Mobicents JSLEE Twiddle User Guide

by Bartosz Baranowski

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts* [https://fedorahosted.org/liberation-fonts/] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press **Enter** to execute the command.
>
> Press **Ctrl**+**Alt**+**F1** to switch to the first virtual terminal. Press **Ctrl**+**Alt**+**F7** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in `Mono-spaced Bold`. For example:

> File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

`Mono-spaced Bold Italic` or ***Proportional Bold Italic***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type `ssh` *`username@domain.name`* at a shell prompt. If the remote machine is `example.com` and your username on that machine is john, type `ssh john@example.com`.
>
> The `mount -o remount` *`file-system`* command remounts the named file system. For example, to remount the `/home` file system, the command is `mount -o remount /home`.
>
> To see the version of a currently installed package, use the `rpm -q` *`package`* command. It will return a result as follows: *`package-version-release`*.

Note the words in bold italics above username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as

a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }

}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

A note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.

Warning

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

2. Provide feedback to the authors!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in the the *Issue Tracker* [http://code.google.com/p/mobicents/issues/list], against the product **Mobicents JSLEE Twiddle** , or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: JSLEETwiddleCLI_User_Guide

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Introduction to Mobicents JSLEE Twiddle

JAIN SLEE is the specification for a Java Service Logic and Execution Environment (SLEE). A SLEE is an application server, or service container, which defines a component model for structuring the logic of communications services as a collection of reusable components, and for composing these components into even more sophisticated services.

As such, it requires management tools to access and manage different aspects of container. JSLEE Twiddle is command line tool enabling user to interact with container in various way, through standard management interfaces.

Setup

2.1. Pre-Install Requirements and Prerequisites

Ensure that the following requirements have been met before continuing with the install.

2.1.1. Hardware Requirements

The JSLEE Twiddle doesn't change the Mobicents Hardware Requirements.

2.1.2. Software Prerequisites

JSLEE Twiddle does not have software dependencies.

2.2. Mobicents JSLEE Twiddle Source Code

2.2.1. Release Source Code Building

1. **Downloading the source code**

Important

Subversion is used to manage its source code. Instructions for using Subversion, including install, can be found at *http://svnbook.red-bean.com*

Use SVN to checkout a specific release source, the base URL is http://mobicents.googlecode.com/svn/tags/servers/jain-slee/tools/twiddle, then add the specific release version, lets consider 1.0.0.CR1.

```
[usr]$ svn co http://mobicents.googlecode.com/svn/tags/servers/jain-slee/tools/twiddle/1.0.0.CR1 twiddle-1.0.0.CR1
```

2. **Building the source code**

Important

Maven 2.0.9 (or higher) is used to build the release. Instructions for using Maven2, including install, can be found at *http://maven.apache.org*

Use Maven to build the binaries.

```
[usr]$ cd twiddle-1.0.0.CR1
[usr]$ mvn install
```

Once the process finishes you should have the `binary` jar file in the `target` directory. Also `${tool.binary.dir}` directory should be present in `target` directory. It contains required libraries and scripts which allow to invoke JSLEE Twiddle.

2.2.2. Development Trunk Source Building

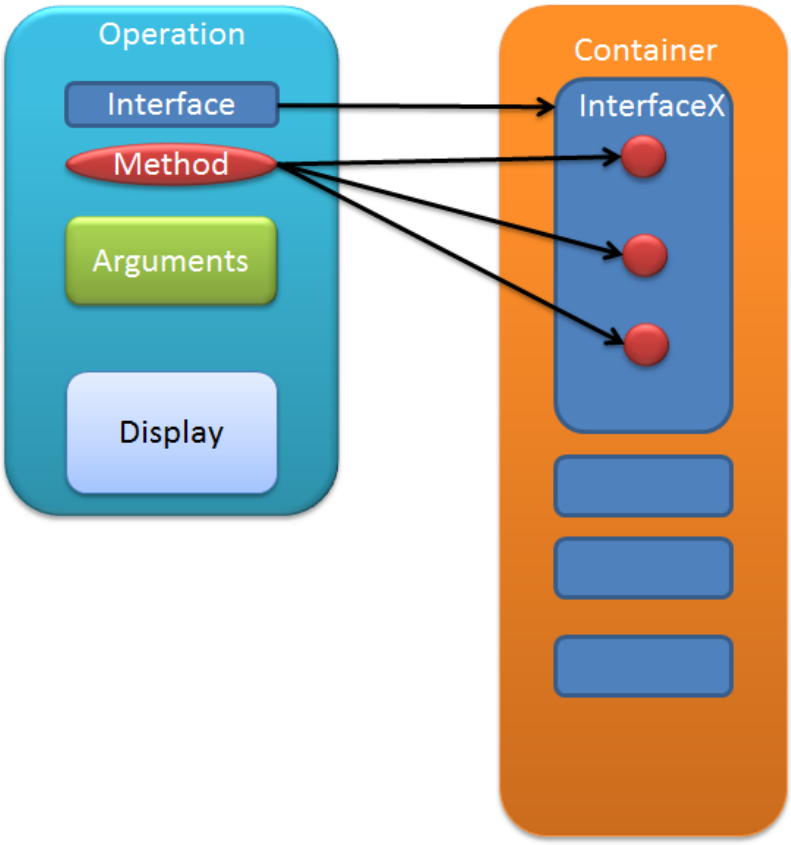
Similar process as for *Section 2.2.1, "Release Source Code Building"*, the only change is the SVN source code URL, which is http://mobicents.googlecode.com/svn/trunk/servers/jain-slee/tools/twiddle.

