

# Kikarc

Version : 1.0

Date : 19/08/2006

Author : Patrick Germain Placidoux

Copyright (c) 2007-2008, Patrick Germain Placidoux

All rights reserved.

## SUMMARY

1	<a href="#">1 Introduction</a>	4
2	<a href="#">2 Overview</a>	5
2.1	<a href="#">2.1 The Custom xml file</a>	5
3	<a href="#">3 Kikarc options</a>	9
3.1	<a href="#">3.1 Usage</a>	9
3.2	<a href="#">3.2 Help and logs</a>	10
3.3	<a href="#">3.3 Paths</a>	11
3.4	<a href="#">3.4 Operations</a>	13
3.4.1	<a href="#">3.4.1 inject (-o inject):</a>	14
3.4.2	<a href="#">3.4.2 run (-o run):</a>	15
3.4.3	<a href="#">3.4.3 inrun (-o inrun): this is the default.</a>	16
3.4.4	<a href="#">3.4.4 extract (-o extract):</a>	17
3.4.4.1	<a href="#">3.4.4.1 Action keys</a>	18
3.4.4.2	<a href="#">3.4.4.2 Extract output</a>	21
3.4.4.3	<a href="#">3.4.4.3 Link with the kikact command</a>	21
3.4.5	<a href="#">3.4.5 remove (-o remove):</a>	23
3.5	<a href="#">3.5 Exits</a>	25
3.5.1	<a href="#">3.5.1 server</a>	25
3.5.2	<a href="#">3.5.2 cluster</a>	25
3.5.3	<a href="#">3.5.3 application</a>	25
3.5.4	<a href="#">3.5.4 Link with the kikact command</a>	25
3.6	<a href="#">3.6 Export</a>	26
3.6.1	<a href="#">3.6.1 Link with the kikact command</a>	26
3.7	<a href="#">3.7 Partial custom file execution</a>	27

4	<a href="#">4 Annexe 1: Kikarc all options</a>	30
5	<a href="#">5 Annexe 2: Action plugins</a>	34
6	<a href="#">6 Annexe 3: Exit plugins</a>	35

## **1 INTRODUCTION**

---

Kikarc stands for ***kick architecture !***

Kikarc is the kikonf command for custom xml files management.

## 2 OVERVIEW

---

### 2.1 THE CUSTOM XML FILE

---

A custom xml file is any xml file containing one or more Action tag(s).

Because of the adaptive ability of the kikonf parser, a kikarc command can be fed with any xml file including one or more Action tag(s).

Note:

Actually if the custom xml file includes no Action tag at all, nothing will simply happen running the kikarc command.

Reading the xml file the Action tags are treated in their appearance order per injector.

Note:

Actions are associated to a specific software and it exists one injector per software.

See the kikonf core documentation for more information.

Anyway you don't need to know this to run it.

Example 1:

Given this xml file, that could be any xml file: c.xml

```
<mytag1>
  <mytag2/>
  <mytag3/>
  <mytag4>
    </mytag5>
  </mytag4>
</mytag1>
```

Let's run it:

```
> kikarc c.xml -v3
```

Nothing happens because there is no node supporting any Action tag.

Example 2:

Let's include an Action node somewhere let's say at mytag1.mytag4.mytag5 node.  
And another one at mytag1 node, but near to the end tag.

```
<mytag1>
  <mytag2/>
  <mytag3/>
  <mytag4>
    <mytag5>
      <!-- this is an Action tag node -->
      <crtserver type='action' bal='was.crtserver'>
        <scope server='myserver' node='localhostNode01'/>
      </crtserver>
    </mytag5>
  </mytag4>

  <!-- this is another one -->
  <jdbc type='action' bal='was.jdbc' name='myprovider' description='mydesc'
path='/my/database/jdbc/path'>
  <scope node='localhostNode01' server='myserver'/>
  <db2 xa='true'/>
</jdbc>

</mytag1>
```

Let's run it:

```
> kikarc c.xml -v3
```

*Begin Actions ...*

*Action:crtserver retrieved.*

*Inject Operation*

*Application Server at scope: node:localhostNode01 server:myserver cluster:.*

*ApplicationServer:myserver created.*

*Action:jdbc retrieved.*

*Inject Operation*

*Scope target is Server: myservers at node: localhostNode01.*

*JDBCProvider:myprovider, at scope:cluster: None, cell: false, node: localhostNode01, server: myservers.*

*JDBCProvider:myprovider created.*

*VariableMap:DB2\_JDBC\_DRIVER\_PATH created.*

*... End Actions*

We observe that Action crtserver is run then Action jdbc is run.

They are run in that order because they appear in that order, from the top to the bottom of the c.xml file.

If Action nodes are duplicated they are operated as much times as they appear.

An Action node is a node with this Attribute:  
type = 'action'

Within an action node the bal Attribute is required to locate the Action:  
bal='was.crtserver'

Note:

For more information about the bal (Basic Action Locator) syntax see the kikonf core documentation.

