Base Language - Feature #1650

process launching support for Windows (JNI library changes and any associated modifications to the Java code as needed)

10/21/2012 01:11 PM - Greg Shah

Status:	Closed	**Start date:**	01/07/2013
Priority:	Normal	**Due date:**	05/03/2013
Assignee:	Eugenie Lyzenko	**% Done:**	100%
Category:		**Estimated time:**	224.00 hours
Target version:	Runtime Support for Server Features		
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Base Language - Feature #1611: implement QUOTER utility	**Closed**	**10/20/2012**

History

#1 - 10/31/2012 01:31 PM - Greg Shah

- Target version set to Milestone 7

#2 - 12/04/2012 08:49 AM - Greg Shah

See https://projsrv01/redmine/issues/1611#note-36 for some useful notes.

#3 - 01/07/2013 02:04 PM - Greg Shah

- Start date changed from 10/21/2012 to 01/07/2013

- Assignee set to Eugenie Lyzenko

Please implement full process launching support for Windows. This means that src/native/process.c will need modifications. Make sure to isolate the platform differences by using one of these techniques:

1. If the difference is just a single line of code or a very small amount of differential code, then you can use conditional preprocessor statements:

```
#if TARGET_OS == WINDOWS
   ...stuff for windows...
#else
   ...stuff for linux/unix...
#endif
```

2. You can create functions that perform a generic version of the task and then have platform specific versions of the functions. These can be in a separate C source file and the makefile could be changed to compile the correct one for the platform.

Please be advised that the Windows and Linux/UNIX process launching seems to have some differences in the 4GL. We must not break the Linux/UNIX implementation that we already have. But we must also get the Windows launching exactly correct.

The makefile will need some changes. For example, on Windows we would have a p2j.dll instead of a libp2j.so. And the compile/link tools may be different. The code should compile for both 32-bit and 64-bit Windows. Please try to use this toolchain (instead of MS Visual C):

http://www.mingw.org/

MinGW is a "minimalist GNU for Windows" which basically provides a working gcc environment (including both complier and linker) on Windows. The code that is written will still need to use the WIN32 API. I would consider the use of Cygwin (a more complete POSIX compliant environment for Windows), BUT I know that there are many WIN32 APIs that we will need to support in this project that cannot be provided by Cygwin.

The Java code that calls the native methods, will also need changes. For example, the code currently assumes that the shell for OS-COMMAND is always "sh", which won't work on Windows.

These language statements must be handled:

OS-COMMAND
DOS
UNIX (don't break this, it already works)
INPUT THROUGH
INPUT-OUTPUT THROUGH
OUTPUT THROUGH

Make sure all the options that are currently supported, work properly in the result.

#4 - 01/15/2013 10:56 AM - Greg Shah

Please provide a summary of your findings so far.

I know that the DOS statement needs conversion support. I would like to know if there are any other conversion changes needed. I would like you to propose the conversion changes and AFTER we have discussed them, you will implement the changes. Then you will temporarily put this task on hold to work on other tasks.

#5 - 01/15/2013 02:32 PM - Eugenie Lyzenko

At this time I have not found any other statements except DOS to add the conversion support. As far as I understood the statements OS-COMMAND, I/O THROUGH, UNIX and OS-COMMAND that currently converted are using the common ProcessOps.launch() wrapper to pass the control to native code.

And for now I see two possible ways to implement Windows specific support:

1. We detect the Windows inside the Java code just before calling process related command and then execute Windows alternative for ProcessOps.launch(), something like ProcessOps.launchWin(). This new wrapper is implementing in Windows version of the process.c file and will be located in p2j.dll.

2. We do not make any separation on the conversion level. Just add the support for DOS statement by adding(or extending another words) dos commands to the the UNIX commands subset. As the result - we will have the common set of the OS Shell commands including DOS, UNIX, OS-COMMANDS... calling to ProcessOps.launch(). If the OS is matches the requested command - it will be executed in native code, if not - we will have error/exception generating error info to P2J.

From the options above I'm inclining to choose #2. I think(correct me if I'm wrong) the native Progress 4GL does not check this correspondence on the compilation level. The responsibility to choose the right command for appropriate OS is up to Progress programmer. This has the dark side of course - conversion error propagation up to the runtime code when for example we can try to execute Windows command under Linux and vise versa.

I have installed the MinGW development toolkit on my Windows system but did not experiment with it long enough. The point I need to investigate: if the API of this toolkit is fully compatible with GNU C we use to compile the native code under Linux. If the answer is positive - we can use the similar process/pipe calls in both code version(Linux and Windows) with possible modifications to reflect Windows specific behavior of the native 4GL code. Probably this means we will have to distribute the C runtime(that translates GNU C calls to Win API ones) from MinGW to the customers.

#6 - 01/15/2013 02:50 PM - Greg Shah

Option 2 is certainly how we want to do it. Yes, it is up to the application to only try to run child processes that can be run.

