User Interface - Support #6856

user interface tests

10/18/2022 06:51 PM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Marian Edu	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Subtasks:

Support # 7337: UI - Events	**New**
Support # 7338: UI - Layout and Sizing	**New**
Support # 7339: UI - Focus	**New**

History

#1 - 11/02/2022 01:55 PM - Greg Shah

We need testcases that explore the following as comprehensively as possible. The intention is to provide complete coverage of the 4GL compatibility of the "user interface" support.

There are multiple problems to be considered for this set of tests.

- interactive vs non-interactive
  - as much as possible, we want to leverage non-interactive tests since these can be most easily automated
  - non-interactive tests can be implemented with ABLUnit
  - there is no substitute for some amount of interactive tests
  - interactive tests will be implemented with Sikuli
- ChUI vs GUI
  - most UI features can be executed in both "modalities", with some slight differences in behavior
  - unless a feature is unambiguously impossible to execute in both modalities, we will implement tests such that they can be run twice (once in ChUI and another time in GUI)
- dynamic and static
  - widget/frame/window creation should be tested in both dynamic and static forms

Testcases Needed

- All Widget Types
  - all built-in 4GL widgets
    - data usage
      - each widget should have tests for all valid data types that it supports
      - both variable and field lvalues should be tested
      - complex expressions where supported (e.g. in DISPLAY statements)
    - widgets
      - BROWSE including columns and cells
      - BUTTON
      - COMBO-BOX
      - DIALOG-BOX
      - FIELD-GROUP
      - FILL-IN
      - FRAME
      - EDITOR
      - IMAGE
      - LITERAL
      - MENU
      - MENU-ITEM
      - RADIO-SET
      - RECTANGLE
      - SELECTION-LIST
      - SLIDER (note: ChUI support is missing)
      - SUB-MENU

            - TEXT
            - TOGGLE-BOX
            - WINDOW
        - OCX replacements
            - as a 4GL widget
            - as an OCX control using CONTROL-FRAME and COM properties/methods
            - TYPE reports correct value for a 4GL widget as well as OCX converted control
            - controls
                - BUTTON-LIST (SSListBar)
                - CALENDAR
                - HTML-BROWSER
                - IMAGE-LIST
                - MSGBLASTER
                - Progress Indicator
                - PSTIMER
                - SPREADSHEET
                - TAB-SET
                - TREEVIEW
                - TREELIST
        - System UI elements
            - Scrolling UI elements (including scroll context menus)
            - Tooltips where applicable
- Window Support
    - default window
    - current window
    - moving
    - sizing
    - modal windows/dialogs
    - message boxes (including info, warning, error)
    - multi-window operations including focus handling and activation/deactivation
    - decorations including min/max/restore, titlebar, the system menu and window border
    - message lines
    - status lines
    - z-order (including TOP-ONLY)
- Old School Statements
    - BELL
    - CHOOSE (both row and field modes)
    - CLEAR
    - DISPLAY
    - DOWN
    - EDITING blocks
    - ENTERED function
    - GO-PENDING function
    - HIDE
    - MESSAGE
    - PAUSE
    - PROMPT-FOR
    - PUT CURSOR
    - PUT SCREEN
    - READKEY
    - SCROLL
    - SET
    - STATUS
    - UNDERLINE
    - UP
    - UPDATE
    - VIEW
- Event Based UI
    - Triggers
        - ON statement
        - TRIGGERS phrase
    - APPLY
    - DISABLE
    - ENABLE
    - ENTERED function
    - NEXT-PROMPT
    - PROCESS EVENTS
    - WAIT-FOR
    - SET-WAIT-STATE method
    - nested WAIT-FOR
    - default processing of events
- Screen Buffers
    - ASSIGN statement
    - DISPLAY statement
    - FRAME-VALUE function and statement