This kikonf adaptative aspect help kikonf to adapt to your business specifics.

Let's say you decide to organize your xml file like the xml below because it reflects your actual architecture.

According your business structure, let's you say decide that for one given Application named: auction, one cluster of two servers is supported with a specific set of resources.  
Example 3 illustrates this case.

Example 3: see file:kikarc\_example\_3.xml

As you can see there is not real limit to your structure organization.

Ok I heard you said that example 3 is quite a big file and you do not really have time to create one specific xml file for all your pre-existing Application.

No soucy kikonf genrates them for you.

Kikarc comes with 3 default exits wich allow you to retrieve your pre-existing configuration to well strutured xml files.

Theses exits are:

- application: for a given installed Application's name, will retrieve the configuration of this Application and the configuration of all of the Application's targets (servers, clusters).
- cluster: for a given cluster name, will retrieve the cluster configuration and the configuration of all the cluster's members (servers).
- server: for a given server retrieves the server configuration and the configuration of all of its resources.

Note: these exits are inclusive from application to server.

Example 4: This example retrieve all known Actions for Application auction.

```
kikarc -o extract -e was.application --name auction -v10
```

Note: the information below explains the detail of the kikarc command options.

## 3 KIKARC OPTIONS

---

### 3.1 USAGE

---

#### >kikarc

Usage show a short view of kikarc.

Usage:

type -h for help.

type -H for extended Help.

kikarc <CUSTOM\_XML\_FILE>

This injects the Actions of the xml file into the target software configuration:

kikarc my.xml -v3

-v3: verbose level 3, verbose is available from 0 to 30.

kikarc <CUSTOM\_XML\_FILE> -o remove

This removes the Actions of the xml file from the target software configuration:

kikarc my.xml -o remove

kikarc <CUSTOM\_XML\_FILE> -o extract

This updates the xml file with the corresponding configuration Actions extracted from the software and shows the result to the output:

kikarc my.xml -o extract

kikarc <CUSTOM\_XML\_FILE> -x

This exports the Actions, from the xml file, to standalone action files and shows the result to the output:

kikarc my.xml -x

## 3.2 HELP AND LOGS

---

>**kikarc -h**

**-h, --help** *show this help message and exit.*

This option shows help on all options.

**-H HELP, --HELP=HELP** Extended help.

This options shows the complete kikarc document ation.

**-v VERBOSE, --verbose=VERBOSE** *The verbose level. A number from 0 to 30.*

The highter this number is the more verbose is shown.

For instance -v3 shows a block of text per Action injected or extracted.

Example :

>**kikarc c.xml -v3** or **kikarc c.xml -v 3**

**-l LOG\_FILE, --log\_file=LOG\_FILE** *(optional) A file where to log the output.*

All verbose is redirected to this file, although regular outputs are still printed.

### 3.3 PATHS

**-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, kikarc tries to retrieve its value from an environment variable named KIKONF\_CATTRS.*

*If neither of the option or the environment variable are set kikarc use the default kikonf.attrs file into the*

*<KIKONF\_INSTALL\_DIR>/conf directory.*

You may use this option if you want to specify a custom kikonf.attrs file, covering another scope of softwares. This would be the case if you supports more than one binary for the same software.

For more information about the kikonf.attrs file see the kikonf core documentation.

**-c ACTION\_DIR, --cxml=ACTION\_DIR**

*(optional) The path to the directory of the action files. If not given, sample action files are retrieved from the <KIKONF\_INSTALL\_DIR>/actions directory.*

*Note: If this option is not set, kikarc tries to retrieve its value from an environment variable named KIKONF\_CXML.*

*If neither the option or the environment variable are set kikarc use the default*

*<KIKONF\_INSTALL\_DIR>/actions directory.*

This action directory may be used by kikarc only when options `-overwrite` is given.

**-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, kikarc tries to retrieve its value from an environment variable named KIKONF\_RST.*

Generally you very less likely need to use restrictor directory than you may need to use custom action directory.

You may want to use restrictor directories when occurs the need to restrict a singular user's (or group) rights over a singular action.

This would restrict his access to (whole or) a specific tag(s) or Attribute(s).

You can see a restrictor file as a mask within a few fields are allowed.

Actually restrictor file are Action descriptor file, customized .

You can take a descriptor file for a given Action from:

`<PLUGINS_DIR>/actions/<CATEGORY>/<ACTION_NAME>/by/WHO/ACT_INF/action.xml`

and place it into your restrictor directory.

Obviously if not customized it would restrict nothing.

Note:

As you can see in the path (<PLUGINS\_DIR>/actions) above have a certain structure , your restrictor directory must reflect this structure.

```
<MY_RESTRICTOR_DIR>  
  <CATEGORY>  
    <ACTION_NAME>  
      by  
      <WHO>  
      action.xml
```

For more information about how to make restrictor files see the kikonf core documentation.

