User Interface - Feature #1818

implement a timer service to replace the PSTimer OCX

10/30/2012 12:32 PM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	24.00 hours
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Base Language - Feature #3262: implement COM/OLE Automation support	**Closed**
Related to User Interface - Bug #5254: FWD extension control must follow the ...	**New**

History

#1 - 10/31/2012 02:47 PM - Greg Shah

- Target version set to Milestone 12

#2 - 03/23/2016 12:54 PM - Greg Shah

- Target version deleted (Milestone 12)

#3 - 04/27/2017 03:01 PM - Greg Shah

See #3257-4.

#4 - 05/19/2017 03:41 PM - Ovidiu Maxiniuc

- Assignee set to Ovidiu Maxiniuc

I created task branch 1818a and committed my stable intermediary work. The implementation for PSTimer can be found in com.goldencode.p2j.activex.PSTimer. I understand that this OCX is somehow just provided by OE so it would be accessible to any ABL installation.

While the runtime implementation for this replacement is in test phase, I have some issues accessing it in converted code. At this moment there is little support if any for com-handle s. I don't know if this is the right place to discuss but the big issue is not the PSTimer per-se, but how to code access to it, which is in fact the implementation of com-handle.

We start by using LoadControls. For the moment, the conversion recognise the special handle type and block my attempt to quick patch it to unwrapControlFrame and call existing LoadControls on it. The ABL code looks like:

```
UIB_S = chCtrlFrame:LoadControls(OCXFile, "CtrlFrame":U).
```

I think the generated code should be something like:

```
uibS.assign(chCtrlFrame.unwrapControlFrame().loadControls(ocxfile, "CtrlFrame"));
```

Another problem is the chaining that can be quite complicated since the access to activex is generally resolved at runtime, much like :: operator, but more generic. While leaf nodes like access to scalar properties seems to be simple:

```
ABL: pstimer:ENABLED = YES.

FWD: pstimer.unwrapComHandle().setProperty("ENABLED", new logical(true));
```

it become more complicated with method/chaining:

```
ABL: chCtrlFrame:PSTimer:INTERVAL = 4.

FWD: chCtrlFrame.unwrapComHandle().getChainedProperty("PSTimer").unwrapComHandle().setProperty("INTERVAL", new
 integer(4));
```

A final issue. In loadControls we need to identify the OCX-es for which we have a java replacement and wire them. Otherwise the native solution should be used (if possible to access the legacy activex code) before returning with error.

#5 - 05/25/2017 02:49 PM - Ovidiu Maxiniuc

- Status changed from New to WIP

The conversion works fine at this moment. At least to support PSTimer. I added an almost complete runtime support: COM-properties can be set and method invoked. The problem is the callbacks. The PSTimer uses a background thread for timing events which must be delivered to main execution thread.

At this moment I realized that the ActiveX should actually run on client side (where it may draw in its ControlFrame) or spawn dialogs (like PSTimer:AboutBox). This means that the JeAx must, in fact, be executed on client completely, the comhandle and ControlFrameWidget will have to deliver the set/get properties and method execution messages to client and wait for their execution. It's still not cleat to me how the callback/events will be supported.

Current revision is 11155.

#6 - 05/25/2017 04:21 PM - Greg Shah

I was not intending to actually provide any ActiveX support here. The idea was to implement a pure Java equivalent for a timer service with a callback when the timer pops. The callback would be on the server, calling converted business logic.

There is no visual usage of PSTimer in this application, so it doesn't have to be on the client.

I am OK using the COM automation support to map to the runtime Java classes. I don't think we want to use the LoadControls or any of that. The idea was to make it much simpler:

1. Define a new timer.
2. Set its configuration (e.g. at least the interval).
3. Give it a callback (internal procedure) and start it running.

On a separate timer thread (in Java) we generate a server event when the timer pops. This leads to calling the internal procedure that is the callback, but it does so on the conversation thread. The setting of the callback and invocation of it, should be using the same kind approach as is done with PUB/SUB.

#7 - 05/26/2017 09:35 AM - Constantin Asofiei

Ovidiu, the rules to determine the program where the PSTimer.tick procedure exists is:

1. use the <ctrlframe:name>.PSTimer.tick as the internal procedure name, where ctrlframe:name is determined at the time when the procedure needs to be invoked (the current value for the name attribute)
2. if ctrlframe:instantiating-procedure is no longer alive (i.e. was not ran persistent or was deleted), then do nothing
3. search ctrlframe:instantiating-procedure for the proc
4. search super-procedures of ctrlframe:instantiating-procedure

So, basically, to invoke the procedure you need to follow these steps, and FWD knows already how to do the search:

1. first check if ctrlframe:instantiating-procedure is alive (handle._isValid() call on the ctrlframe:instantiating-procedure).
2. compute the procname as <ctrlframe:name>.PSTimer.tick. Note that 4GL will not allow to set a ? value for name attr.
3. simulate a RUN <procname> in ctrlframe:instantiating-procedure via ControlFlowOps.invokeIn(<procname>, ctrlframe:instantiating-procedure). The call is without parameters, the matching is done only via the name, and FWD already knows to check the parameters properly.

#8 - 05/26/2017 10:32 AM - Greg Shah

I really don't want to maintain a control-frame widget or any of the other OCX/ActiveX nonsense.

Our objective here is to provide a **clean** extension to the 4GL syntax to setup a new timer. Something like this:

```
DEFINE VARIABLE h-timer AS HANDLE.

CREATE TIMER h-timer
       ASSIGN INTERVAL = 4
              CALLBACK = "some-internal-procedure-name".

...

h-timer.start().

...

h-timer.stop().

PROCEDURE some-internal-procedure-name:
   /* called every timer pop */
END.
```

#9 - 05/26/2017 10:37 AM - Greg Shah

If we have to hand-edit the 4GL code to replace the OCX stuff it is OK. Typically, this is only used in a small-ish number of places in an application.