    - INPUT function
    - INPUT-VALUE attribute
    - SCREEN-VALUE attribute
- Down Frames
- Nested Frames
- Key and Mouse Input (including all mouse buttons and scroll wheel)
- Format Strings
- Validation
- Focus
- Mouse hover highlighting
- Layout and Sizing
- Drag & Drop where applicable including related events
- Accelerator keys and mnemonics
- Built-In Functions and Other Helpers
    - CURRENT-RESULT-ROW
    - FRAME-COL
    - FRAME-DB
    - FRAME-DOWN
    - FRAME-FIELD
    - FRAME-FILE
    - FRAME-INDEX
    - FRAME-LINE
    - FRAME-NAME
    - FRAME-ROW
    - IS-ATTR-SPACE
    - KBLABEL
    - KEYCODE
    - KEYFUNCTION
    - KEYLABEL
    - LASTKEY
    - LIST-EVENTS
    - LIST-QUERY-ATTRS
    - LIST-SET-ATTRS
    - LIST-WIDGETS
    - RGB-VALUE()
    - SCREEN-LINES
    - TERMINAL built-in function and statement
- @-base fields
- Accumulators
    - ACCUM function
    - ACCUMULATE statement
    - all aggregate phrase types
    - integration into DISPLAY and down frame processing
- Graphics Resources
    - loading, unloading, sizing, manipulation
    - icons
    - images
- Colors
- Fonts
- Widget Pools and Resource Lifetimes
    - CREATE <widget>
    - CREATE WIDGET-POOL
    - DEFINE *
    - DELETE WIDGET
    - DELETE WIDGET-POOL
- Redirected Terminal (we have a start on these tests in the testcases/uast/redirected* and testcases/uast/io/*)
- SYSTEM-HELP
- System Dialogs
    - SYSTEM-DIALOG COLOR
    - SYSTEM-DIALOG FONT
    - SYSTEM-DIALOG GET-DIR
    - SYSTEM-DIALOG GET-FILE
    - SYSTEM-DIALOG PRINTER-SETUP (note: only partial support is available in FWD)
- COM Automation
- Direct Manipulation
    - SELECTABLE, MOVABLE, RESIZABLE widget attributes
    - Global and frame-only direct manipulation events
- Widget enumeration (FIRST-CHILD and related)
- FRAME and widgets z-order
- Web GUI specific
    - Session life time (for example the web sessions won't time out unexpectedly)
    - Session resiliency (for example temporary network errors)
    - File upload
    - Browser window resize and related events

I expect these tests to be split into smaller functional groupings that can be run on their own. I don't expect a single set of tests which includes all of these categories.

#2 - 03/15/2023 09:20 AM - Hynek Cihlar

The items with the highest priorities are (in the order from highest):

- Event Based UI
- Layout and Sizing
- Focus
- Window Support
- Screen Buffers
- All Widget Types
- Old School Statements

#3 - 04/26/2023 10:10 AM - Greg Shah

- Assignee set to Marian Edu

#4 - 06/07/2023 09:35 AM - Greg Shah

For details on using Sikuli, see Automating GUI Testing.

Let's discuss the issues that the Acorn team has found and come up with solutions. Roger will help with this since he is our leading Sikuli expert. If we learn things that are not in the wiki page, please add them as we go.

#5 - 06/08/2023 08:19 AM - Greg Shah

Additional items to discuss:

- Should we use ABLUnit for tests that also need Sikuli or the Harness to handle the interactive parts? If so, how do we launch and manage Sikuli/Harness from ABLUnit?
- How can we launch a FWD web client from ABLUnit?
- What would be needed to use our same testing approach on OE as well as FWD? There is a value to doing this, though it is not an absolute requirement. If we do rely upon anything specific to FWD, that might be OK but it would probably mean that using these tests under OE would not be possible.
- What problems have Acorn found with Sikuli and how can we resolve these?

#6 - 08/07/2023 09:50 AM - Vladimir Tsichevski

Greg Shah wrote:

> Additional items to discuss:
>
> - Should we use ABLUnit for tests that also need Sikuli to handle the interactive parts?

