User Interface - Bug #6267

StackOverflowError in BROWSE widget

04/08/2022 02:24 PM - Vladimir Tsichevski

Status:	Review	**Start date:**	
Priority:	High	**Due date:**	
Assignee:	Sergey Ivanovskiy	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

History

#1 - 04/08/2022 02:27 PM - Vladimir Tsichevski

- File browse.p added

Stack overflow situation in Browse widget:

1. create an application with an editable BROWSE widget (example browse.p attached).
2. click a browse cell to open cell editor
3. press BACK-TAB or CURSOR-UP or CURSOR-DOWN.

The client application crashes with stack overflow.

I do not know if this is a regression or not.

Analysis for BACK-TAB:

1. FillInGuiImpl receives key event and passes it to getBrowse().processKeyEvent()
2. BrowseGuiImpl.processKeyEvent passes this to current, which is a ColumnSetContainer:

```
  if (current != null && current != this)
         current.processKeyEvent(event);
```

3. ColumnSetContainer.processKeyEvent passes this to current, which is the same FillInGuiImpl.

Also, FillInGuiImpl.processKeyEvent has code, which IMO makes no sense: super.processKeyEvent(ke) is called twice in a row:

```
public void processKeyEvent(KeyInput ke)
   {
      super.processKeyEvent(ke);
      ^^^^^^^^^^^^^^^^^^^^^^^^^

      // Do not skip widget that is not enabled, if focus landed on such a widget there would be
      // no way (no key event) to move focus out. Focus can be set to a non-enabled widget with
      // setFocus method in NativeAPIEmulation.
      if (ke.isConsumed())
      {
         return;
      }

      @SuppressWarnings("unchecked")
      Frame<O> frame = (Frame<O>) UiUtils.locateFrame(this);

      if (frame == null)
      {
          return;
      }
      else if (frame.isInChoose(this))
      {
```

```
            frame.processKeyEvent(ke);
            return;
        }

        // First call all KeyListener objects that may have been registered
        // for this component.
        super.processKeyEvent(ke);
        ^^^^^^^^^^^^^^^^^^^^^^^^^^

        // Check if any of the KeyListeners consumed the KeyEvent.
        if (ke.isConsumed())
        {
            return;
        }
```

#3 - 04/08/2022 02:28 PM - Vladimir Tsichevski

This issue was first published as #5034-1982.

#4 - 04/08/2022 05:38 PM - Greg Shah

Vladimir: Please use bzr blame and report who wrote the code in question.

#6 - 04/14/2022 03:13 PM - Stanislav Lomany

- Assignee set to Stanislav Lomany

#7 - 04/14/2022 04:34 PM - Stanislav Lomany

Vladimir, I cannot help saying that the issue you was analyzing is caused by your change 13716 (3821c). The change is pretty straightforward:

```
-          id = (key > 255 || key < ' ') ? EventType.KEY_PRESSED :
-                                          EventType.KEY_TYPED;
+          id = Keyboard.isPrintableKey(key, isReal)
+                   ? EventType.KEY_TYPED
+                   : EventType.KEY_PRESSED;
```

If we take "arrow up" key with code 501 it was going EventType.KEY_PRESSED before the change and EventType.KEY_TYPED after the change. The problem here is that "arrow up" evaluates as a "printable key".

Greg, could you provide key-related guidance? To me, more correct version of the condition looks like Keyboard.isPrintableKey(Keyboard.keyAction(key), isReal). But there's a related TODO left by someone above:

```
//TODO: fix character check here, at present it handles only ASCII
//Perhaps Character.isValidCodePoint(key) + some additional check may
//help us filter out correct keys.
```

#8 - 04/14/2022 05:34 PM - Greg Shah

Stanislav Lomany wrote:

> Vladimir, I cannot help saying that the issue you was analyzing is caused by your change 13716 (3821c). The change is pretty straightforward:
> [...]
> If we take "arrow up" key with code 501 it was going EventType.KEY_PRESSED before the change and EventType.KEY_TYPED after the change. The problem here is that "arrow up" evaluates as a "printable key".
>
> Greg, could you provide key-related guidance? To me, more correct version of the condition looks like Keyboard.isPrintableKey(Keyboard.keyAction(key), isReal). But there's a related TODO left by someone above:
> [...]

I don't have the answer.

