User Interface - Bug #7657

8-bit character entry problem in ChUI

08/03/2023 12:18 PM - Greg Shah

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

Description

Related issues:

Related to Base Language - Feature #6431: implement support to allow CPINTERN...	**WIP**

History

#2 - 08/03/2023 12:18 PM - Greg Shah

As reported by one of our customers:

> I did notice one other thing. It occurs both in the Hotel sample app as well as our own app. When running client-terminal.sh, I can only enter 7 bit ASCII values. When running via swing, I can enter 8 bit ISO8859-1 characters like ä.
>
> Both have CPSTREAM as ISO8859-1.

#3 - 02/01/2024 12:08 PM - Robert Jensen

Any news on this? We support customers all over the world and to support UTF8 clients.

#4 - 02/01/2024 12:44 PM - Greg Shah

- Assignee set to Eugenie Lyzenko

#5 - 02/01/2024 01:06 PM - Greg Shah

At a high level, I think we need to implement (at least) these things:

- Support for "wide characters" in NCURSES. There are alternate versions of the ncurses libraries that are compiled with wide character support. We need to ensure we can enable this support and successfully use it with UTF-8 input.
- FWD client support for setting CPINTERNAL and CPSTREAM to UTF-8. Without this we will improperly process the input. The actual setting of these CP values can be done normally in the directory or via the matching bootstrap config startup parameters. I think the issue here is that we may not properly honor these settings today.

#6 - 02/01/2024 01:15 PM - Robert Jensen

I see the display is correct for UTF8 when I put data in the database. But I can't enter these charactersfrom the terminal screen (Swing window) I can't input anything but ISO8859-1 characters. I suspect this is a limitation of the "P2J ChUI CLient", not Golden Code directly. Any advice here?

The client-termimnal.sh script does not display UTF8, only 7 bit ASCII characters.

#7 - 02/01/2024 01:18 PM - Greg Shah

I suspect that item 2 in #7657-5 will address this issue. If not, we will fix the bug.

#8 - 02/02/2024 08:51 PM - Eugenie Lyzenko

The small sample to demonstrate the issue will be great here. Or recreation instruction for Hotel application.

#9 - 02/05/2024 11:57 AM - Robert Jensen

I apologize for my ignorance, but for the Hotel app, where exactly do I set the encoding for the client? I assume it is in directory.xml, but I do not know which section. It does appear to default to ISO8859-1

For the terminal app, I am getting a failure now in terminal start up: "java: symbol lookup error: /home/mfg/projects/hotel/deploy/lib/libp2j.so: undefined symbol: auto_getch_refresh

Swing is ok.

Which may be related to the p2j updates, or I did something wrong in my setup. I think I need to rebuild/recompile the hotel app. Altough we will be running as terminal most likely as we need to get the input/output streams programatically.

I agree that Note 5 does sound like the answer.

#10 - 02/09/2024 06:53 PM - Theodoros Theodorou

Robert Jensen wrote:

> I apologize for my ignorance, but for the Hotel app, where exactly do I set the encoding for the client? I assume it is in directory.xml, but I do not know which section. It does appear to default to ISO8859-1

You can set cpinternal or cpstream through directory.xml using:

```
<node class="container" name="standard">
   <node class="container" name="runtime">
      <node class="container" name="default">
         <node class="container" name="i18n">
            <node class="string" name="cpinternal">
               <node-attribute name="value" value="UTF-8"/>
```

or thorugh client.xml using:

```
<client>
   <cmd-line-option cpinternal="UTF-8"/>
</client>
```

Setting command line parameters through directory.xml is preferred because sometimes some effects are missed through client.xml.

#11 - 02/10/2024 08:13 AM - Greg Shah

- Related to Feature #6431: implement support to allow CPINTERNAL set to UTF-8 and CP936 added

#12 - 02/10/2024 08:20 AM - Greg Shah

> I apologize for my ignorance, but for the Hotel app, where exactly do I set the encoding for the client? I assume it is in directory.xml, but I do not know which section. It does appear to default to ISO8859-1

In addition to Theodoros' comments, anything that goes into the bootstrap configuration (e.g. client.xml) can also be passed at the end of the ClientDriver command line in the form client:cmd-line-option:cpinternal=UTF-8. The client.sh can also take these same parameters at the end of its command line and it will pass them through to the ClientDriver.

