Kikact

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

SUMMARY

1 INTRODUCTION

Kikact stands for ***kick actions !***
Kikact is the kikonf command to work with stanadalone Actions.

2 OVERVIEW

Kikact is the command to work with standalone Action nodes.

This the usage output when kikact is run with no parameter:

>kikact

kikact <BAL>[,BAL]
<BAL>[,BAL]: A comma separted list of action BALs (Basic Action Locator).

This injects the Actions crtserver, jdbc and datasrc into the target software configuration:
kikact was.crtserver,was.jdbc,was.datasrc -v3
-v3: verbose level 3, verbose is available from 0 to 30.

kikact <BAL>[,BAL] -o remove
This removes the Actions crtserver,jdbc,datasrc from the target software configuration:
kikact was.crtserver,was.jdbc,was.datasrc -v3 -o remove

kikact <BAL>[,BAL] -o extract
This updates the Actions crtserver,jdbc,datasrc with the corresponding configurations extracted from the software and shows the result to the output:
kikact was.crtserver,was.jdbc,was.datasrc -o extract

Note:
There are no spaces wrapping BALs.
To avoid repeting the category use the --category (-g) option:
ex:**kikact -g was crtserver,jdbc,datasrc**

One can run just one action:

kikact was.crtserver

or a comma separated list of Actions:

kikact was.crtserver,was.jdbc,was.datasrc

They are five operations:
inject, run, inrun (inject + run this is the default), **extract and remove**.

Operations are signalized by the --operation (-o) option.

Example:
kikact was.crtserver -o remove

This would remove the server configuration from the target software.

The syntax for an Action (ex: was.crtserver) is define by the BAL (Basic Action Locator) syntax.

See chapter Annex 1 The BAL (Basic Action Locator) for more information.

Quickly says, a BAL starts with the **Category** (also called provider or target software).
Within **was.**crtserver, the Category is: was.

After the Category, follows up the **Action name**.
Within was.**crtserver**, the Action name is: crtserver.

To list all supported Actions sorted per Category, type:

>kikupd -p

Note: the -p (--plugins) options, list all supported Action plugins by the current kikonf installation.

Where are Action files ?

What we call an Action file is the configuration file for the action (ex: crtserver).
For configuration file, read: the configuration you want for this Action.

Action files are searched into the Actions Directory.
The Actions Directory can be given as parameter or as an environment variable.

If neither of above is given, the Action directory is the default: *<KIKONF_INSTALL_DIR>/actions* directory.

Note: By default the <KIKONF_INSTALL_DIR>/actions directory supports one sample Action file for each installed Actions.

Hence if we run

>kikact was.crtserver

A file named: crtserver.xml is searched and retreived into the Action directory and injected into the target software configuration.

If you want to support more than one Action files for the same action you can use the **Label** facility of the **BAL** syntax.

You can name your files :

was.crtserver$01.xml
was.crtserver$my_first_test.xml
was.crtserver$preprod.xml

Note: All stuffs coming after $ is constituing the Label part and is discarded from the action name. The Label part is any ASCII string.

Hence the command:
>kikact was.crtserver

Will run the crtserver Action for these 3 configuration files (was.crtserver$01.xml, was.crtserver$my_first_test.xml, was.crtserver$preprod.xml).

If you want to run just one of the labelized Action files type:

>kikact was.crtserver$preprod.xml

ore more:

>kikact was.crtserver$01.xml,was.crtserver$preprod.xml

What are Action file ?

An action file is an xml file wich says the settings you want for a speficic action.

For instance within action was.crtserver the configuration may be:

<**crtserver type='action'**>
  <scope server='myserver' node='localhostNode01/>
</crtserver>

With this Action file, the command

>kikact was.crtserver -v3

Begin Actions ...

Action:crtserver retreived.
Inject Operation
Application Server at scope: node:localhostNode01 server:myserver cluster:.
ApplicationServer:myserver removed.
ApplicationServer:myserver created.

... End Actions

Creates the Application Server named: **myserver** into the Node named: **localhostNode01** into the target software configuration pointed by was.

Another usefull tools is extraction, this functionality allows you to extract your software configuration to standalone Action.
A sample use of this is:

>kikact was.crtserver -o extract -v3

```
<crtserver type='action' bal='was.crtserver'>
   <scope server='myserver' node='localhostNode01' cluster='None'/>
</crtserver>
```

Extract if further more powerful that that and it's more explolred into next chapter "kikact options".

3 KIKACT OPTIONS

3.1 USAGE

Usage show a short view of kikact.