Hynek/Sergey: What do you think?

#9 - 04/14/2022 05:35 PM - Vladimir Tsichevski

Stanislav Lomany wrote:

> Vladimir, I cannot help saying that the issue you was analyzing is caused by your change 13716 (3821c). The change is pretty straightforward:

I see. But how comes BACK-TAB key is printable in FWD, and TAB is not?

#10 - 04/14/2022 05:37 PM - Vladimir Tsichevski

Vladimir Tsichevski wrote:

> Stanislav Lomany wrote:
>
> > Vladimir, I cannot help saying that the issue you was analyzing is caused by your change 13716 (3821c). The change is pretty straightforward:
>
> I see. But how comes BACK-TAB key is printable in FWD, and TAB is not?

And, in any case, the program should be guarded against unlimited recursion.

#11 - 04/14/2022 06:32 PM - Sergey Ivanovskiy

I would propose to fix Keyboard.isPrintableKey(key, isReal) to limit this condition
Character.isAlphabetic(key) && !isFunctionKey(key) to Character.isAlphabetic(key) && !isFunctionKey(key) && !isNavigationKey(key) && !isSpecialKey(key)?

#12 - 04/14/2022 07:00 PM - Vladimir Tsichevski

Sergey Ivanovskiy wrote:

> I would propose to fix Keyboard.isPrintableKey(key, isReal) to limit this condition
> Character.isAlphabetic(key) && !isFunctionKey(key) to Character.isAlphabetic(key) && !isFunctionKey(key) && !isNavigationKey(key) && !isSpecialKey(key)?

Agreed, seems, the problem is in this function. But other problem with recursion should be addressed either somehow IMO. The code is too fragile.

#13 - 04/15/2022 05:40 AM - Sergey Ivanovskiy

Committed revision 13785 (3821c) please review.

#14 - 04/15/2022 06:19 AM - Stanislav Lomany

> Committed revision 13785 (3821c) please review.

Sergey, Character.isAlphabetic states that the following keys are considered to be alphabetical:

- UPPERCASE_LETTER
- LOWERCASE_LETTER
- TITLECASE_LETTER
- MODIFIER_LETTER
- OTHER_LETTER
- LETTER_NUMBER
- or it has contributory property Other_Alphabetic as defined by the Unicode Standard.

An arrow doesn't seem to fall into any of these categories. So it should return true for an arrow. My theory is that we're feeding a wrong key code to Keyboard.isPrintableKey.
If you have a look at Keyboard.isPrintableKey usages, it receives KeyInput.actionCode() which is Keyboard.keyAction(key) rather than the key itself.
Your thoughts?

#15 - 04/15/2022 06:30 AM - Stanislav Lomany

> And, in any case, the program should be guarded against unlimited recursion.

I would say the keys in the browse have been working fine so far. I don't want to introduce something that may have a new impact if there's no need and not clear how to test it.

#16 - 04/15/2022 06:34 AM - Stanislav Lomany

Igor, please take a look at the code which Vladimir posted in #6267-1. We have double super.processKeyEvent(ke) call which is weird. The top call was presumably added later by you (pretty long ago). I don't know which call should I remove. Any advice?

Constantin, any advice from you on this double calls topic?

#17 - 04/15/2022 08:09 AM - Sergey Ivanovskiy

Stanislav Lomany wrote:

> An arrow doesn't seem to fall into any of these categories. So it should return true for an arrow. My theory is that we're feeding a wrong key code to Keyboard.isPrintableKey.
> If you have a look at Keyboard.isPrintableKey usages, it receives KeyInput.actionCode() which is Keyboard.keyAction(key) rather than the key itself. Your thoughts?

I found that this unicode symbol ģ has decimal code 501. It seems that my changes are correct.

#18 - 04/15/2022 08:51 AM - Stanislav Lomany

> I found that this unicode symbol ģ has decimal code 501. It seems that my changes are correct.

Right, but we don't type ģ, we press arrow up. Shouldn't arrow up code be Keyboard.KA_CURSOR_UP = -119 and not 501?

#19 - 04/15/2022 08:53 AM - Stanislav Lomany

So it looks like you excluded ģ and some other chars from the printable range.