The core problem remains in #6431, we know that setting CPINTERNAL to UTF-8 will not fully work properly. We will fix that.

> For the terminal app, I am getting a failure now in terminal start up: "java: symbol lookup error: /home/mfg/projects/hotel/deploy/lib/libp2j.so: undefined symbol: auto_getch_refresh

This means you have not patched ncurses. That is a requirement for the native terminal client. Please see Patching NCURSES Using Static Linking, Patching NCURSES and Patching TERMINFO.

#13 - 02/15/2024 05:46 PM - Robert Jensen

When running my App, I see that Unicode is supported. I put in Unicode (beyond 8 bit) into Item Descriptions in Maria DB

client-swing.sh will show the characters. But I can not enter them through the terminal.
client-terminal.sh will not show or enter anything beyond 7 bit ASCII.

So I suspect the problem is in my terminal setup. I did try setting cpinternal:UTF-8 in when calling client-swing.sh, it did nothing.
directory.xml already has cpinternal as UTF-8.

I think the problem is in the client-swing and client-terminal scripts themselves.
An oddity here. I can enter the 8 bit ISO characters such as ä in client-swing. I suspect the client script is ISO8859-1 and something is converting the character to UTF-8 as they are entered. Encoding is always difficult.

#14 - 02/18/2024 01:13 PM - Greg Shah

> client-swing.sh will show the characters. But I can not enter them through the terminal.
> client-terminal.sh will not show or enter anything beyond 7 bit ASCII.
>
> So I suspect the problem is in my terminal setup. I did try setting cpinternal:UTF-8 in when calling client-swing.sh, it did nothing.
> directory.xml already has cpinternal as UTF-8.

No, you are not doing anything wrong. The items in #7657-5 need to be implemented before it will work. The reason you get further with Swing is that our Swing client is not dependent upon NCURSES. The ncurses changes are absolutely needed for this to work.

#15 - 03/28/2024 05:45 PM - Robert Jensen

I fianally got around to trying the ncurses update. No change. The setup_ncurses6x.sh script did appear to work correctly. The.bashrc file has the "export NCURSES_FWD_STATIC=/home/mfg/ncurses/ncurses-6.3" line in it. But the terminal output is unchanged. I see "MM" trash in fields where a valid 8 bit character appears. The startup is:

./client-terminal.sh client:cmd-line-option:startup-procedure=com/qad/qra/core/ClientBootstrap.p
client:cmd-line-option:parameter="cpinternal:UTF-8,cpstream:UTF-8,startup=mf.p,mfgwrapper=true"

Also, another small note. We rely on control chracters appearing in the message area to alert of specical processing. These 0x2, 0x3, 0x4 0x5. But the display in terminal shows:
^b ^c (and they do seem to be two characters, not a single control character. This is a bit tricky to deterime)
In swing, they do not appear (they are control chars after all), but they are present.

I may have missed a step here. I am running from Intellij, which I did restart after the changes to .bashrd But maybe I need to reboot? Is there any way fro me to deterime if the patch really got installed and p2j sees it? I'm open to suggestions.

#16 - 03/28/2024 06:49 PM - Robert Jensen

The comand line is actually:
./client-terminal.sh client:cmd-line-option:startup-procedure=com/qad/qra/core/ClientBootstrap.p
client:cmd-line-option:parameter="cpinternal=UTF-8,cpstream=UTF-8,startup=mf.p,mfgwrapper=true"

I wonder if that is correct.

#17 - 03/29/2024 06:17 AM - Greg Shah

In #7657-5, I mentioned some changes that were needed. Let me make it more clear: these changes that are needed are mostly modifications to the FWD source code. Unless you have written those changes (unlikely), then it is not expected to work (yet).