>kikact

Usage:

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

kikact <BAL>[,BAL]
<BAL>[,BAL]: A comma separted list of action BALs (Basic Action Locator).

This injects the Actions crtserver,jdbc,datasrc into the target software configuration:
kikact was.crtserver,was.jdbc,was.datasrc -v3
-v3: verbose level 3, verbose is available from 0 to 30.

kikact <BAL>[,BAL] -o remove
This removes the Actions crtserver,jdbc,datasrc from the target software configuration:
kikact was.crtserver,was.jdbc,was.datasrc -v3 -o remove

kikact <BAL>[,BAL] -o extract
This updates the Actions crtserver,jdbc,datasrc with the corresponding configurations extracted from the software and shows the result to the output:
kikact was.crtserver,was.jdbc,was.datasrc -o extract

Note:
There are no spaces wrapping BALs.
To avoid repeting the category use the --category (-g) option:
ex:**kikact -g was crtserver,jdbc,datasrc**

3.2 HELP AND LOGS

>kikact -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 kikact 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 :
>kikact c.xml -v3 or kikact 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, althought 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, kikact tries to retreive its value from an environment variable named KIKONF_CATTRS.
If neither of the option or the environment variable are set kikact 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 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.

An Action file is an xml file wich defines the settings for a singular Action.
Actually it is where you put your setup for a singular Action.
For more information about Action files see (chapter above "overview" or) the kikonf core documentation.

The default <KIKONF_INSTALL_DIR>/actions directory contains a sample for each existing Actions.
Beware that if no custom directory is found these samples are run for each corresponding Actions.

Note:
The structue of a custom action directory is:

<MY_CUSTOM_ACTION_DIR>
<BAL1>.xml
<BAL2>.xml
<BALn>.xml
...
For more information about custom Action directory see the kikonf core documentation.

-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.

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

You may want to use restrictor directories when occures 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 descritpor file is the file wich is used to run syntaxic checks on custom action files, to avoid trivial configurations being injected into the target software.
An Action descritpor 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 these sample action files: was.crtserver.xml, was.jdbc.xml and was.starts.xml for Actions crtserver, jdbc and starts.

was.crtserver.xml:

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

was.jdbc.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>
```

was.starts.xml:

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

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:

```
>kikact was.crtserver,was.jdbc,was.starts -o inject
```

this is the equivalent to (because nothing would happened for was.starts):

```
>kikact was.crtserver,was.jdbc -o inject
```

Let's run it:

>kikact was.crtserver,was.jdbc,was.starts -o inject -v3

Begin Actions ...

Action:crtserver retreived.
Inject Operation
Application Server at scope: node:localhostNode01 server:myserver cluster:.
ApplicationServer:myserver removed.
ApplicationServer:myserver created.

Action:jdbc retreived.
Inject Operation
Scope target is Server: myserver at node: localhostNode01.
JDBCProvider:myprovider, at scope:cluster: None, cell: false, node: localhostNode01, server: myserver.
JDBCProvider:myprovider created.
VariableMap:DB2_JDBC_DRIVER_PATH created.

Action:starts retreived.

... 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:

>kikact was.crtserver,was.jdbc,was.starts -o run

this is the equivalent to (because nothing would happened for the others):

>kikact was.starts -o run

Let's run it:

>kikact was.crtserver,was.jdbc,was.starts -o run -v3

Begin Actions ...

Action:crtserver retreived.

Action:jdbc retreived.

Action:starts retreived.
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:

> kikact was.crtserver,was.jdbc,was.starts -o inrun

or (because inrun is the default):

> kikact was.crtserver,was.jdbc,was.starts

Let's run it:

>kikact was.crtserver,was.jdbc,was.starts -v3

Begin Actions ...

Action:crtserver retreived.
Inject Operation
Application Server at scope: node:localhostNode01 server:myserver cluster:.
ApplicationServer:myserver removed.
ApplicationServer:myserver created.

Action:jdbc retreived.
Inject Operation
Scope target is Server: myserver at node: localhostNode01.
JDBCProvider:myprovider, at scope:cluster: None, cell: false, node: localhostNode01, server: myserver.
JDBCProvider:myprovider created.
VariableMap:DB2_JDBC_DRIVER_PATH created.

Action:starts retreived.
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

3.4.4 extract (-o extract):

This will extract all the configuration set of the requested Actions from the target software and built a new Action files.

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:

> kikact was.crtserver,was.jdbc,was.starts -o extract

this is the equivalent to (because nothing would happened for was.starts):

> kikact was.crtserver,was.jdbc -o extract

Let's run it:

> kikact was.crtserver,was.jdbc -o extract

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

<crtserver type='action' bal='was.crtserver'>
   <scope server='myserver' node='localhostNode01' cluster='None'/>
</crtserver>
```

3.4.4.1 Action keys

How does kikact manage to extract up to date Action file from the software configuration based on my source Action files ?