#20 - 04/15/2022 09:17 AM - Sergey Ivanovskiy

Stanislav, it seems that KeyInput is created from TypeAHead with help of EventManager.eventFromKey. So Keyboard.isPrintableKey(key, isReal) is used first for creating KeyInput event and this method Keyboard.isPrintableKey(KeyInput) is used after the KeyInput has been created.

#21 - 04/15/2022 09:18 AM - Sergey Ivanovskiy

Stanislav Lomany wrote:

> So it looks like you excluded ģ and some other chars from the printable range.

Yes.

#22 - 04/15/2022 09:57 AM - Stanislav Lomany

> Stanislav, it seems that KeyInput is created from TypeAHead with help of EventManager.eventFromKey. So Keyboard.isPrintableKey(key, isReal) is used first for creating KeyInput event and this method Keyboard.isPrintableKey(KeyInput) is used after the KeyInput has been created.

Right. Keyboard.isPrintableKey(KeyInput evt) passes action code evt.actionCode() to isPrintableKey(int key) and we can returning to my suggestion above to use Keyboard.isPrintableKey(Keyboard.keyAction(key), isReal) in the discussed code.

#23 - 04/15/2022 11:05 AM - Sergey Ivanovskiy

In our system the key codes from 501 to 509 belong the the navigation key set and if KeyCode belongs this range, then one of navigation keys has been pressed.

#24 - 04/15/2022 01:17 PM - Stanislav Lomany

Consider we have a KeyInput with KeyInput.key = 501 and KeyInput.actionCode = -119. I'm telling that it looks like isPrintableKey takes KeyInput.actionCode and we're feeding KeyInput.key to it. To me, you fix is trying to filter actionCode using the space of key values.

#25 - 04/15/2022 01:36 PM - Sergey Ivanovskiy

Stanislav Lomany wrote:

> Consider we have a KeyInput with KeyInput.key = 501 and KeyInput.actionCode = -119. I'm telling that it looks like isPrintableKey takes KeyInput.actionCode and we're feeding KeyInput.key to it. To me, you fix is trying to filter actionCode using the space of key values.

For this usage my changes look consistent

```
   private static KeyInput getKeyEvent(int key, Widget<?> src, boolean isReal, boolean isReleased)
   {
      //TODO: fix character check here, at present it handles only ASCII
      //Perhaps Character.isValidCodePoint(key) + some additional check may
      //help us filter out correct keys.
      int id;
      if (isReleased)
      {
         id = EventType.KEY_RELEASED;
      }
      else
      {
         id = Keyboard.isPrintableKey(key, isReal)
                  ? EventType.KEY_TYPED
                  : EventType.KEY_PRESSED;
      }

      return new KeyInput(src, id, key, false, isReal);
   }
```

because actionCode value of events can be changed if new event will be added to evNames1 (This is possible, for an example, PEN_DOWN, PEN_UP have been added recently). For me this code

```
   public static boolean isPrintableKey(KeyInput evt)
   {
      return isPrintableKey(evt.actionCode(), evt.isRealKey());
   }
```

doesn't make sense. We should consult the author of this code.

#26 - 04/15/2022 01:49 PM - Sergey Ivanovskiy

It seems that this function should be implemented using evt.keyCode()

```
   /**
    * Determine if the given key is a printable key.
    *
    * @param    evt
    *           The event to test.
    *
```

```
    * @return   true if the key is in the [0x20, 0xff] interval.
    */
   public static boolean isPrintableKey(KeyInput evt)
   {
      return isPrintableKey(evt.keyCode(), evt.isRealKey());
   }
```

#27 - 04/15/2022 02:37 PM - Stanislav Lomany

- Status changed from New to WIP

OK, now we're on the same page.

Greg, I checked usages of isPrintableKey(int key) and half of them pass KeyInput.key as the parameter and the other half - KeyInput.actionCode. These are different codes linked by actionCode = Keyboard.keyAction(key). Sergey disagrees with me, but I think isPrintableKey is written as if takes actionCode. I don't know what it originally supposed take, though. Your thoughts?

#28 - 04/15/2022 02:46 PM - Greg Shah

Unfortunately, Nick Saxon is the original author of the majority of the Keyboard class, including the key action/function processing. He left GCD many years ago, so we have no way to consult him.

Constantin was heavily involved with the isPrintableKey() work, so he may have some thoughts on this. He will be available tomorrow to respond.

