Base Language - Feature #1613

finish FILE-INFO system handle support

10/20/2012 09:34 AM - Greg Shah

Status:	Closed	**Start date:**	10/20/2012
Priority:	Normal	**Due date:**	
Assignee:	Costin Savin	**% Done:**	100%
Category:		**Estimated time:**	24.00 hours
Target version:	Runtime Support for Server Features		
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Base Language - Feature #2132: implement any missing filesys.c fea...	**Closed**	**07/04/2013**	**07/18/2013**

History

#1 - 10/20/2012 09:37 AM - Greg Shah

- Estimated time changed from 8.00 to 24.00

Implement the following:

FILE-INFO:FILE-TYPE (SMLP are not supported in P2J right now, there are limitations in Java 6, check new features in Java 7 to see if these can be supported, the alternative is to implement JNI helper methods)

FILE-INFO:FILE-CREATE-*

#2 - 10/31/2012 01:45 PM - Greg Shah

- Target version set to Milestone 7

#3 - 01/11/2013 05:55 PM - Greg Shah

- Assignee set to Costin Savin

- Status changed from New to WIP

Assuming new JNI methods are needed, they can be placed into filesys.c.

The conversion support is in rules/convert/methods_attributes.rules and it will need changes too, as well as the runtime.

#4 - 01/12/2013 09:09 AM - Costin Savin

For FILE-INFO:FILE-TYPE we can reuse the native code for OS-DIR as main type and just keep the read/write type logic from previous implementation in FileSystemDaemon like this:

```
            String primeType = FileChecker.getFileType(info.pathname);
            StringBuilder sb = new StringBuilder(primeType);

            if (local.canRead())
               sb.append("R");

            try
            {
```

```
            if (local.canWrite())
               sb.append("W");
         }
```

This way adding another native method is not necessary.

Previous implementation does not seem to take into account when we try to obtain file-info for a directory, while on windows Progress it can also be done for directories.
Was this intended from Unix Progress behavior or not implemented?

#5 - 01/14/2013 10:18 AM - Greg Shah

I am OK with reusing the native method, so long as the result is 100% compatible. Does the FILE-TYPE attribute actually generate the exact same results as FileChecker.getFileType() plus the RW? Even if the attribute characters are exactly the same, does it generate them in the same order as OS-DIR?

```
Previous implementation does not seem to take into account when we try to obtain file-info for a directory, wh
ile on windows Progress it can also be done for directories.
Was this intended from Unix Progress behavior or not implemented?
```

I think this was an oversight. Please implement it.

#6 - 01/14/2013 10:44 AM - Costin Savin

> I am OK with reusing the native method, so long as the result is 100% compatible. Does the FILE-TYPE attribute actually generate the exact same results as FileChecker.getFileType() plus the RW?

From what I've tested RW mark always appear at the end of the file status and the first part is the same as OS-DIR, so it is likely in Progress the type part logic is also reused, on the other hand the same as OS-DIR I couldn't test the special cases like: links, pipes and devices.

#7 - 01/14/2013 11:28 AM - Greg Shah

OK, great! Put the update together ASAP and upload it.

#8 - 01/14/2013 12:39 PM - Costin Savin

Finished and tested the FILE-INFO:FILE-TYPE logic wit OS-DIR code reuse.
For FILE-INFO:FILE-CREATE-* there might be necessary to create another JNI function.
In Java 1.7 this:
http://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/BasicFileAttributes.html
seems to have the key for the problems but for 1.6 I haven't found something yet.

#9 - 01/14/2013 12:44 PM - Greg Shah

At this point, I don't think it is a good idea to rely on something only in Java 7. Our current production customer is on Java 6 and I don't want to force them to migrate, just because of this.

Please add a JNI function or functions as needed.

#10 - 01/15/2013 12:11 PM - Costin Savin

On Unix the only date the system keeps regarding files is the last access date which is probably what Progress on Unix returns, for Windows the creation date can be retrieved through other functions then used for Unix.
I finished the implementation for Unix though it still needs to be tested on Unix 4GL.
The update contains changes that relate to the OS-DIR task #1626 which is not yet approved, should I include those files also?

#11 - 01/15/2013 12:19 PM - Greg Shah

Yes, do build the changes on top of the files from #1626 as this is in test and should pass soon. We can always make changes later if needed, but in the best case no merging will be needed.

Upload the update and do finish the UNIX 4GL testing to ensure the implementation is as expected.

#12 - 01/15/2013 12:43 PM - Costin Savin

- File cs_upd20130115a.zip added

Added proposed update , will test on Unix 4GL for matching behavior.

#13 - 01/15/2013 02:43 PM - Greg Shah

Feedback:

1. The build.xml and makefile are not changed in your update. Remove them unless you have real changes.

2. Line 253 of FileSystemDaemon will cause a regression when used with relative filenames. The following is from the Progress documentation:

```
If you set the FILE-NAME attribute to a relative pathname, the FILE-INFO handle
searches the current PROPATH to locate the file.
```

The searchPath() is very important.

3. The FileSystemOps.fileInfoGetCreationTime() should call the date.secondsSinceMidnight(GregorianCalendar) static method to reuse that code instead of duplicating the logic.

Let me know how your testing goes. Please add relative filenames into the testcases to ensure you aren't regressing something.

#14 - 01/16/2013 03:22 AM - Costin Savin

2. The problem with searchPath is that it only returns the value, if the target is a file, otherwise it returns null, (I'm not yet sure what is the reason behind the logic).

Should the file checking be removed from the logic of searchPath or create a duplicate of searchPath (like searchAll) which returns value for everything, not only file?

#15 - 01/16/2013 05:21 AM - Costin Savin

Tested on Unix 4GL:
It seems that FILE-INFO:FILE-CREATE-* doesn't actually return any value on Unix.
File types for this and OS-DIR work slightly different then on windows and with some bugs that will need to be replicated as behavior

#16 - 01/16/2013 07:22 AM - Greg Shah

```
2. The problem with searchPath is that it only returns the value, if the target is a file, otherwise it return
s null, (I'm not yet sure what is the reason behind the logic).

Should the file checking be removed from the logic of searchPath or create a duplicate of searchPath (like sea
rchAll) which returns value for everything, not only file?
```

I don't understand why getExistFilePath() was put there in the first place. As far as I know, the original use of searchPath() should work fine.

```
Tested on Unix 4GL:
It seems that FILE-INFO:FILE-CREATE-* doesn't actually return any value on Unix.
File types for this and OS-DIR work slightly different then on windows and with some bugs that will need to be
 replicated as behavior
```

Fix it but add clear documentation about the exact behavior in Windows and UNIX so that the implementation on Windows will be correct later.

#17 - 01/17/2013 11:03 AM - Costin Savin

Finished the update, however modification to search(String[] paths, boolean caseSens, String filename) from FileSystemDaemon is required.

