# The Search Order for External Files

## 34th International Rexx Language Symposium

Amsterdam, May 14-17

Josep Maria Blasco
jose.maria.blasco@gmail.com

EPBCN – ESPACIO PSICOANALÍTICO DE BARCELONA
C/ BALMES, 32, 2º 1ª — 08007 BARCELONA

May the 16th, 2023

# The Search Order for External Files

## Part I

# Introduction: An anomaly

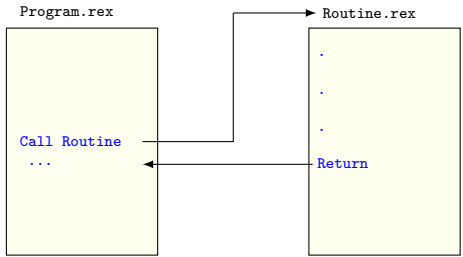
An anomaly in ooRexx
- The search order for external files
- Directories to search
- Introducing `same`, `curr` and `path`
- The anomaly

Reasons for the anomaly: a peek at the source code of the ooRexx interpreter
- `primitiveSearchName`
- `hasDirectory`

How to handle the anomaly
- A bug, or a feature?
- Deepening our understanding of the problem

# Introduction: An anomaly in ooRexx (1/4)

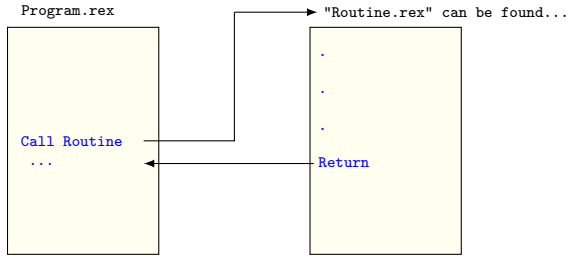
- We will start by studying an anomaly in ooRexx.
- The anomaly presents itself when calling a Rexx function located in an external file:




- The search order for external files uses a list of directories and a list of extensions to locate the file.

# Introduction: An anomaly in ooRexx (2/4)

- The searched directories are, in this order:




- (1) the same (or caller's) directory;
- (2) the current directory;
- (3) an optional, application-defined path;
- (4) the contents of the `REXX_PATH` environment variable;
- (5) the contents of the `PATH` environment variable.

# Introduction: An anomaly in ooRexx (3/4)

- To simplify, and without loss of generality, we will assume that

| Same directory | Current directory | App.-defined path | REXX_PATH | PATH |
|---|---|---|---|---|
| same | curr | (empty) | (empty) | path |

- PATH contains only one directory, called precisely path,
- the application-defined path is not set (or it is empty), and similarly
- the REXX_PATH environment variable is not set (or it is empty).
- Furthermore, let us assume that the same directory is indeed called same,
- and that the current directory is called curr.
- We will only have to examine three directories: same, curr and path.
- A Call Routine statement will search for Routine first in same, then in curr, and then in path. If Routine is not found, a syntax error will be raised.

# Introduction: An anomaly in ooRexx (4/4)

- `Call Routine` searches in `same`, `curr`, and then `path`.
- What happens if our statement is `Call "lib/Routine"` instead? We will have to search for `Routine` *in the* `"lib"` *subdirectory*.
- In the `"lib"` subdirectory of what? — Of `same`, `curr`, and `path`. *In exactly the same way as when the statement is* `Call Routine`.
- Now for the anomaly: when the instruction is `Call "../Routine"`, we should expect that `".."`, that is, the *parent* directory of `same`, `curr`, and `path` would be searched.
- BUT in this case, only the *current* directory `curr` is searched.

| Searched in... | 1st | 2nd | 3rd |
|---|---|---|---|
| `Call Routine` | `same` | `curr` | `path` |
| `Call "lib/Routine"` | `same` | `curr` | `path` |
| `Call "../Routine"` | ~~`same`~~ | `curr` | ~~`path`~~ |

- Why?