### 3.4 OPERATIONS

-o OPERATION, --operation=OPERATION (optional) Operation, allowed operations are inrun (default), run, inject, extract or remove.

Info:

Remember that Action descriptor files are located at :

<PLUGINS\_DIR>/actions/<CATEGORY>/<ACTION\_NAME>/by/WHO/ACT\_INF/action.xml

The Action descriptor file is the file which is used to run syntactic checks on custom action files, to avoid trivial configurations being injected into the target software.

An Action descriptor file can be seen as a WYSIWYG dtd.

For more information about descriptor files see the kikonf core documentation.

The example below are run using this sample file: c.xml

```
<mytag1>
  <mytag2/>
  <mytag3/>
  <mytag4>
    <mytag5>
      <!-- this is an Action tag node -->
      <rtserver type='action' bal='was.crtserver'>
        <scope server='myserver' node='localhostNode01' />
      </rtserver>
    </mytag5>
  </mytag4>

  <!-- this is another one -->
  <jdbc type='action' bal='was.jdbc' name='myprovider' description='mydesc'
  path='/my/database/jdbc/path'>
    <scope node='localhostNode01' server='myserver' />
    <db2 xa='true' />
  </jdbc>

  <starts type='action' bal='was.starts'>
    <scope node = 'localhostNode01' server = 'myserver' />
  </starts>

</mytag1>
```

**Note:**

was.crtserver and was.jdbc are configuration Actions.

was.starts is a control Action.

### **3.4.1 inject (-o inject):**

This will inject all the configuration set of the requested Actions into the target software.

Advanced: The method inject (if supported) is called on the class of the corresponding Action.

Impacted Actions : Actions with sub\_type=configuration into their descriptor file.  
Nothing happen for other sub\_type.

Example:

```
> kikarc c.xml -o inject
```

Let's run it:

```
> kikarc c.xml -o inject -v3
```

*Begin Actions ...*

*Action:crtserver retrieved.*

*Inject Operation*

*Application Server at scope: node:localhostNode01 server:myserver cluster:.*

*ApplicationServer:myserver removed.*

*ApplicationServer:myserver created.*

*Action:jdbc retrieved.*

*Inject Operation*

*Scope target is Server: myservers at node: localhostNode01.*

*JDBCProvider:myprovider, at scope:cluster: None, cell: false, node: localhostNode01, server: myservers.*

*JDBCProvider:myprovider created.*

*VariableMap:DB2\_JDBC\_DRIVER\_PATH created.*

*Action:starts retrieved.*

*... End Actions*

### **3.4.2 run (-o run):**

This will run the requested Actions on the target software.

Advanced: The method run (if supported) is called on the class of the corresponding Action.

Impacted Actions : Actions with sub\_type=control into their descriptor file.  
Nothing happen for other sub\_type.

Example:

```
> kikarc c.xml -o run
```

Let's run it:

```
> kikarc c.xml -o run -v3
```

*Begin Actions ...*

*Action:crtserver retrieved.*

*Action:jdbc retrieved.*

*Action:starts retrieved.*

*Run Operation*

*Scope target is Server: myserver at node: localhostNode01.*

*Application Server at scope:node: localhostNode01, server: myserver.*

*Aplication Server:myserver started on node:localhostNode01.*

*... End Actions*

### **3.4.3 inrun (-o inrun): this is the default.**

This will inject all the configuration set of the requested Actions into the target software and run the requested Actions on the target software.

Advanced: The method inject (if supported) is called on the class of the corresponding Action and the method run (if supported) is called on the class of the corresponding Action.

Impacted Actions : Actions with sub\_type=configuration into their descriptor file and Actions with sub\_type=control into their descriptor file.

Nothing happen for other sub\_type.

#### Example:

```
> kikarc c.xml -o inrun  
    or  
> kikarc c.xml
```

#### Let's run it:

```
> kikarc c.xml -v3
```

*Begin Actions ...*

*Action:crtserver retrieved.*

*Inject Operation*

*Application Server at scope: node:localhostNode01 server:mysqlserver cluster:.*

*ApplicationServer:mysqlserver removed.*

*ApplicationServer:mysqlserver created.*

*Action:jdbc retrieved.*

*Inject Operation*

*Scope target is Server: mysqlserver at node: localhostNode01.*

*JDBCProvider:myprovider, at scope:cluster: None, cell: false, node: localhostNode01, server: mysqlserver.*

*JDBCProvider:myprovider created.*

*VariableMap:DB2\_JDBC\_DRIVER\_PATH created.*

*Action:starts retrieved.*

*Run Operation*

*Scope target is Server: mysqlserver at node: localhostNode01.*

*Application Server at scope:node: localhostNode01, server: mysqlserver.*

*Application Server:mysqlserver started on node:localhostNode01.*

*... End Actions*

**3.4.4 extract (-o extract):**

This will extract all the configuration set of the requested Actions from the target software and built a new custom xml file.

