Base Language - Feature #2108

event model rationalization

04/08/2013 11:07 AM - Greg Shah

Status:	Closed	**Start date:**	06/08/2013
Priority:	Normal	**Due date:**	06/14/2013
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	40.00 hours
Target version:	Runtime Support for Server Features		
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #2024: implement runtime support for database t...	**Closed**	**06/19/2013**	**07/01/2013**
Related to Base Language - Feature #1991: runtime support for appserver	**Closed**	**04/29/2013**	**05/24/2013**
Related to Base Language - Feature #1958: implement the runtime support for s...	**Closed**	**04/03/2013**	**05/15/2013**

History

#1 - 04/08/2013 11:40 AM - Greg Shah

The 4GL event model is inherently "client" oriented. It's primary usage is built around the user-interface and the implementation of triggers for a call-back UI model that was needed for GUI support in the v7 timeframe. This is the WAIT-FOR/PROCESS EVENTS/APPLY event loop implementation.

As other event processing has been added for non-UI purposes, it seems that the same event model was just extended. This is possible in the 4GL because the system is inherently designed such that the entire application runs on the client side and just accesses data from a remote DB server. The (relatively) more recent additions server-side processing (appserver and WebSpeed) has not changed this model, but in such cases the UI portions are essentially null or unused, so there is no conflict.

The problem for us is that we have split the UI off into its own remote thin-client/presentation engine, where we handle all the UI event processing. This includes all the keyboard (low level key labels, key presses and also including the high level key functions like "GO"), mouse, high level events (e.g. LEAVE) and direct manipulation (i.e. drag and drop). These are naturally handled in our current approach.

But all of the real application processing, including all of the business logic runs on the P2J application server. The following types of event processing would normally process on the server in P2J.

1. sockets: we should consider whether this processing is really is a client-side activity like file system access, perhaps only the "client socket" stuff qualifies, but I am not sure

2. database triggers: I am hopeful this one can be split off cleanly, if (as I suspect) the database trigger events are executed independent of a WAIT-FOR

3. appserver (e.g. PROCEDURE COMPLETE)

4. procedure processing (e.g. ON "close" OF THIS-PROCEDURE): note that this includes all the different forms of procedure handles AND it also seems to be similar to the processing of CURRENT-WINDOW/ACTIVE-WINDOW/DEFAULT-WINDOW (e.g. ON "close" OF CURRENT-WINDOW)

5. "user" or "developer" events (e.g. "U1")

6. DDE events (not needed for milestone 7)

7. ProDataSet events (not needed for milestone 7)

I believe the "named events" (PUBLISH/SUBSCRIBE) event model is separate and can be safely implemented on the server. I suspect the database triggers (even those implemented using the ON statement) are also separate, but testing will have to be done to determine if this is correct and any dependencies that may exist if it is not correct.

The objective of this task is to determine the exact requirements for the above issues, design a proper solution and implement it. The items for DDE and ProDataSets will be ignored for now.

#2 - 04/26/2013 08:02 AM - Greg Shah

- Start date set to 06/08/2013

- Estimated time set to 40.00

- Due date set to 06/14/2013

- Assignee set to Constantin Asofiei

We have already determined that the sockets support needs to be client-side.

Appserver will be largely handled during #1991.

Assuming database triggers can be safely placed on the server (something that should be determined in #2024), the primary work left would be for procedures and user-defined events.