# Introduction: Reasons for the anomaly (1/3)

- Why is this happening? Let's take a look at the interpreter source code. In the Unix version of `SysFileSystem::primitiveSearchName`, we find:

```
1  // do the direct search if this is qualified enough;
2  // if not, try to locate it along the path
3  if ( hasDirectory(asIs) ?
4         checkCurrentFile(asIs, resolvedName) :
5         searchPath(asIs, path, resolvedName)
6     )
7  {
8      return true;
9  }
```

- If the boolean function `hasDirectory` returns true, *the path search is bypassed*.

# Introduction: Reasons for the anomaly (2/3)

- And what are the tests implemented by `hasDirectory`, exactly? Well, the source reads as follows (`name` is the file name being examined):

```
1  // hasDirectory() means we have enough absolute directory
2  // information at the beginning to bypass performing path searches.
3  return name[0] == '~' || name[0] == '/' ||
4         (name[0] == '.' && name[1] == '/') ||
5         (name[0] == '.' && name[1] == '.' && name[2] == '/');
```

- But file names starting with `"../"` (or with `"./"`, for that matter), are not *absolute*, but *relative*!
- What is going on?

# Introduction: Reasons for the anomaly (3/3)

- If we suppress the check for `"../"` and `"./"` in the Unix version of `hasDirectory`, the interpreter starts to work as expected and as documented. Additionally, all the tests in the test suite pass.[1]
- If, correspondingly, we suppress the check for `"..\"` and `".\"` in the Windows version of `hasDirectory`, the interpreter **does not** work as expected and documented (it starts to produce unexpected results).
- The Unix version of the interpreter constructs the filenames manually, i.e., it collates the directories with the supplied filename. Contrary to that, the Windows version resorts to the `SearchPath` Windows API.
- But `SearchPath` does **not** work as expected when the filename starts with `"..\"` or `".\"`: it searches the *current* directory, but not the supplied path.

[1] I submitted a proof-of-concept patch to test it, see bug no. 1865.

# Introduction: Ways of handling the anomaly (1/2)

- A possibility is to stop using `SearchPath` in the Windows version of the interpreter, and start constructing the filenames manually, like in the Unix version.
- This would suppress the anomaly, but (1) it may conceivably break existing programs, and (2) it would introduce new aspects of the interpreter behavior that some users do not find "natural".
- Another possibility is to decide that the interpreter works as intended, and then document this anomaly as a feature.
- This has two disadvantages: (1) it's asymmetrical (i.e., difficult to explain and remember), and (2) it represents opting for a limitation, instead of providing maximal freedom and letting the user limit herself.

# Introduction: Ways of handling the anomaly (2/2)

- It's difficult to reach a decision. Several advanced users believed that the interpreter behaves as one should expect by reading the documentation.
- Some other users (and in some cases, the same) expect that `"."` refers to the current directory — a contradiction.
- Maybe we could look elsewhere to gather more data? We could take a peek at how other interpreters work, and compare with ooRexx. We could, in fact, take a look at other products and environments, too. Maybe there is "a Rexx way of doing things" that we are not aware enough of, and this "Rexx way" will emerge by itself, as we deepen our understanding of the problem.
- To that purpose, we will implement and run a number of tests.

# The Search Order for External Files

## Part II

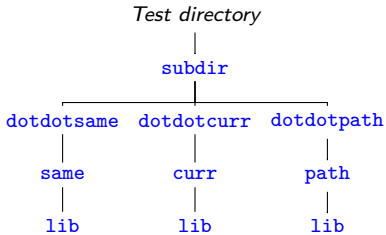
# Preparing the tests

# Preparing the tests: Basic assumptions

- To test other interpreters, we will continue to use the simplifying assumptions stated above: there will be three designated directories, `same`, `curr`, and `path`, and they will be checked in this order.
- Additionally, we will assume that each of these directories has a `lib` subdirectory, i.e., that `same/lib`, `curr/lib`, and `path/lib` exist.
- We will also assume that we can count on the *parent* directories of the three primary directories. We will call them `dotdotsame`, `dotdotcurr`, and `dotdotpath`; `same` will be a subdirectory of `dotdotsame`, and so on.
- The three "dotdot" directories will be located in a subdirectory called `subdir`, and `subdir` will in turn be located in our test directory.