- Support for "wide characters" in NCURSES. There are alternate versions of the ncurses libraries that are compiled with wide character support. We need to ensure we can enable this support and successfully use it with UTF-8 input.
  - We have to ensure that those versions of the library are installed **and patched**.
  - The FWD build (native portion) needs to bind to the wide versions of the library.
  - The FWD native code (.h and .c files) must be modified to process wide characters. That code is very likely to be limited to single byte processing in some areas, including:
    - the interfaces called in NCURSES
    - internal variables/memory allocations on the heap (or local vars on the stack)
    - internal interfaces and processing
  - The FWD Java code may need edits. In particular, it may have some assumptions that each character coming back or going out is a single byte (e.g. we might apply bitmasks to values to clear the most signficant bytes).
- FWD client support for setting CPINTERNAL and CPSTREAM to UTF-8. Without this we will improperly process the input. The actual setting of these CP values can be done normally in the directory or via the matching bootstrap config startup parameters. I think the issue here is that we may not properly honor these settings today. This work is described in #6431. At this point it is largely a testing effort that remains. I don't know if any code changes are needed in FWD.

#18 - 03/29/2024 06:26 AM - Greg Shah

> I may have missed a step here. I am running from Intellij, which I did restart after the changes to .bashrd But maybe I need to reboot? Is there any way fro me to deterime if the patch really got installed and p2j sees it? I'm open to suggestions.

When you say "if the patch really got installed", I assume you mean the patches to NCURSES that are required to make FWD compile (and run in native mode) properly. We have these patches documented in Patching NCURSES Using Static Linking, Patching NCURSES and Patching TERMINFO.

You do not need to reboot. But you MUST recompile FWD to properly build the libp2j.so module.

If the FWD native build succeeded, then you have a working libp2j.so and it must have linked to a patched version of NCURSES in some way. You can always force load the module using ldd build/lib/libp2j.so to see what it reports as its dependencies. That is a good "check".

#19 - 03/29/2024 06:39 AM - Greg Shah

Robert Jensen wrote:

> The comand line is actually:
> ./client-terminal.sh client:cmd-line-option:startup-procedure=com/qad/qra/core/ClientBootstrap.p
> client:cmd-line-option:parameter="cpinternal=UTF-8,cpstream=UTF-8,startup=mf.p,mfgwrapper=true"
>
> I wonder if that is correct.

No, it is not correct if you are trying to override the cpinternal and the cpstream. For those, you must add the client:cmd-line-option:cpinternal=UTF-8 client:cmd-line-option:cpstream=UTF-8 to the command line for the script. Or you can configure them in the directory as noted above.

But please note that without the FWD code changes described in #7657-17, this will not work yet.

We will allocate some time next week to write these changes.

#20 - 03/29/2024 01:23 PM - Robert Jensen

I am open to trying out and testing any FWD changes here on my end. Let me know if I can help in any way. I have worked on terminal emulators at various times in the past.

#21 - 04/11/2024 12:16 PM - Eugenie Lyzenko

Guys,

During testing Hotel ChUI with wide chars environment I'm getting the following warning when starting server:

```
...
24/04/11 19:07:57.647+0300 | WARNING | com.goldencode.p2j.security.SecurityCache | ThreadName:main, Session:00
000000, Thread:00000001, User:standard | Can't find or can't instantiate the user-provided implementation for
SsoAuthenticator.
java.lang.ClassNotFoundException: com.goldencode.hotel.HotelChuiSsoAuthenticator
    at java.net.URLClassLoader.findClass(URLClassLoader.java:382)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:352)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Class.java:264)
    at com.goldencode.p2j.security.SecurityCache.instantiateSsoAuthenticator(SecurityCache.java:2136)
    at com.goldencode.p2j.security.SecurityCache.readAuthMode(SecurityCache.java:2104)
    at com.goldencode.p2j.security.SecurityCache.<init>(SecurityCache.java:460)
    at com.goldencode.p2j.security.SecurityAdmin.currentCache(SecurityAdmin.java:5941)
    at com.goldencode.p2j.security.SecurityAdmin.addUser(SecurityAdmin.java:2101)
    at com.goldencode.p2j.admin.AdminServerImpl.addUser(AdminServerImpl.java:1493)
    at com.goldencode.p2j.main.TemporaryAccountPool.createUser(TemporaryAccountPool.java:245)
    at com.goldencode.p2j.main.TemporaryAccountPool.createTemporaryAccounts(TemporaryAccountPool.java:178)
    at com.goldencode.p2j.main.TemporaryAccountPool.<clinit>(TemporaryAccountPool.java:152)
    at com.goldencode.p2j.main.StandardServer$14.initialize(StandardServer.java:1576)
    at com.goldencode.p2j.main.StandardServer.hookInitialize(StandardServer.java:2503)
    at com.goldencode.p2j.main.StandardServer.bootstrap(StandardServer.java:1230)
    at com.goldencode.p2j.main.ServerDriver.start(ServerDriver.java:534)
    at com.goldencode.p2j.main.CommonDriver.process(CommonDriver.java:593)
    at com.goldencode.p2j.main.ServerDriver.process(ServerDriver.java:225)
    at com.goldencode.p2j.main.ServerDriver.main(ServerDriver.java:1010)
...
```

