Repoz

Version : 1.0
Date : 19/08/2006
Author : ¨Patrick Germain Placidoux

SUMMARY

1 OBJECTIVE

This document describes the Repoz purpose and the Repoz global commands.

Note: repoz is an OpenSource project at sourceforge.net, contact : repoz@gmx.com.

Anticipating the following content of this documentation, Repoz global commands are those commands that are available all along the interpreter, for any Repoz mode.

2 INTRODUCTION

Repoz is a plateform where to manage bunches of files in an integrated manner, as they would within a former repository.

Files can be opened into Repoz, changed and saved one by one, or globally.
Hence the save operation will check that none of the files is locked or has changed during the interval.

The set of files can be backuped at their initial state before any change occured, using the -b lauching the repoz command (e.g.: **$ repoz -b -B /where/to/backup).**

3 PRERIQUISITES

Oses (every where python works)

Linux, AIX ®, Windows ®

Langages

Python >= 2.4 < 3

4 INSTALLATION

Download repoz at www.sourceforge.net

Unzip the file repoz_#.##.zip in the directory of your choice.

Put the path <REPOZ_INSTALL_PATH>/bin in your path environment variable.

Type :

> repoz on Windows
or
> repoz .sh on Unixes
(dont forget to run chmod ugo+x *.sh in the bin directory)

5 REPOZ SESSION

5.1 THE REPOZ SESSION LIFE CYCLE

A Repoz Session starts with the interpreter launched by the repoz command and ends with "Ctrl z" or "exit".

```
$ repoz (Begin the Repoz Session)
#-------------------------------------------------------------------------------#
#                                                                               #
#                        WELCOME TO REPOZ                                       #
#                  The file content management factory                          #
#                                                                               #
#                                                                               #
#  Type H or HELP for help on global commands.                                  #
#  Type H <command> for help on a global command.                               #
#  Type h or help for help on a specific processor.                             #
#  Type h <command> for help on a specific processor command.                   #
# ...                                                                           #
#-------------------------------------------------------------------------------#

>>> Repoz commands >>>

:>^Z (End the Repoz Session)
```

All the operations done during a Repoz session may be recorded using the -r (--record) option.

e.g.:
$ repoz -r -B /where/to/backup

Note :
The record option use the same backup directory as the --with_backup option (-b) to store the recorded log file.

5.2 REPLAYING A REPOZ SESSION

When a repoz Session is launched with the -r (--record) option :

e.g.:
$ repoz -r -B /where/to/backup

All the commands typed into the Repoz interpreter are recorded into a file named like the following into the backup directory.

/where/to/backup/Repoz_<time_stamp>_record.log

e.g.:
/where/to/backup/Repoz_D20100522T145812_record.log

Afterward, this file may **be played again** on Repoz like this :

$ cat /where/to/backup/Repoz_D20100522T145812_record.log | repoz (for unixes)
C:\> type /where/to/backup/Repoz_D20100522T145812_record.log | repoz (for windows)

or

$ repoz -f /where/to/backup/Repoz_D20100522T145812_record.log

This shows that a set of batch commands from any text file can be played on repoz
either throught a pipe or from a file using the -f (--file) option.

Note:
The content below covers the commands and functionalities available during a Repoz Session.

6 PROCESSORS

To provide a standard way to manage file's content, Repoz supports a set of processors.
There is one processor per type of file:

- the **xpc** processor for xml files (.xml files)
- the **apc** processor for property file (.attrs files
- the **tpc** processor for template files (.tmpl files)

Notes for apc :
In python scheme properties are better called: Attributes than properties.
This is why we prefer to sufix this kind of files by .attrs instead of .properties.
But dont worry the way you sufix your file do not matter at all considering that you always pass the complete file name to the command.
What really matter at all is to choose the wright processor for the wright type of the file content.

Each processor matches a particular file type.

e.g.:
:>**xpc** -F /where/is/my/xml/file.xml to manage an **xml file**.
:>**apc** -F /where/is/my/attrs/file.attrs to manage an **attrs file**.
:>**tpc** -D /where/is/my/template/file.tmpl to manage a **template file**.

Except for template (tpc), the -F (or --file_source) option point to the input source file.

Each file **may come** with another file called its **descriptor file**.
Think about a descriptor file as an equivalence (but more intuitive) representation of the classical xml schema's that are xsd or dtd files.