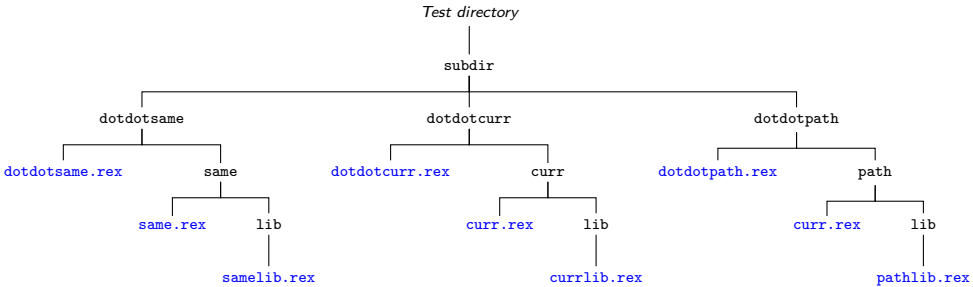
# Preparing the tests: The directory structure

- This is the directory structure we will be using. The `subdir` subdirectory is not really needed for the Rexx tests, but it will come in handy when testing other environments.



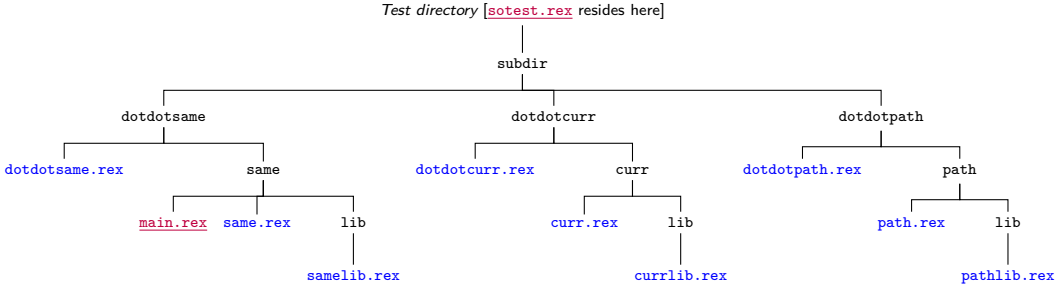
# Preparing the tests: Populating directories

- We will place a small Rexx program in all testable directories. Every program returns its own name. For example, `same.rex` returns the string `"same"`.



# Preparing the tests: Where to place the tests

- The test initiator program, sotest.rex will be located in the test directory.
- After doing some housekeeping, like setting the current directory to subdir/dotdotcurr/curr, it will call the real test program, main.rex, located in the same directory.



# Interpreters and operating systems

| Operating system | Interpreter | Version string |
|---|---|---|
| OS/2 4.52 (ArcaOS 5.0.7) | Classic Rexx | `REXXSAA 4.00 3 Feb 1999` |
| | Object Rexx | `OBJREXX 6.00 18 May 1999` |
| | Regina Rexx | `REXX-Regina_3.9.5 5.00 25 Jun 2022` |
| Ubuntu 22.04.01 LTS | Regina Rexx | `REXX-Regina_3.9.5 5.00 25 Jun 2022` |
| | ooRexx | `REXX-ooRexx_5.0.0(MT)_64-bit 6.05 23 Dec 2022` |
| Windows 11 Pro 10.0.22621.1413 | Regina Rexx | `REXX-Regina_3.9.5 5.00 25 Jun 2022` |
| | ooRexx | `REXX-ooRexx_5.1.0(MT)_64-bit 6.05 10 Mar 2023` |

- Operating systems: OS/2, Ubuntu and Windows.
- Interpreters: Classic Rexx, Object Rexx, Regina Rexx, ooRexx.