The process launching code in ProcessDaemon will need modification since it has some Linux/UNIX specific stuff (like a hard coded "sh").

```
 The point I need to investigate: if the API of this toolkit is fully compatible with GNU C we use to compile
the native code under Linux. If the answer is positive - we can use the similar process/pipe calls in both cod
e version(Linux and Windows) with possible modifications to reflect Windows specific behavior of the native 4G
L code. Probably this means we will have to distribute the C runtime(that translates GNU C calls to Win API on
es) from MinGW to the customers.
```

It is my understanding that mingw does NOT HAVE its own compatible GNU runtime environment. In fact, that is exactly why we are picking it. We know that there are many native features of Windows that cannot be supported or duplicated using the GNU runtime. Instead of trying to start down a path that can never get us to the proper destination, I want to implement using the WIN32 APIs directly.

For now, just make sure that the DOS command is fully supported in conversion. Put everything else (the runtime changes) on hold.

#7 - 01/15/2013 06:32 PM - Eugenie Lyzenko

Looks like we now already have properly conversion for DOS statement. The following line
dos dir /w > dir.log.
converts now to:

```
ProcessOps.launch(new String[] {
"dir",
"/w",
">",
"dir.log"
}, false, true);
```

And the code can be compiled.

So any arbitrary DOS a b c ... x y z. command will be converted to launch(new String[] { "a", "b", "c", ... , "x", "y", "z" }). Is this the support we are looking for on conversion level?

As to native support I would suggest to separate the platform specific code to separate file(or even two). Then each JNI:

```
Java_com_goldencode_p2j_util_LaunchManager_someFunction()
#if TARGET_OS == WINDOWS
someFunctionWin()
#else
someFunctionLinux()
#endif
```

This allows us to leave the current working Linux code untouched and concentrate on Windows implementation instead of mixing two platform code within the same function.

And looks like at the moment of native implementation we will have to modify ProcessDaemon.prepareCommandLine() method replacing "sh" with "cmd.exe" for Windows case.

#8 - 01/16/2013 06:46 AM - Greg Shah

```
So any arbitrary DOS a b c ... x y z. command will be converted to launch( new String[] { "a", "b", "c", ... ,
 "x", "y", "z" } ). Is this the support we are looking for on conversion level?
```

Yes, you are right. We already have the proper support in the conversion for the DOS command (and also the undocumented OS2, BTOS and VMS <g>). The rules are in rules/convert/process_launch.rules.

You don't need to do anything on the conversion.

```
As to native support I would suggest to separate the platform specific code to separate file(or even two). The
n each JNI:
Java_com_goldencode_p2j_util_LaunchManager_someFunction()
#if TARGET_OS == WINDOWS
someFunctionWin()
#else
someFunctionLinux()
#endif
```

OK.

```
And looks like at the moment of native implementation we will have to modify ProcessDaemon.prepareCommandLine(
) method replacing "sh" with "cmd.exe" for Windows case.
```

Yes. And I would be surprised if we didn't find some other things that need some modification. But for now, you can put this on hold and work on other conversion tasks for milestone 4.

#9 - 03/27/2013 04:21 PM - Eugenie Lyzenko

The question. I have idea to install The Windows inside the KVM Virtual Machine on my new Ubuntu box. Will it also conflict(as having dual boot system) with the current security requirements for the customer's development system?

Yes, I have another standalone system with Windows on the board for such experiments but this is a bit less convenient.

#10 - 03/28/2013 08:37 AM - Greg Shah

You are allowed to use Windows in KVM on the same Ubuntu system. It will be allowed in the standards. Thanks for asking.

#11 - 03/29/2013 07:06 PM - Eugenie Lyzenko

- File evl_upd20130329a.zip added

This is update just for review to see the approach. It compiles native DLL in Windows 32-bit using mingw32-gcc. Planning to do the same for 64-bit Windows version. The Windows native code has only stubs for now to compile. I can install both 32 and 64 bit Windows in virtual machines for testing without any problems.

What is required for Windows system:
1. Java 6 SDK
2. Apache Ant(need to configure system variables)
3. Download and install mingw-get-inst-20120426.exe(need to modify system PATH variable).

#12 - 04/02/2013 06:09 AM - Greg Shah

Feedback:

It is OK so far. Make sure you merge with the latest version of P2J (for example the build.xml that you started from is out of date). Also, I want you to integrate your changes into the main build.xml (we don't want a separate build.xml for win32), but it is fine to make a separate copy for experiments. The final result must automatically detect the platform on which the build is occurring and build the right native module (.so or .dll) for that platform.

In regard to the structure of the windows vs linux code, I want to do the following:

1. Each JNI function can have as much common code as is possible to be used on both platforms.

2. Any code that needs to be different for each platform should be in a separate .c file that is platform specific (one for each platform). For example, you would have a file_system_helper_win.c and a file_system_helper_linux.c and only one would be compiled and linked into the result, depending on the platform. If you need to differentiate signatures (e.g. for calling conventions or something like that), you can have separate header files too.

3. The names of the platform specific functions should not have the platform names in them. Instead, the platform-specific functions will have the same names. For example, both file_system_helper_win.c and file_system_helper_linux.c would have a checkFileType() function with the same signature.