#29 - 04/15/2022 04:57 PM - Vladimir Tsichevski

Greg Shah wrote:

> Vladimir: Please use bzr blame and report who wrote the code in question.

```
                     |    @Override
                     |    public void processKeyEvent(KeyInput ke)
                     |    {
10895       ias@gol  |       super.processKeyEvent(ke);
                     |
```

#30 - 04/16/2022 09:06 AM - Constantin Asofiei

A problem with Keyboard.isPrintable originates from having to distinguish between characters being pressed by the user, at the keyboard, and events raised by the application, like 'APPLY'.

My understanding is that in OE, APPLY can work with:

- an event name - this can actually be the 'key function' for events like GO, HELP, or otherwise the label for the event; see the testcases/uast/keyboards/win-gui.txt for the full list of codes.
- a printable key code: this key code is not a real unicode key code, and from my testing looks like only ASCII chars can be applied - something greater than 1 byte can make OE unstable. For example, doing a apply 8364 to ch, where ch is a FILL-IN, will be a no-op in 4GL (or maybe abend). But in FWD, the euro unicode char € is applied (this is incorrect and should be fixed).

So, for an application-driven event, we look for a printable char with the actual key code resolved from the APPLY (which can be explicitly set or otherwise computed from the event name).

Keep in mind that KeyInput.key has the actual key pressed by the user, and KeyInput.actionCode is the computed value in OE terms. For example, a CURSOR-RIGHT key press will have a 503 code from the OS, but the action code will be -118. So that's why we need to use the KeyInput.actionCode.

The stack overflow for the browse.p program is that FWD sees the CURSOR-UP as a printable key (incorrectly as 501 key is printable in unicode), and for these we have KEY_TYPED in EventManager.getKeyEvent:

```
         id = Keyboard.isPrintableKey(key, isReal)
                  ? EventType.KEY_TYPED
                  : EventType.KEY_PRESSED;
```

The problem is not with the double super.processKeyEvent(ke); (which I can't explain at this time why it was added). The fact that FWD misrepresents the key as typed, goes into a different code path (as there are different methods in KeyListener, onKeyTyped and onKeyPressed).

Stanislav, the only place where Keyboard.isPrintableKey has a OS-generated key code (and not an action code) is from EventManager.getKeyEvent. This is the part which can give an incorrect result, as the OS code can collide with a unicode printable char, and as we found, the cursor keys must be excluded.

Greg: to solve this for good, I think we need to go through all the 0 to 4095 codes, create the list of printable codes in unicode (Character.isAlphabetic(key)), and after that compare this with the 4GL key code mappings: I think if they are mapped with a 'Key Function' in the testcases/uast/keyboards/win-gui.txt list, then we should explicitly exclude them in Keyboard.isPrintable for real keys. But it should be confirmed with testcases.

#31 - 04/17/2022 03:45 PM - Sergey Ivanovskiy

Constantin, the decimal code 8364 for the euro sign is just a representation of the euro sign but its key code can be mapped on 32 to 255 depending on the code page. For Dutch(Netherland) with WINDOWS-1252 codepage the decimal key code for the euro sign is 128 (0x80).

#32 - 04/17/2022 03:52 PM - Sergey Ivanovskiy

So if you would apply APPLY 128 TO ch. using codepage Windows-1252, then you got the euro sign in the target field.

#33 - 04/18/2022 02:57 AM - Sergey Ivanovskiy

It can be more clear if someone run this java program

```
        Charset win1252 = Charset.forName("WINDOWS-1252");
        String euro1252 = new String(new byte[] {(byte) 0x80}, win1252);
        System.out.println("euro1252=" + euro1252);
        System.out.println("euro1252.codePointAt(0)=" + euro1252.codePointAt(0));
```

Obviously the output should be

```
euro1252=€
euro1252.codePointAt(0)=8364
```

#34 - 04/18/2022 04:49 AM - Stanislav Lomany

Constantin, so we have

- a "key code" which is a number that can be supplied to APPLY or generated by a key press;
- an "action code" which is some FWD-specific number.

Right?

Is isPrintableKey(int key, boolean isReal) supposed to take "key code" rather than "action code"? I see that in different places "key code" as well as "action code" are supplied to this function.