# Types of tests

(1) Common tests (2) Drive-relative tests (3) Special tests

- We will run three types of tests.
- *Common tests* will apply to all interpreters and operating systems.
- *Drive-relative* tests will only apply to operating systems that use drive letters (i.e., to OS/2 and Windows).
- *Special tests* will allow us to compare the behavior of Rexx to other products and environments.

# Common tests



1. *Simple calls*: `same.rex`, `curr.rex`, and `path.rex`.
2. *Downward-relative calls*: `"lib/samelib.rex"`, etc.
3. *Dot-relative calls*: `"./same.rex"`, etc.
4. *Upward-relative calls*: `"../dotdotsame.rex"`, etc.
5. *Upward-relative calls with a trick*: `"lib/../../dotdotsame.rex"`, etc.

- In all cases we will call the same program *with and without an extension*.

# Drive-relative tests



To run these tests, we will need to assign new drive letters to some of our directories. We can do that using the `SUBST` command under Windows, but under OS/2 we will have to assign drive letters by external means.

1. *Backslash-relative calls*: calling `"\path\to\my.rex"` is *relative* to the different drives present in the super-path.
2. *Letter-relative calls*: calling `"D:my.rex"` is *relative* to the current directory of the `D:` drive (every drive has a current directory under Windows and OS/2).
3. *Drive-absolute calls*: using absolute filenames with different drive letters.

# Special tests

(1) Common tests (2) Drive-relative tests (3) Special tests

These tests allow us to widen our perspective by comparing the behavior of the various Rexx interpreters to the behavior of different products and environments.

1. `CMD.EXE`: We will test the behavior of the `CMD.EXE` Command Line Interpreter under Windows (test included in `sotest.rex`).
2. `SearchPath`: We will test the behavior of the `SearchPath` Windows API (test included in `sotest.rex`).
3. *C/C++ compilers*: We will test the behavior of the GNU `gcc` compiler and of the Microsoft Visual Studio `cl` compiler.
4. *Python*: We will test the behavior of the `pathlib` module.

# Format of a test results file [fragments]

```
1   /**************************************************************************
2    sotest.rex -- A Search Order test suite
3
4    Interpreter:      REXX-ooRexx_5.0.0(MT)_64-bit 6.05 23 Dec 2022
5    Operating system: LINUX
…
11   The following values have been set:
12
13   Same directory:    '/home/sam/sotest/subdir/dotdotsame/same'
14   Current directory: '/home/sam/sotest/subdir/dotdotcurr/curr'
15   Path:              '/home/sam/sotest/subdir/dotdotpath/path'
…
19   **************************************************************************/
20  Pass.1  = .true;  Pass.1.test  = 'same'
21  Pass.2  = .true;  Pass.2.test  = 'same.rex'
…
48  Pass.29 = .true;  Pass.29.test = 'lib/../../dotdotpath'
49  Pass.30 = .true;  Pass.30.test = 'lib/../../dotdotpath.rex'
50  Pass.0 = 30
51  Return Pass.
```

- Test results are themselves programs. When called, they return a stem. This allows for easy comparisons, tabulations, etc.

# Two bugs

Our tests have unveiled bugs in two different interpreters.

1. The Windows version of ooRexx has a very subtle bug: when a filename has (1) no extension and (2) a dot in a directory (for example, `"my.dir\file"`, or `"..\file"`), the interpreter erroneously thinks that it has an extension, and therefore no additional extensions are tested.[2]

2. The `REXXSAA` interpreter does not search "in the current directory, with the current extension" and later "along environment PATH, with the current extension", as documented. We will refer to this bug as "the SAA bug".