Does it mean something serious issue with my application config or I can ignore this?

#22 - 04/11/2024 12:23 PM - Greg Shah

It must be something pulled over from the Hotel GUI directory. In October 2023, Galya implemented single signon for Hotel GUI. That is a web based thing and there is no equivalent at this time that has been implemented for Hotel ChUI in web mode.

In Hotel GUI, the class is embedded/src/com/goldencode/hotel/HotelGuiSsoAuthenticator.java (actually, I think it is a mistake that it was put in the embedded directory). Anyway, there is no such class for Hotel ChUI.

#23 - 04/11/2024 01:06 PM - Eugenie Lyzenko

Greg Shah wrote:

> It must be something pulled over from the Hotel GUI directory. In October 2023, Galya implemented single signon for Hotel GUI. That is a web based thing and there is no equivalent at this time that has been implemented for Hotel ChUI in web mode.
>
> In Hotel GUI, the class is embedded/src/com/goldencode/hotel/HotelGuiSsoAuthenticator.java (actually, I think it is a mistake that it was put in the embedded directory). Anyway, there is no such class for Hotel ChUI.

Thanks for clarification. Disabling SSO in directory.xml resolves this issue.

#24 - 04/11/2024 04:46 PM - Eugenie Lyzenko

The news so far.

After experimenting I would like to share some finding at this time. For minimal usage we need to:

1. To use UTF-8 or other char sets outside ASCII we use in ChUI we need to build special NCURSES set with configure command:

```
make clean
./configure --with-termlib CFLAGS='-fPIC -O2' --with-abi-version=6 --enable-widec
make
```

This will create wide char capable version of the NCURSES libraries to kink with. I mean to use static linking for MCURSES libraries with w name postfix in this case.

2. The FWD build config file for native code (makefile) is need to be changes to use this version of the NCURSES:

```
...
# linux section
ifeq "$(OS)" "Linux"
   override CFLAGS+=-fpic

   # NCURSES library is a requirement in the project anyway so the C code
   # calls functions in that interface directly instead of exec'ing command
   # line utilities for the same purpose (to avoid the hard requirement of
   # having extra utility programs installed in addition to P2J); this is
   # the reason why libp2j depends on libncurses:
   override LDFLAGS+=-ldl -lutil
   ifdef NCURSES_FWD_STATIC
      override INCLUDES+=-I${NCURSES_FWD_STATIC}/include
      override LDFLAGS+=-L${NCURSES_FWD_STATIC}/lib -l:libncursesw.a -l:libtinfow.a
   else
      override LDFLAGS+=-lncursesw
```

```
   endif
   # this option is valid in Linux but not in Solaris
   override RMCMD+=v
endif
...
```

3. The native code in FWD is need to be changed to use alternative API calls to get/put data from/to ChUI terminal(terminal_linux.c):

```
+#define NCURSES_WIDECHAR 1
+#include <wchar.h>
...
+#include <locale.h>
...
void addchNative(int chrToDraw)
{
-   addch(chrToDraw);
+   // the lowest 8 bits is the char, while the rest
+   const cchar_t wch = {chrToDraw & 0xFFFFFF00, {chrToDraw & 0x000000FF, 0, 0, 0, 0}};
+   add_wch(&wch);
}
...
void initConsole(JNIEnv *env, char * termname)
{
...
+   // do we need this call?
+   setlocale(LC_ALL, "");
...
}
...
jint readKey()
{
-   return (jint) getch();
+   jint retValKey = -1;
+   int rc = get_wch((wint_t*)&retValKey);
+   if (rc == OK || rc == KEY_CODE_YES)
+   {
+       // TODO: need to transform incoming data to use in Java?
+   }
+   else
+   {
+       // assuming rc == ERR here
+       retValKey = -1;
+   }
+
+   return retValKey;
}
...
```

This is the minimal set of changes to start using of the Hotel ChUI demo application. Note both read/write function have changed the usage approach. For wide char version the character attribute and key value are separated in two fields. In current FWD version we combine/pack both values into single 32-bit integer.