#3 - 07/08/2013 04:23 PM - Constantin Asofiei

- Status changed from New to WIP

I've been looking for common points between the various non-UI events and my findings are these:

1. events like PROCEDURE-COMPLETE, READ-RESPONSE, CONNECT, CLOSE, user-events can be used in a WAIT-FOR <event> of <handle>. clause, even if the resource currently mapped by the <handle> can not implicitly receive the <event>. These can be explicitly sent using APPLY <event> TO <handle>, even if the resource referred by the <handle> is not documented to listen for these events. Thus, something like this will work:

```
def var i as int.
form i with frame f1.
view frame f1.
h = frame f1:handle.

on 1 anywhere do:
   apply "read-response" to h.  /* when 1 is pressed, the wait-for will end */
end.

wait-for read-response of h. /* the handle is a frame */
```

2. when using the aforementioned events, the <handle> in the WAIT-FOR ... OF <handle>. statement needs to be in a state so that it can receive events (i.e. SENSITIVE). Note that the WAIT-FOR is not linked to the handle, but is linked to the resource referred by the handle.
3. in the docs, the PROCEURE-COMPLETE event (for async requests) is recommended to be used with loops listening for UI events. As I recall the server project does not use async requests, but if the client project uses them in such a manner, there is no option but rewrite the event listening loops from the client side (read further for more details).
4. the fast and simple approach to solve this is to create separate event managers for each event type (i.e. socket, procedure-complete, user-defined, etc). Thus, the conversion rules will emit different APIs, based on the event type. The restriction is that these "custom" wait-for implementations will listen only for that events, which will not allow different events to be processed (i.e. READ-RESPONSE events to be processed by a WAIT-FOR PROCEDURE-COMPLETE loop).
5. the complex approach is to rewrite the event processing/listening loops in ThinClient to allow these new event types. But, considering that some events originate on the client side (like socket events) and other on the server side (like procedure-complete and user events), we will need to rewrite the event management so that events can be pushed in the event queue on the P2J client by the P2J server (or by the client too, in case of socket events). Problem is that I currently see no easy way of hooking this into i.e. TC.waitForWorker, processEventsWorker, etc, and I think a first approach would be to generalize the event management to work with generic events, and after this hook the custom events into place.

At this time, if we implement different event queues, at some point we will have no choice but rewrite the event management to use a single queue; and this may be required for the client project, if it uses generic WAIT-FOR loops or UPDATE statements, while listening for async requests too.

#4 - 07/08/2013 06:32 PM - Greg Shah

> when using the aforementioned events, the <handle> in the WAIT-FOR ... OF <handle>. statement needs to be in a state so that it can receive events (i.e. SENSITIVE)

1. What types of handle can be used there?
2. Can you write "traditional" 4GL triggers for all of these event types?
3. What default processing is there for these events?

> the complex approach is to rewrite the event processing/listening loops in ThinClient to allow these new event types. But, considering that some events originate on the client side (like socket events) and other on the server side (like procedure-complete and user events), we will need to rewrite the event management so that events can be pushed in the event queue on the P2J client by the P2J server (or by the client too, in case of socket events).

I think we have to do this "the right way". Otherwise we will be spending time on another short term solution that is throw-away. We always get burned when we take short cuts. Please provide more thoughts on the kind of changes you plan to make.

In general, please do consider that we prefer to move away from the use of the AWT-like events for managing the UI. The 4GL UI is completely synchronous, but we have an asynchronous event model as the basis. This has caused a great deal of difficulty in trying to get the UI to work the way the 4GL works. In particular, enter/leave, focus management, validation, drawing and trigger processing are probably much more complicated than needed.

#5 - 07/09/2013 01:45 PM - Constantin Asofiei

> 1. What types of handle can be used there?

I found only these handles to be accepted by WAIT-FOR:

- active-window
- current-window
- frame
- widgets
- procedure
- socket and server-socket (only after they are connected, i.e. the SENSITIVE flag is set to true).

WAIT-FOR does not accept PSEUDO-WIDGETS (like SESSION, FILE-INFO, etc) and widgets which do not have the SENSITIVE attribute (the exception to this rule is the PROCEDURE handle, which can be used, although it doesn't accept the SENSITIVE attribute).