3. The test results have been amended according to the bugs: the `REXXSAA` results have been patched as if the SAA bug did not exist, and the tests for ooRexx under Windows have been run again with the patched interpreter.

[2] I have reported this bug (SourceForge bug no. 1870), and provided a fix and an additional test, which were committed on March the 10th 2023 (r12651).

# The Search Order for External Files

Part III

# Interpreting the results

# Classifying common tests: The equivalence classes

(1) REXXSAA/Regina/`CMD.EXE` (2) Object Rexx (3) ooRexx/SearchPath

- Common tests results form three equivalence classes.
- The first class includes the REXXSAA ("Classic Rexx") interpreter for OS/2 (with the effect of the SAA bug manually amended), the three versions of the Regina Rexx interpreter, and the Windows `CMD.EXE` Command Line Interpreter.
- The only element of the second class is the Object Rexx interpreter for OS/2.
- To the third class belong the ooRexx interpreter (with the `hasExtension` bug fixed) and the `SearchPath` Windows API.

# Class 1: REXXSAA, Regina and CMD.EXE



- The differentiating characteristics of this class are the following:
- There is no notion of the "same" directory. The only interpreter that uses it is ooRexx. The same concept is also used in other environments, for example in some C/C++ compilers.
- As soon as the supplied filename includes a separator character ("\" or "/"), only the current directory is checked, and all path searches are bypassed.
- This is the most restrictive behavior, which should not be surprising since REXXSAA and Regina are older than Object Rexx and ooRexx.

# Class 2: Object Rexx for OS/2



- The differentiating characteristics of this class are the following:
- Similarly to REXXSAA and Regina Rexx, Object Rexx does not have a concept of the "same" directory. Files are only checked against the current directory and the path. This puts it on the restrictive side.
- On the other hand, Object Rexx does *not* have any limitation regarding filenames that start with `".\"` or `"..\"`: it checks them against the current directory and, if not found, against all the directories contained in the `PATH` environment variable. In this sense, it is the most advanced interpreter in the whole set.

# Class 3: ooRexx and the `SearchPath` Windows API

(1) REXXSAA/Regina/`CMD.EXE` (2) Object Rexx (3) ooRexx/SearchPath

- The differentiating characteristics of this class are the following:
- There is a concept of the "same" directory. As we mentioned earlier, this concept, in the Rexx world, is unique to ooRexx, although it can be found in other environmens.
- Filenames are checked against the same, current and (extended) path directories, *but* the path (and same) searches are bypassed when the filename starts with `".\"` or `"..\"`. This is undocumented behavior.

# Classifying drive-relative tests: Python's role (1/2)

- When one writes `"\path\to.file"` in the Windows or OS/2 CLI, this is a *relative* filename. Relative to what? To the current *drive*, e.g., if the current directory is `"E:\dir1\dir2"`, the drive part of this directory (`"E:"`) is concatenated to the supplied filename, and one gets `"E:\path\to.file"`.
- What should it mean, in this case, that the "same" directory is checked before the current directory? It could well mean the following: if the same directory is `"S:\dir3"`, we extract the drive part (`"S:"`) and we concatenate it to the supplied filename, to get `"S:\path\to.file"`, and similarly for all the directories in the different paths.
- Currently, no Rexx interpreter exhibits such a behavior, but Python's `pathlib` module does: for Python, `Path("D:/test") / "/this/file"` (under Windows) is `WindowsPath('D:/this/file')`.
- Is this behavior desirable for Rexx?

# Classifying drive-relative tests: Python's role (2/2)

- Similarly, when one writes `"D:path\to.file"`, this is a *relative* filename. Relative to what? To *the current directory of the (*`"D:"`*) drive*. If that is `"D:\dir"`, then we get `"D:\dir\path\to.file"`.
- What should it mean, in this case, that the "same" directory is checked before the current directory? It could well mean the following: if the same directory was `"D:\dir2"`, we could construct `"D:\dir2\path\to.file"`, and similarly for all the directories in the different paths (when the drive letter would not match, the current drive and directory would be used).
- Currently, no Rexx interpreter exhibits such a behavior, but Python's `pathlib` module does: for Python, `Path("D:/this/is") / "D:a/long/path"` (under Windows) is `WindowsPath('D:/this/is/a/long/path')`.
- Is this behavior desirable for Rexx?