A descriptor file **describes all the Attributes (and/or tags), types and default values supported** by a particular file.

Using descriptors allows **judicious validation checks when updating the content of the file**.

Note:
If no descriptor file is used, no check at all is performed, when changes are made into the file.

A Descriptor file is supported on a processor command using the **-D (--files_desc) option**.

e.g.:
:>xpc -F /where/is/my/xml/file.xml -D /where/is/my/descriptor/file.desc.xml
:>apc -F /where/is/my/attrs/file.attrs -D /where/is/my/descriptor/file.desc.attrs

The following concerns the xpc processor only:
There is another kind of descriptor file called the **restrictor file**, that can be used to apply custom autorisation policy checks on specific tags/attributes/values of an xml file.

A Restrictor file is supported on the xpc processor command only, using the **-R (--files_desc) option**.

e.g.:
:>xpc -F /where/is/my/xml/file.xml -D /where/is/my/descriptor/file.desc.xml -D /where/is/my/restrictor/file.rest.xml

On any processor command the **option -f (file),** can be used to signal the target file to be writen on **save**.

e.g.:
:>xpc -F /where/is/my/xml/file.xml -f /where/is/my/xml/new_file.xml

If -f is not guiven the **source file /where/is/my/xml/file.xml is overwriten on save**.

Note:
Dont confuse, -F in upper case stands for descriptor file, -f in lower case stands for the saved target file.

6.1 CREATING/DESTROYING PROCESSORS

Once repoz is launched the first thing to do is to create a processor for any file you want to manage.

6.1.1 *Creating processor*

e.g.:
:>xpc -F <repoz_intallation_dir>/repoz/samples/test.xml
*New processor with **alias:test**, created and mounted.*

:test:/tag1>xpc -F <repoz_intallation_dir>/repoz/samples/test.xml
*New processor with **alias:test0**, created. (use mount test0, to mount it)*

:test:/tag1>xpc -F <repoz_intallation_dir>/repoz/samples/test.xml -s
*New processor with **alias:test1**, created. (use mount test1, to mount it)*

:test:/tag1>apc -F <repoz_intallation_dir>/repoz/samples/test.attrs -s
*New processor with **alias:test2**, created. (use mount test2, to mount it)*

:test:/tag1>tpc -D <repoz_intallation_dir>/samples/test.tmpl -a myalias
*New processor with **alias:myalias**, created. (use mount myalias, to mount it)*

Note about the sample directory:
To run the Repoz samples of this documentation into a Kikonf installation directory replace :
<repoz_intallation_dir>/samples by: **<kikonf_intsallation_dir>/samples/repoz**.

- Alias

The first processor is created and aliased by the name "test".
Each new processor is aliased by an unique name in repoz.
If the option -a (--alias) is not used with the processor command (unlike in the last processor myalias), the alias is guessed by Repoz, from the name of the file, with no sufix.

If the same file name appears more than once (like for the second processor test0),
a digit from 0-->n is concatenated with this name.

- Multiple processors for the same input file:

From alias test to test1, 3 processors are mounted on the same file: test.xml.
This is permitted as doing like this allows background update operations over one or more file(s), then paste the relevant portion to the file destinated to be save.

Please note that the proecessor test1 is created with option -s,
this indicates that only this one is allowed to the save operation.

Afterward using the specific save or the global: save all command, <== A FAIRE
will save the content of the processor test1 to the file test.xml.

In that situation only one of the processor must be marked to be saved.

6.1.2 Destroying processor

:test:/tag1>unpc test (todo)
Processor with alias:test, unmounted and destroyed.

:test:/tag1>unpc test0
Processor with alias:test0, destroyed.

:test:/tag1>unpc myalias
Processor with alias:myalias, destroyed.

Destroying processors during a Session allows to free the memory.

6.2 MOUNTING/UNMOUNTING PROCESSORS

Once one processor is created you can **switch from one to another using the mount/unmount** command.

(continuing from the previous sample):

6.2.1 Mounting

e.g.:

```
:test:/tag1>mount test0
:test0:/tag1>
:test0:/tag1>mount myalias
:myalias>
```

The mount command may be replaced by "**<ALIAS>:**"

e.g.:

```
:test:/tag1>test0:
:test0:/tag1>
:test0:/tag1>myalias:
:myalias>
```

6.2.2 Unmounting

e.g.:

```
:test:/tag1>unmount test0 (todo)
:>
```

7 THE REPOZ INTERPRETER

Once your processor is mounted you can run content management operations on it.