2. Can you write "traditional" 4GL triggers for all of these event types?

Yes. This works:

```
on procedure-complete of h do:
   message "procedure-complete" self:handle.
end.
on read-response of h do:
   message "read-response" self:handle.
end.
on connect of h do:
   message "connect" self:handle.
end.
```

The ON ... ANYWHERE form works too. Note that a PROCEDURE-COMPLETE trigger abends in 4GL, as it tries to resolve the context of the trigger (I guess is the handle where the procedure resides, set by the EVENT-PROCEDURE procname IN handle clause). Anyway, even if my tests might not make sens in a 4GL-developer's point of view, this do show some info about how 4GL treats the events.

3. What default processing is there for these events?

1. procedure-complete - this event is raised when the async request is completed; if the EVENT-PROCEDURE procname clause is used with the RUN ASYNC statement, then procname is invoked. Else, nothing is invoked.
2. read-response - raised when data is received on the socket or the socket disconnects; if the set-read-response-procedure was used to specify a procedure, that procedure is invoked.
3. connect - raised when a server-socket has received a new connection; if the set-connect-procedure was used to specify a procedure, that procedure is invoked.

User-defined events looks like are implemented like key-events; U1-U10 have the 551-560 key codes and for the CLOSE event the docs state that is mapped to the F8/ESC-Z/CTRL-ALT-Z keys, but I couldn't duplicate it.

#6 - 08/08/2013 09:04 AM - Greg Shah

Please post an update on your research findings and your planned approach.

#7 - 08/09/2013 06:07 PM - Constantin Asofiei

First, when working on understanding how the 4GL event model behaves, I noticed that P2J has some differences when PROCESS EVENTS statement and WAIT-FOR-like statements (WAIT-FOR, UPDATE, etc) process key events. In P2J, keys are buffered in the TypeAhead class and picked up only when a key is expected by a user-blocking statement. In 4GL, they are picked up by the PROCESS EVENTS statement too; from testing, even if keys are pressed by the user before the PROCESS EVENTS statement, their target is not determined until the PROCESS EVENTS is encountered. This means one of two cases:

1. when key is pressed, an event is raised for it, but the target widget to which it applies is not computed until an event-processing statement is executed
2. when key is pressed, an event is not raised for it; instead, the keys are buffered (as P2J does now), and events are raised only when an event-processing statement is executed

Why I ended up investigating this: the two methods which would handle the "server-side events" are TC.processEventsWorker and TC.waitForWorker. The differences between these two are:

1. TC.waitForWorker picks keys supplied by the TypeAhead buffer and posts events to the EventManager
2. TC.processEventsWorker only picks events supplied by the EventManager

To hook up the "server-side events", these two should behave in a similar manner, as I'm thinking of posting the "server-side events" directly to the EventManager. Thus, changes are needed:

1. TC.waitForWorker should not only listen for keys, but for "server-side event" posting too.
2. TC.processEventsWorker should be fixed so that it can access the keys supplied by TypeAhead.

When a server-side event is generated, we will need to post it to the client-side too. The event will need to be added to the EventManager, and for this we will need to:

1. send the event name (CONNECT, PROCEDURE-COMPLETE, READ-RESPONSE), the event ID and the resource ID to the client side; this will be used to create a ServerEvent instance, and post it to the EventManager, but will not process it.
2. on server side, any "server-side event" will be registered on a map; when the client-side processes a ServerEvent instance, it will call back to the server to process it. The server side will identify the event based on the provided ID.
3. if any triggers are registered for the event, invoke them too (some behavior in 4GL is undetermined, as i.e. input variables, context can't be determined, but we can at least inform the user that trigger invocation is not possible, if one is registered).