Design Overview

Mobicents JSLEE Twiddle is command line tool. It is based on `JBoss` JMX `Twiddle`. Mobicents specific interaction capabilities are created over base provided by `JBoss` `Twiddle`. Top overview of design is depicted on diagram below:

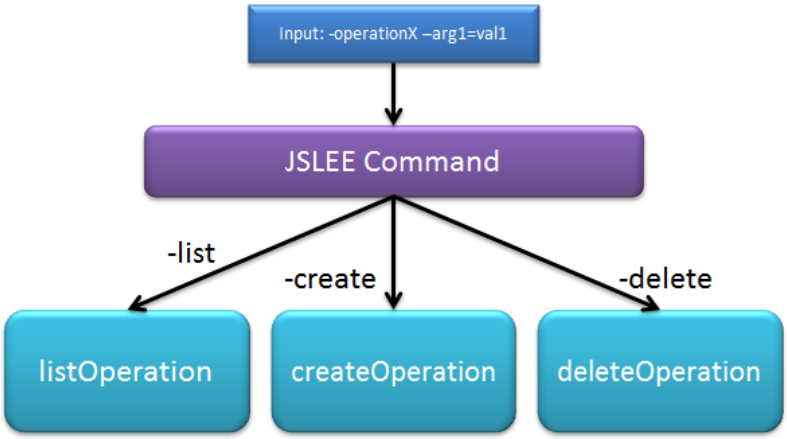
Mobicents JSLEE Twiddle General design

Mobicents JSLEE Twiddle acts on base of `command` and `operation`. `Command` supports various operations. Operation is aware of specific resource that needs to be called and how it should be accessed to perform certain task. It also knows how to present specific result to user.



Mobicents JSLEE Twiddle Operation

Based on user input, `command` makes decision which operation should be created and invoked. Top overview of `command` and `operation` relation is depicted on diagram below:



Mobicents JSLEE Twiddle JSLEE Command - Operation relation

`Commands` group operation which logically belong to one category or act on same set of resources, ie. services.

Commands

This chapter explains how to use features inherited from `Twiddle`. Also it contains list of supported `commands` along with comprehensive description of each `command` and `operations` supported by specific `command`. Note that commands in this chapter operate over standard JSLEE management interfaces.

Important

White spaces ARE NOT supported within argument values.

Important

Specific shell characters must be escaped or value of argument must be quoted.

Important

Some terminal input include "\" - it indicates that command is continued from new line, if copy/paste is used on example, this sign should be removed.

Note

Options and arguments can have long and short form. Short form `-s`, accept parameter without equal sign, ie: -slocalhost. Long options require equals sign, ie: --server=localhost

Note

Mobicents JSLEE Twiddle `operations` support long and short form. Operation options are specified ONLY in long form.

4.1. Twiddle

Mobicents JSLEE Twiddle provide two scripts to run command line, `Windows` batch and `*NIX` bash script. Both accept the same set of options/parameters:

```
twiddle.sh [options] <command> [command_arguments]
```

- options - options which affect how `Twiddle` executes, ie. address of server, user name/password
- command - command which should handle CLI invocation
- command_arguments - options and arguments specific for command

Following `options` can be passed directly to `twiddle`:

--help-commands
: displays available commands along with simple description.