On lines 125-126 and 163-164 we have the following code:

```
// is it a file and does it exist?
if (file.exists() && file.isFile())
```

This check prevents from using searchPath() for the FILE INFO purpose.

The reason behind it seems to be coming from the method searchPath(character) -> searchPath(String) which has specified that it only returns value if it is a file.
My solution would be to move this check from search to searchPath so we can safely use search() method for FILE-INFO.

#18 - 01/17/2013 11:27 AM - Costin Savin

- File cs_upd20130117b.zip added

Added proposed update with described changes (also contains FileChecker and filesys.c hopefully it will not collide with #1626 submit -> corrected)

#19 - 01/17/2013 05:09 PM - Greg Shah

See the code feedback notes in #1626. As noted there, DO NO FURTHER WORK ON THIS UNTIL AFTER THE MILESTONE 4 DELIVERY. Uploading the same update in both places is the right way to go.

#20 - 03/25/2013 04:13 PM - Costin Savin

- File cs_20130325a.zip added

Added proposed update.The update is for both #1613 and #1626

#21 - 03/26/2013 11:13 AM - Greg Shah

Code Feedback:

1. StreamWrapper has no real changes other than the copyright date and the history entry. Does it need to be in this update?

2. Have you fully tested this with all the related 4GL testcases? You would also need to have code that tests the SEARCH function since that is changing. I ask, because it seems to me that the FileSystemDaemon implementation is broken. allowDir should be false for SEARCH (the searchPath() case) but it is set true now. And accessFileInfo() should have allowDir as true, right? Do not post an update before you have it fully working (unless it has been explicitly requested for early review). If I am incorrect, help me understand where I went wrong in my analysis.

3. You have not handled the issues numbered 3-5 listed in #1626 note 48.

I would also point out that in C, it is common to do this (and it is preferred):

```
if (val & IS_WINDOWS)
```

4. In filesys.c:

```
if((fd = open(file_path, O_NONBLOCK)) < 0)
```

should be this:

```
if((fd = open(file_path, (O_RDONLY | O_NONBLOCK))) < 0)
```

right?

5. In filesys.c, why did you remove the close(fd) calls from the function (including the error return paths)? Isn't that going to cause a file descriptor leak?

#22 - 03/27/2013 02:58 PM - Costin Savin

There seems to be an error in the search() logic of FileSystemDaemon.
If the given paths Array is non null it will search the file within it's elements, the problem is the default search path is "." which when combined with the path will result in an invalid file. As a solution I created a constant in EnvironmentOps DEFAULT_SEARCH_PATH = "." and checked also if path elements are equal with default value before adding the path to them.
I'll do some more tests and prepare the update if this change is ok.

#23 - 03/28/2013 08:48 AM - Greg Shah

I don't understand why the comma is included in the default path. We already have protection against the path being "." in FSD.search(). I wonder if the real problem is that the comma is being left behind in the paths array before it is stored using FSD.setSearchPath() when it should be stripped off. We need to be very careful here to avoid regressions because other code relies upon this logic. But the comma (used as a platform-neutral path separator in the 4GL) probably doesn't need to be there (and probably should not be there). Please investigate this by using other testcases (see testcases/uast/) that use the SEARCH function and PROPATH assignment.

#24 - 03/28/2013 08:53 AM - Costin Savin

In the Javadoc of EnvironmentOps.searchPath() there is this saying:

```
    * If no value is found via this lookup, then the default value of
    * "," will be returned.
```

I'm thinking this is probably Progress behavior though it seems weird.

#25 - 03/28/2013 01:26 PM - Costin Savin

I retested the cases from UAST which contained search function or Propath and there weren't any difference,
However the test in which the logic could fail was not present which is searching for a file using a relative path:
Ex:

```
message search("../costins/pro.sh").
```

This example returns ../costins/pro.sh in Progress 4GL (if the file exists at that location of course)
Without the changes I made the p2j version returned unknown: ? and with the changes to ignore if the path is ., it works.

My change doesn't fix all the changes though

In this case:

```
propath = "/home"
message search("../costins/pro.sh").
```

the default search path will not be "." but /home and p2j will return null instead of "../costins/pro.sh" which is the 4GL result

This would not happen if the check file.isAbsolute() would not be a requirement to check if the file exists and return the name in this case.

My solution removes the check isAbsolute and replaces it with checking if the file exists which I think is logic, if the files does not exist relative to the current directory then we search for it in the Propath.

```
   public static String search(String[] paths,
                               boolean  caseSens,
```

```
                              String   filename,
                              boolean allowDir)
  {
     String result = null;

     // make sure there is something to search for (the code below needs to
     // be protected)
     if (filename == null)
        return null;

     File file = new File(filename);

     try
     {
        // is it a file does it exist are directory allowed?
        if (file.exists() && (allowDir || file.isFile()))
        {
           if (caseSens)
           {
                 result = filename;
           }
           else
           {
              result = getCaseInsensitiveMatch(file, false);
           }
        }
        else
        {
           int     len     = ((paths == null) ? 0 : paths.length);
           boolean current = false;

           // paths may be null but if so, the len will be 0 and this code
           // won't execute so it is safe
           for (int i = 0; i < len; i++)
           {
              if (paths[i] == null)
                 continue;

              if (paths[i].length() == 0 || ".".equals(paths[i]) ||
                  paths[i].equals(EnvironmentOps.DEFAULT_SEARCH_PATH))
              {
                 // check a relative file in the current directory
                 file = new File(filename);
                 current = true;
              }
              else
              {
                 // check a relative file in the specified path
                 file = new File(paths[i], filename);
                 current = false;
              }

              if (caseSens)
              {
                 // is it a file does it exist are directory allowed?
                 if (file.exists() && (allowDir || file.isFile()))
                 {
                    // current dir returns the relative filename and anywhere
                    // else returns the absolute filename
                    result = (current ? filename : file.getCanonicalPath());
                    break;
                 }
              }
              else
              {
                 result = getCaseInsensitiveMatch(file, false);

                 if (result != null)
                    break;
              }
           }
        }
     }

     catch (IOException ioe)
```

```
        {
           // let null be returned
        }

        return result;
    }
```

If this change is ok I will attach the update

#26 - 04/01/2013 09:29 AM - Greg Shah

> In the Javadoc of EnvironmentOps.searchPath() there is this saying:

I think you mean EnvironmentOps.getSearchPath(), which returns the current PROPATH setting for the user's session.

> I'm thinking this is probably Progress behavior though it seems weird.

There is nothing weird about this. You must understand the following facts:

1. The PROPATH in Progress is a list of paths in which to search for files (programs during RUN statements, include files during compilation or just any file via SEARCH).