# Special tests

- We have already examined the behavior of `CMD.EXE`, the `SearchPath` Windows API, and Python's `pathlib` module.
- If we now examine the behavior of the `#include` directive in C/C++ (both for `gcc` and `cl`), we will see that all the comon tests pass.
- C/C++ uses the concept of the same directory, and has no problems wih dot-relative or upward relative filenames (indeed they are quite common, although there are religious wars about their convenience).
- This should mean something to us: `#include` and `::requires` play similar roles.
- As a curiosity, the concept of sameness in Visual Studio is *recursive*.

# Finally, is there a Rexx way of doing things? (1/2)

- After examining the results of our tests, I do not think we have enough elements to support the idea that there is "a Rexx way of doing things" for external search.
- What we can see is a slow (and sometimes hesitant) evolution from a more limited scripting language paradigm to a full programming language paradigm.
- Older versions fall in the scripting language side. That is probably the reason why their behavior is the same as the Command Line Interpreter.
- Object Rexx makes the jump to the full language paradigm, and it incorporates constructs found in other programming languages.
- ooRexx goes one step beyond (with the concept of the "same" directory), and then backpedals when handling the ".\" or "..\" cases.

# Finally, is there a Rexx way of doing things? (2/2)

- Maybe there is no "Rexx way of doing things" for external search, but we have learned many things!
- And maybe by expanding our understanding still more, we will be able to come to a set of insights and decisions:
- A decision concerning the ooRexx anomaly, and
- A set of recommendations for a future Rexx standard (Language level 7?).
- And, in general, new knowledge for the RexxLA Architecture Review Board (ARB) to ponder.
- This might be very useful for new implementations and variants of Rexx.

# The Search Order for External Files

Part IV

# Modeling external search algorithms

The ontological question: What is an external search algorithm?
- Locations and qualifiers
- Location-first and qualifier-first algorithms. Location exception clauses.
- Qualifier exception clauses. The composition algorithm

A system of classes to model external search algorithms

A proof-of concept prototype for a pluggable external search algorithms system

# What is an external search algorithm? (1/3)

- To expand our understanding of external search, we have to address the ontological question: What is an external search algorithm?
- We will have to identify the aspects that are common to all the search algorithms (or, at least, to the algorithms we are studying).
- For example, all the algorithms manage a list of locations (e.g., directories) and a list of qualifiers (e.g., file extensions).
- [We use the term "location" instead of "directory" to be able to accommodate, in the future, operating systems where the fundamental file collection unit is not a directory (for example, VM minidisks), and similarly for "qualifiers", which will be *extensions* under Windows, for example but may be *filetypes* and *filemodes* under VM.]

# What is an external search algorithm? (2/3)

- Some algorithms are location-first (i.e., they search for all qualifiers in a given location, and then proceed to the next location), and some others are qualifier-first (they look for a qualifier in all the locations, and then they proceed with the next qualifier).
- [For example, Regina Rexx is directory-first, while ooRexx is extension-first.]
- Most algorithms have a location exception clause: when the filename to search for has a certain form, not all locations are checked, but only a designated subset of those.
- [For example, Regina Rexx limits its search to the current directory as soon as it finds a separator like `"/"` in the filename, while the ooRexx location exception algorithm is much more nuanced (and partially undocumented).]