If we can easily write code to do that conversion for us, even better.

#10 - 05/26/2017 10:57 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> If we have to hand-edit the 4GL code to replace the OCX stuff it is OK. Typically, this is only used in a small-ish number of places in an application.
>
> If we can easily write code to do that conversion for us, even better.

I understand. I got it wrong :(.

I started by implementing the timer service but I could not find a solution to integrate it cleanly, without altering the P4GL code. I studied the OE Programming Interfaces and tried to find a solution. There is small protocol for loading OCX in ABL, usually automatically generated by AppBuilder. My intention was to create a framework for loading and running any Java-emulated COMs, without altering at all the legacy code. This would allow emulation of any OCX rewritten in Java in the future.

I will adjust my changes so the Timer will be an extension to 4GL and drop all support for OCX-s. Then, I will also prepare a set of patches for updating the customer's application code for this new extension.

#11 - 05/26/2017 11:57 AM - Greg Shah

In #3262 we will be implementing full COM automation support. The OCX part is not being done and we don't plan to do it because it would only work on a Windows system and the integration would be nasty and it is not clear how easy that might be to do for the web client.

But COM automation is going to be supported and your code will probably be a big part of the solution. So don't throw it away.

If you can keep it while changing the PSTimer approach, I'd appreciate it.

Finally, in regard to the customer's code, make any conversion support generic. The PSTimer is a Progress OCX and it is used in many applications. So any automation to modify code should be reusable.

#12 - 05/26/2017 12:36 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> In #3262 we will be implementing full COM automation support. The OCX part is not being done and we don't plan to do it because it would only work on a Windows system and the integration would be nasty and it is not clear how easy that might be to do for the web client.

I read #3262 while doing the research. The current revision from 1818a does the conversion as described there. However, it expects that the heavy work to be performed by some java class on server side. However, this is not an issue, changing the runtime worker does not need the already converted code to changed.

> But COM automation is going to be supported and your code will probably be a big part of the solution. So don't throw it away.

I will not.

> If you can keep it while changing the PSTimer approach, I'd appreciate it.

There are two kinds of OCX supported by 4GL:

- the COM Automations. These do not have an UI so they may be fully implemented on server-side. The syntax to create such objects is the one described in #3262. Since the timer also does not have an UI (except for the useless AboutBox method) I tried at first to instantiate it this way on test server. For unknown causes, it did not work, so I suspended work for creating/releasing the COM this way.
- an ActiveX with UI loaded in ControlFrame. All usages of PSTimer (including the client and pdf docs) are using this approach, so I was biased to it. Except for create/release part, the usage of these COMs are the same as for COM Automations.

The implementation for first kind of COM objects is in repository but some parts were overwritten. I am switching now from regarding the PSTimer as an ActiveX to a COM automation. This will require less changes in ABL code: just drop the ControlFrame code and replace it with CREATE "PSTimer" timer-comhandle.

> Finally, in regard to the customer's code, make any conversion support generic. The PSTimer is a Progress OCX and it is used in many applications. So any automation to modify code should be reusable.

I will try to.

#13 - 05/26/2017 01:37 PM - Constantin Asofiei

Greg Shah wrote:

> Our objective here is to provide a **clean** extension to the 4GL syntax to setup a new timer. Something like this:

Greg, I think we can re-use the rules I noted in note 7 to find the target internal procedure. Also, beside the timer:callback attribute, we can use a timer:callback-handle attribute too, which can reference the external program where the callback resides. If is unknown (not set), we can default to the timer:instantiating-procedure.

In regards to this new TIMER resource, it will act like like any other legacy resource, on the low-level (and conversion side). Its implementation will extend HandleChain, with its own interface to define its specific attributes and methods; also, the interface will be registered in CommonHandle, etc.

We might want to use another marker annotation instead of LegacyResource - maybe FWDResource, which extends LegacyResource?

#14 - 05/26/2017 01:50 PM - Greg Shah

> We might want to use another marker annotation instead of LegacyResource - maybe FWDResource, which extends LegacyResource?

Sounds good.

#15 - 05/29/2017 03:17 PM - Ovidiu Maxiniuc

Constantin,

I am working on the runtime of this issue: the asynchronous callback to event procedure.

I tried to postServerEvent() but this doesn't work: if the client is blocked in a wait-for loop, it will handle only UI events while the client socket reader cannot push the post message to be dispatched on the main thread. So, even if ThinClient.waitForEvent() checks for server events, they cannot be queued to EventManager's WorkArea.serverEvents until the current method is over.

If the 4GL code is rewritten so as the server posted events have the chance to be processed by client and resent back, FWD enter a peculiar state where the messages are no more processed synchronously and breaking the communication protocol. As result, exceptions like this are generated: com.goldencode.p2j.net.ProtocolViolation: Expected ID 81 but rec'd ID 80.

The good part is sometime the 'ping-pong' process works and apparently the correct internal procedure is called as callback. I did not try any fancy configurations with multiple external procedures or persistent invocations, though. But I encountered some unexpected cases like this:

```
java.lang.NullPointerException
        at com.goldencode.p2j.ui.chui.ThinClient.processMnemonic(ThinClient.java:16367)
        at com.goldencode.p2j.ui.chui.ThinClient.waitForEvent(ThinClient.java:13985)
        [...]
```

Any ideas here? Am I doing something wrong?

#16 - 05/29/2017 03:30 PM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> Constantin,
>
> I am working on the runtime of this issue: the asynchronous callback to event procedure.
>
> I tried to postServerEvent() but this doesn't work:

From where are you calling this? It must be from the server-side (as the ServerEvent needs to have the Runnable task set).