#35 - 04/18/2022 05:04 AM - Constantin Asofiei

Not exactly. The only place in FWD where isPrintableKey receives the key code generated as it was typed by the user at the keyboard is in EventManager.eventFromKey which calls EventManager.getKeyEvent, and that calls Keyboard.isPrintableKey. In all other places, the 'action code' is received (which, for a 'real' printable key, is the same as the KeyInput.key as far as I can tell). If you see other places where isPrintableKey receives the code generated by a generated key press, please specify.

Also, the 'action code' is a FWD specific number, as you state, which maps certain key codes to 'actions' (like cursors, GO, HELP, etc - all the Keyboard.KA_* constants). These are negative values in FWD, and AFAIK key codes are always positive numbers.

Maybe we should have a separate isPrintableKey for the case when the user-generated key code is interpreted. And leave the current one for all the

other cases.

About the euro sign: you are right, pressing CTRL-ALT-5 generates the euro sign in OE (with dutch keyboard and win1252 codepage - use prowin -cpstream 1252 -cpinternal 1252), but in FWD (with a standalone testcase) this doesn't work, even if I set cpinternal to 1252 in the directory.

#36 - 04/18/2022 05:42 AM - Sergey Ivanovskiy

Constantin Asofiei wrote:

> About the euro sign: you are right, pressing CTRL-ALT-5 generates the euro sign in OE (with dutch keyboard and win1252 codepage - use prowin -cpstream 1252 -cpinternal 1252), but in FWD (with a standalone testcase) this doesn't work, even if I set cpinternal to 1252 in the directory.

This is not related to #6267. Working on this issue within #6018 and #6153.

#37 - 04/18/2022 05:44 AM - Stanislav Lomany

> If you see other places where isPrintableKey receives the code generated by a generated key press, please specify.

There're many such places: Editor/FillIn/SelectionListBody/BrowseGuiImpl/DropDownGuiImpl etc. use the following path: processKeyEvent(KeyInput ke) -> Keyboard.isPrintableKey(ke) -> isPrintableKey(evt.actionCode(), evt.isRealKey()).

#38 - 04/19/2022 09:28 AM - Greg Shah

Why not separate these cases and do the safe thing at this point? Getting rid of the regression is important.

#39 - 04/19/2022 09:30 AM - Greg Shah

What I mean by separate, is create a replacement for isPrintableKey() for the user-typed key. It should be named differently so that its purpose is clear.

#40 - 04/19/2022 10:43 AM - Constantin Asofiei

Greg Shah wrote:

> Getting rid of the regression is important.

The fix from 3821c/13785 solves the regression.

> What I mean by separate, is create a replacement for isPrintableKey() for the user-typed key. It should be named differently so that its purpose is clear.

I agree, a replacement to be called by EventManager.getKeyEvent, when isReal flag is true. But what I noted before still applies: there may be other combinations of OS key codes which conflict 'printable' characters from Character.isAlphabetic, which will need to be handled explicitly, as we do for function keys and cursors.

Stanislav: for the code paths you mentioned, at the time processKeyEvent is called, the KeyInput holds the 'processed' key in actionCode, this is not the OS generated key code.

#41 - 04/20/2022 03:45 AM - Stanislav Lomany

> I agree, a replacement to be called by EventManager.getKeyEvent, when isReal flag is true.

Constantin, are you saying that is the only place when isReal flag is actually true?

> But what I noted before still applies: there may be other combinations of OS key codes which conflict 'printable' characters from Character.isAlphabetic, which will need to be handled explicitly, as we do for function keys and cursors.

Greg, should I check all the other keys?

#42 - 04/20/2022 03:58 AM - Constantin Asofiei

Stanislav Lomany wrote:

> > I agree, a replacement to be called by EventManager.getKeyEvent, when isReal flag is true.
>
> Constantin, are you saying that is the only place when isReal flag is actually true?

No, what I'm saying is this is the only place where the key code is the real OS-generated key code, and not the preprocessed one.

The KeyInput.isRealKey() will still return true, for real user key presses, but at that time the KeyInput.actionCode is no longer the OS-generated key code.

#43 - 04/20/2022 07:44 AM - Greg Shah

> Greg, should I check all the other keys?

It is worth checking the obvious cases that can be generated by Firefox/Chrome with your keyboard. Perhaps there are some keys like Home, End or ESC which we don't handle properly.

The real place I expect issues is from Sergey's work in #6018 and #6153 where I18N layouts, dead keys and other support will cause problems, possibly different results from Chrome vs Firefox. We will let him handle those differences.