4. Using this approach, the JNI functions will call the helper function without needing to IFDEF.

```
JNIEXPORT jint JNICALL
       Java_com_goldencode_p2j_util_FileChecker_checkFileType(JNIEnv  *env,
                                                              jclass  jc,
                                                              jstring fname)
{
   checkFileType(env, jc, fname);
}
```

This should greatly reduce the amount of conditional preprocessing that has to occur. But it means that you must design the platform-specific helper API to cleanly separate the common code from the platform-specific code. Of course, please maximize the common code.

#13 - 04/02/2013 08:11 PM - Eugenie Lyzenko

- File evl_upd20130402a.zip added

OK. This drop contains only changes in build.xml project build file to review. Plus one small change for filesys.c for missing return value. I'll rework the code according to your notes 1-3.

The suggested approach for single build.xml is based on the fact the 64-bit Windows has the property ProgramFiles(x86) defined and 32-bit Windows does not have it. We can not trust the ${os.arch} value from ANT because if we run 32-bit Java in 64-bit Windows this will return "x86". Another possibility is to check system wide PROCESSOR_ARCHITECTURE value. My KVM instances give the "AMD64" and "x86" values for 64-bit and 32-bit Windows running inside Ubuntu and I'm not sure what is this value if we run 32-bit OS on 64-bit capable hardware in native mode(not in virtual machine).

The side effect of the ProgramFiles(x86) approach - we have one ignoring warning for exec if we building in 32-bit Windows.

So the new build.xml works in following:
Ubuntu 12.10 64-bit
Windows XP 32-bit in virtual machine
Windows 7 64-bit in virtual machine(Windows XP 64-bit is very unstable to run in KVM, it is a lot of pain to work with).

#14 - 04/03/2013 09:25 AM - Greg Shah

I am generally fine with the build.xml approach. The only thing I see is this:

```
    <javah class="com.goldencode.p2j.util.LaunchManager"
           destdir="${src.home}/native"
           classpath="${build.home}/classes"/>
      <javah class="com.goldencode.p2j.util.FileChecker"
           destdir="${src.home}/native"
           classpath="${build.home}/classes"/>
    <copy todir="${build.home}/native">
      <fileset dir="${src.home}/native" includes="makefile" />
    </copy>
```

This code is common and should be in a target that is shared by all the other native tasks as a "depends".

In regard to the problems detecting win 32-bit vs 64-bit, go with your approach for now. Please put some comments into the build.xml to explain why you have made the choices you did.

#15 - 04/03/2013 06:40 PM - Eugenie Lyzenko

- File evl_upd20130403a.zip added

Today drop has been uploaded. Includes:
1. Rework for build.xml to separate common native tasks. Adding required comments.
2. The *.c sources have been reworked to eliminate conditional compilation in sources. It is happening only in makefile. The helper *.c source files have been added to accumulate platform specifics.

The update is verified for compilation in Linux and both Windows architectures. Now I'm shifting to the windows implementation itself. The code common for all OS will be moved to the platform neutral object.

#16 - 04/05/2013 08:48 AM - Greg Shah

I like it. Keep going in this direction.

#17 - 04/07/2013 01:33 PM - Eugenie Lyzenko

- File evl_upd20130407a.zip added

The next drop changes:
1. P2J build.xml file has been changed to pass OS architecture to the makefile for Windows compilation. This is to be able to choose whether we need to include -fpic compiler option. This is required for 64-bit but is default for 32-bit and produces warnings.
2. In the filesys.c commonly used code has been moved to platform independent file. The symlink part is working from Windows Vista, the implementation is under investigation.
3. For the process.c module functions the experimental implementations have been added. There are Linux calls we use without direct counterpart in Windows:

```
- sigemptyset(), sigaction()
- SIGTSTP, SIGTTIN, SIGTTOU constants
```

for now I've replaced with signal call for SIGINT exception.
4. The main difference is missing fork call in Windows. So the pseudoTerminalLaunch need to be implemented separately for each OS. Now I'm trying to use CreateProcess to replace fork/execvp pair we use in Linux.

Continue working.

#18 - 04/08/2013 06:59 PM - Greg Shah

Code Feedback:

1. Please remove the following from future updates:

```
src/native/com_goldencode_p2j_util_FileChecker.h
src/native/com_goldencode_p2j_util_LaunchManager.h
```

Those are generated files and should not be distributed.

2. Please use the "simple" process.h and filesys.h instead of the longer *_helper.h versions.

3. Please use a simpler process_linux.c/process_win.c instead of the ***helper***.c versions. Do the same thing with the file_system_*.c files.

4. This code is missing from process_helper_linux.c:

```
   // due to a bug in the JVM 1.4.2 and earlier, we must override any
   // signal handler for SIGCHLD that may exist because otherwise the JVM
   // may reap our process instead of letting us do it (see Sun J2SE bugs
   // #4945203, #4785154 and #4763362)
   signal(SIGCHLD, SIG_IGN);
```

I think it is pretty important and should be put back in, unless you have a good reason that it needs to be removed.