# What is an external search algorithm? (3/3)

- Similarly, most algorithms include a qualifier exception clause: depending on the form of the filename, only a designated subset of the qualifiers is checked.
- [For example, Regina Rexx does not try to add an extension when the file has *a known extension* (that is, one of the predefined extensions, or one of the extensions supplied in the `REGINA_SUFFIXES` environment variable). For ooRexx the test is much simpler: if the filename contains a dot (`"."`), `then no extensions are added.`]
- Every search algorithm defines a composition algorithm that combines a certain location, a certain filename and a certain qualifier (which may be empty) and produces a list of (hopefully absolute) file names.
- [For example, the Unix-like version of ooRexx tries the supplied filename as-is, and then, if different, in lowercase.]

# Modeling external search algorithms (1/3)

- We have written a set of External Search classes to model the behavior of the different interpreters and environments. Instances are created by providing a class-specific set of parameters. Once initialized, they provide a `search` method that resolves a filename according to the specified external search algorithm.

```
/* Our base class will be an abstract class...                                  */
::Class ExternalSearch Public Abstract
/* ...with two direct subclasses, also abstract.                                */
::Class QualifierFirstExternalSearch Subclass ExternalSearch Public
::Class LocationFirstExternalSearch  Subclass ExternalSearch Public
/* ooRexx external search is extension- (qualifier-)first                       */
::Class ooRexxExternalSearch         Subclass QualifierFirstExternalSearch Public
/* Regina Rexx external search is location- (directory-)first                   */
::Class ReginaRexxExternalSearch     Subclass LocationFirstExternalSearch  Public

mySearch  = ReginaRexxExternalSearch~new  /* This sequence is equivalent to      */
myRoutine = mySearch~search(Routine)      /*   "Call Routine args", but          */
.Routine~newFile(myRoutine)~call(args)    /*   with the Regina search order      */
```

# Modeling external search algorithms (2/3)

- External search objects can be instantiated with their default values, which are those documented in the respective manuals

```
1  mySearch = ooRexxExternalSearch~new                    /* Default values */
```

- Or they can be fully customized at object creation time..

```
1  mySearch = ooRexxExternalSearch~new(                    -
2      (                                                   - /* Directories,   */
3          "same=<same directory>",                        -
4          "current=<current directory>",                  -
5          "application=<application-defined path>",       - /* paths, and     */
6          "rexx_path=<path>",                             -
7          "path=<path>",                                  -
8      ),                                                  -
9      (                                                   - /* extensions    */
10         "same=<same extension>",                        -
11         "application=<application-defined extensions>", -
12     )                                                   -
13 )
```

# Modeling external search algorithms (3/3)

- Every search algorithm class has to be instantiated with its own set of class-specific parameters.

```
1  mySearch = ReginaRexxExternalSearch~new(                    -
2      (                                                       -
3          "regina_macros=<path>",                             - /* Paths,          */
4          "current=<current directory>",                      - /* directories     */
5          "path=<path>",                                      - /* and             */
6      ),                                                      -
7      (                                                       - /* extensions      */
8          "same=<same extension>",                            -
9          "regina_suffixes=<list>",                           -
10     )                                                       -
11 )
```

# The composition operation. Advanced modeling

- The `compose` method of `ExternalSearch` takes as arguments a directory, a filename and a possibly empty extension, and attempts to compose them into a (hopefully absolute) filename. This operation is not so simple as it may seem at first glance, because it has to consider the cases where the filename may itself be absolute, the directory part can be relative, and so on.
- `ExternalSearch` has a settable boolean attribute called `driveRelative`, with a default value of `.false`. When `driveRelative` is `.true`, the composition operation is slightly modified, so that drive-relative filenames are resolved like in the `pathlib` Python module. This is experimental at the moment.

# Class ooRexxEnhancedExternalSearch

- Class `ooRexxEnhancedExternalSearch` fixes the anomaly by removing the checks for `".\"` and `"..\"`, and by additionally setting `driveRelative` to `.true`. All 48 tests pass when we use this enhanced external search algorithm.