#44 - 04/20/2022 07:40 PM - Stanislav Lomany

> It is worth checking the obvious cases that can be generated by Firefox/Chrome with your keyboard. Perhaps there are some keys like Home, End or ESC which we don't handle properly.

I've checked the keys and found that the following keys probably have issues:

- INSERT - considered as printable. It has code 510 which can be added to the Sergey's isNavigationKey.
- DELETE - considered as printable. It has code 127 and I have to say that it was considered as printable before the last changes: (key > 255 || key < ' ').
- For a change, I typed Cyrillic characters and found that some of them are not accepted because isControlKey -> isInRangeOfKeys "removes" CTRL, ALT, SHIFT modifiers from, say, 1049 code and compares resulting number against a given range.

#45 - 04/20/2022 07:46 PM - Stanislav Lomany

> - DELETE - considered as printable. It has code 127 and I have to say that it was considered as printable before the last changes: (key > 255 || key < ' ').

I mean it is considered as printable *plus* it was considered as typed in the discussed snipped beforehand.

#46 - 04/20/2022 08:14 PM - Stanislav Lomany

> I agree, a replacement to be called by EventManager.getKeyEvent, when isReal flag is true.

Constantin, do you mean to create, say, isPrintableUserKey(key) (which will call isPrintableKey(key, true)) for this specific case? If not, please tell me what you mean so that I don't bother you anymore.

#47 - 04/21/2022 04:02 AM - Sergey Ivanovskiy

Stanislav, I would propose to discuss the following topics:

1. To add a special keyboard layout to handle Cyrillic characters for the web client.
2. It would be correct to take into account the codepage provided by these option -cpstream 1252 -cpinternal 1252 so the extending range of Latin characters [128, 255] could be translated into correct unicode by the java web client. This way supposed to add codepage to the keyboard layouts and perform this transformation new String(new byte[] {(byte) key}, codepage);. In this case only these characters from [32, 255] can be printable. Now we don't associate a keyboard layout with its codepage and for the euro sign the jscript client sends its unicode code instead of its extended Latin code. Another way to permit the java web client to know the correct codepage in order to transform the unicode code to its extended Latin code from [128, 255] set new String(Character.toChars(key)).toBytes(codepage).

#48 - 04/21/2022 05:47 AM - Constantin Asofiei

Stanislav Lomany wrote:

> > I agree, a replacement to be called by EventManager.getKeyEvent, when isReal flag is true.
>
> Constantin, do you mean to create, say, isPrintableUserKey(key) (which will call isPrintableKey(key, true)) for this specific case? If not, please tell me what you mean so that I don't bother you anymore.

Yes, something like this:

- add a new isPrintableUserKey(key) which will be called only for EventManager.getKeyEvent case, when the isReal flag is true. In this case, its implementation will be:

```
         // add required character groups
         return (key >= 0x20 && key <= 0xff)                          ||
                (Character.isAlphabetic(key) && !isFunctionKey(key) && !isNavigationKey(key) && !isControl
Key(key))||
                Character.getType(key) == Character.CURRENCY_SYMBOL ||
                isQuotientOfEightKey(key) || isSingleQuotation(key);
```

- the remaining Keyboard.isPrintableKey maybe can have just return isPrintable(keyLabel(key));, but I'm not sure how keyLabel(key) will work for dutch characters, the euro currency symbol or other cases. So this needs to be tested.

#49 - 04/21/2022 10:44 AM - Stanislav Lomany

What about the cases when isReal flag is true, but it's not called from EventManager.getKeyEvent? Do we preserve isPrintableKey(int key, boolean isReal) version of the function?

#50 - 04/21/2022 10:53 AM - Constantin Asofiei

Stanislav Lomany wrote:

> What about the cases when isReal flag is true, but it's not called from EventManager.getKeyEvent? Do we preserve isPrintableKey(int key, boolean isReal) version of the function?

Theoretically, no, return isPrintable(keyLabel(key)); should be enough, as those places have already the proper actionCode, and this can be checked directly.

But, depending on testing, we may find that only the && !isFunctionKey(key) && !isNavigationKey(key) tests can be dropped from the current isPrintableKey code. I don't recall exactly how FWD behaves later on in the code to give more advice.

