of resources that get deleted, an optional name, an optional persistent flag and an association with a block scope at exists at a "top-level" block (external proc, internal proc, user-defined function and trigger).
   2. By its very nature, this only affects resources that are dynamically created (and thus must be removed using DELETE).
   3. Which resources have widget-pool support in the 4GL?
   4. It should be made clear (also in the scopeStart() and scopeFinished() javadoc) that the scopes only open and close at top-level blocks. Inner blocks don't cause any scoping behavior.
   5. All widget-pool behavior is context-local. In other words, it is specific to the current user's session.
   6. Resources can only ever exist in one pool.
   7. New widget-pools are created in the 4GL in which cases? If I understand correctly, 1 unnamed is created by default when the session starts and then every time CREATE WIDGET-POOL is called a new one is added.
   8. How many unnamed persistent pools can be created? (1 by default and then 1 more explicitly per external proc?)
   9. How do unnamed persistent pools ever get deleted? It seems like they will always get leaked upward in the scopes and can't ever be deleted.

#15 - 09/20/2013 04:32 AM - Constantin Asofiei

- File ca_upd20130920a.zip added

1. Why throw IllegalStateException in WidgetPool.addResource()?

The exception is thrown because this method must always find the pool, when a named pool is specified. The exception will be thrown only if someone forgets to validate the pool before adding the resource to the pool, and the named pool doesn't exist. This is needed because the pool must be validated first, before resource-related validation (like buffer name for CREATE BUFFER statement)

2. Why does WidgetPool.newPool() have an unnamed boolean flag if name == null can be used to detect if this should be unnamed?

You are right, is redundant, I've removed it now.

3. WidgetPool.getPool(String name, boolean currentScope) may be better named as findActivePool() to make it clear that there is a scope-based search going on.

findActivePool is better, I've renamed it.

4. Do we need code to do a final cleanup/removal of unnamed persistent pools when the user's session exits?

From the list of resources which can be pooled, the only one which links it to an external dependency is the temp-table. But this one is already context-local, so will be discarded when the session ends. So as the resources already get discarded (and they should clean too) when the user's session exits, the pool doesn't need to clean up after them.

5. Where do we create the initial unnamed widget pool (that is implicitly persistent)?

There is no reason for us to add overhead for the default persistent unnamed pool associated with the session. As this is always on the bottom of the stack of unnamed pools, if reached, one will not be able to delete it. Thus, P2J gains nothing from adding resources to a default unnamed pool, as 4GL doesn't offer any mechanism to interrogate the pool's resources or the pool stack.

6. Please enhance the class javadoc to address the following:

Done.

Note that 0919a.zip has passed regression testing.

#16 - 09/20/2013 08:53 AM - Greg Shah

The changes are really good. The class javadoc improvements are excellent!

> The exception is thrown because this method must always find the pool, when a named pool is specified. The exception will be thrown only if someone forgets to validate the pool before adding the resource to the pool, and the named pool doesn't exist. This is needed because the pool must be validated first, before resource-related validation (like buffer name for CREATE BUFFER statement)

If I understand correctly, you are saying that it would be a P2J error for this to occur. If so, please document this in comments in this method.

> 4. Do we need code to do a final cleanup/removal of unnamed persistent pools when the user's session exits?

> From the list of resources which can be pooled, the only one which links it to an external dependency is the temp-table. But this one is already context-local, so will be discarded when the session ends. So as the resources already get discarded (and they should clean too) when the user's session exits, the pool doesn't need to clean up after them.

My concern here is that the fact that the current implementation of these resources is safe cannot be easily determined by reading the code. Please some javadoc to explain that:

> No session-level widget-pool cleanup is implemented at this time. The current implementation of all the poolable resources does not require any session-level cleanup processing that is driven by widget pools. Should the implementation of any one of the poolable resources be modified to be dependent upon session-level widget-pool cleanup, then this feature will have to be added.

After adding the comments, you can check in and distribute the update without further testing. The changes should be safe.

#17 - 09/20/2013 09:16 AM - Constantin Asofiei

- File ca_upd20130920c.zip added

ca_upd20130920c.zip was committed to bzr revision 10385.

Final Notes:
1. A bug was fixed for CREATE BUFFER ... IN WIDGET-POOL cases when the BUFFER-NAME clause is missing. For such cases, a java null is emitted for the buffer name. When implementing this statement, you need to assume a null buffer name means that the BUFFER-NAME clause is missing.
2. Don't assume that an empty pool name means the unnamed pool is used.
3. For all the CREATE statements which accept a IN WIDGET-POOL clause, a null widget pool name will mean that the unnamed pool is used (i.e. the IN WIDGET-POOL clause is missing). Do not convert a java null to an unknown 4GL-style character value, as this will mean the named pool is used.
4. Before creating the resource, you need to validate the widget pool by calling WidgetPool.validWidgetPool(name). If this returns false, DON'T create the resource.
5. After the resource is created, use WidgetPool.addResource(WrappedResource, name) to add the resource to the pool.
5. The Java code related to resource creation was changed for all the poolable resources so that it follows the above rules. If you need to merge, make sure that the above rules are maintained.

#18 - 09/20/2013 09:18 AM - Greg Shah

- Status changed from New to Closed

#19 - 11/16/2016 11:43 AM - Greg Shah

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

Files

ca_upd20130131b.zip	21.7 KB	01/31/2013	Constantin Asofiei
ca_upd20130131e.zip	21.8 KB	01/31/2013	Constantin Asofiei
ca_upd20130919a.zip	169 KB	09/19/2013	Constantin Asofiei
widget_pool_tests_20130919a.zip	13.5 KB	09/19/2013	Constantin Asofiei
ca_upd20130920a.zip	171 KB	09/20/2013	Constantin Asofiei
ca_upd20130920c.zip	171 KB	09/20/2013	Constantin Asofiei