Advanced: The method extract (if supported) is called on the class of the corresponding Action.

Impacted Actions : Actions with sub\_type=configuration into their descriptor file.

Nothing happen for other sub\_type.

**Example 1:**

```
> kikarc c.xml -o extract
```

**Let's run it:**

```
> kikarc c.xml -o extract -v3
```

*Begin Actions ...*

*Action:crtserver retrieved.*

*Extract Operation*

*Application Server at scope: node:localhostNode01 server:myserver cluster:.*

*Action:jdbc retrieved.*

*Extract Operation*

*JDBCProvider:myprovider, at scope:cluster: None, cell: false, node: localhostNode01, server: mys server.*

*Scope target is Server: mys server at node: localhostNode01.*

*JDBCProvider:myprovider retrieved.*

*Action:starts retrieved.*

*Extract Operation is not supported.*

*... End Actions*

```
<mytag1>
```

```
<mytag2/>
```

```
<mytag3/>
```

```
<mytag4>
```

```
<mytag5>
```

```
<crtserver type='action' bal='was.crtserver' sub_type='configuration' softwares='None'
  template='default' weight='2'>
```

```
<scope server='mys server' node='localhostNode01' cluster='None'/>
```

```
</crtserver>
```

```
</mytag5>
```

```
</mytag4>
```

```
<jdbc type='action' bal='was.jdbc' sub_type='configuration' softwares='None' name='myprovider'
  description='mydesc' path='/my/database/jdbc/path' prefix='None'>
```

```
<scope cell='false' node='localhostNode01' server='mys server' cluster='None'/>
```

```
<db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar'/>
```

```
</jdbc>
```

```
</mytag1>
```

#### 3.4.4.1 Action keys

### How does kikarc manage to extract an up to date custom file from the software configuration based on my source custom file ?

As overligheted in green (above) each Action supportes a scope, this scope is the fondemental part on an Action key.

Actually an Action key is structured like the following:

**\_ name** (optional) : if the Action has an Attribute name supported by the top Action tag it is taken in consideration to be part of the Action key.

**\_ prefix** (optional) : if the Action has an Attribute prefix supported by the top Action tag it is taken in consideration to be part of the Action key.

**\_ scope**

Note:

Operation remove also relay on Action keys to known witch resources to remove from the software configuration.

These keys from the provided custom file are used to retrieve Actions from the software configuration. Afterwards those Actions are matched back to their righth place into the source xml file and print to output.

#### Example 2 illustrating Action keys :

Let's update the jdbc configuration interactively updating the jdbc path: *+"Another\_path"* and removing the jdbc description.

```
> kikarc c.xml -o extract
```

```
<mytag1>
  <mytag2/>
  <mytag3/>
  <mytag4>
    <mytag5>
      <crtserver type='action' bal='was.crtserver' sub_type='configuration' softwares='None'
        template='default' weight='2'>
        <scope server='myserver' node='localhostNode01' cluster='None'/>
      </crtserver>
    </mytag5>
  </mytag4>
  <jdbc type='action' bal='was.jdbc' sub_type='configuration' softwares='None' name='myprovider'
    description='None' path='/my/database/jdbc/path' prefix='None'>
```

```
<scope cell='false' node='localhostNode01' server='myserver' cluster='None'/>
<db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;Another_path'/>
</jdbc>
```

```
</mytag1>
```

As we can see the provider

named: "myprovider" from

scope: "<scope cell='false' node='localhostNode01' server='myserver' cluster='None'/>"

is retrieved with its new values from the software configuration.

### Default values:

If I compare the extracted jdbc block with the original jdbc block from c.xml :

#### **extracted jdbc block :**

```
<jdbc type='action' bal='was.jdbc' sub_type='configuration' softwares='None' name='myprovider'
description='None' path='/my/database/jdbc/path' prefix='None'>
<scope cell='false' node='localhostNode01' server='myserver' cluster='None'/>
<db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;Another_path'/>
</jdbc>
```

#### **original jdbc block from c.xml :**

```
<jdbc type='action' bal='was.jdbc' name='myprovider' description='mydesc' path='/my/database/jdbc/path'>
<scope node='localhostNode01' server='myserver'/>
<db2 xa='true'/>
</jdbc>
```

Some new Attributes that was not used into the original c.xml file now appear into the extracted file.

In fact a fraction of Action Attributes (marked in green above) are default attributes and do not need to be filled.

To wipe away these optional Attributes use operation extract in conjunction with option --no\_dft

### Example 3 using extract operation in conjunction with option --no\_dft:

```
> kikarc c.xml -o extract -v3 --no_dft
```

```
<mytag1>
<mytag2/>
<mytag3/>
<mytag4>
<mytag5>

<certserver type='action' bal='was.certserver'>
<scope server='myserver' node='localhostNode01'/>
</certserver>

</mytag5>
</mytag4>
```

```
<jdbc type='action' bal='was.jdbc' name='myprovider' path='/my/database/jdbc/path'>
  <scope node='localhostNode01' server='myserver'/>
  <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;Another_path'/>
</jdbc>

</mytag1>
```

Not all default has disappeared and the extract is really more sexy.