Other than the above 4 items, I like the way this is shaping up. Keep going.

#19 - 04/08/2013 07:16 PM - Eugenie Lyzenko

> 2. Please use the "simple" process.h and filesys.h instead of the longer *_helper.h versions.

I had the same idea. But I did worry process.h can conflict with mingw internal include file with the same name - process.h. I made just name separation.

> 3. Please use a simpler process_linux.c/process_win.c instead of the helper.c versions. Do the same thing with the file_system_*.c files.

OK.

> 4. This code is missing from process_helper_linux.c:

```
> // due to a bug in the JVM 1.4.2 and earlier, we must override any
> // signal handler for SIGCHLD that may exist because otherwise the JVM
> // may reap our process instead of letting us do it (see Sun J2SE bugs
> // #4945203, #4785154 and #4763362)
> signal(SIGCHLD, SIG_IGN);
>
```

> I think it is pretty important and should be put back in, unless you have a good reason that it needs to be removed.

OK. Sorry I've just lost this code during migration to the platform independent coed and back. Will be restored.

#20 - 04/08/2013 08:30 PM - Eugenie Lyzenko

- File evl_upd20130408a.zip added

The next drop includes:
1. Restored missing code for linux process file to handle SIGCHLD
2. Reworked file names to simplify. Unfortunately for now process.h has compilation issue in Windows. So the file is: process_.h
3. The Java source modification has been included. We have to modify two placed inside ProcessDaemon.java: one for command choosing - sh or cmd.exe, the other is option choosing - /c or -c.
4. The process launching has some degree of working. The simple test:

```
dos silent "dir /w > dir.log".
```

converted and produces correct results for Windows.

Continue working.

What do you think about C code GUI debugger for Windows we can use in this project? It will be good to have something other than gdb provided with mingw. For now I use simple printf calls to show info but this is very restricted approach.

#21 - 04/09/2013 01:28 PM - Greg Shah

> Reworked file names to simplify. Unfortunately for now process.h has compilation issue in Windows. So the file is: process_.h

Let's call it process_launch.h so that it is descriptive but still won't conflict.

Overall, the latest version is really looking good. Keep going.

> 4. The process launching has some degree of working.

That is great!!! Well done!

> What do you think about C code GUI debugger for Windows we can use in this project? It will be good to have something other than gdb provided with mingw.

Look at this:

http://www.gnu.org/software/ddd/
http://www.drdobbs.com/tools/visual-debugging-with-ddd/184404519 (at the end it describes that using Cygwin, one can get DDD running)

I have used it on Linux as my primary C/C++ debugger. I haven't used it much, but it has done the job. It uses gdb as its backend, so it should work for debugging mingw created binaries too. But there will be a bit of extra work in getting it running under Windows.

#22 - 04/10/2013 12:28 PM - Eugenie Lyzenko

The issues(related or partially) to discuss so far:

1. The source file com/goldencode/p2j/util/SourceNameMapper.java method initMappingData, line 832:
...
sb.append(pkgroot.replaceAll("\\.", fileSep));
...
This call generates exception in Windows(for fileSep == "\"). This is known issue we fixed some time ago.
I would suggest to replace this call with:
...
sb.append(pkgroot.replaceAll("\\.", StringHelper.fixupRegex(fileSep)));
...
This will add second "\" for Windows and do nothing for Linux.

2. The source file com/goldencode/p2j/util/ProcessDaemon.java method launch, line 156:
...
terminal.suspend();
...
Do we really need this call here? The result of this call is invoking TypeAhead.suspend which calls Object.wait() and suspend the thread execution before launch the command. And it is required to press spacebar to continue. This is not happening neither in Windows 4GL nor in Linux 4GL(in both silent and non-silent modes). To properly implement 4GL behaviour we need to remove this(fully or partially by substitution with another thin client clearing code).

#23 - 04/11/2013 04:20 PM - Greg Shah

> I would suggest to replace this call with:
> ...
> sb.append(pkgroot.replaceAll("\\.", StringHelper.fixupRegex(fileSep)));

OK, include it.

> terminal.suspend();
> ...
> Do we really need this call here? The result of this call is invoking TypeAhead.suspend which calls Object.wait() and suspend the thread execution before launch the command.

Yes, this code is essential. It does the following:

1. It stops the TypeAhead class from trying to read keystrokes. If it did not do this, then we would be "fighting" with the child process for input.

2. It notifies the Window class that it is suspended. This disables drawing to the window (except for the status line).

3. It clears the terminal to remove all the previous 4GL output.

4. It suspends processing at the NCURSES level.

All of these things are essential. Of course, we have no such terminal facilities written yet for Windows, so in that case we may have to bypass this stuff. But it MUST be there for Linux.

> And it is required to press spacebar to continue. This is not happening neither in Windows 4GL nor in Linux 4GL(in both silent and non-silent modes).