> if the client is blocked in a wait-for loop, it will handle only UI events while the client socket reader cannot push the post message to be dispatched on the main thread. So, even if ThinClient.waitForEvent() checks for server events, they cannot be queued to EventManager's WorkArea.serverEvents until the current method is over.
> If the 4GL code is rewritten so as the server posted events have the chance to be processed by client and resent back, FWD enter a peculiar state where the messages are no more processed synchronously and breaking the communication protocol. As result, exceptions like this are generated: com.goldencode.p2j.net.ProtocolViolation: Expected ID 81 but rec'd ID 80.

I don't understand this one. For the socket case you mention, the READ-RESPONSE (via server.readResponseEvent) is executed as an async call, from the client-side - see SessionManager.registerAsyncThread and deregisterAsyncThread). If you are calling server-side methods from a non-main client-side thread, then yes, you will get errors like that.

> The good part is sometime the 'ping-pong' process works and apparently the correct internal procedure is called as callback. I did not try any fancy configurations with multiple external procedures or persistent invocations, though. But I encountered some unexpected cases like this:
> [...]

Is currentEventList in the currentEventList.lookup(frameId, widgetId, true, true, match); null? This is not OK, as the state gets corrupted.

> Any ideas here? Am I doing something wrong?

See above. Please specify where is the timer being executed - client or server-side - and if on client-side, you need to send an async request to the server-side to create the server event, which in turn will register it back on the client-side.

#17 - 05/29/2017 03:43 PM - Ovidiu Maxiniuc

Constantin,

The events are fired from server-side, from an AssociatedThread. It was not registerAsyncThread() but am I trying this right now. Client code is not altered.

I am also investigating the NPE from processMnemonic.

#18 - 06/02/2017 03:39 PM - Ovidiu Maxiniuc

Greg,

Apparently the .wrx files hold some information that is not available in the source code (.p or .w files). The initial properties of PSTimers set at design time in AppBuilder are saved there. In the customer code, the COM objects (ActiveXes) are changed rarely if ever, they are supposed to be just loaded with the correct values from .wrx.

The problem is that these are binary files and I can only guess (using a HEX viewer and some reverse engineering) the configuration values for ENABLED and INTERVAL properties of PSTimer ActiveX in order to change customer code to our extension. I don't think I can load the provided .wrx in P4GL test box.

#19 - 06/02/2017 04:09 PM - Greg Shah

> I don't think I can load the provided .wrx in P4GL test box.

You probably can. They were originally created in the appbuilder. Try to load one there.

Either way, we will have to provide hints for this that can be read at conversion time and turned into setter calls in the converted code. Why one would want to put these in a binary wrx file, I don't know. But we will make it simple and put it in the code.

#20 - 06/09/2017 09:37 AM - Ovidiu Maxiniuc

As with revision 11159 of branch 1818a, FWD looks for a special .ext-hints file (I can optionally move the content to normal .hints) and read information on ControlFrames. It uses those to convert ActiveX references to FWDTimer extension early in the conversion process (at the end of post-parse-fixups stage). After this step, there should be no more references to legacy CtrlFrame and chCtrlFrame.

The new .ext-hints files are standard xml files. The values for INTERVAL and ENABLED properties were extracted from associated .wrx binary legacy files using reverse engineering. Note that some timers start automatically when created, some other are started programmatically on some events. As result, the original .wrx files are not needed any more in FWD.

#21 - 06/09/2017 10:14 AM - Ovidiu Maxiniuc

Technical details for the summary from note 20.
The .ext-hints looks like this

```
<extension>
   <timer control="CtrlFrame"   comhandle="chCtrlFrame"   handle="hTimer"  interval="1000" enabled="True" call
back="CtrlFrame.PSTimer.Tick"   />
   <drop-procedure name="control_load" />
</extension>
```

If the attribute value is present it will be injected in intermediary AST, otherwise the default value will be used. AST tree for declarations of ControlFrame will be processed as follows:

Original code	Result

Definition of legacy variables:
 	
`DEFINE VARIABLE CtrlFrame AS WIDGET-H ANDLE NO-UNDO.`
`DEFINE VARIABLE chCtrlFrame AS COMPON ENT-HANDLE NO-UNDO.`
	They are replaced with definition of a standard handle for the new object.
	
	`DEFINE VARIABLE hTimer AS HANDLE NO-U NDO.`
	
	Note that the com-handle variable is gone!

Creation of the CONTROL-FRAME to hold the ActiveX
 	
`CREATE CONTROL-FRAME CtrlFrame ASSIGN`
`  FRAME  = FRAME DEFAUL T-FRAME:HANDLE`
`  ROW  = 1`
`  COLUMN  = 1`
`  HEIGHT  = 1`
`  WIDTH  = 7`
`  HIDDEN  = yes`
`  SENSITIVE  = yes.`
	A new-type TIMER object part of the FWD Extension is dynamically created using a syntax similar to other kind of ABL objects stored in a handle:
	
	`CREATE TIMER <hTimer>`
	`  ASSIGN`
	`  INTERVAL = <N>`
	`  ENABLED = <Yes/No>`
	`  CALLBACK = "<callback>".`
	
	The handle name and <attr-val> values are read from .ext-hints file.

The ActiveX event callback procedure:
 	
`PROCEDURE CtrlFrame.PSTimer.Tick .`
`  [...]`
`END PROCEDURE.`
	Remains unchanged.
	The name of this procedure must be set in CALLBACK of respective FWDTimer in .ext-hints file.

The procedure that AppBuilder injects for loading the ActiveX:
 	
`PROCEDURE control_load :`
`  /* Purpose:  Load the OCXs */`
`  [...]`
`END PROCEDURE.`
	Procedure dropped completely. There may be multiple procedures that can be dropped. They are added to .ext-hints list after analysing the ABL code.
	TODO:
	Check if programmer added additional code here after the code was automatically generated.