2. Each path in the list is separated by a COMMA. As an example, "c:\this\is\a\path,d:\another\path," or "/home/costins,/bin,/usr/bin,".

3. If the propath is not otherwise set, it will default to a single directory in the list and that directory is the current directory.

4. In the file system on both Windows, UNIX and Linux, a "." represents the current directory.

5. To represent a COMMA-separated list with only 1 element which is the current directory, the propath would be set to ".,".

The key point here is that the COMMA is used for parsing out the paths. And when the PROPATH is read by 4GL code (or converted code) using the PROPATH "function", the resulting value must be comma separated. BUT the paths variable stored on the client in FSD should not have the COMMA, since it is a String[] and it is already split.

I suspect the problems you are having are related to the splitPath() code and the fact that it is not processing based on a COMMA. We made the COMMA changes recently, but this code seems to have been overlooked. Please do NOT remove the isAbsolute() check and do not make the other changes you have proposed. They are incorrect.

Please look further based on the above notes.

#27 - 04/01/2013 10:53 AM - Costin Savin

- File cs_20130401a.zip added

Added proposed update, with the modified search function, eliminated the change in EnvironmentOps since it wasn't necessary anymore.
The problem was the search looked if the path was absolute or not and if it wasn't then it looked in the propath, when actually it should be searched if the file with the given path exists and if it doesn't then we search in the propath.

Also when we search for a file using a relative path the returned result is the relative path ex message search("../costins/pro.sh"). -> ../costins/pro.sh
The only thing I'm not sure it will have the correct behavior is when we have case insensitive filesystem in which case the getCaseInsensitiveMatch will build on the canonical path of the file instead of the relative path.

The 4GL bug which sees .. directory as hidden for os dir is described in DirStream class (it happpends because the relative path is used when checking file type, replicated this in java).
Update is for #1626 also

#28 - 04/02/2013 06:24 AM - Greg Shah

> The 4GL bug which sees .. directory as hidden for os dir is described in DirStream class (it happpends because the relative path is used when checking file type, replicated this in java).

I see this comment:

```
         // replicate the behavior in UNIX the upper dir appears as unknown which is a
         // result of not using the absolute path when checking for type
         // normal will be to use * FileChecker.getFileType(file.getAbsolutePath())
         // not using absolute path will not affect windows behaviour since hidden files
         // are not marked as starting with . in Windows.
         // In FILE-INFO logic the type is computed for absolute value and the upper dir
         // will not appear as hidden.
```

I don't fully understand what you are saying here. Based on this comment, I cannot tell the answers to these questions:

1. What happens when the user specifies an absolute file name on UNIX in the 4GL?
2. What happens when the user specifies an absolute file name on Windows in the 4GL?
3. What happens when the user specifies a relative file name on UNIX in the 4GL?
4. What happens when the user specifies a relative file name on Windows in the 4GL?
5. How does P2J duplicate these same differences in behavior?

#29 - 04/02/2013 06:58 AM - Greg Shah

Code Feedback:

1. This code in filesys.c still needs to close the file descriptor before returning:

```
   // Use fstat to detect files and directories or links that point to files
   // and directories as files or directories
   if (fstat(fd, &st))
   {
      return -1;
   }
```

2. Your changes at the top of FSD.search() are on the right track. But as you note:

> The only thing I'm not sure it will have the correct behavior is when we have case insensitive filesystem in which case the getCaseInsensitiveMatch will build on the canonical path of the file instead of the relative path.

You are right, the case insensitive path is now broken with your changes. Please fix it.

3. This code looks wrong and it definitely breaks behavior that we rely upon today:

```
               // if the path length is not empty check for the file inside the path
               if (paths[i].length() != 0)
               {
                  // check a relative file in the specified path
                  file = new File(paths[i], filename);
                  current = false;
               }
```

Why did you remove the check for the current directory? That is definitely needed in cases today. Also note how current cannot ever be set to true in your code. This doesn't seem correct at all. What are you trying to do here? You need to detail that and we can discuss it. You should also make sure you are using testcases that will trigger all the previous code paths (that you removed or changed) so that you can see if you are regressing things.

4. Please don't randomly delete blank lines that were intentionally placed in code for readability. For example, in FSD.search() your code looks like this:

```
               if (paths[i] == null)
                  continue;
               // if the path length is not empty check for the file inside the path
               if (paths[i].length() != 0)
               {
```

It should have looked like this:

```
               if (paths[i] == null)
                  continue;

               // if the path length is not empty check for the file inside the path
               if (paths[i].length() != 0)
               {
```

5. HandleCommon is missing a history entry.

6. Remove the extra character added to history 014 in FileSystemOps.

#30 - 04/02/2013 07:49 AM - Greg Shah

Also, please note that your update zip filename is not matching our standard naming conventions. You don't need to change the ones that are already uploaded. Just please make sure the next one uploaded is correct.

#31 - 04/02/2013 10:37 AM - Costin Savin

> 3. This code looks wrong and it definitely breaks behavior that we rely upon today:
>
> // if the path length is not empty check for the file inside the path
> if (paths[i].length() != 0) {
> // check a relative file in the specified path
> file = new File(paths[i], filename);
> current = false;
> }
>
> Why did you remove the check for current directory? That is definitely needed in cases today. Also note how current cannot ever be set to true in your code. This doesn't seem correct at all. What are you trying to do here? You need to detail that and we can discuss it.

I removed the check for current directory because I considered it is already checked in the begining with file.exists() which will relate to the current directory in case of a relative path:

```
         // is it a file, does it exist, are directory allowed?
         if (file.exists() && (allowDir || file.isFile())) // if the file is in the current dir it should be d
iscovered here
         {
            if (caseSens)
            {
               result = filename;
            }
            else
            {
               result = getCaseInsensitiveMatch(file, false);
            }
         }
         // if file doesn't exist look in the search path
         else
         {
            ...
```

The comments regarding the upper dir seen as hidden in unix do not belong in DirStream but in filesys.c and we reproduce it by just not treating the .. case as special when detecting hidnen files and directories, removed the coments from DirStream and added a comment to filesys.c which describes it

On the behavior
When a user specifies an absolute or relative file in both Unix and windows OS-DIR will display the contents of that file plus the upper dir .. using the full path (nonrelative).
In windows the current folder "." is also added

os-dir("../test_dir")

Unix results: considering the current dir is "/home/costins/work"

```
".." "/home/costins/test_dir/.." HD
".hidden" /home/costins/test_dir/.hidden" "HF"
...
```

Windows results: considering the current dir is "D:\costins\work"

```
".." "D:\costins\test_dir\.." D
```

```
"." "D:\costins\test_dir\." D
"hidden" D:\costins\test_dir\.hidden" "HF"
...
```

I also discovered a bug (DirStream.getPath) in the case where we get the canonical path of a link the result will be the path to what the link is pointing to.