These changes does not mean the full UTF-8 support. I think we need to change Java classes that responsible for preparing data to put in terminal and handling the key obtained from native key reader. Currently we assume the character value can be represented with 8-bit single byte. This will not work for wide chars. So we need to have another I/O processing inside Java classes for wide chars. So we will have to get two versions of FWD classes, one for single byte chars and other - for wide chars. Or may be we will drop usage of the regular NCURSES and always use *w version of the libraries having backward compatibility with previous projects. I have no clear picture here.

But at runtime we need to have the ability to know whether ChUI terminal is wide or not to choose respective approach to pack/unpack the data to display or got. Also in wide version there is a difference between wide character(rc OK) and function key(rc KEY_CODE_YES) returned from get_wch(). This might need special attention too.

The further work will depend on what we want as result. In any case I'm expecting this can take more than one day to complete.

#25 - 04/12/2024 06:47 AM - Greg Shah

Good work!

Some thoughts:

- We should add a build-time option to control whether the native module will use wide chars or not.
- We can't switch exclusively to UTF-8. The existing environments mostly are not UTF-8 and this even includes hardware terminals. These things must be handled with full compatibility. We are just adding the option to support UTF-8 and wide chars.
- On the Java side, it seems like we can make the changes to always pass the data down in wide mode. Only the JNI code needs to know how to process the result.
  - The idea is that we only need a single API for the Java code to call.
  - Do you see a reason that won't work?

#26 - 04/12/2024 01:45 PM - Eugenie Lyzenko

Greg Shah wrote:

> Good work!
>
> Some thoughts:
>
> - We should add a build-time option to control whether the native module will use wide chars or not.

Agreed.

> - We can't switch exclusively to UTF-8. The existing environments mostly are not UTF-8 and this even includes hardware terminals. These things must be handled with full compatibility. We are just adding the option to support UTF-8 and wide chars.

OK. I think we will need to change JNI signatures to use more input options to note the wide char mode is in use and to pass attribute in a separate variable to have 32-bit variable we currently use for character code completely.

> - On the Java side, it seems like we can make the changes to always pass the data down in wide mode. Only the JNI code needs to know how to process the result.
>   - The idea is that we only need a single API for the Java code to call.
>   - Do you see a reason that won't work?

It is difficult to say for sure. For now I think it is possible (with JNI methods input options change). I will have the exact answer during implementation.

#27 - 04/24/2024 04:42 PM - Eugenie Lyzenko

Making required code changes to add ChUI terminal support with optional UTF-8 characters for input/output.

1. The first things to do is to have proper building environment to make correct linking with NCURSES libraries, static or dynamic. At this time I offer to introduce new system variable NCURSES_FWD_WIDE_CHARS to separate required libraries to link for native code. If we need to have the ability to compile for both NCURSES versions (wide and legacy) we should have two separate location for static NCURSES libraries, one for wide chars aware and other for legacy one. The variable can be added alongside with NCURSES_FWD_STATIC in .bashrc. The value is not important, just assign something like yes. The respective FWD makefile changes will be:

```
...
ifeq "$(OS)" "Linux"
   override CFLAGS+=-fpic

   # NCURSES library is a requirement in the project anyway so the C code
   # calls functions in that interface directly instead of exec'ing command
   # line utilities for the same purpose (to avoid the hard requirement of
   # having extra utility programs installed in addition to P2J); this is
   # the reason why libp2j depends on libncurses:
   override LDFLAGS+=-ldl -lutil
   ifdef NCURSES_FWD_STATIC
      override INCLUDES+=-I${NCURSES_FWD_STATIC}/include
      ifdef NCURSES_FWD_WIDE_CHARS
         override LDFLAGS+=-L${NCURSES_FWD_STATIC}/lib -l:libncursesw.a -l:libtinfow.a
      else
         override LDFLAGS+=-L${NCURSES_FWD_STATIC}/lib -l:libncurses.a -l:libtinfo.a
      endif
   else
      ifdef NCURSES_FWD_WIDE_CHARS
         override LDFLAGS+=-lncursesw
      else
         override LDFLAGS+=-lncurses
      endif
   endif
   # this option is valid in Linux but not in Solaris
   override RMCMD+=v
endif
...
```

This way we can easily switch between two different NCURSES libraries depending on current building requirements.