OTOH, keep in mind that euro code is 0x80 in codepage 1252, and APPLY 128 works in OE (see #6267-30 and later). As this will not be a 'real key', what will Keyboard.keyLabel(128) return for the euro sign?

#51 - 04/21/2022 05:15 PM - Stanislav Lomany

> Theoretically, no, return isPrintable(keyLabel(key)); should be enough, as those places have already the proper actionCode, and this can be checked directly.

I don't know, majority of the key presses has isReal set to true when they reach isPrintableKey(int key, boolean isReal), so they go return (key >= 0x20 && key <= 0xff) ... branch rather than return isPrintable(keyLabel(key));.

Greg, I don't feel productive in this task. I investigated the issue, it was fixed, also I'll commit the fix for double super.processKeyEvent(ke) because I don't like key listeners being notified twice. The remaining potential issue I suggest to transfer to Sergey or Constantin.

#52 - 04/21/2022 05:51 PM - Stanislav Lomany

> I'll commit the fix for double super.processKeyEvent(ke)

Committed as 3821c rev 13808.

#53 - 04/22/2022 11:31 AM - Greg Shah

- Assignee changed from Stanislav Lomany to Sergey Ivanovskiy

#54 - 04/22/2022 01:36 PM - Sergey Ivanovskiy

Could you point me what is code that is responsible for providing cpinternal and cpstream codepage settings? I would expect that these settings could be provided by EnvironmentOps, but didn't find the related code.

#55 - 04/22/2022 01:39 PM - Greg Shah

Sergey Ivanovskiy wrote:

> Could you point me what is code that is responsible for providing cpinternal and cpstream codepage settings? I would expect that these settings could be provided by EnvironmentOps, but didn't find the related code.

See I18NOps.

#56 - 04/22/2022 01:51 PM - Sergey Ivanovskiy

Thank you. I have found this code I18nOps too.

#57 - 04/22/2022 03:22 PM - Sergey Ivanovskiy

It looks like the code from I18nOps isn't used by Keyboard but String cset = Utils.getCharsetOverride(); is used instead of I18nOps.getCharset() so it needs to convert a unicode code point into this encoding Utils.getCharsetOverride() in order to get correct key labels for the pressed key. It can be done by new String(Character.toChars(key)).getBytes(cset)

#58 - 04/25/2022 11:00 AM - Sergey Ivanovskiy

I cannot answer these two questions. In this code

```
                  // this must be done only if we are in an APPLY call
                  Widget next = frame.getNextEnabledWidget(src, FocusManager.FOCUS_NOWRAP);
                  if (next != null)
                  {
                     applyWorker(new KeyInput(next,
                                              evt.id(),
                                              evt.isRealKey() ? key : action,
                                              evt.isSpecial(),
                                              evt.isRealKey()));
                  }
```

if evt.isRealKey() is false, then the key code is replaced by the action code but this code is not clear because the constructor has these assignments

```
     this.key = key;
     ...........
```

```
    this.actionCode = Keyboard.keyAction(key);
```

If key is actually action code, then it seems that Keyboard.keyAction(key) == key. Does it make sense to create a special constructor that accepts an action code?

The javascript client doesn't know the client internal codepage and sends the unicode codes for extended latin codes, for an example for Ö it sends 0xD6 but for IBM037 it is 0xEC. It seems for me that the java web client should convert to the internal codepage or we must suppose that key is a unicode key point so we need transform APPLY 0xEC TO EDITOR. to key 0xD6. Is it correct? Now we don't use I18nOps.getJavaCharset() for setup basic labels for key codes.

#59 - 04/25/2022 11:58 AM - Greg Shah

Our approach is to use Unicode throughout all internal Java/Javascript code and we only convert to/from the CPINTERNAL, CPSTREAM... when we do input/output operations. For example, we read a text file that is encoded in CPSTREAM. When we read those characters, we know we must interpret them as the encoding defined in CPSTREAM. When they are read into memory, we must convert these from CPSTREAM into Unicode. Then all of our internal operations can operate on Unicode from there.

It is only at the locations where the legacy 4GL would be expected to read or write the specific encoding that we must honor it. If the UI has some dependencies, then we should implement that conversion as close to the input reading as is reasonable.