IMO we should not. ABLUnit and Sikuli are two approaches, which never intersect. ABLUnit is pure 4gl, which mean there is no way to reach Sikuli API from ABLUnit. Sikuli approach works for any GUI application on any platform, and it cannot use 4gl.

So we should use one or another approach depending on the problem we are dealing with, but never try to make a hybrid of the two.

The JUnit5 is quite natural to orchestrate Sikuli tests.

> If so, how do we launch and manage Sikuli from ABLUnit?

IMO we can not and need not doing this.

- What would be needed to use our same testing approach on OE as well as FWD? There is a value to doing this, though it is not an absolute requirement. If we do rely upon anything specific to FWD, that might be OK but it would probably mean that using these tests under OE would not be possible.

With Sikuli this problem does not exist. You can run the same tests (some parametrization is required though) with FWD and "native" OE provided Sikuli can access the OE screen.

- What problems have Acorn found with Sikuli and how can we resolve these?

I'd like to know this either. Probably, I have some solutions already.

#7 - 08/07/2023 10:19 AM - Marian Edu

Vladimir Tsichevski wrote:

- Should we use ABLUnit for tests that also need Sikuli to handle the interactive parts?

> IMO we should not. ABLUnit and Sikuli are two approaches, which never intersect. ABLUnit is pure 4gl, which mean there is no way to reach Sikuli API from ABLUnit. Sikuli approach works for any GUI application on any platform, and it cannot use 4gl.

There was no plan to use 4gl from Sikuli, more likely we see Sikuli just like a surogate for user interaction - so just input, no real tests done on Sikuli part.

> So we should use one or another approach depending on the problem we are dealing with, but never try to make a hybrid of the two.

I have nothing against this approach, for UI tests that require user interaction we can only build the screen and then do the testing in Sikuli. This is more or less what one customer is doing with the cucumber/gerkin dsl & Sikuli but this is for testing real application not writing simple 'unit-tests' that

happens to require some user interaction.

If so, how do we launch and manage Sikuli from ABLUnit?

IMO we can not and need not doing this.

The idea was something down that line... inside the test method start sikuli script that waits for something to show, then we start the user interface and the wait-for on it will block until the UI is closed. This will be done by the Sikuli script that will need to do a series of action, last of witch will be to close the UI screen. After that the test will resume and we can check the outcome (some output parameter on the UI screen).

- What would be needed to use our same testing approach on OE as well as FWD? There is a value to doing this, though it is not an absolute requirement. If we do rely upon anything specific to FWD, that might be OK but it would probably mean that using these tests under OE would not be possible.

With Sikuli this problem does not exist. You can run the same tests (some parametrization is required though) with FWD and "native" OE provided Sikuli can access the OE screen.

- What problems have Acorn found with Sikuli and how can we resolve these?

I'd like to know this either. Probably, I have some solutions already.

We've only scratched the surface with Sikuli (the Java API) and issues we're mostly caused by the OCR - seems to be better to narrow down the region for text to be recognised and then there are some issue there, the font and size can play a role and also similar letters (radio set items can also get on 'O' in front of each items sometimes).

But, as said I think we're not going to invest more time on this approach with Sikuli - if tests are to be written in Sikuli then we can stick to just provide the UI screens and the test scenarios as Word documents. I will give this new idea a try and see if we can continue to use ABLUnit for testing and use Sikuli only to perform user actions but do not test the outcome inside Sikuli.

This might work with Swing client (gui/chui) although I don't know if the FWD implementation of ABLUnit will allow the wait-for in the first place. For Web client I think the only approach will be to write Sikuli tests against an application like 'hotel' maybe or create one specifically for testing, have that running on a server and run Sikuli tests on the client with that app open in the web browser. For one application for instance they always started from the login screen, so new web browser window, do the login and navigate to the screen to test as the first steps of Sikuli test.