For this purpose repoz provide multiple ways to extend your strength.

Actually the Repoz interpreter is a multi-interpreters.

7.1 MODES

Repoz supports langages interpreters :

- the **xpath** langage
- the **query** langage
- the **local** os shells langage
- the **Python** langage

The **default** interpreter is **xpath**.

The command: **mode** switches from an interpreter to another.

:> is the **xpath mode** (switches to the xpath langage. The prompt **symbol** is **":"**)
%> is the **ql mode** (switches to the query langage. The prompt **symbol** is **"%"**)
!> is the **os mode** (switches to the Local os shells langage. The prompt **symbol** is **"!"**)
?> is the **python mode** (switches to the Python langage. The prompt **symbol** is **"?"**)

Note:
The xpath mode is actually an xpath implementation.
Type h or help under any of these modes to see how to use them.
These modes are also seen deeper farther below.

The xpath (:), ql (%) and shell (!) modes are "command" interpreters and run commands.
The python mode is a script interpreter and runs python instructions.

The simplest way to switch from a mode to another, is to use the **"mode"** symbole.

e.g.:
:>**%** *(typing "%" from the xpath mode and Enter key)*
%> *(switches to the ql mode)*

%>! *(typing "!" from the ql mode and Enter key)*
!> *(switches to the os mode)*

Note:
xpath and ql langages implementations are specifics per processor to view their specific help per processor:
create the processor, mount it and type h or help.
Shell and python interpreters are generics for all processors.

7.1.1 *Arbitrary Modes*

It is possible eventually to run a mode from anoher.

e.g.:

```
:> # This is the default xpath mode.
:>xpc -F <repoz_intallation_dir>/repoz/samples/test.xml  # This creates a processor on the xml file:  test.xml and mount it.
New processor with alias:test, created and mounted.
```

In this mode we are suppose to run xpath commands like the following.

e.g.:

```
:test:/tag1>ls
/tag1> attr1:a attr2:b
tag2
tag3
tag2
tag2
```

or

```
:test:/tag1> cd tag2
:test:/tag1/tag2>ls
```

or

```
:test:/tag1>ls tag3
/tag1/tag3> attr1:f attr2:g attr3:A value with spaces !
```

or

```
:test:/tag1>ls tag2@attr1=h/tag4
/tag1/tag2/tag4>
tag5
```

Actually this mode like in any of the other modes, it is also allowed to **Arbitrary** run another interpreter prefixing its command by its symbole.

e.g.:

```
:test:/tag1>!dir # This runs a Local os shell command, try "ls" for unixes.
 Le volume dans le lecteur E s'appelle BIG
 Le numéro de série du volume est CCCD-48CE

 Répertoire de <repoz_intallation_dir>\bin

23/05/2010  23:25    <REP>          .
23/05/2010  23:25    <REP>          ..
23/05/2010  23:19               94 repoz
23/05/2010  23:18              138 repoz.bat
```

or

```
:test:/tag1>%select * at tag3 # This runs an xql command.
        *tun    attr2   attr3                           attr1                           *text
/tag1/tag3
        I3      g       A value with spaces ! f         [aaaa1,bbbbbbbb2]
```

7.1.2 Complex Python instructions using Arbitrary Modes

```
:test:/tag1>?   # This switches to the Python interpreter.
?test:/tag1>
?test:/tag1>for i in range(1,4): #  Now we type an imbricated Python instruction.
?test:/tag1>    :ls tag3 # continuing here !
?test:/tag1>
/tag1/tag3> attr1:f attr2:g attr3:A value with spaces !

/tag1/tag3> attr1:f attr2:g attr3:A value with spaces !

/tag1/tag3> attr1:f attr2:g attr3:A value with spaces !
```

This runs the **xpath** command : **"ls tag3"** , 3 times (into the range from 1 to 4) as required by the "for" instruction.

Note:
This mechanism will become more powerfull when we'll figure out, how to use command returned values further below in this documentation.
Especially in the chapter : *"Complex Python instructions using Arbitrary Modes and Returned Values"*.