Chained access to ActiveX properties
 	
`chCtrlFrame:PSTimer:Interval = 100.`
`a = chCtrlFrame:PSTimer:Interval.`
`chCtrlFrame:PSTimer:Enabled = Yes.`
`b = chCtrlFrame:PSTimer:Enabled.`
	Are converted to accesses (get/set) to attributes of the FWDTimer object from FWD Extension:
	
	`hTimer:Interval = 100.`
	`a = hTimer:Interval.`
	`hTimer:Enabled = Yes.`
	`b = hTimer:Enabled.`

Any access to old CtrlFrame
 	
`CtrlFrame:NAME = "CtrlFrame":U .`
`CtrlFrame:MOVE-AFTER(btnHelp:HANDLE I N FRAME schedulerFrame).`
	are dropped since there is no such object. For special case when the name attribute is changed, the callback attribute from .ext-hints file should be adjusted.

Note that all above resulting ABL code is at syntactic level. The code from right column is never generated/visible.

#23 - 06/09/2017 01:12 PM - Ovidiu Maxiniuc

- Related to Feature #3262: implement COM/OLE Automation support added

#24 - 06/14/2017 12:41 PM - Greg Shah

Code Review Task Branch 1818a Revision 11160

As mentioned before, this is really good work. Some feedback:

1. I understand the interest in making changes to the code at a very early stage (post-parse fixups) so that the annotations processing is possibly simpler. The problem with doing it that early is that it affects the "original" AST that can be used for more than conversion. For example, we use it for reporting and call graph analysis. That processing requires unmodified original ASTs as input. Please move the rewriting of the PSTimer code to annotations. If it is done early enough, it should be pretty safe.

2. com.goldencode.p2j.com is potentially a confusing package name since most people would guess it is part of a .com domain name. Please change this package name to com.goldencode.p2j.comauto which seems pretty unambiguous.

3. In common-progress.rules, the read_only_attribute states that following attributes are write-enabled only in FWD extension! but I don't see that the extension is being checked there. For any attribute that would not be for the PSTImer, would still be reported as writable. ENABLED will fall into this category.

4. In progress.g, I think that commenting out the active_x assignment is incorrect (line 21027: | KW_CNTRL_FR { is_widget = true; /* active_x = true; */ }). A control frame can only be used with ActiveX and we need to know this downstream.

5. In commhandle.setProperty(), is the use of setter.getParameterTypes()[0] always safe? I guess if the annotations are incorrect and/or there are no parameters, then there will be an array of length 0.

6. commhandle.setProperty() and other such methods in that class (call, marshall...) are missing javadoc.

7. What is the advantage to keeping the ControlFrameWidget, ActivexFrame and related support classes? I see that one might be able to convert OCX code, but it won't ever run. Since we don't intend to support OCX/Active-X, is the idea to provide stubs that will allow conversion and compilation? This could be useful. But I would want to make sure it is clear that this is just stubbed out and the original code must be modified. In regard to the ActivexFrame class, I'm not sure it is needed for this purpose.

8. In ComObject javadoc, what does the s refer to in Base class for COM objects that will be stored using s?

9. ComServer.emit() assumes there will always be a ControlFrameWidget instance for each ComObject. That doesn't seem right. For non-OCX cases, the COM Automation support is completely non-visual and has no control-frame.

10. In PSTimer.run(), any spurious interruption of the wait() will cause a new interval to be started. I suspect that instead, the toSleep value should be decremented and a partial interval should be attempted. This would make it more likely that the interval value will be honored in the presence of spurious interrupts.

11. In PSTimer.run(), toSleep is never re-assigned. For this reason, the if (enabled && toSleep != interval) check seems like dead code.

#25 - 06/21/2017 02:02 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> Code Review Task Branch 1818a Revision 11160

Thank you for the review. I adjusted the code in 11162. This revision is rebased on trunk 11153. Here are my answers and other notes.

> 1. I understand the interest in making changes to the code at a very early stage (post-parse fixups) so that the annotations processing is possibly simpler. The problem with doing it that early is that it affects the "original" AST that can be used for more than conversion. For example, we use it for reporting and call graph analysis. That processing requires unmodified original ASTs as input. Please move the rewriting of the PSTimer code to annotations. If it is done early enough, it should be pretty safe.

Done. Moved just after dropping unreachable code.

> 2. com.goldencode.p2j.com is potentially a confusing package name since most people would guess it is part of a .com domain name. Please change this package name to com.goldencode.p2j.comauto which seems pretty unambiguous.

Done.

> 3. In common-progress.rules, the read_only_attribute states that following attributes are write-enabled only in FWD extension! but I don't see that the extension is being checked there. For any attribute that would not be for the PSTImer, would still be reported as writable. ENABLED will fall into this category.

This was a little tricky. FWD is unable to tell whether a handle holds a timer or any other object. To fix this I moved the decision to runtime by adding a new interface, Enableable. The implementations of R/O ENABLED attribute will simply call readOnlyError("ENABLED"); as the conversion would have done. However, I found no such attribute ENABLED, only ENABLE method and statements in ABL Reference. The only getter for such attribute reports it as not implemented.

> 4. In progress.g, I think that commenting out the active_x assignment is incorrect (line 21027: | KW_CNTRL_FR { is_widget = true; /* active_x = true; */ }). A control frame can only be used with ActiveX and we need to know this downstream.

I think that is correct, though. The CONTROL-FRAME exposes some methods and attributes that are accessible via ControlFrame interface. This is kept in a normal handle and ABL programmer can reuse it. In fact, it is used only to load the foreign object and allocate it some space on the window surface. All accesses to contained object(s) are performed using com-handle s.