#32 - 04/02/2013 01:29 PM - Costin Savin

> 2.
> You are right, the case insensitive path is now broken with your changes. Please fix it.

I'm not sure it's "broken" now and wasn't before when relative path were not included in the search logic. In this case the getCaseInsensitiveMatch will need to be modified to match.
How can I call a 4GL case insensitive search?

#33 - 04/03/2013 09:03 AM - Greg Shah

> I removed the check for current directory because I considered it is already checked in the begining with file.exists() which will relate to the current directory in case of a relative path:

That is not correct. The 4GL will search for a non-absolute file match using each path in the PROPATH. In fact, if the "." is included deeper in the PROPATH (instead of in front), then I think your outermost check for file.exists() is going to be wrong, because the 4GL may find the same file first in the preceding PROPATH entries.

If the PROPATH is /path1/,/path2/,/path3/,. then a search for relative/path/to/file.txt would find /path1/relative/path/to/file.txt before it finds ./relative/path/to/file.txt, right? Please create testcases for different scnearios, where the . is explicitly placed in front, in the middle and at the end of the propath. Make sure you have conflicting files in the different paths so that you can see which one is being found. Also make sure to check if the 4GL will place an implicit . into the propath if it is not there already.

The way you have coded it, there is an implicit . at the beginning of the propath. I'm not convinced that is correct.

> I'm not sure it's "broken" now and wasn't before when relative path were not included in the search logic. In this case the getCaseInsensitiveMatch will need to be modified to match.

How can I call a 4GL case insensitive search?

All usage of the 4GL searching on Windows is case-insensitive. There is nothing you can do explicitly to make SEARCH case-insensitive. The 4GL does that implicitly on Windows because the Windows file system is case-insensitive. NBow, consider that when we convert a 4GL program that was coded to operate on Windows, the customer may be trying to run that converted Java program on Linux or UNIX. We provide support for this by

configuring whether the original 4GL code was run on a case-insensitive file system (e.g. Windows) and if so, we implement a special search algorithm to case-insensitively search (even if the actual file system is now case-sensitive).

This is why we will have a problem with placing the file.exists() at the outermost level. Please carefully consider these cases when making your changes.

#34 - 04/03/2013 12:55 PM - Costin Savin

Tested by having this structure /workcs/file.txt in the current folder(/home/costins)
and setting propath variable to /home/costins/propath, inside this there also is a /workcs/file.txt structure, and indeed when searching for "workcs/file.txt" the following will be returned "/home/costins/propath/workcs/file.txt" what this means is that the search function probably looks in.
If we search for "./workcs/file.txt" the result will be "./workcs/file.txt" which resolves to -> home/costins/workcs/file.txt
however if we search for "workcs/file.txt" and we don't have any folder in propath containing this, the one from current directory will not be found and the result will be unknown.

Now about search for relative paths on case insensitive os:
if we have the current folder set to "D:\work" and inside we have file.txt
if we search ".\file.txt" the result in 4GL windows will be ".\file.txt", p2j will use canonical path for this and return "D:\work\file.txt" instead.
I think the solution would be to check if the file is absolute in the getCaseInsensitiveMatch and use the file path instead of the canonicalPath if it is not absolute.

#35 - 04/04/2013 07:35 AM - Costin Savin

- File cs_upd20130404a.zip added

> If the PROPATH is /path1/,/path2/,/path3/,. then a search for relative/path/to/file.txt would find /path1/relative/path/to/file.txt before it finds ./relative/path/to/file.txt, right

Tested and this is true search matches with the paths in the order in which they are found in propath.

> Also make sure to check if the 4GL will place an implicit . into the propath if it is not there already.

This is not true. If we set PROPATH to empty and the current dir contains "file.txt"
search("file.txt") will return unknown, if we search("./file.txt") it will return "./file.txt", if we set Propath to contain "."
search("file.txt") will return the full (canonical path) to the file not just "./file.txt". Fixed p2j behavior by adding this additional check:
(file.isAbsolute() || file.getPath().startsWith(".")) to file.exists() which checks if it's absolute or a relative path to the current dir.

Also in the getCaseInsensitiveMatch I modified it to use the canonical path only if the path it's absolute and otherwise use the regular path to match the 4GL windows behavior.

I also attached the update with these chages for review.

PS:
The HandleCommon class doesn't have anything to do with functionality for this update I just added some interfaces that were not there. I can remove it if you wish.

#36 - 04/05/2013 08:21 AM - Greg Shah

Code Feedback:

1. See note 52 in #1626.

2. The FSD.search() change to use file.getPath().startsWith(".") is not equivalent to testing ".".equals(paths[i]) in the original "else" block. For example, a path ../whatever.txt will match this too. Please explain.

3. I am still trying to understand why you feel it is necessary to make the isAbsolute() section into a very complicated test. What specifically are you trying to do here? The test is so complex that the reader can no longer understand what the code is doing and why it must be changed in this way. In particular, what specific problem are you trying to address and why is this the best solution to the problem?

4. If we did go with this approach, we certainly wouldn't leave the dead "current" variable in use in the "else" block. This suggests to me that you are not reading the code carefully and that your proposed change has not been 100% figured out before you wrote it. Do not just write code and hope it works. I expect your code to be logically correct when it is submitted. That means you must understand all of its behavior in advance.

#37 - 04/09/2013 09:48 AM - Costin Savin

After I retested to see what I've missed (both Unix and Windows) I realized that the behavior of search for files and directories is actually different. My mistake was that I mixed FILE-INFO paths tests for directories with SEARCH tests which applies to files , considering they both yield the same type of result, instead of doing the same test for both and see the differences.

Considering the following tests:

```
DEFINE VAR fileN AS CHAR.
fileN = "file.txt".
PROPATH = ".".
FILE-INFO:FILE-NAME = fileN.
MESSAGE "SEARCH: " SEARCH(fileN).
MESSAGE "FILE-INFO:PATHNAME" FILE-INFO:PATHNAME.
MESSAGE "FILE-INFO:FULL-PATHNAME" FILE-INFO:FULL-PATHNAME.
```

```
SEARCH:  ./file.txt
FILE-INFO:PATHNAME ./file.txt
FILE-INFO:FULL-PATHNAME /home/costins/file.txt
```

now if we have PROPATH set to "" and we search

```
DEFINE VAR fileN AS CHAR.
fileN = "file.txt".
PROPATH = "".
FILE-INFO:FILE-NAME = fileN.
MESSAGE "SEARCH: " SEARCH("fileN").
MESSAGE "FILE-INFO:FULL-PATHNAME" FILE-INFO:FULL-PATHNAME.
```

The results will be different:

```
SEARCH:  file.txt
FILE-INFO:PATHNAME file.txt
FILE-INFO:FULL-PATHNAME /home/costins/file.txt
```

We will still find file.txt (instead of ./file.text).