# Pluggable external search algorithms: a prototype

- Using the security manager feature of ooRexx, we have devised an experimental system of pluggable external search order algorithms. This is implemented, as a proof-of-concept, by the "[]=" class method of the ExternalSearch class. The following code fragment illustrates the technique.

```
1  /* To be able to plug a security manager, we need a Routine object        */
2  routine = .Routine~newFile("/path/to/my/program.rex")
3
4  /* Routine will be called, but with our enhanced search order in effect   */
5  .ExternalSearch[routine] = .ooRexxEnhancedExternalSearch
6
7  /* Now call our routine with the appropriate parameters. Every CALL       */
8  /* in "program.rex" will be resolved according to the Rexx Enhanced       */
9  /* External Search algorithm.                                             */
10 routine~call(parameters)
11
12 ::Requires ExternalSearch
```

# Pluggable external search algorithms: a prototype

- This technique allows to run the `sotest.rex` test program against a number of different external search algorithms, without having to switch interpreters or operating systems.

```
1  program = "D:\Dropbox\ooRexx\sotest\sotest.rex"  /* Maybe                  */
2  routine = .Routine~newFile(program)
3  /* Install the Regina search order using the ooRexx security manager      */
4  .ExternalSearch[routine] = .ReginaRexxExternalSearch
5  /* All programs called from sotest.rex will be searched using the         */
6  /* Regina search order algorithm                                          */
7  routine~call()
8
9  ::Requires ExternalSearch
```

Part V

# Appendices

# Further work

- Drive-relative tests need more work.
- Study what happens when the components of a path are themselves relative (i.e., “dir”).
- Can our algorithms be adapted to the the z/VM world? To z/OS? To z/VSE?...
- Improve the security manager integration (but see bug #1886).
- ...

# Acknowledgements (1/2)


- I would like to start by thanking Joan Batet, Lisa Mc Connell, Silvina Fernández, Xavier Navarro, Joel Padulles, David Palau and Francesc Rosés, who have contributed to enhance this document by patiently reading several drafts, finding typos, indicating paragraphs that were unclear, and suggesting all kinds of corrections and enhancements.
- To Erich Steinböck, who pointed me to the relevant parts of the interpreter code to continue my investigation.
- To Rony G. Flatscher, who was especially kind and encouraging, and patiently introduced me to the arcanes of preparing and submitting documentation, code and test patches for ooRexx.
- To René Vincent Jansen, who invited me to participate in the RexxLA Architecture Advisory Council, a.k.a Architecture Review Board (ARB), and provided resources in the GitHub `rexx-repository` of RexxLA for me to store my programs, test results, etc.

# Acknowledgements (2/2)


- To the members of the Architecture Review Board itself, for their critical comments and encouragement.
- To my colleagues at Espacio Psicoanalítico de Barcelona, for bearing with me while I submerged myself in this research, listening to my musings, and being loving and supportive.
- To the participants of the different mailing lists, especially the ARB list, the developers list, and the RexxLA list.
- To the Rexx Language Association, for stimulating my creativity.
- And of course to the ooRexx developers, for maintaining and enhancing a wonderful version of the Rexx language.

Questions?

# References

- Most of the material covered here is addressed in more detail in the companion document, *The Search Order for External Files*, `https://www.epbcn.com/pdf/josep-maria-blasco/2023-05-16-The-search-order-for-external-rexx-files.pdf`.
- The test programs, result sets, etc., can be downloaded from the `https://github.com/RexxLA/rexx-repository/tree/master/ARB/standards/work-in-progress/search-order` GitHub directory and from `http://www.epbcn.com/pdf/josep-maria-blasco/2023-05-16/`.

[**Editor's note**: This is a shortened version, prepared for publication, of the original presentation slides, which can be downloaded at `https://www.epbcn.com/pdf/josep-maria-blasco/2023-05-16-The-search-order-for-external-rexx-files-Slides.pdf`. Animated slides have been substituted by static slides. Apart from this minor modification, the two versions are identical.]