As overlighted 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 retreive Actions from the software configuration.

Example 2 illustrating Action keys :

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

> kikact was.jdbc -o extract

```
<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>
```

As we can see the provider
named: *"myprovider"* from
scope: *"<scope cell='false'* ***node='localhostNode01' server='myserver'*** *cluster='None'/>"*
is retrevied with its new values from the software configuration.

Default values:

If I compare the extracted A ction file was.jdbc with the original Action file :

extracted was.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>
```

original jdbc was.jdbc.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 extrated file.
In fact a fraction of Action Attibutes (marked in green above) are default attributes and do no need to be filled.

To wipe away these optional Attibutes use operation extract in conjunction with option --no_dft

Example 3 using extract operation in conjunction with option --no_dft:

> kikact was.jdbc -o extract **--no_dft**

```
<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>
```

Not all default has desappeared 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):

```
>kikact.py was.jdbc -o extract --no_dft -n

<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>
```

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:
>kikact was.crtserver,was.jdbc -o extract --overwrite

Will overwrite sample Action files (creating a backup for the original files).

To write the extract output to a specificed directory use option: --to_dir (-d) :
>kikact was.crtserver,was.jdbc -o extract -d path/to/my/directory
Will overwrite sample Action files (creating a backup for the original files).

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:

> kikact was.crtserver,was.jdbc -o remove

Let's run it:

>kikact was.crtserver,was.jdbc -o remove -v3

Begin Actions ...

Action:crtserver retreived.
Remove Operation
Application Server at scope: node:localhostNode01 server:myserver cluster:.
ApplicationServer:myserver removed.

Action:jdbc retreived.
Remove Operation
Scope target is Server: myserver at node: localhostNode01.

... End Actions

Please note that Action jdbc is doing nothing because Action crtserver preceding in the stack has already removed all the server's (myserver) 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:

>kikact was.crtserver,was.jdbc -o remove -v3 --check 3

Begin Actions ...

Action:crtserver retreived.

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

Action:jdbc retreived.
Remove Operation
Scope target is Server: myserver at node: localhostNode01.

EKIKONWAS: Scope not found ! Your values:"{'cluster': None, 'cell': 'false', 'node': 'localhostNode01', 'server': 'myserver'}".
EKIKONF: Error running at least one action !

Scope is not found because server myser has been destroyed with all its resources (including jdbc wich 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 a list of given Action files.

It is aslo possible to generate Actrion files 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.

>kikact -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.

>kikact -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.

>kikact -o extract -e was.application --name myapp

4 ANNEX 1 THE BAL (BASIC ACTION LOCATOR)

4.1 SYNTAX

BAL (Basic Action Locator)	IN COMMANDS (kikact, kikarc,...)	MAPPED ACTION FILES	MAPPED ACTION RESTRICTOR FILES	ACTION TAG IN CUSTOM XML FILE
CATEGORY.ACTION [.by.WHO]	BAL[.$LABEL]	BAL[.$LABEL].xml **Top tag:** ACTION	BAL.xml **Top tag:** ACTION	**Action tag:** ACTION **Bal attribute:** BAL

4.2 SAMPLES

BAL	IN COMMANDS	SAMPLES	MAPPED ACTION CONFIGURATION FILES	MAPPED ACTION RESTRICTOR FILES	ACTION TAG IN CUSTOM XML FILES
CATEGORY.ACTION	BAL	was.datasrc	was.datasrc.xml or was.datasrc.$LABEL.xml (all labelized action files are treated as well) **Top tag:** datasrc	was.datasrc.xml **Top tag:** datasrc	**Action tag:**datasrc **Bal attribute:** was.datasrc
	BAL.$LABEL	was.datasrc.$1 (or tmc.datasrc. $invoices	was.datasrc.$1.xml **Top tag:** datasrc		
Advanced (use it if you know what you are doing):					
CATEGORY.ACTION .by.WHO	BAL	was.datasrc.by.shandra	was.datasrc.by.shandra.xml or was.datasrc.by.shandra. $LABEL.xml (all labelized action files are treated as well)	was.datasrc.(+) by.shandra.xml **Top tag:** datasrc	**Action tag:**datasrc **Bal attribute:** was.datasrc.(+) by.shandra
	BAL.$LABEL	was.datasrc.by.shandra.$1	was.datasrc.by.shandra. $1.xml		

Note:

If you launch a command (ex: kikact) with the -g (or --category) CATEGORY option (ex: -g was), you do not need to prefix each BAL with CATEGORY.
kikact -g was jvm,datasrc,vhost instead of