7.2 VARIABLES

The Global variables is a mechanism to **share variables accross modes**.

7.2.1 *Variables Declaration*

:>var myvar = myvalue # *Creates the global variable myvar.*

:>var myvar # *shows the content of the global variable myvar.*
myvalue

:>unvar myvar # *unset the global variable myvar.*

:>var # *shows all the global variables.*
var: ***repoz****, value: <__main__.Shell instance at 0x00B70B20>*
var: rs, value: None
var: re, value: None
var: ***r****, value: {'re': None, 'ro': None, 'rc': None, 'rs': None}*
var: rc, value: None
var: ro, value: None
:>

The special variable: **repoz,** represents the repoz instance itself that could be manipulated within the python mode (experts only).

The special variable **r** also splitted into 4 standalone variables represents the values returned by the last previous command.

7.2.2 *Content evaluation*

Because xpath (:), ql (%) and shell (!) modes are string literal interpreters,
the variable contents are evaluated as CoolTyped expressions.
This is not the case for the python (?) mode which expects real python expressions.

Note about :
Talking shortly:
- CoolTyping is a serializing/deserializing tools for python.
- A CoolTyped Expression is a leteral expression with no: **"** and no: **'.**

Then the following characters **","**, **":"**, **"("** and **"{"** are interpreted as python artefacts.

For more information about CoolTyping see the CoolTyping project's documentation at sourceforge.net or into the directory <REPOZ_INSTALL_PATH>/doc, or into the directory

<KIKONF_INSTALL_PATH>/doc if you are using Kikonf, or on the www.kikonf.com site.

e.g:

```
:>var myvar = a value with spaces
:>var myvar
:>a value with spaces
```

In python this should be :

```
?>var myvar = 'a value with spaces'
?>var myvar
?>a value with spaces
```

And using complexe expressions:

e.g:

```
!>var myvar = (a, another string with spaces, 1, False)
!>var myvar
('a', ' another string with spaces', ' 1', ' False') # CoolTyping turns the expression to a tuple.
```

In python this should be :

```
?>var myvar = ('a', 'another string with spaces', 1, False)
?>var myvar
('a', ' another string with spaces', ' 1', ' False')
```

Note:
For xpath (:), ql (%) and shell (!) modes, contents are reverse CoolTyped when used.
e.g (with the example above) :

```
!>echo $myvar
Var replacement: myvar to:(a, another string with spaces, 1, False)
Var replacement: new line is:echo (a, another string with spaces, 1, False)
(a, another string with spaces, 1, False)
```

This is not the case in the python (?) mode:

```
?>$myvar
Var replacement: myvar to:('a', 'another string with spaces', 1, False)
Var replacement: new line is:('a', 'another string with spaces', 1, False)
('a', 'another string with spaces', 1, False)
```

- **Only in python mode**, Variables can be use straightforward:

e.g.:

```
?>print $myvar
Var replacement: myvar to:('a', 'another string with spaces', 1, False)
Var replacement: new line is:print ('a', 'another string with spaces', 1, False)
('a', 'another string with spaces', 1, False)
```

is equivalent to :

```
?>print myvar
('a', 'another string with spaces', 1, False)
?>
```

7.2.3 Usage

A variable may be re-used into any of the interpreter commands like this : $VAR or ${VAR}

e.g.:

```
:>xpc -F <repoz_intallation_dir>/samples/test.xml -D <repoz_intallation_dir>/samples/test.desc.xml
New processor with alias:test, created and mounted.
:test:/tag1>ls tag3
/tag1/tag3> attr1:f attr2:g attr3:A value with spaces !

:test:/tag1>var myvar = tag3         # Creating the variable: myvar
:test:/tag1>ls $myvar                # Using the Variable: myvar into the ls command.
Var replacement: myvar to:tag3
Var replacement: new line is:ls tag3 # Operates the Variable replacement, runs the resulting
command and shows the result:
/tag1/tag3> attr1:f attr2:g attr3:A value with spaces !
```

e.g.:

```
:test:/tag1>ls tag2@attr1=h/tag4
/tag1/tag2/tag4>
tag5
:test:/tag1> var a=tag2              # Declaring the variables: a and b.
:test:/tag1> var b=h/tag4

:test:/tag1>ls $a@attr1=${b}         # Using a and b into the ls command.
Var replacement: a to:tag2
Var replacement: new line is:ls tag2@attr1=${b}
Var replacement: b to:h/tag4
Var replacement: new line is:ls tag2@attr1=h/tag4  # Operates the Variable replacement, runs the
resulting command and shows the result:
/tag1/tag2/tag4>
tag5
```