> 5. In commhandle.setProperty(), is the use of setter.getParameterTypes()[0] always safe? I guess if the annotations are incorrect and/or there are no parameters, then there will be an array of length 0.

Indeed, I was relying on correct annotations. I added in this release a double-check, just in case.

> 6. commhandle.setProperty() and other such methods in that class (call, marshall...) are missing javadoc.

Sorry, somehow I forgot about then. Added now.

> 7. What is the advantage to keeping the ControlFrameWidget, ActivexFrame and related support classes? I see that one might be able to convert OCX code, but it won't ever run. Since we don't intend to support OCX/Active-X, is the idea to provide stubs that will allow conversion and compilation? This could be useful. But I would want to make sure it is clear that this is just stubbed out and the original code must be modified. In regard to the ActivexFrame class, I'm not sure it is needed for this purpose.

At first, I started the implementation thinking that we will support some kind of OCX rewritten in Java. Then I let those classes, indeed, to provide

stubs so the generated code is fully convertible, the result compilable and even runnable, even if not with the expected results. If the runtime is really not needed, the client-side classes can be dropped. I suggest to keep them, but document this more explicitly.

> 8. In ComObject javadoc, what does the s refer to in Base class for COM objects that will be stored using s?

Sorry, this is a typo. The s was intended to be the plural for following comhandle in link tag.

> 9. ComServer.emit() assumes there will always be a ControlFrameWidget instance for each ComObject. That doesn't seem right. For non-OCX cases, the COM Automation support is completely non-visual and has no control-frame.

Indeed. The timer seems like a COM Automation object, but they decided to make it an ActiveX. I did not study the COM Automation very well as I could not instantiate any on the test machine. The event handling must be added if we will support them. In case of the PSTimer they are not needed as the event is delivered via the callback method.

> 10. In PSTimer.run(), any spurious interruption of the wait() will cause a new interval to be started. I suspect that instead, the toSleep value should be decremented and a partial interval should be attempted. This would make it more likely that the interval value will be honored in the presence of spurious interrupts.

This is correct. I added an internal loop for precisely count the requested amount of time. I am testing now the new implementation.

> 11. In PSTimer.run(), toSleep is never re-assigned. For this reason, the if (enabled && toSleep != interval) check seems like dead code.

toSleep was never re-assigned but the filed interval might. The conditional was testing whether the interval was changed while the thread was suspended, in which case it had to restart with the new interval value. Now I added a dedicated flag for this case.

#26 - 07/19/2017 10:59 AM - Greg Shah

Code Review Task Branch 1818a Revision 11162

I am generally OK with the changes.

> 3. In common-progress.rules, the read_only_attribute states that following attributes are write-enabled only in FWD extension! but I don't see that the extension is being checked there. For any attribute that would not be for the PSTImer, would still be reported as writable. ENABLED will fall into this category.

> This was a little tricky. FWD is unable to tell whether a handle holds a timer or any other object. To fix this I moved the decision to runtime by adding a new interface, Enableable. The implementations of R/O ENABLED attribute will simply call readOnlyError("ENABLED"); as the conversion would have done. However, I found no such attribute ENABLED, only ENABLE method and statements in ABL Reference. The only getter for such attribute reports it as not implemented.

Although I do like that the attribute is being handled (for the pstimer) like a regular attribute. But why is CommonWidget including Enableable? No widgets implement this and they don't need to do so. Because HandleCommon has Enableable included, by default all resources (widgets or otherwise) will report that the attribute does not exist. Only the PSTimer will respond differently and that is correct.

I think you can just remove the Enableable from CommonWidget / GenericWidget.

> 4. In progress.g, I think that commenting out the active_x assignment is incorrect (line 21027: | KW_CNTRL_FR { is_widget = true; /* active_x = true; */ }). A control frame can only be used with ActiveX and we need to know this downstream.

> I think that is correct, though. The CONTROL-FRAME exposes some methods and attributes that are accessible via ControlFrame interface. This is kept in a normal handle and ABL programmer can reuse it. In fact, it is used only to load the foreign object and allocate it some space on the window surface. All accesses to contained object(s) are performed using com-handle s.

With this change, sym.setPossibleActiveX() can never be called and this means that inside chained_object_members the isComHandleType() will now act differently. The handle that referneces a control-frame is a special type of resource in the 4GL. It actually acts as both a regular 4GL handle and well as a com-handle. LoadControls() is a COM method. It also has COM attributes Control-Name, Controls, Height, Left, Name, Top, Widget-Handle and Width. This is according to the 4GL documentation. The problem is that we have to be able to differentiate between this usage of a regular 4GL handle and versus a com-handle. In chained_object_members, when the refnode is a var_handle or field_handle that was initialized (at least once) using a control-frame, then it may be chained with those COM methods and attributes. Without the active_x flag being set on line 21027, this will be broken.

Does re-enabling this cause problems for your testcases?

#27 - 08/01/2017 09:54 AM - Greg Shah

From Ovidiu (with edits by me to remove customer code references):

> I did some investigations related to this "ENABLED" attribute. It was added about 2 years ago because some code was found in a customer project. It is documented in progress.g as: missing from keyword index, found in customer code and UNTESTED at this time.
> At this moment, I don't have any base to support the existence of this attribute any more. The code still exist in the customer application but the occurrences seem like dead code because the code can only be executed in an else branch that seems like it can never be reached. if it would ever be reached, an error 4052 will probably raised: "**ENABLED is not a queryable> attribute for <widget id>".
>
> I tried to create some testcases with this attribute in different widgets (assuming the other, SCREEN-VALUE is valid as well), but I get this error each time when the execution reaches it. ABL compiles the code (lhField is a handle, after all), but it will fail at runtime.
>
> The existence of this attribute is the reason I added the Enableable interface, but I am thinking now to remove this completely. The problem is, because we are forcing the cast of handle to the interface assumed at conversion time, the conversion/built of the customer's project will fail, causing a regression.
>
> The correct action I think is one of the following:
>
> - fix the ABL code by dropping the invalid code;
> - create the correct interface and place the ENABLED attribute there, if there is indeed an object type in ABL that supports this attribute and I failed to identify.