Now if we create interactively a new provider on the same scope we wont be able to extract it because based on the Action keys from the source xml file, the match will keep going on :

```
named: "myprovider" from
scope: "<scope cell='false' node='localhostNode01' server='myserver' cluster='None'/>"
```

The workaround is to use one of the two options : -n (--no\_name) or -N (--no\_name\_no\_prefix) in conjunction with option extract.

Using these option extarct will ignore the name and/or prefix part of the Action keys.

Example 4 using extract operation in conjunction with option --no\_name (-n):

```
> kikarc c.xml -o extract -n -v3
```

*Begin Actions ...*

*Action:crtserver retrieved.*

*Extract Operation*

*Application Server at scope: node:localhostNode01 server:myserver cluster:.*

*Action:jdbc retrieved.*

*Extract Operation*

*JDBCProviders at scope:cluster: None, cell: false, node: localhostNode01, server: myservers.*

*Scope target is Server: myservers at node: localhostNode01.*

*JDBCProvider:my other provider retrieved.*

*JDBCProvider:myprovider retrieved.*

*JDBCProvider:Derby JDBC Provider retrieved.*

*Unmanaged provider:Derby JDBC Provider found, skipping !*

*Action:starts retrieved.*

*Extract Operation is not supported.*

*... End Actions*

```
<mytag1>
  <mytag2/>
  <mytag3/>
  <mytag4>
  <mytag5>

  <crtserver type='action' bal='was.crtserver'>
```

```

    <scope server='myserver' node='localhostNode01' cluster='None'/>
  </crtserver>

  </mytag5>
</mytag4>

<jdbc type='action' bal='was.jdbc' name='my other provider' description=""My description 2""
path='/my/database/jdbc/path' prefix='None'>
  <scope cell='false' node='localhostNode01' server='myserver' cluster='None'/>
  <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;db2jcc_license_cisuz.jar'/>
</jdbc>

<jdbc type='action' bal='was.jdbc' name='myprovider' description='None' path='/my/database/jdbc/path'
prefix='None'>
  <scope cell='false' node='localhostNode01' server='myserver' cluster='None'/>
  <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;Another_path'/>
</jdbc>

</mytag1>

```

Because name is ignored not only the jdbc named "myprovider" is matched.

#### 3.4.4.2 Extract output

So with no special option extract is dumped to the stdout.

To overwrite the original file use extract in conjunction with the option: --overwrite:

```
> kikarc c.xml -o extract --overwrite
```

Will overwrite the c.xml file (creating a backup for the original file).

To write the extract output to a specified file use option: --to\_file (-f) :

```
> kikarc c.xml -o extract -f path/to/my/new/file.xml
```

Will overwrite the c.xml file (creating a backup for the original file).

To write the extract output as standalones splitted Action files, to a specified directory see the next chapter below.

#### 3.4.4.3 Link with the kikact command

To write the extract output as standalone Action files in order to be delegated later or be used with the kikact command, run the extract operation with the --export (-x) option.

Example 4 using extract operation in conjunction with option --export (-x):

**> kikarc c.xml -o extract -x --no\_dft**

```
<crtserver type='action' bal='was.crtserver'>  
  <scope server='myserver' node='localhostNode01'/>  
</crtserver>
```

```
<jdbc type='action' bal='was.jdbc' name='myprovider' path='/my/database/jdbc/path'>  
  <scope node='localhostNode01' server='myserver'/>  
  <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;Another_path'/>  
</jdbc>
```

The Action nodes are sorted out.

To redirect the output to a directory where to store the standalone Action files, use the extract operation in conjunction with the options --to\_dir (-d) :

**> kikarc c.xml -o extract -x -d path/to/my/action/directory**

### **3.4.5 remove (-o remove):**

This will remove all the configuration set of the requested Actions from the target software. Beware using this one.

Or eventually test it switching test from False to True into the kikonf.attrs file (test=True). For more information on the kikonf.attrs file see the kikonf core documentation.

Advanced: The method remove (if supported) is called on the class of the corresponding Action.

Impacted Actions : Actions with sub\_type=configuration into their descriptor file. Nothing happen for other sub\_type.

#### Example 1:

```
> kikarc c.xml -o remove
```

#### Let's run it:

```
> kikarc c.xml -o remove -v3
```

*Begin Actions ...*

*Action:crtserver retrieved.*

*Remove Operation*

*Application Server at scope: node:localhostNode01 server:mysqlserver cluster:.*

*ApplicationServer:mysqlserver removed.*

*Action:jdbc retrieved.*

*Remove Operation*

*Scope target is Server: mysqlserver at node: localhostNode01.*

*Action:starts retrieved.*

*... End Actions*

Please note that Action jdbc is doing nothing because Action crtserver preceding in the stack has already removed all the server's (mysqlserver) associated resources.