kikact was.jvm,was.datasrc,was.vhost

Advanced trick:

In the advanced part, the clause "by" in CATEGORY.ACTION.by.WHO is used when one action has multiple providers (or authors).
ex: The action was.jvm may have been writen by kikonf, but another version more adapted for specific needs may have also been provided by shandra.

WHO is the name of this provider.
Action plugins are clasified per category/action/provider.
The kikupd option -P shows a detailled list of all plugins per providers.

A directory entitled by WHO exists at
<KIKONF_INSTALL_DIR>/plugins/actions/<CATEGORY>/<ACTION>/by/<WHO>.

If the file <KIKONF_INSTALL_DIR>/plugins/actions/<CATEGORY>/<ACTION>/default.txt is filled with WHO.
WHO becomes the default plugin for this action and the clause "by" is no more needed.

On delivry of the kikonf binary, **kikonf** is this default provider.

5 ANNEXE 2: KIKACT ALL OPTIONS

>kikact -h

Usage:

type -H for extended Help.

kikact <BAL>[,BAL]

<BAL>[,BAL]: A comma separted list of action BALs (Basic Action Locator).

This injects the Actions crtserver,jdbc,datasrc into the target software configuration:
kikact was.crtserver,was.jdbc,was.datasrc -v3
-v3: verbose level 3, verbose is available from 0 to 30.

kikact <BAL>[,BAL] -o remove

This removes the Actions crtserver,jdbc,datasrc from the target software configuration:
kikact was.crtserver,was.jdbc,was.datasrc -v3 -o remove

kikact <BAL>[,BAL] -o extract

This updates the Actions crtserver,jdbc,datasrc with the corresponding configurations extracted from the software and shows the result to the output:
kikact was.crtserver,was.jdbc,was.datasrc -o extract

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.

-g CATEGORY, --category=CATEGORY (Optional) Category, if you use this option you no more need to prefix you Action BAL or exit BEL with the software category.

-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.

-l LOG_FILE, --log_file=LOG_FILE (Optional) A file where to log the output.

Extract or remove extended options:
The following options are allowed combined with the extract (-o extract) or remove (-o remove) operations.

--check=CHECK In conjontion with the extract or remove operations (-o extract). A number >= 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/descritpors check all always run !

-n, --no_name In conjontion with the extract or remove operations (-o extract/remove). Will retreive 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 retreived from Action file(s) (Custom xml or standalone Action xml), to match software configuration elements.
-N, --no_name_no_prefix In conjontion with the extract or remove operations (-o extract/remove).

Will retreive 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 retreived from Action file(s) (Custom xml or standalone Action xml), to match software configuration elements.

Extract extended options:

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

--overwrite In conjontion with the extract operation (-o extract). Extracted file(s) will overwrite the original files.
Be sure of what you doing using this options. Using this option with no --to_dir (-d) parameter you may overwrite your custom action directory or the default one !!!

-d TO_DIR, --to_dir=TO_DIR Allowed in conjontion with the extract operation (-o extract). A path to a directory where to write extracted action xml files. In this sample xml files extracted for the was.jvm was.jdbc and was.datasrc actions are stored into the /my/dir directory: > kikact was.jvm,was.jdbc,was.datasrc -o extract -d /my/dir. In this sample xml files extracted from the was.application exit are stored into the /my/dir directory: > kikact was.jvm,was.jdbc,was.datasrc -o extract -d /my/dir.

--no_dft In conjontion 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 whome value matches to the res/descriptor's default value for this attribute are not shown !

-e EXIT, --exit=EXIT
In conjontion 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 conjontion with the --exit (-e) option. A name.

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

--scope_server=SCOPE_SERVER
In conjontion with the --exit (-e) option. A server name.
--scope_node=SCOPE_NODE
In conjontion with the --exit (-e) option. A node name.
--scope_cluster=SCOPE_CLUSTER
In conjontion with the --exit (-e) option. A cluster name.

--scope_application=SCOPE_APPLICATION
In conjontion with the --exit (-e) option. An application name.

--scope_war=SCOPE_WAR In conjontion with the --exit (-e) option. An application war name.

--scope_type=SCOPE_TYPE In conjontion with the --exit (-e) option. A server type. Allowed values are as: for Application Server and ws: for WebServer.

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