-H<commandName>
: displays detailed help message for command with specified `commandName`.

-D<name>[=<value>]
: creates system property with specific name. If value is passed it is assigned to it.

-s<url> or --server=<url>
: specifies `URL` of RMI container. `URL` has following form: <IP[:PORT]>

-a<name> or --adapter=<name>
: specifies RMI adapter to be used. Example value: jmx/rmi/RMIAdaptor

-u<name> or --user=<name>
: specifies name of user to be used for AA

-p<name> or --password=<name>
: specifies password for user to be used for AA

-q or --quiet
: instructs `Twiddle` to execute without output

4.2. slee

`slee` command interacts with `javax.slee.management:name=SleeManagement` MBean. It is standard JSLEE MBean. `slee` allows basic operations like start/stop.

It supports following operations:

Table 4.1. Operations table for `slee` command

Operation	Short form	Long form	Argument	Options
start	-r	--start	No	No
stop	-s	--stop	No	No
shutdown	-d	--shutdown	No	No
info	-i	--info	No	No

4.2.1. start

`start` operation starts container. Container has to be in `STOPPED` state. This operation does not accept options nor arguments. Example invocation:

Example 4.1. Start container

```
./twiddle.sh slee -r
```

```
./twiddle.sh slee --start
```

4.2.2. stop

`stop` operation stops container. Container has to be in `RUNNING` state. This operation does not accept options nor arguments.Example invocation:

Example 4.2. Stop container

```
./twiddle.sh slee -s
```

```
./twiddle.sh slee --stop
```

4.2.3. shutdown

`shutdown` operation terminates container. Difference to `stop` is that container is no longer accessible. This operation does not accept options nor arguments.Example invocation:

Example 4.3. Shutdown container

```
./twiddle.sh slee -d
```

```
./twiddle.sh slee --shutdown
```

4.2.4. info

`info` operation displays information about container. This information depends on implementation. This operation does not accept options nor arguments.Example invocation:

Example 4.4. Print info about container

```
./twiddle.sh slee -i
```

```
./twiddle.sh slee --info
```

4.3. deploy

`deploy` command interacts with `javax.slee.management:name=Deployment` Bean. It is standard SLEE MBean. `deploy` allows to access information about deployed components and change list of deployed components. Also it allows access to components meta data. It supports following operations:

Table 4.2. Operations table for `deploy` command

Operation	Short form	Long form	Argument	Options
list	-l	--list	No	Yes
installed	-y	--installed	No	Yes
install	-i	--install	Yes	No
un-install	-u	--un-install	Yes	No
desc	-d	--desc	No	Yes
ref	-r	--ref	Yes	No

4.3.1. list

`list` operation as name suggests lists various information about deployed components. It does not take argument. It requires options to be present, to indicate type of information that has to be listed. `list` supports following options:

--sbbs
: instructs to list deployed `SbbIDs`. Optionally this option takes argument. Argument must be in form of `ServiceID`: ServiceID[name=CallBlockingService,vendor=org.mobicents,version=0.1] . In case argument is present, `SbbIDs` belonging to service are listed. Example calls:

Example 4.5. List all deployed SBBs

```
twiddle.sh deploy -l --sbbs
```

Example 4.6. List SBBs for service

```
twiddle.sh deploy -l \
--sbbs=ServiceID[name=CallBlockingService,vendor=org.mobicents,version=0.1]
```

--services
: instructs command to list deployed services(ServiceID). Does not take argument.

--events
: instructs command to list deployed event (EventTypeID). Does not take argument.

--ra-types
: instructs command to list deployed RA Types (ResourceAdaptorTypeID). Does not take argument.

--ras
: instructs command to list deployed RAs (ResourceAdaptorID). Does not take argument.

--dus
: instructs command to list deployed DUs (DeployableUnitID). Does not take argument.

--profile-spec
: instructs command to list deployed profile specifications (ProfileSpecifiactionID). Does not take argument.

4.3.2. installed

`installed` operation checks if certain component is deployed in container. Does not take argument. Requires one of options to be present. Supports following options:

--duid
: checks if DeployableUnit is deployed. Option requires argument of form: DeployableUnitID[url=file:/E:/servers/jboss-5.1.0.GA/server/default/deploy/mgcp-ra-DU-2.2.0-SNAPSHOT.jar/]

--cid
: checks if component is deployed. Option requires argument which is valid component ID, that is ServiceID, SbbID, etc.