This comes from the following code (line 194 of ProcessDaemon):

```
         // if not silent we display a default message and pause indefinitely
         if (!silent)
         {
            lastkey = terminal.suspendedPause();
         }
```

This is only done in non-silent mode. But I am sure it is needed. You can see this in a very simple testcase. Run this on lindev01 (Linux 4GL environment):

```
os-command bash.
```

Do something at the command line (e.g. ls -l). Then end the child process by typing "exit" and hitting ENTER. You will see the "Press space bar to continue." message and the indefinite pause.

It also happens in something like this:

```
os-command ls -l.
```

We know for sure this is working today. Perhaps there is a difference in Windows, but for Linux it is required.

> To properly implement 4GL behaviour we need to remove this(fully or partially by substitution with another thin client clearing code).

It should still work in the Swing CHUI client, right?

If there are specific differences in Windows, please detail how they are different from the Linux 4GL environment. Show 4GL example code and the differing results so I can understand.

#24 - 04/12/2013 07:02 PM - Eugenie Lyzenko

- File evl_upd20130412a.zip added

```
sb.append(pkgroot.replaceAll("\\.", StringHelper.fixupRegex(fileSep)));
```

OK, include it.

The new drop has been uploaded with this change. In addition The code for I/O redirection has been changed to reflect Windows specifics. Now I'm debugging the different testcases for INPUT-OUTPUT THROUGH 4GL statements.

If there are specific differences in Windows, please detail how they are different from the Linux 4GL environment. Show 4GL example code and the differing results so I can understand.

OK. I'll prepare the testcases.

#25 - 04/14/2013 10:48 AM - Greg Shah

It looks good. Keep going.

#26 - 04/17/2013 09:06 PM - Eugenie Lyzenko

- File evl_upd20130417a.zip added

The new drop has been updated after lots of debugging. Windows is a kind of black box. The FileDescriptor Java implementation approach does not work for Windows standard output redirection. It gives the EOF exception and InputStreamReader.ready() == false for correct fd field set. Or bad descriptor exception. So I had to add new function to the process_win.c named initializeFileHandle. When we set up handle field for FileDescriptor object - the redirection code become working. The next issue I need to debug - why this does not work for standard error stream redirection and to find if this is OK for standard input.

The current code was verified to work in both 32-bit and 64-bit virtual Windows. And we need more samples to check. So continue working. The following code now works fine:
define variable chVar as character format "x(70)".

```
input through echo %COMPUTERNAME% no-echo.
set chVar with frame indata no-box no-labels.
display chVar.
input close.

pause.
```

#27 - 04/18/2013 12:42 PM - Greg Shah

Good work! Keep going.

#28 - 04/22/2013 08:50 PM - Eugenie Lyzenko

- File evl_upd20130422a.zip added

The new drop. Includes:
1. Merge with recent code for filesys update, the open() call was replaced with open_native() because there is no O_NONBLOCK constant in Windows, the P_NOWAIT should be used instead.
2. The OUTPUT THROUGH statements implementation in native library.
3. The process launching in Windows was reworked to get rid of the wrap calls. The file descriptor initialization code does not work in Windows - the debugging findings. So because initializeFileDescriptor call is useless in Windows it is completely moved to the Linux native code. We have to use initializeFileHandle for all In/Out/Err pipes in Windows. And this is checked as working code in bot 32-bit and 64-bit Windows.
4. So the simple Progress code:

```
define variable chVar as character format "x(70)"
   init "echo %PROCESSOR_IDENTIFIER%\nexit\n".

pause "Press any key to start".

output through cmd.exe > out_thru_log.txt no-echo.
display chVar with no-labels no-box.
output close.
```

Converted and run in Windows. The result - the value of the PROCESSOR_IDENTIFIER system variable is written into out_thru_log.txt file.

Continue working. The next steps will be checking for the INPUT-OUTPUT THROUGH statements and to check if the OUTPUT THROUGH statement is working identically in P2J and 4GL in Windows. The question here is do we need to add "\n" character during conversion of the passed variable for OUTPUT THROUGH statement. Or may be we need to use something like wrileLn() instead of just write in ProcessStream Writer methods.

About INPUT-OUTPUT THROUGH. Hope this is just a combination of previous two working cases and there will be no showstoppers here.

#29 - 04/23/2013 06:21 PM - Eugenie Lyzenko

Update status for INPUT-OUTPUT THROUGH. The idea of this 4GL statement is to use external program instead of console to display data, modify and commit the result back. Consider the simple test example:

```
define variable intVar as integer initial 4.

display intVar.
message "Press any key to change the value".
pause.
```

```
input-output through chvalue.exe unbuffered.
export intVar.
set intVar.
input-output close.

display intVar.
message "The value should be changed by external program".
pause.
```

The chvalue.c program used here is:

```
#include <stdio.h>

main()
{
   int value;

   setbuf(stdout, (char *) NULL);

   while (scanf("%d", &value) == 1)
   {
      value = value * 2 + 5;
      printf("%d\n", value);
   }
}
```

Converted and run on Windows 32-bit and 64-bit. So the INPUT-OUTPUT THROUGH is working with the recent update. The only question here is: We have the Waited thread in ../p2j/util/ProcessStream.java. It is OK when we have either only INPUT THROUGH or OUTPUT THROUGH redirection - we have only one reader or writer to wait for. But in the case of INPUT-OUTPUT THROUGH we have both writer and reader and as results two Waiter threads for single external process. I just afraid of concurrency for these two wait calls in different threads. Not sure if this is a problem for native code calls.