If we search for a **directory**(that exist in the current directory "workcs" for ex) instead of a file there will be nothing found in FILE-INFO paths.

With these test result I can see that the previous logic of search stands , except searching for a file from the current directory when Propath does not contain the current directory,
For directories the logic will have to be slightly modified.

I'll post the update after some more testing.

#38 - 04/09/2013 12:37 PM - Costin Savin

This part of the previous code

```
 if (paths[i].length() == 0 || ".".equals(paths[i]))
 {
      // check a relative file in the current directory
      file = new File(filename);
      current = true;
 }

...
// current dir returns the relative filename and anywhere
// else returns the absolute filename
result = (current ? filename : file.getCanonicalPath());
```

seems wrong .

The first condition paths[i].length() == 0 is true for returning the file name, but if the PROPATH contains the "." it would never return the simple file name but the filename combined with the current path like "./filename".

Now for the canonicalPath part, we can have in propath paths like ".." , "../../bla/bla" and the path for files found by search in those folders will not be brought to canonicalPath form and will be instead presented in relative form "../file1", "../../bla/bla"
Actually I can give you an example where searching for canonical path will fain in p2j:

Lets say we have a file "/home/costins/pro.sh" and we make a symbolic link to it lnk.s. and we have propath set to "/home/costins"

When we do this: SEARCH("lnk.s"). it will get to the canonicalPath logic which will return "/home/costins/pro.sh" which is acualy the file to which the link is pointing.

Without those faulty parts the code would look like this:

```
  public static String search(String[] paths,
                              boolean  caseSens,
                              String   filename,
                              boolean  allowDir)
  {
     String result = null;

     // make sure there is something to search for (the code below needs to
     // be protected)
     if (filename == null)
        return null;

     File file = new File(filename);
     if (file.isAbsolute())
     {
```

```
         if (caseSens)
         {
            // is it a file and does it exist, are dir allowed?
            if (file.exists() && (file.isFile() || allowDir))
            {
               result = filename;
            }
         }
         else
         {
            result = getCaseInsensitiveMatch(file, false);
         }
      }
      else
      {
         int len = ((paths == null) ? 0 : paths.length);
         boolean relativePath = false;

         // paths may be null but if so, the len will be 0 and this code
         // won't execute so it is safe
         for (int i = 0; i < len; i++)
         {
            if (paths[i] == null)
               continue;
            File path = new File(paths[i]);
            // check if path is empty and filename exists and it's file
            // this doesn't apply for directories.
            if (paths[i].length() == 0 && new File(filename).isFile())
            {
               file = new File(filename);
            }
            else
            {
               file = new File(paths[i], filename);
            }

            if (caseSens)
            {
               // is it a file and does it exist, are dir allowed
               if (file.exists() && (file.isFile() || allowDir))
               {
                  result = file.getPath();
                  break;
               }
            }
            else
            {
               result = getCaseInsensitiveMatch(file, false);

               if (result != null)
                  break;
            }
         }
      }
```

#39 - 04/09/2013 02:03 PM - Greg Shah

These investigations are all going in the right direction. Continue your analysis and make a list of the problems in the current algorithm. For each problem, make sure there is a testcase that demonstrates the issue and output for these 3 scenarios:

1. P2J
2. 4GL Linux
3. 4GL Windows

Once you have all the issues identified and documented, then we can discuss the solution.

#40 - 04/09/2013 04:56 PM - Greg Shah

While you are working on the final fixes to FSD.search(), I would like to get a safe version of all the other changes checked in. That would allow us to close out most of #1613 and all of #1626. Most importantly, it will allow others on the team to merge with your changes, since we have people working on the native code that are dependent on you.

Please prepare an update that has all of the safe changes included, but avoids any of the dangerous stuff in FSD.search().

#41 - 04/10/2013 11:24 AM - Costin Savin

- File cs_upd20130410b.zip added

- File cs_upd20130410a.zip added

After some tests I found some more problems in the search algorithm for example the file.isAbsolute() check should also include files that are relative to current path that means they begin with "." + OS path separator, ".." + OS path separator or are equal to "." or ".."
Hidden files and directories seem to be detected without any bugs in 4GL.

My question is the "clean" update should only contain the dirAllowed changes without any fix to the logic even if it will result in wrong results?
To make sure I added both versions
10a contains my latest proposed update, including changes which might be considered "dangerous"
10b has no change to search besides allowDir and also no changes to getCaseInsensitiveMatch even if the behaviour is wrong.

#42 - 04/11/2013 01:25 PM - Costin Savin

Considering we have the following directory structure:

```
..(upper dir)
  |->.(Current dir)
     |->file.txt (file)
     |->propath(dir)
        |->file.txt (file)
  |->workcs(dir)
     |->file.txt (file)
```

And the following testcase:

```
message "No Propath change".
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH("./file.txt"): ' search("./file.txt").
```

```
propath = ".".
message 'PROPATH ="."'.
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH("./file.txt"): ' search("./file.txt").
message 'SEARCH("../workcs/file.txt"): ' search("../workcs/file.txt").

propath = "./propath".
message 'PROPATH ="./propath,"'.
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH("./file.txt"): ' search("./file.txt").
message 'PROPATH =""'.
propath = "".
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH("./file.txt"): ' search("./file.txt").
```

Here are the results for the unchanged search function on P2J , 4GL Unix and 4GL Windows

P2J (UNIX)

```
No Propath change
SEARCH("file.txt"): ?
SEARCH("./file.txt"): ?
PROPATH = "."
SEARCH("file.txt"): file.txt
SEARCH("./file.txt"): ./file.txt
SEARCH("../workcs/file.txt"): ../workcs/file.txt
PROPATH = "./propath,"
SEARCH("file.txt"): ?
SEARCH("./file.txt"): ?
PROPATH = ""
SEARCH("file.txt"): file.txt
SEARCH("./file.txt"): ./file.txt
```

4GL UNIX

```
No Propath change
SEARCH("file.txt"):  ?
SEARCH("./file.txt"):  ./file.txt
PROPATH ="."
SEARCH("file.txt"):  ./file.txt
SEARCH("./file.txt"):  ./file.txt
SEARCH("../workcs/file.txt"): ../workcs/file.txt
PROPATH = "./propath,"
SEARCH("file.txt"):  ./propath/file.txt
SEARCH("./file.txt"):  ./file.txt
PROPATH =""
SEARCH("file.txt"):  file.txt
SEARCH("./file.txt"):  ./file.txt
```

4GL WINDOWS

```
No Propath change
SEARCH("file.txt"):  ?
SEARCH("./file.txt"):  .\file.txt
PROPATH ="."
SEARCH("file.txt"):  .\file.txt
SEARCH("./file.txt"):  .\file.txt
SEARCH("../workcs/file.txt"): ..\workcs\file.txt
PROPATH = "./propath,"
SEARCH("file.txt"):  .\propath\file.txt
SEARCH("./file.txt"):  .\file.txt
PROPATH =""
```

```
SEARCH("file.txt"):  file.txt
SEARCH("./file.txt"):  .\file.txt
```

#43 - 04/11/2013 02:42 PM - Greg Shah

Which bzr revision of P2J did you use to get the results of note 42?