2. The second step is to find out when the wide char version is to be used in FWD environment. On the terminal setup we can use for every client session the CPINTERNAL variable defined in directory.xml and checking for Linux OS behind. If defined as UTF-8 then boolean flag is passing to native FWD library to inform the wide chars calls is to be used. The check can be done in ConsoleHelper:

```
   /**
    * Initialize the native layer.
    */
   public static void initChui()
   {
-      initConsole(); // call native function to initialize console.
+      // call native function to initialize console.
+      initConsole(!EnvironmentOps.isUnderWindowsFamily() &&
+                  "UTF-8".equalsIgnoreCase(I18nOps._getCPInternal()));
   }
```

This way we can let the native FWD layer to know what NCURSES API to use(teminal_linux.c):

```
...
void addchNative(int chrToDraw)
{
-   addch(chrToDraw);
+   if (useWideChars)
+   {
+      // the lowest 8 bits is the char, while the rest
+      const cchar_t wch = {chrToDraw & 0xFFFFFF00, {chrToDraw & 0x000000FF, 0, 0, 0, 0}};
+      add_wch(&wch);
```

```
+    }
+    else
+    {
+       addch(chrToDraw);
+    }
}
...
jint readKey()
{
-    return (jint) getch();
+    jint retValKey = -1;
+    int rc = get_wch((wint_t*)&retValKey);
+    if (rc == OK || rc == KEY_CODE_YES)
+    {
+        // TODO: need to transform incoming data to use in Java?
+    }
+    else
+    {
+        // assuming rc == ERR here
+        retValKey = -1;
+    }
+
+    return retValKey;
}
```

3. The next step is to ensure the Java code layer properly handle UTF-8 as subset of generic chars processing for In/Out with native ChUI console. Now I'm thinking about good representation of the Java Strings for output to ChUI. We currently convert Strings to byte array with String.getBytes() call, meaning every byte is a single ASCII char. I think for generic purpose we need to represent the String as array of chars, letting native code to do some post-processing before terminal output depending on whether the UTF-8 in use or not. On the other hand what we will get as result of the String.getBytes("UTF-8")? Every byte will represent single character? Or amount of bytes per char will depend on particular char Unicode point? Should we completely shift to splitting single String into array of Unicode chars(16-bit) used internally in Java?

Please let me know what you think.

#28 - 04/24/2024 05:51 PM - Eugenie Lyzenko

Created task branch 7657a from trunk revision 15162.

#29 - 04/25/2024 09:21 AM - Robert Jensen

UTF-8 encoding has a variable number of bytes per character. By the way, Java actually uses UTF-16 internally. It can also have characters taking up more then one 16 bit word. But this rarely happens with most common character sets.

So yes, the number of bytes used by a character depends on the Unicode code point.

#30 - 04/25/2024 10:51 AM - Eugenie Lyzenko

The 7657a updated for review to revision 15163. Rebased with recent trunk. New revision is 15165.

This is the first steps of changes at this time to compile and test. Continue working.

#31 - 05/02/2024 03:13 PM - Greg Shah

> 1. The first things to do is to have proper building environment to make correct linking with NCURSES libraries, static or dynamic. At this time I offer to introduce new system variable NCURSES_FWD_WIDE_CHARS to separate required libraries to link for native code. If we need to have the ability to compile for both NCURSES versions (wide and legacy) we should have two separate location for static NCURSES libraries, one for wide chars aware and other for legacy one. The variable can be added alongside with NCURSES_FWD_STATIC in .bashrc. The value is not important, just assign something like yes. The respective FWD makefile changes will be:
> [...]
>
> This way we can easily switch between two different NCURSES libraries depending on current building requirements.

This generally seems correct.