7.3 RETURNED VALUES

As seen above, for the xpath (:), ql (%) and shell (!) modes, the values returned by the commands are stored into special global variables.

```
!>var
var: repoz, value: <__main__.Shell instance at 0x00B70B20>
var: rs, value: None
var: re, value: None
var: r, value: {'re': None, 'ro': None, 'rc': None, 'rs': None}
var: rc, value: None
var: ro, value: None
```

The special Variables **r, rs, re, rc, ro** store the values returned by the commands:

- **rc** : returned code of the previous command, if relevant.
- **rs** : contains the stdout output of the previous command, if relevant.
- **re** : contains the returned exception raised by the previous command, if relevant.
- **ro** : contains the returned object by the previous command, if relevant.
-
- r : Is a dict representation of the previous variables.

e.g.:

```
!test:/tag1>echo 'what to say ?'
'what to say ?'

!test:/tag1>var
var: repoz, value: <__main__.Shell instance at 0x00B70B20>
var: rs, value: 'what to say ?'

var: re, value: None
var: r, value: {'re': None, 'ro': None, 'rc': 0, 'rs': "'what to say ?'\n"}
var: rc, value: 0
var: ro, value: None
```

e.g.:

```
:>xpc -F <repoz_intallation_dir>/samples/test.xml -D <repoz_intallation_dir>/samples/test.desc.xml
New processor with alias:test, created and mounted.
:test:/tag1>ls
/tag1> attr1:a attr2:b
tag2
tag3
tag2
```

```
tag2

:test:/tag1>var
var: repoz, value: <__main__.Shell instance at 0x00B70B20>
var: rs, value: /tag1> attr1:a attr2:b
tag2
tag3
tag2
tag2

var: re, value: None
var: r, value: ...
var: rc, value: 0
var: ro, value: [<lib.epicxmlp.Node instance at 0x00BFD580>] # A list of the found xml nodes (may
be used in the python mode).
...
```

7.4 COMPLEX PYTHON INSTRUCTIONS USING ARBITRARY MODES AND RETURNED VALUES

This python imbricated command works on the object returned by the xpath command:ls.

```
:>xpc -F <repoz_intallation_dir>/samples/test.xml -D <repoz_intallation_dir>/samples/test.desc.xml
New processor with alias:test, created and mounted.
?test:/tag1>
?test:/tag1> # Running the complex python instruction
?test:/tag1>
?test:/tag1>for tag in ('tag3', 'tag2@attr1=h/tag4'):
?test:/tag1>    print '[Passe for tag: ' + tag + ']'
?test:/tag1>    var tag = tag
?test:/tag1>    :ls $tag
?test:/tag1>    print 'Printed by python>>> Tag Name:', ro[0].getTag(), ' Attributes:', ro[0].getdAttrs(), '\n\n'
?test:/tag1>
?test:/tag1> # result
?test:/tag1>
[Passe for tag: tag3]
Var replacement: tag to:tag3
Var replacement: new line is:ls tag3                # Replacement for :ls $tag
/tag1/tag3> attr1:f attr2:g attr3:A value with spaces !   # Output for: :ls $tag

Printed by python>>> Tag Name: tag3  Attributes: {'attr2': 'g', 'attr3': 'A value with spaces !', 'attr1': 'f'}
```

```
[Passe for tag: tag2@attr1=h/tag4]
Var replacement: tag to:tag2@attr1=h/tag4
Var replacement: new line is:ls tag2@attr1=h/tag4          # Replacement for :ls $tag
/tag1/tag2/tag4>                                           # Output for: :ls $tag
tag5

Printed by python>>> Tag Name: tag4  Attributes: {}
```

8 SHOW AND SAVE COMMANDS

8.1 SHOW

To show the content of a file, type the command: **show**.

e.g.:
:>xpc -F <repoz_intallation_dir>/samples/**test.xml**
New processor with alias:test, created and mounted.