Were any of your changes applied at the time?

What happens if you display the "effective" PROPATH each time? Instead of lines like this:

```
message 'PROPATH ="."'.
```

do this:

```
message 'PROPATH ="." (' + PROPATH + ')'.
```

#44 - 04/11/2013 02:48 PM - Costin Savin

> Were any of your changes applied at the time?

Yes the allowDir additional check only but that should not affect anything from the logic since for search we call this with false. .

> What happens if you display the "effective" PROPATH each time? Instead of lines like this:
>
> message 'PROPATH ="."'.

In 4GL after the path we set "." there will be some additional builtin paths to files that PROGRESS 4GL uses.

#45 - 04/11/2013 03:48 PM - Greg Shah

Code Review of cs_upd20130410a.zip:

Generally I am fine with this update.

The resulting code in the FSD.search() method has some blank lines removed where they previously existed. Please put them back to aid readability.

Also, please make the following minor changes to FileChecker:

```
   /**
    * Utility method that checks if a path in in a form that is relative to the current dir
    *
    * @param  pathName
    *         The path which we want to check.
    *
    * @return true if the path is in a form that is relative to current dir or
    *         false otherwise
    */
   public static boolean isPathRealativeToCurrentDir(String pathName)
   {
      String separator = System.getProperty("file.separator");
      // check if the path is relative to the current dir
      if (pathName.startsWith("." + separator) || pathName.startsWith(".." + separator) ||
          pathName.equals(".") || pathName.equals(".."))
      {
         return true;
      }
      else
      {
         return false;
      }
   }
```

1. It has a spelling error in the name.

2. The body is written inefficiently. It would be better to look like this:

```
   /**
    * Utility method that checks if a path in in a form that is relative to the current
    * directory.
    *
    * @param  pathName
    *         The path which we want to check.
    *
    * @return true if the path is in a form that is relative to current dir or
    *         false otherwise.
    */
   public static boolean isPathRelativeToCurrentDir(String pathName)
   {
      String separator = System.getProperty("file.separator");

      // check if the path is relative to the current dir
      return (pathName.startsWith("." + separator)  ||
              pathName.startsWith(".." + separator) ||
              pathName.equals(".")                  ||
              pathName.equals(".."));
   }
```

3. The reference to the file separator is not as simple as you have it here. If you are operating on a name that was generated by the converted 4GL code, then you should probably be using the legacy file separator instead of the file.separator property. In this case, it is probably OK to use the local platform's file separator, but you should use the File.separator instead of "file.separator" system property.

Upload that version here. Then get it CONVERSION and RUNTIME regression tested. This one has the potential to break many things, so it is risky. We need to prove it is compatible before it gets checked in.

#46 - 04/12/2013 04:58 AM - Costin Savin

- File cs_upd20130412a.zip added

Added proposed update.

#47 - 04/12/2013 09:05 AM - Greg Shah

> The resulting code in the FSD.search() method has some blank lines removed where they previously existed. Please put them back to aid readability.

Some of these lines are still missing. I don't have the time to show you line by line where you deleted things inappropriately. It seems like you are not comparing your work with the original files before you send it. Is there some problem here that I should know about?

Please go ahead and get this version regression tested (both conversion and runtime).

While that is happening, please examine the following issue: please modify your tests to pass input to SEARCH and FILE-INFO:FILENAME that deliberately uses the Windows File.separator on Linux and the Linux File.separator on Windows. In other words, change ALL the file separators in the program to be wrong for the given platform and test the result on 4GL and P2J. Then report the results. I suspect there are some problems with our implementation.

#48 - 04/12/2013 12:41 PM - Costin Savin

Tested with the file separator changed to opposite of what the OS uses,

The results are:

- Unix P2J and 4GL will not find anything when given the path with the opposite separator,
- On Windows 4GL you can use either / or \ separators and you will find the path so this means for windows in FileChecker.isPathRelativeToCurrentDir() we have to check for both operators:

```
   public static boolean isPathRelativeToCurrentDir(String pathName)
   {
      String separator = File.separator;

      // check if the path is relative to the current dir
      return (pathName.startsWith("." + separator)   ||
              pathName.startsWith(".." + separator)  ||
              pathName.equals(".")                   ||
              pathName.equals(".."))                 ||
              (PlatformHelper.isUnderWindowsFamily() &&
              (pathName.startsWith("./" + separator) ||
              pathName.startsWith("../" + separator)));
   }
```

Is this solution ok with you?

> Some of these lines are still missing.

I re-checked and indeed I missed one space in the search function (compare tool didn't show the difference too well because of the missing try block), found another space in the getCaseInsensitiveMatch and one in the accessFileInfo (which doesn't seem to make sense but I put it back to be sure).

Ran the regression conversion testing, there are a lot of differences, because the latest converted code I have as reference to compare to, is from 5'th February 2013.

None of the differences seem to be related to my changes.
(I can attach the diff file if needed).

#49 - 04/12/2013 01:54 PM - Greg Shah

- Unix P2J and 4GL will not find anything when given the path with the opposite separator,
- On Windows 4GL you can use either / or \ separators and you will find the path

Please post the testcases and the output you got.

```
public static boolean isPathRelativeToCurrentDir(String pathName)
   {
      String separator = File.separator;

      // check if the path is relative to the current dir
      return (pathName.startsWith("." + separator)   ||
              pathName.startsWith(".." + separator)  ||
              pathName.equals(".")                   ||
              pathName.equals(".."))                 ||
              (PlatformHelper.isUnderWindowsFamily() &&
              (pathName.startsWith("./" + separator) ||
              pathName.startsWith("../" + separator)));
   }
```

I don't think this is right. For example, why are you appending the \ to ./ in this code: pathName.startsWith("./" + separator) || ?

In addition, much of this depends on how the 4GL output looks. The key question is if they "canonicalize" the inputs. In other words, do they replace all / characters with \ BEFORE they start searching in the filesystem?

Also: even if you fix this one method, I wonder if the search algorithm is still going to fail.

> Some of these lines are still missing.

> I re-checked and indeed I missed one space in the search function (compare tool didn't show the difference too well because of the missing try block), found another space in the getCaseInsensitiveMatch and one in the accessFileInfo (which doesn't seem to make sense but I put it back to be sure).