And one more conclusion: MinGW is OK for this native implementation. Continue working.

#30 - 04/24/2013 06:35 PM - Greg Shah

The code continues to look good. Your approach makes sense. Good stuff. Keep going.

#31 - 04/25/2013 09:27 AM - Greg Shah

- Due date set to 05/31/2013

- Status changed from New to WIP

- % Done changed from 0 to 80

- Estimated time changed from 80.00 to 224.00

#32 - 04/25/2013 09:32 AM - Greg Shah

- Due date changed from 05/31/2013 to 05/03/2013

#33 - 04/26/2013 03:40 PM - Eugenie Lyzenko

- File evl_upd20130426a.zip added

The next drop for review. Includes:
1. Merging with the recent code tree. The file build.xml modified.
2. The first version of the clearSreen native function has been implemented as replacement of the functionality provided by NCURSES.
3. The ../p2j/util/ProcessStream.java has been modified to check if the Waiter already exists for the INPUT-OUTPUT THROUGH case. Now we wait only once for two redirected streams.
4. General code modifications. The investigation shows we need to provide more checking for resource allocation/deallocation to avoid process crash. We can not trust the Windows backend for any clean up actions to avoid irregular and unpredictable crashes in ntdll system code. Every opened handle should be closed, every allocated heap should be freed, we can not trust the dup(string) function to work properly on all Windows versions. This is good because we have to take full control on what is going on.
5. I've still left the printf debugging stuffs for further debugging.
6. The native waiter call was changed to WaitForSingleObjectEx. This function is extended waiter with control in I/O operations.
7. The code was tested on 32-bit and 64-bit Windows.

Continue working. We need to have more testscases and implement file system features for Windows.

#34 - 04/29/2013 03:27 PM - Greg Shah

Please review this:

http://support.microsoft.com/kb/94248

#35 - 04/29/2013 07:02 PM - Eugenie Lyzenko

- File evl_upd20130429a.zip added

The next drop has been uploaded with symlink status check support. See the appropriate history for details.

#36 - 04/29/2013 07:15 PM - Eugenie Lyzenko

> Please review this:
>
> http://support.microsoft.com/kb/94248

The question. I have checked out P2J.DLL. Both 32-bit and 64-bit versions are linked to be used with MSVCRT.DLL and KERNEL32.DLL in MinGW if we do not specify explicitly the entry point. These libraries are the part of the Operating System so we do not need to distribute them separately.

Do we need to implement out own DLL Init routine? Or link to the special version of CRT library(for example LIBC.LIB or LIBCMT.LIB)? We consider the P2J.DLL to be multithreaded?

#37 - 04/30/2013 09:01 AM - Greg Shah

> Do we need to implement out own DLL Init routine?

Yes, I think we should. For now, we just need to properly handle the CRT initialization/termination on process attach/detach. This is pretty much the same as we used to do back in OS/2.

> We consider the P2J.DLL to be multithreaded?

Yes, everything in the DLL must be thread-safe and re-entrant.

It is true that the 4GL is single threaded, but I don't want to assume that the DLL will never be accessed by more than 1 thread at a time. So we must make it safe.

#38 - 04/30/2013 09:14 AM - Greg Shah

It looks good. Keep going.

Please make a list of everything that remains to be done.

#39 - 04/30/2013 03:25 PM - Eugenie Lyzenko

> Please make a list of everything that remains to be done.

At this moment I see the following TODO list(for process start related task):

1. Write the DLL Initialization routine and adjust the native code building makefile to link with appropriate Windows LIB/DLL.
2. Investigate the signal handling behaviour, particularly the ability to replace sigemptyset() function and handling the following signals: SIGTSTP, SIGTTIN, SIGTTOU.
3. Investigate the requirement to add replacement for Linux fix with signal(SIGCHLD, SIG_IGN); in Windows code.
4. Write the set of the process-related samples(we certainly need more test than we now have) to verify conversion-runtime cycle.

#40 - 05/01/2013 07:40 AM - Eugenie Lyzenko

The question. I'm planning to separate DLL Windows initialization code in the file entry_win.c. Is this name acceptable?

#41 - 05/01/2013 09:54 AM - Greg Shah

> I'm planning to separate DLL Windows initialization code in the file entry_win.c. Is this name acceptable?

Please use initterm_win.c instead.

#42 - 05/01/2013 04:27 PM - Eugenie Lyzenko

- File evl_upd20130501a.zip added

The next drop uploaded. Includes:
1. Basic skeleton for init/term procedure.
2. Small syntax fixes in most source code for comment/descriptions.
3. Explicit adding linking to MSVCRT.LIB/DLL in module makefile.

Continue working.