Because this kind of issue may happened. An Action with bigger range can already destroy sub resources, errors may occur into the subsequent Actions.

To see those errors, run with options: --check 3 (from 1 to 3) :

#### Example 2 using remove operation in conjunction with option -check:

```
>kikarc c.xml -o remove -v3 --check 3
```

*Begin Actions ...*

*Action:crtserver retrieved.*

*Remove Operation*

*Application Server at scope: node:localhostNode01 server:myserver cluster:.  
ApplicationServer:myserver removed.*

*Action:jdbc retrieved.*

*Remove Operation*

*Scope target is Server: myservers at node: localhostNode01.*

*EKIKONWAS: **Scope not found** ! Your values: '{"cluster': None, 'cell': 'false', 'node': 'localhostNode01', 'server': 'myservers'}".*

*EKIKONF: Error running at least one action !*

Scope is not found because server myservers has been destroyed with all its resources (including jdbc which has been created on the server scope) by previous Action was.crtserver.

Note:

Operation extract also supports the --check option.

## **3.5 EXITS**

---

Now we saw how to extract the software configuration but based on an existing custom xml file.

It is also possible to generate a custom xml file from scratch using exits.

### **3.5.1 server**

This exit will extract the server configuration and all its resources, understand all the resources defined under this server scope.

```
>kikarc -o extract -e was.server --scope_server myserver
```

### **3.5.2 cluster**

This exit will extract the cluster configuration and all its clustermembers. And will recursively extract the same information as done by the exit server, from each clustermember.

```
>kikarc -o extract -e was.cluster --scope_cluster mycluster
```

### **3.5.3 application**

This exit will extract the Application configuration, and all its targets. Understand Servers, Clusters and WebServers. And will recursively extract the same information as done by the exit server or exit cluster from each target.

```
>kikarc -o extract -e was.application --name myapp
```

### **3.5.4 Link with the kikact command**

To generate standalone Action files, straight away from exit just add the --export (-x) options:

```
>kikarc -o extract -e was.application --name myapp -x
```

## 3.6 EXPORT

---

### 3.6.1 Link with the kikact command

As seen in chapters extract and exits the --export (-x) command can be used with extract operation to export Action nodes from a resulting custom xml file to standalone Action nodes.

The export options can also be used alone from a given custom file to turn it to standalone Action nodes ready to be used with kikact or dispatched.

>kikarc -c.xml -x

lets run it with c.xml.

>kikarc -c.xml -x

```
<crtserver type='action' bal='was.crtserver' sub_type='configuration' softwares='None'
template='default' weight='2'>
  <scope server='myserver' node='localhostNode01' cluster='None'/>
</crtserver>
```

```
<jdbc type='action' bal='was.jdbc' sub_type='configuration' softwares='None' name='myprovider'
description='mydesc' path='/my/database/jdbc/path' prefix='None'>
```

```
  <scope cell='false' node='localhostNode01' server='myserver' cluster='None'/>
  <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar'/>
</jdbc>
```

```
<starts type='action' bal='was.starts' sub_type='control' softwares='None'>
  <scope node='localhostNode01' server='myserver'/>
</starts>
```

### 3.7 PARTIAL CUSTOM FILE EXECUTION

Sometimes running a big custom file dont want to run the all file but just a part of it.

Use the following options when you just want to run a fraction of your custom xml file.

**-t TD, --td=TD** Given a custom xml file, joins the specified Action node(s).

**--td (-t)** expects a Top down PxQuery request according the syntax of the picxml documentation (see it for more information).

**-T TDC, --tdc=TDC** Given a custom xml file, joins the specified Action node(s). **--tdc (-T)** expects a Top down complete PxQuery request according the syntax of the picxml documentation (see it for more information).

For instance :

Given this new custom file extracted in chapter extract (example 4 with kikarc.py c.xml -o extract -n --no\_dft) : c.xml

```
<mytag1>
  <mytag2/>
  <mytag3/>
  <mytag4>
    <mytag5>

      <certserver type='action' bal='was.certserver'>
        <scope server='myserver' node='localhostNode01'/>
      </certserver>

    </mytag5>
  </mytag4>

  <jdbc type='action' bal='was.jdbc' name='my other provider' description=""My description 2""
  path='/my/database/jdbc/path'>
    <scope node='localhostNode01' server='myserver'/>
    <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;db2jcc_license_cisuz.jar'/>
  </jdbc>

  <jdbc type='action' bal='was.jdbc' name='myprovider' path='/my/database/jdbc/path'>
    <scope node='localhostNode01' server='myserver'/>
    <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;Another_path'/>
  </jdbc>
</mytag1>
```

>kikarc c.xml -t mytag1.jdbc -v3 -x

```
<jdbc type='action' bal='was.jdbc' name='my other provider' description=""My description 2""
path='/my/database/jdbc/path'>
  <scope node='localhostNode01' server='myserver'/>
  <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;db2jcc_license_cisuz.jar'/>
</jdbc>
```

```
<jdbc type='action' bal='was.jdbc' name='myprovider' path='/my/database/jdbc/path'>
  <scope node='localhostNode01' server='myserver'/>
  <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;Another_path'/>
</jdbc>
```

>kikarc c.xml -t mytag1.mytag4.mytag5.crtserver -v3 -x

```
<crtserver type='action' bal='was.crtserver' sub_type='configuration' softwares='None' template='default'
weight='2'>
  <scope server='myserver' node='localhostNode01' cluster='None'/>
</crtserver>
```

As you can see it's just easy as following the xml tree order until you desired Action node for the -t (top down) syntax.