Example 4.7. Check if profile specification is deployed

```
twiddle.sh deploy -y --profile-spec=\
ProfileSpecificationID[name=ResourceInfoProfileSpec,vendor=javax.slee,version=1.0]
```

4.3.3. install

`install` operation installs DeplyableUnit indicated by path argumnt. Does not accept options. Operation requires argument which is path to deployable unit.

Example 4.8. Deploy DU

```
twiddle.sh deploy -i/data/sip11-ra-DU-2.2.0-SNAPSHOT.jar
```

Example 4.9. Deploy DU

```
twiddle.sh deploy --install=file:///C:/workspace/resources/sip11-ra-DU-2.2.0-SNAPSHOT.jar
```

4.3.4. un-install

`un-install` operation removes DeployableUnit from container. Does not support options. It requires valid deployableUnitID as argument: DeployableUnitID[url=file:/E:/servers/jboss-5.1.0.GA/server/default/deploy/call-controller2-DU-2.2.0-SNAPSHOT.jar/]

Example 4.10. UnDeploy DU

```
twiddle.sh deploy --un-install=DeployableUnitID[url=file:/E:/servers/jboss-5.1.0.GA/server/default/deploy/call-controller2-DU-2.2.0-SNAPSHOT.jar/]
```

4.3.5. desc

`desc` operation fetchese descriptor of component. Does not take argument, requires one of options to be present:

--duid
: fetches for DeployableUnit. Option requires argument in form of DeployableUnitID. Option accepts array argument.

Example 4.11. Fetch descriptors for DUs

```
twiddle.sh deploy -d --duid=DeployableUnitID[url=file:/E:/servers/jboss-5.1.0.GA/server/default/deploy/call-controller2-DU-2.2.0-SNAPSHOT.jar/];DeployableUnitID[url=file:/E:/servers/jboss-5.1.0.GA/server/default/deploy/sip-services-DU-2.2.0-SNAPSHOT.jar/]
```

--cid
: fetches for DeployableUnit. Option requires argument in form of ComponentID. Option accepts array argument.

Example 4.12. Fetch descriptors for event

```
twiddle.sh deploy -d --cid=\
EventTypeID[name=javax.sip.Dialog.REGISTER,vendor=net.java.slee,version=1.2];\
SbbID[name=CallBlockingSbb,vendor=org.mobicents,version=0.1]
```

Example 4.13. Fetch descriptors for event and SBB

```
twiddle.sh deploy -d --cid=\
EventTypeID[name=javax.sip.Dialog.REGISTER,vendor=net.java.slee,version=1.2]
```

4.3.6. ref

`ref` operation determines which components reference specific component. Operation does not support options. It requires ComponentID as argument.

Example 4.14. Check which components reference RA Type

```
twiddle.sh deploy -rResourceAdaptorTypeID[name=jain-mgcp,vendor=net.java,version=2.0]
```

4.4. alarm

`alarm` command interacts with `javax.slee.management:name=Alarm` Bean. It is standard SLEE MBean. `alarm` command allows to manage alarms, for instance check for existence or remove. . It supports following operations:

Table 4.3. Operations table for `alarm` command

Operation	Short form	Long form	Argument	Options
clear	-c	--clear	No	Yes
list	-l	--list	No	Yes
descriptor	-d	--descriptor	Yes	No
active	-a	--active	Yes	No

4.4.1. clear

`clear` operation turns down alarms based on criteria passed in options. It requires either *`--id`* or *`--nsrc`* to be present. `clear` command supports following options:

--id
: this option specifies id of alarm which should be cleared. Option requires string argument, example: *`415f719e-3a3d-42b4-acc1-4e84706f031a`*. This option excludes other options.

--nsrc
: specifies notification source for which alarms should be cleared. It requires argument which is valid notification source: RAEntityNotification[entity=LabRA]. This option excludes *`--id`*. Optionally it can be followed by *`--type`*.

--type
: specifies type of alarm to be cleared. Type is service/component implementation specific. Type of alarm is passed as alarms meta data when it is created. Option requires argument which is string representation of type. Example call

Example 4.15. Clear alarms of certain type for ProfileTable

```
./twiddle.sh alarm -c --nsrc=ProfileTableNotification[table=SMSSubscription] \
    --type=TransientFailure
```

4.4.2. list

`list` operation lists all active alarms(their IDs). It does not require argument or option. It supports following options.