:test:/tag1>show

```
<tag1 attr1='a' attr2='b'>
   <tag2 attr1='c' attr2='d' attr3='e'/>
   <tag3 attr1='f' attr2='g' attr3='A value with spaces !'>
      aaaa1
      bbbbbbbb2
   </tag3>
   <tag2 attr1='h' attr2='i' attr3='j' attr4='k'>
      <tag4>
         <tag5 attr1='True' attr2='m' attr3='n'/>
      </tag4>
   </tag2>
   <tag2 attr1='o' attr2='p'/>
</tag1>
```

:test:/tag1>cd tag3
/tag1/tag3> attr1:f attr2:g attr3:A value with spaces !

:test:/tag1/tag3>show

```
<tag3 attr1='f' attr2='g' attr3='A value with spaces !'>
   aaaa1
   bbbbbbbb2
</tag3>
```

e.g.:
apc -F <repoz_intallation_dir>/samples/**test.attrs**
New processor with alias:test, created and mounted.

:test:/>show

```
#
champ1=valeur1

# champ2_help whith spaces
champ2=[{AAA:ccccA2,BBB:ccccB2},{AAA:bbbbA1,BBB:bbbbB1}]

# champ2_help in more. Than one. lines whith spaces. )
```

champ3={ccc:caa,ddd:daa}

8.2 SAVE

To save the content of a file type the command: **save**

e.g.:
:>xpc -F <repoz_intallation_dir>/samples/**test.xml -s** # *The preocessor must have been created with the -s (--to_save) option to be writable.*
New processor with alias:test, created and mounted.

:test:/tag1>save
File:<repoz_intallation_dir>/samples/test.xml saved !

The processor may be saved to a target distinct from the original file.

e.g.:
:>xpc -F <repoz_intallation_dir>/samples/**test.xml -s -f newfile.xml**
New processor with alias:newfile, created and mounted.

:newfile:/tag1>save
File:newfile saved !

Note:
For the **template processor**, the target file using the option -f creating the processor is required before using the save commmand.

9 ANNEXE KIKACT AND KIKARC SPECIFIC OPTIONS

When repoz is used **inside the Kikonf project**, the following commands, behing the scene, will open the files referenced by the specific explicitally named, Kikonf Action or the Kikonf custom xml file.

9.1 WORKING WITH KIKONF ACTIONS

Syntax:

:> xpc --kikact <BAL>
BAL: Basic Action Locator. e.g.: was.cache

The action, descriptor and restrictor files (if relevent) for this action are automatically opened.

e.g.:
:> xpc --kikact was.cache

9.2 WORKING CUSTOM XML FILE

The following command, will open the named xml file and all its related descriptors.

Syntax:
:> xpc --kikarc </my/custom/xml/file.xml>

Foreach Action defined into the Kikonf custom xml file:
the Action descriptor file and the **Action restrictor file** (if relevent) **are merged** into the global auto **deducted descriptor file** and restrictor file (if relevent) for the custom xml file.

e.g.:
:> xpc --kikarc c.xml

9.3 KIKARC/KIKACT EXTENDED OPTIONS

The following options are the ones supported when using repoz with one of the **--kikact or --kikarc** options (note they are the sames as those defined for the native Kikonf commands (kikact, kikarc, kikupd, ...).

-C KIKONF_ATTRS, --cattrs=KIKONF_ATTRS

(optional) The path to a custom kikonf.attrs file.
When you don't want to use the default one into the <KIKONF_INSTALL_DIR>/conf directory. Note: If this option is not set, kikact tries to retreive its value from an environment variable named KIKONF_CATTRS.
If neither the option or the environment variable are set kikact use the default kikonf.attrs file into the <KIKONF_INSTALL_DIR>/conf directory.

-c ACTION_DIR, --cxml=ACTION_DIR

(optional) The path to the directory of the action files.
If not given, sample action files are retreived from the <KIKONF_INSTALL_DIR>/actions directory.
Note: If this option is not set, kikact tries to retreive its value from an environment variable named KIKONF_CXML.
If neither the option or the environment variable are set kikact use the default <KIKONF_INSTALL_DIR>/actions directory.

-r RESTRICTOR_DIR, --crst=RESTRICTOR_DIR

(optional) The path to a directory of the action's restrictor files. Note: If this option is not set, kikact tries to retreive its value from an environment variable named KIKONF_RST.