Consider this:

```
         for (int i = 0; i < len; i++)
         {
            if (paths[i] == null)
```

```
            continue;
         File path = new File(paths[i]);
         // check if path is empty and filename exists and it's file
         // this doesn't apply for directories.
         if (paths[i].length() == 0 && new File(filename).isFile())
```

In my opinion, it would be better like this:

```
      for (int i = 0; i < len; i++)
      {
         if (paths[i] == null)
            continue;

         File path = new File(paths[i]);

         // check if path is empty and filename exists and it's file
         // this doesn't apply for directories.
         if (paths[i].length() == 0 && new File(filename).isFile())
```

One of these was a line that you removed.

The other is a line that you added, but you added it "right up against" the following comment. That is a poor practice because it makes finding and reading the comments much harder, because your eyes aren't given any natural clues about how things are broken up. That means that you have to read every line instead of letting the spacing help you out.

> Ran the regression conversion testing, there are a lot of differences, because the latest converted code I have as reference to compare to, is from 5'th February 2013.

It is important to always have the conversion from the latest bzr check-in to compare against. Please re-run using the latest bzr revision (without your change). Then compare the results of your recent run to that "baseline" and report the changes.

#50 - 04/12/2013 02:37 PM - Costin Savin

- File cs_upd20130412c.zip added

> I don't think this is right. For example, why are you appending the \ to ./ in this code: pathName.startsWith("./" + separator) || ?

That was not intended : copy error.

> In addition, much of this depends on how the 4GL output looks. The key question is if they "canonicalize" the inputs. In other words, do they replace all / characters with \ BEFORE they start searching in the filesystem?

Yes this seems to happen (replace all / characters with \ BEFORE they start searching) because the search paths have \ instead instead of /
I wouldn't say "canonical" because relative path still stay relative.

File path = new File(paths[i]);
This is a leftover I forgot to remove, it is not needed.

> It is important to always have the conversion from the latest bzr check-in to compare against. Please re-run using the latest bzr revision

Having an already existent conversion (as long as it's recent enough) saves some time (about 1:20 -1:30 h) because I don't have to run it twice but I agree that comparing to latest bzr is the best choice.

I'll probably get the results from doing that after the weekend.

#51 - 04/12/2013 03:18 PM - Greg Shah

- Unix P2J and 4GL will not find anything when given the path with the opposite separator,
- On Windows 4GL you can use either / or \ separators and you will find the path

> Please post the testcases and the output you got.

You missed this part. I want to see the actual testcases and output.

> Yes this seems to happen (replace all / characters with \ BEFORE they start searching) because the search paths have \ instead instead of /
> I wouldn't say "canonical" because relative path still stay relative.

Then the proper solution is not achieved by changing isPathRelativeToCurrentDir(). Think about the behavior and then determine where in the search() process the change needs to be made.

Also: why is isPathRelativeToCurrentDir() in FileChecker? What is the purpose of FileChecker? Is it a suitable location for that method? Think about what the code does, where it is used and propose a more appropriate location in which it should reside.

#52 - 04/15/2013 07:59 AM - Costin Savin

The testcases for using the wrong File separator:

```
Unix code

message "No Propath change".
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH(".\\file.txt"): ' search(".\\file.txt").
propath = ".".
message 'PROPATH =" (' + PROPATH + ')'.
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH(".\\file.txt"): ' search(".\\file.txt").
message 'SEARCH("..\\workcs\\file.txt"): ' search("..\\workcs\\file.txt").

propath = ".\\propath".
message 'PROPATH =".\\propath" (' + PROPATH + ')'.
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH(".\\file.txt"): ' search(".\\file.txt").
message 'PROPATH =""'.
propath = "".
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH(".\\file.txt"): ' search(".\\file.txt").
```

4GL Unix result

```
No Propath change
SEARCH("file.txt"):  ./file.txt
SEARCH(".\file.txt"):  ?
PROPATH =" (.,/usr/progress/oe10.2B/dlc/tty,/usr/progress/oe10.2B/dlc/tty/adecomp.pl,/usr/progress/oe10.2B/d
lc/tty/adeshar.pl,/usr/progress/oe10.2B/dlc/tty/adeedit.pl,/usr/progress/oe10.2B/dlc/tt
y/prodict.pl,/usr/progress/oe10.2B/dlc/tty/adecomm.pl,/usr/progress/oe10.2B/dlc,/usr/progress/oe10.2B/dlc/bin)
SEARCH("file.txt"):  ./file.txt
SEARCH(".\file.txt"):  ?
SEARCH("..\workcs\file.txt"):  ?
PROPATH =".\propath" (.\propath,/usr/progress/oe10.2B/dlc/tty,/usr/progress/oe10.2B/dlc/tty/adecomp.pl,/usr/pr
```

```
ogress/oe10.2B/dlc/tty/adeshar.pl,/usr/progress/oe10.2B/dlc/tty/adeedit.pl,/usr/progress/oe10.2
B/dlc/tty/prodict.pl,/usr/progress/oe10.2B/dlc/tty/adecomm.pl,/usr/progress/oe10.2B/dlc,/usr/progress/oe10.2B/
dlc/bin)
SEARCH("file.txt"):  ?
SEARCH(".\file.txt"):  ?
PROPATH =""
SEARCH("file.txt"):  file.txt
SEARCH(".\file.txt"):  ?
```

P2J Unix result:

```
No Propath change
SEARCH("file.txt"):  ?
SEARCH(".\file.txt"):  ?
PROPATH ="." (.)
SEARCH("file.txt"):  file.txt
SEARCH(".\file.txt"):  ?
SEARCH("..\workcs\file.txt"):  ?
PROPATH =".\propath" (.\propath)
SEARCH("file.txt"):  ?
SEARCH(".\file.txt"):  ?
PROPATH =""
SEARCH("file.txt"):  file.txt
SEARCH(".\file.txt"):  ?
```

Windows code

```
message "No Propath change".
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH("./file.txt"): ' search("./file.txt").
propath = ".".
message 'PROPATH ="." (' + PROPATH + ')'.
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH("./file.txt"): ' search("./file.txt").
message 'SEARCH("../workcs/file.txt"): ' search("../workcs/file.txt").

propath = "./propath".
message 'PROPATH ="./propath" (' + PROPATH + ')'.
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH("./file.txt"): ' search("./file.txt").
message 'PROPATH =""'.
propath = "".
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH("./file.txt"): ' search("./file.txt").
```