The resource ID is needed to check for exit conditions in the WAIT-FOR loop. The event loop will end when:
- a pair of (event-name, resource id) matches the event list used by the WAIT-FOR loop
- all resources/widgets referenced by the event list are no longer in SENSITIVE state.
This takes us to creating the event list for a WAIT-FOR event of handle or event2 of handle ... statement. Here, I think the EventList needs to differentiate between a resource ID and a widget ID. We can use an EventDefinition subclass or we can extend EventDefinition to hold the resource IDs too. For resource case, whenever is needed to know if the resource is still SENSITIVE, we can either maintain the resource's SENSITIVE state on the client-side too (via a push approach, as we do know with the ScreenDefinition), or we can use a pull approach (i.e. call a server export to check the resource state).

Next, I will explain how the "server-side events" are picked up for socket-related events.

1. Server-Socket handle
   Is incorrect to think that a server socket will start listening for connections only when the event processing statement is called, as the event processing statement does only event processing. Thus, once the server-socket handle is in SENSITIVE state (via ENABLE-CONNECTIONS), the server socket is listening for incoming connections; when an incoming connection is received, an event is posted and will be processed only when the event processing statement is executed. (As a server-socket implementation note, this means the server socket will need to listen for events in a separate thread, and this will start right after ENABLE-CONNECTIONS succeeds).
2. Socket handle
   When CONNECT is called, this will only send a connect request to the socket; it will not wait for the CONNECT event on the server socket to finish. Instead, it will make the resource SENSITIVE and it will start listening for incoming data on the socket (in a separate thread); when data is received, a READ-RESPONSE event is generated and posted to the client-side. The thread listening for incoming data is suspended until the client-side processes this event (and the data will eventually be read from the socket). When the event is finished processing, the data-listening thread is notified and will start listening again for data. This suspension I think is needed because, if the server keeps sending data to the client while the client is in the READ-RESPONSE procedure, no new READ-RESPONSE events are generated (i.e. nested READ-RESPONSE events are not possible, for the same client socket).

For the appserver's PROCEDURE-COMPLETE event, the event posting will be similar to the socket approach. What will be different is how the PROCEDURE-COMPLETE events will be generated. For this, we need to keep in mind that truly-async requests are possible only in State-free mode, as in State-managed mode all "async" requests are queued as FIFO. In State-managed mode, there will be enough a single thread (per appserver resource) to receive and queue the "async" requests, execute them and generate the PROCEDURE-COMPLETE events. For State-free mode, as this is the only truly async mode, I think there is no way but to execute the async request in its own thread and each thread will be responsible of posting PROCEDURE-COMPLETE event.

As implementation approach, I suggest this:

1. make the changes to P2J event processing as part of the async requests changes (the ASYNCHRONOUS EVENT-PROCEDURE clause for the RUN statement). Idea is, the async-related changes look pretty simple and should not take too long.
2. add support for the async handle (in a different update only if I determine it to be more complex than I think of it at this time)
3. add the server-socket and socket handle support

Note that the U1..U10 user events and the CLOSE event work properly in P2J (they are named events which are supported).

#8 - 08/12/2013 08:21 AM - Greg Shah

The findings are interesting. Your approach makes sense. Well done. Please continue as planned.

#9 - 08/22/2013 09:49 AM - Constantin Asofiei

- File ca_upd20130822a.zip added

This update is good for review and includes:

1. changes to the net protocol to support async requests; this is needed because the server-side event is posted to the P2J client from a thread other than the Conversation thread.
2. server-side event processing on client side.
3. RUN ... ASYNC, PROCEDURE-COMPLETE event and ASYNCHRONOUS-REQUEST handle support; the following is currently missing for ASYNCHRONOUS-REQUEST:

- STOP and QUIT attribute implementation (they track when the remote call ends via a STOP or QUIT condition)
- CANCEL-REQUESTS and CANCEL-REQUESTS-AFTER (still couldn't duplicate it in 4GL env); I added support as described in the docs, and what is currently missing is sending the QUIT condition to the remote side

From my testing, I think is reasonbly to think a developer will not use the server-side events in user-defined triggers or APPLY statement; this is because in all cases the SELF handle is set to a specific resource and also the target procedure may have parameters. With this in mind, I didn't duplicate this behavior, as in most cases in 4GL env it can end in abnormal ways (as the SELF handle/parameters are not available) or with error messages that the procedure context is not available.