#60 - 04/25/2022 01:59 PM - Sergey Ivanovskiy

I checked that Apply statement interprets integer expression according to cpinternal code page so APPLY 225 TO EDITOR.(0xE1) for IBM437 codepage appends ß that has 0xDF unicode value. We need to convert this integer expression to unicode value using IBM437 codepage.

#61 - 04/25/2022 02:09 PM - Greg Shah

OK, fix it.

#62 - 04/26/2022 06:36 AM - Sergey Ivanovskiy

If integer value is within [0, 255], then it can be decoded using the current internal codepage given by cpinternal but the other values out of this range: negative integers and > 255 are not encoded by a single byte codepage.
Although 4GL documentation said that APPLY -2. is the same as APPLY ENDKEY. I cannot detect the rule that can be applied here to convert integer to action. Could you help me?

#63 - 04/26/2022 06:41 AM - Sergey Ivanovskiy

It seems that the Keyboard can use unicode codepoints when it initializes basic keys

```
      for (char i = 32; i <= 126; i ++)
      {
         basicKeys[i] = new String(Character.toChars(i));
      }
      for (char i = 128; i <= 255; i ++)
      {
         basicKeys[i] = new String(Character.toChars(i));
      }
```

but APPLY key TO EDITOR. should be converted using the cpinternal codepage.

#64 - 04/26/2022 02:32 PM - Sergey Ivanovskiy

Sergey Ivanovskiy wrote:

> It seems that the Keyboard can use unicode codepoints when it initializes basic keys
> [...]
> but APPLY key TO EDITOR. should be converted using the cpinternal codepage.

I returned to the previous idea that key values represent characters using cpinternal codeset.
Hynek, it seems that you developed I18 support. Could you give me examples of i18n/cpinternal usages in the directory?

#65 - 04/26/2022 02:47 PM - Hynek Cihlar

Sergey Ivanovskiy wrote:

> Sergey Ivanovskiy wrote:
>
> > It seems that the Keyboard can use unicode codepoints when it initializes basic keys
> > [...]
> > but APPLY key TO EDITOR. should be converted using the cpinternal codepage.
>
> I returned to the previous idea that key values represent characters using cpinternal codeset.
> Hynek, it seems that you developed I18 support. Could you give me examples of i18n/cpinternal usages in the directory?

I only worked in the translation department, not much on character encoding :-). But if you want to see the possible values for cpinternal and their mappings to Java code pages, look in I18nOps's static constructor and the block where convmap2JavaDefault is initialized.

#66 - 04/26/2022 03:56 PM - Sergey Ivanovskiy

It is okey. I cannot find how to setup of i18n node in the directory. Is i18n a node of container class?

```
        <node class="container" name="i18n">
          <node class="string" name="cpinternal">
            <node-attribute name="value" value="1252"/>
          </node>
        </node>
```

#67 - 04/27/2022 05:57 AM - Sergey Ivanovskiy

I tested FWD with WINDOWS-1252 and the changes discussed in this thread and found many new issues for Swing and Web clients that related to encoding and keys processing. There were observed white boxes in the editor while key code and label were reported correctly. SHIFT + " (Dead key) causes the Swing client to fail. Some issues are unseen in the current version while the current implementation doesn't take into account cpinternal codeset and uses unicode codepoints. So I didn't commit my changes and investigated why they are still not enough for representing keys printed in Dutch (Netherlands) as an example.

#68 - 04/28/2022 05:39 AM - Sergey Ivanovskiy

I found that the issue is in Utils.toChar(int) that is used for converting a codepoint to a character. If an input value is within [0, 255], then it is supposed that it is a code in the charset given by Utils.getCharsetOverride();, otherwise it is supposed to be a unicode codepoint. It seems it is not consistent.

#69 - 04/29/2022 04:15 AM - Sergey Ivanovskiy

If cpinternal is UTF-8, then 4GL calculates a key codepoint using reverse mapping of bytes to integer in reverse order when the least significant byte becomes the most significant one. For an example, the euro sign (unicode point 8364) is encoded by UTF-8 into sequence of 0xe2, 0x82, 0xac becomes 0xe282ac that is 14844588 in the decimal number system.

#70 - 05/02/2022 06:46 AM - Sergey Ivanovskiy

- Status changed from WIP to Review

- % Done changed from 0 to 100

Constantin, please review the committed revision 13835(3821c).

Files

browse.p	525 Bytes	04/08/2022	Vladimir Tsichevski