However, I would ask this: is there a problem if we always use wide mode? All modern Linux systems probably support it. Is there any problem it would cause? If it slows things down, that would be a problem. But if it would just work the same way, even for non-wide character sets, then perhaps we should always use wide mode.

If wide mode limits the implementation in some way, then we probably don't want to use it always. One way it would be a problem is if wide mode did not support all terminal types.

> 2. The second step is to find out when the wide char version is to be used in FWD environment. On the terminal setup we can use for every client session the CPINTERNAL variable defined in directory.xml and checking for Linux OS behind. If defined as UTF-8 then boolean flag is passing to native FWD library to inform the wide chars calls is to be used. The check can be done in ConsoleHelper:
>
> [...]
>
> This way we can let the native FWD layer to know what NCURSES API to use(teminal_linux.c):

This part I don't fully understand. The native code can't use wide chars if it isn't linked to the wide version of ncurses, right? And we know which version of ncurses we've linked with at compile time, so shouldn't we just conditionallly preprocess the code so that only the correct API is used?

> 3. The next step is to ensure the Java code layer properly handle UTF-8 as subset of generic chars processing for In/Out with native ChUI console. Now I'm thinking about good representation of the Java Strings for output to ChUI. We currently convert Strings to byte array with String.getBytes() call, meaning every byte is a single ASCII char. I think for generic purpose we need to represent the String as array of chars, letting native code to do some post-processing before terminal output depending on whether the UTF-8 in use or not. On the other hand what we will get as result of the String.getBytes("UTF-8")? Every byte will represent single character? Or amount of bytes per char will depend on particular char Unicode point? Should we completely shift to splitting single String into array of Unicode chars(16-bit) used internally in Java?

As Robert points out, Java already has its String in 16-bit Unicode format and there the "supplementary characters" cases where a single character is larger than 16-bits (so it takes more than one char element in the char[] that is the internal representation of a String).

Consider that OE has a concept of setting the CPTERM codepage explicitly. This is meant to be the codepage in which character data is output on the terminal. If not set explicitly, then the CPINTERNAL value is used for CPTERM. In ChUI and in redirected terminal streaming in GUI, if there is a difference between CPTERM and CPINTERNAL, then the characters are converted from CPINTERNAL into CPTERM before writing them out. We currently don't support this but I think we now need to figure it out.

Consider that our existing customers (that use ChUI) have a mixture of hardware terminals and software based terminal emulators. We currently support VT100, VT220, VT320 and xterm terminal types. The VT series were ASCII terminals (or maybe extended ASCII) as far as I remember. I don't know it they could be configured to handle a wider range of character encodings (e.g. like 8859-15 which is similar to Windows 1252). I think xterm is different and can in fact even support UTF-8. My point here is that we have existing users that have external terminals (hardware and software) that we must be able to support. I suspect OE needed the CPTERM so that the output could be forced into a specific codepage that the terminals would accept. We need to honor that same idea of translating the "internal" Java Unicode characters into the CPTERM codepage.

We need to plan that characters can be multibyte and pass the data accordingly. Using the Java char or char[] is not OK because these are 16-bit Unicode and they require special handling to deal with specific characters (like those that take more than 2 bytes). Instead, the Java code should do any codepage conversion and then write the results into a form that the native code can just write out to NCURSES.

#32 - 05/02/2024 03:51 PM - Robert Jensen

You bring up a very good point about using a vt220/320 terminal. They techincally do not support Unicode characters at all. We had to limit our virtual vt320 in Java to only process 7-bit commands.

We had cases (Japanese?) where the CSI charcater (0x9b) would appear in messages. The terminal thought this was a control character and it caused a good deal of trouble on certain messages. Using <ESC>[was far more stable.

It is odd in that Linux think we are an xterm, while Progress sees vt320. They are very close, but different.

From what I have seen in our widget walk code, we do see the UTF-8 characters when we examine the screen widgets, even if they do not display correctly. But when we attempt to enter these values through the terminal, they get corrupted. This may simplify your problem, although the message area may be an issue. However, there are times we need to see the screen. At spacebar pauses for example.

I do feel your pain, we had a good deal of trouble back in 2012 when we went to UTF-8 databases. We appreciate your looking into this. It is a challenge.