#10 - 08/27/2013 12:12 PM - Constantin Asofiei

- File ca_upd20130827a.zip added

The most important change here is related to the fact that when Session-free is used and a external procedure is invoked RUN ... PERSISTENT SET hp ON SERVER ... ASYNC, 4GL does this in two steps:

1. obtains a proxy to the procedure, which is returned to the application right after the RUN is executed; on the appserver side, it associates a dedicated Agent to it
2. initializes the proxy procedure
   If between step 1 and 2 there are async calls for this procedure, as in RUN ... IN hp, the command will be sent to the same agent and is guaranteed to run after the Agent finishes the initialization of the proxy. More, Session-free will always bind an Agent and an external procedure ran persistent; thus, in these cases, we need to associate the Agent with the procedure, even if the procedure is not ran async.

Other fixes are related to synchronization problems and some problems discovered during MAJIC testing (TypeAhead.getKeyStroke loop was not OK).

Beside the runtime regression which is currently running, I need to run one more round with this update.

#11 - 08/28/2013 04:33 PM - Greg Shah

I am in the middle of the code review. So far I have finished the ui and net package code. I have these initial questions:

1. Do key events always get processed in preference to server events in Progress? This seems to be the way that the TypeAhead.getKeystroke() method is implemented. Also, that code needs to be documented to explain that if honorServerEvents is true and there is no key in the typeahead buffer, then this will return null to allow the caller to process the server event.

2. Is it important for the new EventManager server-event methods to synchronize on wa instead of wa.serverEvents? This seems more granular and just as safe. Likewise, the other methods would synchronize on the contained wa.queue instance instead of wa.

3. What are "exit conditions" in terms of the EventList or EventDefinition? This is mentioned several times, but it is not clear what this means from just reading the code.

#12 - 08/28/2013 04:41 PM - Greg Shah

More questions:

4. In regard to your assumption in note 9:

> From my testing, I think is reasonbly to think a developer will not use the server-side events in user-defined triggers or APPLY statement; this is because in all cases the SELF handle is set to a specific resource and also the target procedure may have parameters. With this in mind, I didn't duplicate this behavior, as in most cases in 4GL env it can end in abnormal ways (as the SELF handle/parameters are not available) or with error messages that the procedure context is not available.

I don't have a problem with this. However, this needs to be documented somewhere in the code as a potential deviation from the 4GL. If there are ways of successfully using this (probably unintended) "feature" of Progress, we will eventually encounter 4GL code that does so. I would like future readers of our implementation to know that this was deliberately bypassed.

5. Does the issue in 4 above lead directly to the choice to keep the resource IDs separate from widget IDs? The resulting event processing implementation is more complex because of this split. I wonder what the negative implications are this decision AND if this is something different from how Progress does it.

6. What is the status of these (your status reports suggested you solved these problems, but the history here does not document it):

> STOP and QUIT attribute implementation (they track when the remote call ends via a STOP or QUIT condition)

> CANCEL-REQUESTS and CANCEL-REQUESTS-AFTER (still couldn't duplicate it in 4GL env); I added support as described in the docs, and what is currently missing is sending the QUIT condition to the remote side

#13 - 08/28/2013 05:59 PM - Greg Shah

OK, I have completed my code review (to the 0827a level). Overall: it is really good! The code is now also insanely complicated. I am still having difficulty keeping track of which parts run in which places (non-P2J Java client, P2J server, appserver agent, P2J interactive client...). Wow.

Last questions:

6. Is it true that the 4GL cancels the "cancel timer" thread (in our code this is in ServerImpl) when the timeout is negative? If so, add javadoc to that effect.