#43 - 05/06/2013 10:02 PM - Eugenie Lyzenko

Experiments with different testcases show the following results for process launch 4GL options for Windows implementation of the P2J native library and runtime:
1. If the option is supported on conversion/runtime level - it works.
- stream
- echo | no-echo
- unbuffered
- paged page-size(this option is converting and at runtime there are no page delimiters in our and real 4GL in Windows - we have the same result as on real system)
2. There is option that is not supported in Windows version of Progress environment
- map protermcap | no-map is not supported in Windows according to Progress documentation
3. There are options that we do not support on conversion level:
- convert (target trg_cp source src_cp) | no-convert - The experiments on Windows system with 4GL show this option should be supported in Windows.
- stream-handle option. According to OpenEdge 11.0 documentation this option should work the same way as stream option using appropriate handle corresponding to the stream value. However we do not convert this option(keyword stream-handle is unknown). Not sure if this is severe or not because as I can see this is just another way to use stream itself.

#44 - 05/09/2013 07:14 PM - Eugenie Lyzenko

- File evl_upd20130509a.zip added

The testcases archive. The statements was tested are INPUT THROUGH, OUTPUT THROUGH and INPUT-OUTPUT THROUGH. The current implementation of the P2J.DLL for Windows passes the testcases and shows the similar behaviour with native Progress implementation for Windows for the statement options that is properly converted.

Another note here. Investigation for signal handling based on Windows development documentation:
1. The signals we use in sigaction() calls are ignored in Windows API.
2. We declare these signals should be ignored in Linux p2j library.

Based on #1 and #2 above the conclusion is: we can just ignore the searching for sigaction() replacement in Windows DLL just doing nothing for signals missing in Windows API.

#45 - 05/10/2013 08:13 AM - Greg Shah

Is there any other work needed before you are done?

#46 - 05/10/2013 09:27 AM - Eugenie Lyzenko

> Is there any other work needed before you are done?

I would say yes if the DLL init-term procedure is completely OK. For now I do not see the things to do here. It is compiled with multithreading in mind and dynamically linked with MSVCRT.DLL and KERNEL32.DLL. Is this functionality good enough? Or we need more? For example - is it required to support old 32 bit Windows like Windows 2000 or NT may be? Or even systems that use Win32s old API?

#47 - 05/10/2013 07:56 PM - Eugenie Lyzenko

- File DLL_bestprac.doc added

The investigation and documentation reading let me conclude the current level of the P2J.DLL init-term implementation reflects the native task we now request from this library. So I guest at this time we can stop working on this task and resume on demand to implement new features to be added or need some special processing on init-term process. The current code allows to take as more control as possible for this process. The interesting document is attached here(taken from public resource).

#48 - 05/28/2013 07:06 PM - Greg Shah

Code Review

1. The approach is fine. Please remove all the logging code from the DllEntryPoint function. In addition, put comments in place of that logging.

For the DLL_PROCESS_ATTACH case:

// it is now safe to access the C runtime

For the DLL_PROCESS_DETACH case:

// final C runtime access must be done here

2. Are we sure that the MSVCRT is always present on Windows systems by default? If I understand correctly, we must distribute it with our app to be sure it is there AND we must be using MS Visual C++ (which we do NOT use).

3. Put comments in the appropriate code about not needing the signal processing on Windows.

4. Merge to the bzr rev 10355 level.

#49 - 05/28/2013 07:33 PM - Eugenie Lyzenko

1. The approach is fine. Please remove all the logging code from the DllEntryPoint function. In addition, put comments in place of that logging.

OK.

2. Are we sure that the MSVCRT is always present on Windows systems by default? If I understand correctly, we must distribute it with our app to be sure it is there AND we must be using MS Visual C++ (which we do NOT use).

As far as I can see MSVCRT.DLL is a standard package at least from XP. Why do we need to distribute it? If the user removed the MSVCRT.DLL how can we guarantee the system is in a good enough shape in general to run the application(if one of the system part was removed, may be even accidentally)? We need to document the requirement, of course.

I have modified makefile to explicitly link with MSVCRT by override LDFLAGS+=-lmsvcrt ... option.

3. Put comments in the appropriate code about not needing the signal processing on Windows.

4. Merge to the bzr rev 10355 level.

OK.

#50 - 05/28/2013 07:46 PM - Greg Shah

> As far as I can see MSVCRT.DLL is a standard package at least from XP. Why do we need to distribute it?

Are you sure it was not installed as part of some other application?

> I have modified makefile to explicitly link with MSVCRT by override LDFLAGS+=-lmsvcrt ... option.

I understand.

Another question: is msvcrt.dll multithread-safe by default?

#51 - 05/28/2013 09:20 PM - Eugenie Lyzenko

- File evl_upd20130528a.zip added

Merged update has been uploaded.

#52 - 05/29/2013 06:53 AM - Greg Shah

It looks good. Please get this runtime regression tested. I have applied the pending changes to staging and I am running conversion there under your userid. In 4.5 hours, it should be complete. You still have to apply your changes (evl_upd20130528a.zip) to /gc/convert/staging/p2j/ and rebuild. Then you can run regression testing in staging, directly.