Windows 4GL result

```
No Propath change
SEARCH("file.txt"):  ?
SEARCH("./file.txt"):  .\file.txt
PROPATH ="." (.,D:\oe10.2B\dlc\tty,D:\oe10.2B\dlc\tty\adecomp.pl,D:\oe10.2B\dlc\tty\adeshar.pl,D:\oe10.2B\dlc\
tty\adeedit.pl,D:\oe10.2
B\dlc\tty\prodict.pl,D:\oe10.2B\dlc\tty\adecomm.pl,D:\oe10.2B\dlc,D:\oe10.2B\dlc\bin)
SEARCH("file.txt"):  .\file.txt
SEARCH("./file.txt"):  .\file.txt
SEARCH("../workcs/file.txt"):  ..\workcs\file.txt
PROPATH ="./propath" (./propath,D:\oe10.2B\dlc\tty,D:\oe10.2B\dlc\tty\adecomp.pl,D:\oe10.2B\dlc\tty\adeshar.pl
,D:\oe10.2B\dlc\tty\adeedit.pl,D:\oe10.2
B\dlc\tty\prodict.pl,D:\oe10.2B\dlc\tty\adecomm.pl,D:\oe10.2B\dlc,D:\oe10.2B\dlc\bin))
SEARCH("file.txt"):  .\propath\file.txt
SEARCH("./file.txt"):  .\file.txt
SEARCH("file.txt"):   file.txt
SEARCH("./file.txt"):  .\file.txt
```

For p2j test the search version doesn't contain any change

#53 - 04/15/2013 09:37 AM - Greg Shah

Something is not right in the output you have posted. You posted this:

```
Unix code

message "No Propath change".
message 'SEARCH("file.txt"): ' search("file.txt").
message 'SEARCH(".\\file.txt"): ' search(".\\file.txt").
...
```

And then you posted these results from running the above code:

4GL Unix result

```
No Propath change
SEARCH("file.txt"):  file.txt
SEARCH("./file.txt"):  ./file.txt   <---- how could the text at the beginning of this line have a / instead of
 \ ???
```

Please explain and fix it, if it is broken.

#54 - 04/15/2013 09:45 AM - Costin Savin

The correct version is:
SEARCH(".\file.txt"): ? Edited it

#55 - 04/15/2013 10:12 AM - Greg Shah

Why was it broken?

#56 - 04/15/2013 10:20 AM - Costin Savin

Editing mistake:
Since I couldn't copy/paste the text from the windows 4GL test I had to just write it down and probably put that part there by mistake, rechecked the results of test against what it's written and it matches now.

#57 - 04/15/2013 11:02 AM - Costin Savin

- File cs_upd20130415a.zip added

Added proposed update: moved the isPathRelativeToCurrentDir() method from FileChecker to FileSystemOps, added check to see if File.separator is the windows separator and if true replace all / with \ of the file name and also replace it in the PROPATH paths when searching there.

#58 - 04/15/2013 12:15 PM - Greg Shah

Code Review:

1. You have chosen the wrong place to put isPathRelativeToCurrentDir(). FileSystemOps runs on the SERVER. The code for isPathRelativeToCurrentDir() directly uses the File.separator. If you put isPathRelativeToCurrentDir() into FileSystemOps, then the algorithm would change (and be wrong sometimes) because it would be using the server's File.separator instead of the client's (which can be different if the client and server are running on different platforms).

The most appropriate place for isPathRelativeToCurrentDir() is FileSystemDaemon, which always runs on the client.

2. When you calculate replaceSeparator in FSD.search(), please just use isWindowsPlatformFamily() instead of comparing the platform's File.separator to "\\". Using the explicit windows test is easier to read AND safer in the case where a future platform uses "\\" as a file separator BUT is not Windows.

#59 - 04/15/2013 12:27 PM - Costin Savin

- File cs_upd20130415b.zip added

Added update. I'll start the conversion test if there isn't anything else.

#60 - 04/15/2013 12:51 PM - Greg Shah

It is OK. Get this tested.

Please also retest all the related testcases to prove that the solution is still valid (and that there are no remaining problems). Make sure you include the "mismatching file separator" tests.

#61 - 04/15/2013 02:34 PM - Costin Savin

Ran conversion, no differences compared to latest from bzr, re-ran file-info and search tests, the results look good.

#62 - 04/15/2013 02:36 PM - Greg Shah

Great! Work with Constantin to get it runtime regression tested.

#63 - 04/18/2013 09:00 AM - Constantin Asofiei

Costin, about cs_upd20130415b.zip - you are missing a semicolon in filesys.c, line 103.

#64 - 04/18/2013 09:34 AM - Costin Savin

- File cs_upd20130418a.zip added

Added the semicolon back

#65 - 04/18/2013 09:48 AM - Greg Shah

How did that update pass standalone test and conversion testing if it did not compile?

#66 - 04/18/2013 09:55 AM - Costin Savin

On my machine it was there, so it's possible I corrected it after I created the 15b archive.

#67 - 04/18/2013 10:07 AM - Greg Shah

Please check that the update you posted here has the same versions that you used in testing.

#68 - 04/18/2013 10:41 AM - Costin Savin

- File cs_upd20130418b.zip added

I checked the files from the update and indeed there were other differences (in FSD) between 15b and the latest code on which I did the regression testing, I'm not exactly sure how this happened, I'm guessing at one point I reverted to an intermediate version when comparing the search() method test results with and without my changes.

#69 - 04/18/2013 11:12 AM - Greg Shah

The 0418b is OK. Please get that regression tested.

#70 - 04/19/2013 12:10 PM - Constantin Asofiei

Runtime regression testing has passed.

#71 - 04/19/2013 12:57 PM - Greg Shah

Great! Costin: please check it in and distribute it.

#72 - 04/19/2013 01:22 PM - Costin Savin

Commited to bzr as revision number 10341.

#73 - 04/19/2013 01:42 PM - Greg Shah

- % Done changed from 0 to 100

- Status changed from WIP to Closed

#74 - 11/16/2016 11:43 AM - Greg Shah

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

Files

cs_upd20130115a.zip	27 KB	01/15/2013	Costin Savin
cs_upd20130117b.zip	18.1 KB	01/17/2013	Costin Savin
cs_20130325a.zip	28.4 KB	03/25/2013	Costin Savin
cs_20130401a.zip	59.1 KB	04/01/2013	Costin Savin
cs_upd20130404a.zip	59 KB	04/04/2013	Costin Savin
cs_upd20130410a.zip	59.2 KB	04/10/2013	Costin Savin
cs_upd20130410b.zip	59.2 KB	04/10/2013	Costin Savin
cs_upd20130412a.zip	59.2 KB	04/12/2013	Costin Savin

cs_upd20130412c.zip	59.3 KB	04/12/2013	Costin Savin
cs_upd20130415a.zip	59.4 KB	04/15/2013	Costin Savin
cs_upd20130415b.zip	59.3 KB	04/15/2013	Costin Savin
cs_upd20130418a.zip	59 KB	04/18/2013	Costin Savin
cs_upd20130418b.zip	59.3 KB	04/18/2013	Costin Savin