7. Is there only 1 possible "current" timer for cancelling requests in the 4GL?

8. Is the use of IllegalArgumentException in ControlFlowOps.invoke*AsyncWithMode() there because converted 4GL code should never trigger this error?

9. Please provide more details behind the TODOs in ProxyProcedureWrapper.delete().

#14 - 08/29/2013 09:13 AM - Constantin Asofiei

- File ca_upd20130829a.zip added

The attached fix solved some bugs found during regression testing and addresses the issues in the review.

1. I couldn't find anything from documentation or testing which would suggest that keys/server events are processed in a certain way or chained in a certain way. Idea is, when a server-event is posted, it will be chained in the event queue, but there is no guarantee where will be chained relative to other non-server events. 4GL enforces the server-event order only relative to other server-events, and this only in Session-managed case; in truly-async requests (Session-free case), the order of the server-events is undefined.
   The downside would be that a constant key press would delay server event processing. TypeAhead.getKeyStroke was coded this way because the server events are pushed in a global queue, separated from the UI/key events, as server-events can be posted even when a WAIT-FOR-like loop is not in effect.
2. Is OK to sync on wa.serverEvents, but the wa lock must be used for the other cases (as syncronized access is needed to wa.stack and wa.queue and operations on them need to be atomic).
3. In an EventList or EventDefinition, an "exit condition" is a pair of (event, widget or resource ID) used to terminate a WAIT-FOR statement.
4. Everything I've tried in 4GL when using these events outside of their goal ended up in abnormal behavior. I've added notes about this to the ServerEvent class.
5. In part is related to 4, but one of the reasons was that widgets on client-side have their own unique IDs which are computed different from their associated resource ID on server-side. Thus, mixing client-side widget IDs and resource IDs would not allow to differentiate which is which and the amount of changes in client-side triggered by this is pretty large. Also, for the same reason, a widget referenced via a handle in a i.e. "wait-for close of h" statement will register it in the set of widget IDs, and not resources.
   The other reason was that P2J uses a stack of UI event-queues which is empty when no wait-for loop is running; thus, server-events can not be posted there because they will get discarded if the event-queue is "trashed".
6. when cancel-requests is executed, only the not-yet-executed requests are marked CANCELLED; the running one is just terminated via STOP (I was expecting for this to mark as cancelled the running request, but this was not the case). STOP, QUIT, ERROR attributes plus the CANCEL-REQUESTS and CANCEL-REQUESTS-AFTER are implemented.
7. Yes, a zero or negative value will stop the current timer.
8. Yes, each time CANCEL-REQUESTS-AFTER is called, it cancels the previous timer before setting up the new one.
9. This was added in P2J as a protection to not allow async calls which do not specify the parameter modes (as in P2J they are optional in some cases). But actually, once the internal-entry/procedure was resolved, I can pick up the parameter modes from the name_map.xml as the default. I've fixed this now.
10. Forgot to add the error for procedure deletion when there are active async requests; it's added now.

#15 - 08/30/2013 01:40 PM - Constantin Asofiei

ca_upd20130829a.zip has passed regression testing.

#16 - 09/02/2013 05:21 PM - Greg Shah

Code Review 0829a

Everything looks good.

1. Please add javadoc to explain the details in note 14 answers 1, 3, 5, 6 and 8.
2. Then commit the change and distribute it.

#17 - 09/03/2013 12:10 PM - Constantin Asofiei

- File ca_upd20130903a.zip added

Attached update is committed to bzr revision 10378.

#18 - 09/03/2013 12:11 PM - Greg Shah

- % Done changed from 0 to 100

- Status changed from WIP 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_upd20130822a.zip	490 KB	08/22/2013	Constantin Asofiei
ca_upd20130827a.zip	513 KB	08/27/2013	Constantin Asofiei
ca_upd20130829a.zip	515 KB	08/29/2013	Constantin Asofiei
ca_upd20130903a.zip	516 KB	09/03/2013	Constantin Asofiei