#53 - 05/29/2013 07:16 AM - Eugenie Lyzenko

> Please get this runtime regression tested. I have applied the pending changes to staging and I am running conversion there under your userid. In 4.5 hours, it should be complete. You still have to apply your changes (evl_upd20130528a.zip) to /gc/convert/staging/p2j/ and rebuild. Then you can run regression testing in staging, directly.

OK.

#54 - 05/30/2013 04:32 PM - Eugenie Lyzenko

> Then you can run regression testing in staging, directly.

The testing completed. There are no regressions, results are 20130529_* - 20130530_* coped to the shared directory. The servers are now stopped.

#55 - 05/30/2013 05:00 PM - Greg Shah

Great! Please check it in and distribute it via email.

#56 - 05/30/2013 05:11 PM - Eugenie Lyzenko

> Please check it in and distribute it via email.

OK. One more question. There are the console logging code(printf() related calls) in process_win.c for info/debugging purpose. Can I left it in the file or I should remove this?

#57 - 05/30/2013 05:31 PM - Greg Shah

I should have caught that. No, that stuff MUST be removed. No debugging output should remain. If that is only in the windows code, then you don't need to regression test again. But you DO need to re-test the final windows process launching with your standalone testcases, to make sure that it wasn't broken when you remove the code.

#58 - 05/30/2013 06:04 PM - Eugenie Lyzenko

- File evl_upd20130530a.zip added

> that stuff MUST be removed. No debugging output should remain. If that is only in the windows code, then you don't need to regression test again. But you DO need to re-test the final windows process launching with your standalone testcases, to make sure that it wasn't broken when you remove the code.

This is the only file process_win.c(only C code for Windows) and is not involved in regression testing in staging. I have tested in both Windows, 32 and 64 bit and everything is OK. So I'm going to commit and distribute in a 20 min. if you do not mind.

#59 - 05/30/2013 06:26 PM - Eugenie Lyzenko

- File deleted (evl_upd20130530a.zip)

#60 - 05/30/2013 06:28 PM - Eugenie Lyzenko

- File evl_upd20130530c.zip added

Replaced the file with correct name - evl_upd20130530c.zip. Sorry for this mess.

#61 - 05/31/2013 08:37 AM - Greg Shah

- Status changed from WIP to Closed

- % Done changed from 80 to 100

Distributed and committed to bzr as rev 10356.

#62 - 11/25/2013 03:17 AM - Constantin Asofiei

Greg/Eugenie: if the client is ran in a non-TTY mode, the process launching will not work. Is this by intent? I think this line is at fault, in process_linux.c:114:

```
   // make sure we have a tty as stdin
   if (!isatty(STDIN_FILENO))
   {
      return -2;
   }
```

#63 - 11/25/2013 09:00 AM - Eugenie Lyzenko

> Greg/Eugenie: if the client is ran in a non-TTY mode, the process launching will not work. Is this by intent?

Yes, it is.

This part of code was transferred without changes from previous CHARVA Toolkit.c modifications. You can find the history of the changes(including this one I guess) looking at CHARVA modified Toolkit.c source.

#64 - 11/25/2013 09:17 AM - Greg Shah

The 4GL process launching requires a controlling terminal or pty. For instance, the programs that are launched from MAJIC will not work otherwise. This is why we have our JNI process launching code instead of relying upon the Java process launching support.

#65 - 11/16/2016 11:43 AM - Greg Shah

- Target version changed from Milestone 7 to Runtime Support for Server Features

Files

evl_upd20130329a.zip	14.4 KB	03/29/2013	Eugenie Lyzenko
evl_upd20130402a.zip	14.8 KB	04/03/2013	Eugenie Lyzenko
evl_upd20130403a.zip	19.8 KB	04/03/2013	Eugenie Lyzenko
evl_upd20130407a.zip	23.1 KB	04/07/2013	Eugenie Lyzenko
evl_upd20130408a.zip	26.1 KB	04/09/2013	Eugenie Lyzenko
evl_upd20130412a.zip	38.6 KB	04/12/2013	Eugenie Lyzenko
evl_upd20130417a.zip	39.2 KB	04/18/2013	Eugenie Lyzenko
evl_upd20130422a.zip	39.8 KB	04/23/2013	Eugenie Lyzenko
evl_upd20130426a.zip	47.9 KB	04/26/2013	Eugenie Lyzenko
evl_upd20130429a.zip	48.3 KB	04/29/2013	Eugenie Lyzenko
evl_upd20130501a.zip	49.4 KB	05/01/2013	Eugenie Lyzenko
evl_upd20130509a.zip	89.9 KB	05/09/2013	Eugenie Lyzenko
DLL_bestprac.doc	152 KB	05/10/2013	Eugenie Lyzenko
evl_upd20130528a.zip	50 KB	05/29/2013	Eugenie Lyzenko
evl_upd20130530c.zip	49.7 KB	05/30/2013	Eugenie Lyzenko