This seems like a special case. I do know that ENABLED is somehow recognized by the compiler as a valid attribute name. For example, this does NOT compile in the 4GL:

```
def var h as handle.
if false then h:whatever.
```

But this DOES compile:

```
def var h as handle.
if false then h:enabled.
```

But as you note, we have not yet found a widget use case that works. I tried with radio-set (because it has the ENABLE() method, but it doesn't work there.

I agree with your assessment:

- edit the customer project to comment out that code
- remove Enableable as a widget interface

#28 - 08/02/2017 11:24 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> 4. In progress.g, I think that commenting out the active_x assignment is incorrect (line 21027: | KW_CNTRL_FR { is_widget = true; /* active_x = true; */ }). A control frame can only be used with ActiveX and we need to know this downstream.

I think that is correct, though. The CONTROL-FRAME exposes some methods and attributes that are accessible via ControlFrame interface. This is kept in a normal handle and ABL programmer can reuse it. In fact, it is used only to load the foreign object and allocate it some space on the window surface. All accesses to contained object(s) are performed using com-handle s.

With this change, sym.setPossibleActiveX() can never be called and this means that inside chained_object_members the isComHandleType() will now act differently. The handle that referneces a control-frame is a special type of resource in the 4GL. It actually acts as both a regular 4GL handle and well as a com-handle. LoadControls() is a COM method. It also has COM attributes Control-Name, Controls, Height, Left, Name, Top, Widget-Handle and Width. This is according to the 4GL documentation. The problem is that we have to be able to differentiate between this usage of a regular 4GL handle and versus a com-handle. In chained_object_members, when the refnode is a var_handle or field_handle that was initialized (at least once) using a control-frame, then it may be chained with those COM methods and attributes. Without the active_x flag being set on line 21027, this will be broken.

Does re-enabling this cause problems for your testcases?

Just a little bit. If forced to be an active_x, the width and height attributes of CtrlFrame are unwrapped with unwrapControlFrame instead of unwrapSizeable. This is not correct, according to my testcases, CtrlFrame:WIDTH and CtrlFrame:HEIGHT return the widget's dimensions in character units. The myCOM:WIDTH and myCOM:HEIGHT (myCOM is a COM-HANDLE) return the size of the ActiveX UI, in pixels. It's just a name collision.

Moreover, I have do disagree with you. The handle that references the activex is really not different from any other handle. Attempting to access any of the attributes: CONTROL-NAME, CONTROLS, WIDGET-HANDLE, TOP and LEFT, and also the method LoadControls will result in the compile-time error 3406: *Unknown attribute <attribute> used in widget:attribute phrase* to be issued. As noted above, the only matches are WIDTH and HEIGHT, but this is just a clash, the result of the attribute is different: int (pixels from com-handle) vs decimal (characters from handle).

I tested the reverse. Attempting to call COM-HANDLE on a handle that point to a something else will result in a natural runtime error ***<attribute> is not a <settable/queryable> attribute for <widget id>*. (4052) to be issued.

In conclusion, the setPossibleActiveX and related code is not correct and need to be dropped. I updated the uast/com-handle/pstime-demo2.w testcase with the invalid calls commented out.

#29 - 08/02/2017 11:31 AM - Greg Shah

> In conclusion, the setPossibleActiveX and related code is not correct and need to be dropped. I updated the uast/com-handle/pstime-demo2.w testcase with the invalid calls commented out.

OK, your testing overrides my coding of the parser based on Progress' documentation. I should not be surprised that their docs are wrong. Go ahead and remove this.

> Attempting to access any of the attributes: CONTROL-NAME, CONTROLS, WIDGET-HANDLE, TOP and LEFT, and also the method LoadControls will result in the compile-time error 3406:

I think these are also dead code too, and they should be removed unless there is also a usage outside of control-frame.

#30 - 08/02/2017 04:23 PM - Ovidiu Maxiniuc

I discovered the reason why ControlFrame was assumed to expose these attributes and fields: there are two entities ~~classes~~ instead.
The ControlFrame is just a normal widget with the addition of COM-HANDLE attribute. The other class here is the *Control-frame COM object*. This new one has in fact the properties (not attributes) and LoadControls method accessible through the comhandle.
In fact, it was wrong that comhandle extended handle. They are not related in P4GL (except that they are both BDTs).

I am working now on fixing the implementation according to new discoveries.

#31 - 08/10/2017 09:12 AM - Greg Shah

Could you summarize the status of this task? What has been done to resolve the known issues? Is it ready for a final code review? What testing remains?

#32 - 08/15/2017 09:35 AM - Greg Shah

Code Review Task Branch 1818a Revision 11171

1. The inclusion of COM_INVOCATION in the ExpressionConversionWorker is needed for our Analytics tools when run on applications that have COM automation access. Can you make your code safe to handle that response instead of removing it?

2. comhandle.unwrapComObject() is missing javadoc.

#33 - 08/15/2017 11:06 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> 1. The inclusion of COM_INVOCATION in the ExpressionConversionWorker is needed for our Analytics tools when run on applications that have COM automation access. Can you make your code safe to handle that response instead of removing it?

COM_INVOCATION was not removed. You can find it below, sharing the same code with DB_REF_NON_STATIC. They both are relying on the context to infer the result at conversion time or use the runtime polymorphic mechanisms, returning the generic BaseDataType.

I understand Analytics uses the com_invocation pseudo-type to distinct these special calls. We need to find a solution that covers both usages. One solution is to include the com_invocation with BaseDataType in all places handling _POLY data, but this seems dirty (lot of changes) to me.