For more information about the -t (top down syntax ) see the picxml documentation.

Remember all operations can be used with the partial syntax (inject, run, inrun (default), extract, remove, ...).

Here an example with default inrun operation :

>kikarc c.xml -t mytag1.jdbc -v3

Begin Actions ...

**Action:jdbc retrieved.**

Inject Operation

Scope target is Server: myservers at node: localhostNode01.

JDBCProvider:my other provider, at scope:cluster: None, cell: false, node: localhostNode01, server: myservers.

JDBCProvider:my other provider removed.

VariableSubstitutionEntry:DB2\_JDBC\_DRIVER\_PATH removed.

JDBCProvider:my other provider created.

VariableMap:DB2\_JDBC\_DRIVER\_PATH created.

**Action:jdbc retrieved.**

Inject Operation

Scope target is Server: myservers at node: localhostNode01.

JDBCProvider:myprovider, at scope:cluster: None, cell: false, node: localhostNode01, server: myservers.

JDBCProvider:myprovider removed.

VariableSubstitutionEntry:DB2\_JDBC\_DRIVER\_PATH removed.

JDBCProvider:myprovider created.

VariableMap:DB2\_JDBC\_DRIVER\_PATH created.

... End Actions

<subset>

```
<jdbc type='action' bal='was.jdbc' sub_type='configuration' softwares='None' name='my other provider'
description=""My description 2"" path='/my/database/jd
bc/path' prefix='None'>
  <scope cell='false' node='localhostNode01' server='myserver' cluster='None'/>
  <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;db2jcc_license_cisuz.jar'/>
</jdbc>
```

```
<jdbc type='action' bal='was.jdbc' sub_type='configuration' softwares='None' name='myprovider'
description='None' path='/my/database/jdbc/path' prefix='None'
'>
  <scope cell='false' node='localhostNode01' server='myserver' cluster='None'/>
  <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;Another_path'/>
</jdbc>
```

</subset>

Actually two Actions are run using **-t mytag1.jdbc**, because at this level they are two jdbc Actions. What if I wanted to run the jdbc action named : **my other provider**, instead of the Action named: **myprovider** ?

I should use the -T (top down complete query) option :

>kikarc c.xml -T "t:mytag1.t:jdbc,a:name=my other provider" -x

```
<jdbc type='action' bal='was.jdbc' sub_type='configuration' softwares='None' name='my other
provider' description=""My description 2"" path='/my/database/jdbc/path' prefix='None'>
  <scope cell='false' node='localhostNode01' server='myserver' cluster='None'/>
  <db2 xa='true' jars='db2jcc.jar;db2jcc_license_cu.jar;db2jcc_license_cisuz.jar'/>
</jdbc>
```

## 4 ANNEXE 1: KIKARC ALL OPTIONS

---

>*kikarc.py -h*

**Usage:**

type -h for help.  
type -H for extended Help.

kikarc <CUSTOM\_XML\_FILE>

This injects the Actions of the xml file into the target software configuration:

kikarc my.xml -v3

-v3: verbose level 3, verbose is available from 0 to 30.

kikarc <CUSTOM\_XML\_FILE> -o remove

This removes the Actions of the xml file from the target software configuration:

kikarc my.xml -o remove

kikarc <CUSTOM\_XML\_FILE> -o extract

This updates the xml file with the corresponding configuration Actions extracted from the software and shows the result to the output:

kikarc my.xml -o extract

kikarc <CUSTOM\_XML\_FILE> -x

This exports the Actions, from the xml file, to standalone action files and shows the result to the output:

kikarc my.xml -x

## Options:

- h, --help** show this help message and exit
- H HELP, --HELP=HELP** Extended help.
- v VERBOSE, --verbose=VERBOSE**  
The verbose level. A number from 0 to 30.
- o OPERATION, --operation=OPERATION** Operation, allowed operations are inrun (default), run, inject, extract or remove.
- 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, kikarc tries to retrieve its value from an environment variable named KIKONF\_CATTRS. If neither the option or the environment variable are set kikarc 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 retrieved from the <KIKONF\_INSTALL\_DIR>/actions directory. Note: If this option is not set, kikarc tries to retrieve its value from an environment variable named KIKONF\_CXML. If neither the option or the environment variable are set kikarc 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, kikarc tries to retrieve its value from an environment variable named KIKONF\_RST.
- x, --export** (optional) Export and split the content of a Custom xml file to standalone action configurations files.  
Allowed with a given custom xml file with no operation, or/and every where the --to\_file (-f) option is allowed !
- l LOG\_FILE, --log\_file=LOG\_FILE** (optional) A file where to log the output.  
Partial custom file execution:  
Use the following options when you just want to run a fraction of your custom xml file.
- t TD, --td=TD** (optional) Given a custom xml file, joins the specified Action node(s). --td (-t) expects a Top down PxQuery request according the syntax of the picxml documentation (see it for more information).
- T TDC, --tdc=TDC** (optional) Given a custom xml file, joins the specified Action node(s). --tdc (-T) expects a Top down complete PxQuery request according the syntax of the picxml documentation (see it for more information).

### Extract or remove extended options:

The following options are allowed combined with the extract (-o extract) or remove (-o remove)

operations.

**--check=CHECK** In conjunction with the extract or remove operation (-o extract/remove). A number  $\geq 0$  (default 0). If greater than 0, at the end of the extraction, res/descriptors checks are run on the resulting xml file. Advice:

On purpose extract is versatile and still going on eventual errors. With this option you can run a strong check on the resulting xml. Please note when running the command on normal operation mode (inrun, run), strong res/descriptors check all always run !

**-n, --no\_name** In conjunction with the extract or remove operation (-o extract/remove). Will retrieve all configuration elements starting with the prefix (if given) with no regard to the name attribute. Info: On an extract operation: name, prefix and scope values are retrieved from Action file(s) (Custom xml or standalone Action xml), to match software configuration elements.

**-N, --no\_name\_no\_prefix**

In conjunction with the extract or remove operation (-o extract/remove). Will retrieve all configuration elements matching the scope with no regard for the name and prefix attributes. Info: On an extract operation: name, prefix and scope values are retrieved from Action file(s) (Custom xml or standalone Action xml), to match software configuration elements.

### **Extract or export extended options:**

The following option is allowed combined with the extract operation or with the --export (-x) option.

**--overwrite** In conjunction with the extract operation (-o extract) or --export (-x) option. Extracted file(s) will overwrite the original files . Be sure of what you doing using this options.

### **Export extended option:**

The following option is allowed combined with the --export (-x) option.

**-d TO\_DIR, --to\_dir=TO\_DIR**

Allowed in conjunction with the --export (-x) option. A path to a directory where to write exported action xml files.

### **Extract extended options:**

The following options are allowed combined with the extract (-o extract) operation.

**-f TO\_FILE, --to\_file=TO\_FILE**

Allowed in conjunction with the extract operation (-o extract). A path to a file where to write the newly extracted custom xml file. In this sample the newly extracted custom xml file based on c.xml provision is stored into the file /my/myfile: > kikarc c.xml -o extract -f /my/myfile. In this sample the newly extracted custom xml file based on the was.application exit provision is stored into the file /my/myfile: kikarc -o extract -e was.application -name myapp -f /my/myfile.

**--no\_dft** In conjunction with the extract operation (-o extract).

By default resulting extracted xml file is filled with all the attributes supported by the action  
If no\_dft is True (default false), Attributes whose value matches to the res/descriptor's default value for this attribute are not shown !

**-e EXIT, --exit=EXIT**

In conjunction with the extract operation (-o extract). Refers to an exit module name on which the method extract is invoked to process the extraction.

When --exit is used, extended options: --name, --prefix and extended scope options: --scope\_server, --scope\_node, scope\_cluster, scope\_application, scope\_type are allowed to feed the target method, called on the associated exit module.

**Exit extended options:**

The following options are allowed combined with --exit (-e) option.

**--name=NAME** In conjunction with the --exit (-e) option. A name.

**--prefix=PREFIX** In conjunction with the --exit (-e) option. A prefix name.

**--scope\_server=SCOPE\_SERVER**

In conjunction with the --exit (-e) option. A server name.

**--scope\_node=SCOPE\_NODE**

In conjunction with the --exit (-e) option. A node name.

**--scope\_cluster=SCOPE\_CLUSTER**

In conjunction with the --exit (-e) option. A cluster name.

**--scope\_application=SCOPE\_APPLICATION**

In conjunction with the --exit (-e) option. An application name.

**--scope\_war=SCOPE\_WAR** In conjunction with the --exit (-e) option. An application war name.

**--scope\_type=SCOPE\_TYPE** In conjunction with the --exit (-e) option. A server type. Allowed values are as: for Application Server and ws: for WebServer.

**--scope\_cell** In conjunction with the --exit (-e) option. An application name.

## **5 ANNEXE 2: ACTION PLUGINS**

---

See the `plugin_howto` documentation.

## **6 ANNEXE 3: EXIT PLUGINS**

---

See the `plugin_howto` documentation.