#34 - 08/15/2017 03:49 PM - Greg Shah

> COM_INVOCATION was not removed. You can find it below, sharing the same code with DB_REF_NON_STATIC. They both are relying on the context to infer the result at conversion time or use the runtime polymorphic mechanisms, returning the generic BaseDataType.

OK, I had missed this.

> I understand Analytics uses the com_invocation pseudo-type to distinct these special calls. We need to find a solution that covers both usages. One solution is to include the com_invocation with BaseDataType in all places handling _POLY data, but this seems dirty (lot of changes) to me.

Actually, what I need for the Analytics is to return non-null. The BDT string is good enough. This should work as is.

#35 - 08/15/2017 03:53 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> > I understand Analytics uses the com_invocation pseudo-type to distinct these special calls. We need to find a solution that covers both usages. One solution is to include the com_invocation with BaseDataType in all places handling _POLY data, but this seems dirty (lot of changes) to me.
>
> Actually, what I need for the Analytics is to return non-null. The BDT string is good enough. This should work as is.

This is a really good news! I was afraid I have to alter a lot of code that was testing for BDT, in both ECW and TRPL (maybe other places).

#36 - 08/16/2017 03:30 PM - Ovidiu Maxiniuc

Task branch 1818a was merged with 3292a, 2676b and 1830a. The minor issues found in preliminary tests were fixed.
Committed revision 11177.

#37 - 08/16/2017 05:08 PM - Ovidiu Maxiniuc

Unified branch r11177 passed standard conversion testing on devsrv01.

I committed the extension hints for PSTimer FWD extension (.ext-hints files) into the repository FWD project of customer. They contain values I extracted from corresponding .wrx files as noted in entry 20 above. Committed as revision 77.

#38 - 08/17/2017 09:58 AM - Ovidiu Maxiniuc

Task branch 1818a was rebased with trunk r11158. Pushed up to revision 11179.

#39 - 08/17/2017 10:15 AM - Greg Shah

Code Review Task Branch 1818a Revision 11180

The date of the 081 history entry in build.xml is incorrect. Otherwise everything looks fine.

#40 - 08/18/2017 05:31 PM - Ovidiu Maxiniuc

Task branch 1818a was updated. Some issues were identified while testing with customer's code. They were fixed.
Committed revision 11181.

#41 - 08/23/2017 11:39 AM - Greg Shah

Code Review Task Branch 1818a Revision 11184

1. Do you have an example AST for DISABLE TRIGGERS which has a SYMBOL as the table reference? I don't think that should ever happen. I'm pretty sure the current code will break when used for temp-table or work-table cases. It doesn't seem like it would even work for TABLE cases. Because the record reference can have different forms, we generally try to allow it to emit naturally (as a buffer reference). From that, the table name can be derived. This is exactly what we do for DatabaseTriggerManager.registerDatabaseTrigger(). Shouldn't we just implement the disable form the same way? It will be simpler and will make more sense because it is the same approach. It would also eliminate the need to bypass buffer emit in database_references.rules.

2. For the convert/expressions.rules change I have a question and a request.

- Under what circumstance is there an COM_METHOD/COM_PARAMETER/EXPRESSION node hierarchy? In the parser, COM_PARAMETER and COM_METHOD can only be matched in com_property_or_method. These can call com_parameter which will call expr. But to get to com_property_or_method, one must be in primary_expr which means there is already a call to expr further up the tree. This level of recursion into expr will suppress the emit of the artificial EXPRESSION node. Perhaps this is some generated AST nodes that don't come from the parser? If so, I prefer to make the nodes look like the parser would generate them, otherwise it is quite confusing.
- Many places in our code need something similar to the code that you put into expressions.rules for reparenting. However, it is done differently so that we don't have 100 different reparenting cases in that code at the bottom of the file. The way we do it is we create or calculate the correct JAST ID and then write that target parent as a "temporary peerid" annotation in the EXPRESSION node (search on "temporary peerid" to see how we do it). Then the code at the end can simply lookup the closestPeerId which will find the proper target. That code will rewrite the "peerid" annotation and set it to the newly created JAST ID value. This is why we call it temporary.

3. I would prefer to avoid all the special case processing of COM_PARAMETER in expressions.rules, literals.rules, operators.rules and variable_references.rules. If you have to put all that code there, it is a sign that we need a better way. We will continue to find other cases that have to be "patched". The better way to do it is to add correct "peerid" annotation (for the correct target JAST parent node) to the COM_PARAMETER node. Then everything underneath it will naturally emit to the correct place. This will also fix the problems in item 2 above.

#42 - 08/23/2017 02:10 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> Code Review Task Branch 1818a Revision 11184
>
> 1. Do you have an example AST for DISABLE TRIGGERS which has a SYMBOL as the table reference?

```
define buffer b-book for book.
DISABLE TRIGGERS FOR LOAD OF book.
DISABLE TRIGGERS FOR LOAD OF b-book.
```

The AST tree for 2nd statement looks like this:

```
    <ast col="0" id="721554505740" line="0" text="statement" type="STATEMENT">
      <ast col="1" id="721554505741" line="3" text="DISABLE" type="DISABLE_TRIGGERS">
        <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
        <ast col="22" id="721554505744" line="3" text="LOAD" type="KW_LOAD"/>
        <ast col="0" id="721554505756" line="0" text="book" type="SYMBOL">
          <annotation datatype="java.lang.String" key="bufclassname" value="Book.Buf"/>
          <annotation datatype="java.lang.String" key="pkgname" value="com.goldencode.testcases.dmo.p2j_test"/
>
        </ast>
      </ast>
    </ast>
```

> I don't think that should ever happen. I'm pretty sure the current code will break when used for temp-table or work-table cases.

Temp-table and work-table cannot have triggers. Attempting this will result in compile error 3335: *You may not associate database triggers with workfiles or temp-tables*.

> It doesn't seem like it would even work for TABLE cases. Because the record reference can have different forms, we generally try to allow it to emit naturally (as a buffer reference). From that, the table name can be derived. This is exactly what we do for DatabaseTriggerManager.registerDatabaseTrigger(). Shouldn't we just implement the disable form the same way? It will be simpler and will make more sense because it is the same approach. It would also eliminate the need to bypass buffer emit in database_references.rules.

I guess you're right. This works at this moment. Should I add this to current branch or defer this in a separate task?

> 2. For the convert/expressions.rules change I have a question and a request.
>
> - Under what circumstance is there an COM_METHOD/COM_PARAMETER/EXPRESSION node hierarchy? In the parser, COM_PARAMETER and COM_METHOD can only be matched in com_property_or_method. These can call com_parameter which will call expr. But to get to com_property_or_method, one must be in primary_expr which means there is already a call to expr further up the tree. This level of recursion into expr will suppress the emit of the artificial EXPRESSION node. Perhaps this is some generated AST nodes that don't come from the parser? If so, I prefer to make the nodes look like the parser would generate them, otherwise it is quite confusing.

The EXPRESSION is created early in conversion, when the file is processed in src/com/goldencode/p2j/uast/AstGenerator.java:1506, the node is already there. I'll be back after deeper investigations.

> - Many places in our code need something similar to the code that you put into expressions.rules for reparenting. However, it is done differently so that we don't have 100 different reparenting cases in that code at the bottom of the file. The way we do it is we create or calculate the correct JAST ID and then write that target parent as a "temporary peerid" annotation in the EXPRESSION node (search on "temporary peerid" to see how we do it). Then the code at the end can simply lookup the closestPeerId which will find the proper target. That code will rewrite the "peerid" annotation and set it to the newly created JAST ID value. This is why we call it temporary.

I will look into this. Indeed, there are about 4 or 5 places where the correct parent is searched and I was thinking about unifying the algorithm somehow.

> 3. I would prefer to avoid all the special case processing of COM_PARAMETER in expressions.rules, literals.rules, operators.rules and variable_references.rules. If you have to put all that code there, it is a sign that we need a better way. We will continue to find other cases that have to be "patched". The better way to do it is to add correct "peerid" annotation (for the correct target JAST parent node) to the COM_PARAMETER node. Then everything underneath it will naturally emit to the correct place. This will also fix the problems in item 2 above.

OK. That would be nice.

#43 - 08/23/2017 02:28 PM - Greg Shah

> 1. Do you have an example AST for DISABLE TRIGGERS which has a SYMBOL as the table reference?
> ...
> The AST tree for 2nd statement looks like this:

Hmmm. This is very wrong. It can't happen in the parser. The record rule should rewrite the SYMBOL (book) to be a TABLE node that includes some annotations like schemaname, bufname and dbname.

I have looked deeper. The problem is in annotations/database_general.rules, which has this comment:

```
2. replace STATEMENT/DISABLE_TRIGGERS/TABLE with STATEMENT/DISABLE_TRIGGERS/SYMBOL
```

That code should be removed.

> Should I add this to current branch or defer this in a separate task?

Go ahead and fix it now, please.

#44 - 08/23/2017 04:06 PM - Ovidiu Maxiniuc

I did. The changes are more extensive. Please see the revision 11186. There are a couple of other fixes (in comments.rules and NameConverter.java) for issues found in customer's code.

#45 - 08/23/2017 04:37 PM - Greg Shah

Code Review Task Branch 1818a Revision 11186

I'm fine with the changes except for the retention of the COM_PARAMETER code that is being put into methods_attributes.rules, expressions.rules, literals.rules, operators.rules and variable_references.rules.

> The changes are more extensive.

Try adding this code to com_access.rules. I think it can go in the walk rules.

```
<rule>type == prog.com_parameter
   <!-- in the case of a COM_PARAMETER, the [closestPeerId] is the COM_METHOD,
        emitted as a simple string/name, and that is not our parent. Instead we
        need to go up one level, to the COM_INVOCATION. There is ONE EXCEPTION
        here, the [LoadControls] is emitted directly as a java method call, instead
        as 1st string argument of [call] -->
   <rule>(#(long) parent.getAnnotation("oldtype")) != prog.kw_loadctrl
      <action>copy.putNote("peerid", getAst(closestPeerId).parent.id)</action>
   </rule>
</rule>
```

Then remove all the COM_PARAMETER stuff from the methods_attributes.rules, expressions.rules, literals.rules, operators.rules and variable_references.rules.

If that doesn't work, help me understand the problem and I will help you fix it.

#46 - 08/24/2017 10:46 AM - Ovidiu Maxiniuc

It was rather late last night to start the investigation on the COM_PARAMETER issue.
After rebasing the two new revisions, I used your code from previous note. It worked for my testcases and I will try it full project now (I guess this will

be reported to #3330).

The task branch 1818a rebased and updated. Committed revision 11189.

#47 - 08/24/2017 11:36 AM - Greg Shah

Code Review Task branch 1818a Revision 11189

The changes look good. Please put this through testing. If this passes, then it can be merged to trunk.

#48 - 08/25/2017 11:38 AM - Ovidiu Maxiniuc

Branch 1818a was merged to trunk as revision 11161. Notification was sent to team. Test result an task branch were archived.

#49 - 08/25/2017 11:44 AM - Greg Shah

- % Done changed from 0 to 100

- Status changed from WIP to Closed

#50 - 04/12/2021 07:23 AM - Greg Shah

- Related to Bug #5254: FWD extension control must follow the lifetime of the counterpart 4GL OCX/ActiveX added