--nsrc
: optional option. It specifies notification source. It limits `list` command output to alarms active for specified source. Requires argument which is valid notification source: ProfileTableNotification[table=SMSSubscription]

Example 4.16. List alarms for ProfileTable

```
./twiddle.sh alarm -l --nsrc=ProfileTableNotification[table=SMSSubscription]
```

4.4.3. descriptor

`descriptor` operation fetches meta data for particular alarm. This includes message, time stamp, level,.. etc. It requires alarm ID as argument. It also accepts array argument. Members of array are separated with `;`. Example calls:

Example 4.17. Fetch descriptor for alarm

```
./twiddle.sh alarm -d415f719e-3a3d-42b4-acc1-4e84706f031a
```

Example 4.18. Fetch descriptor for alarms

```
./twiddle.sh alarm -d415f719e-3a3d-42b4-acc1-4e84706f031a;415f719e-3a3d-42b4-acc1-4e84706f031b;415f719e-3a3d-42b4-acc1-4e84706f031c
```

4.4.4. active

`active` operation simply checks if alarm is still active. It requires alarm id as argument.

4.5. resource

`resource` command interacts with `javax.slee.management:name=ResourceManagement` Bean. It is standard SLEE MBean. `resource` command allows management of Resource Adaptor entities: create, remove, start, stop, update configuration,... . It supports following operations:

Table 4.4. Operations table for `resource` command

Operation	Short form	Long form	Argument	Options
bind	-b	--bind	No	Yes
unbind	-u	--unbind	Yes	No
activate	-a	--activate	Yes	No

Operation	Short form	Long form	Argument	Options
deactivate	-d	--deactivate	Yes	No
create	-c	--create	No	Yes
remove	-r	--remove	Yes	No
update-config	-p	--update-config	No	Yes
list	-l	--list	No	Yes
get	-g	--get	No	Yes

4.5.1. bind

`bind` operation creates binding between Resource Adaptor Entity and specific link(associates RA Entity to link). It does not take argument, however it requires options:

--link-name
: specifies link name to be used in bind operation. This name can be referenced in SBB descriptors. Requires argument. This option is mandatory.

--entity-name
: specifies Resource Adaptor Entity name which will be bound to link. Requires valid entity name as argument. This option is mandatory.

Example 4.19. Create link for SIP RA

```
./twiddle.sh resource --bind --link-name=SIP_30 --entity-name=SipRA
```

4.5.2. unbind

`unbind` operation destroys binding created with *Section 4.5.1, "bind"* operation. It does not support any options. It requires link name as argument.

Example 4.20. Remove link for SIP RA

```
./twiddle.sh resource --unbind=SIP_30
```

4.5.3. activate

`activate` operation activates Resource Adaptor Entity. It does not support options. It requires entity name as argument.

Example 4.21. Activate SIP RA

```
./twiddle.sh resource --activate=SipRA
```

4.5.4. deactivate

`deactivate` operation renders Resource Adaptor Entity inactive. It does not support options. It requires entity name as argument.

Example 4.22. Activate SIP RA

```
./twiddle.sh resource --deactivate=SipRA
```

4.5.5. create

`create` operation creates new Resource Adaptor Entity. This operation does not activate created entity. It does not support argument, it requires all supported options to be present:

--entity-name
: specifies entity name to be used for new Resource Adaptor Entity. Requires argument.

--ra-id
: specifies ResourceAdaptorID to be used as ResourceAdaptorObject for new entity. Requires ResourceAdaptorID as argument.

--config
: Specifies ConfigurationProperties passed to ResourceAdaptorObject during configuration part of its life-cycle. Requires string representation of ConfigProperties. Config property has general form of: (name:java.type=value) and array has different form, than in components, members are separated with `,` : [(cnf.prop),(cnf.prop)]

Example 4.23. Create SIP RA

```
./twiddle.sh resource -c --entity-name=SipRA --ra-id=\
ResourceAdaptorID[name=JainSipResourceAdaptor,vendor=net.java.slee.sip,version=1.2]\
--config=[(javax.sip.TRANSPORT:java.lang.String=UDP),\
(javax.sip.IP_ADDRESS:java.lang.String=),\
(javax.sip.PORT:java.lang.Integer=5060)]
```

Example 4.24. Create SIP RA with empty config

```
./twiddle.sh resource -c --entity-name=SipRA --ra-id=\
ResourceAdaptorID[name=JainSipResourceAdaptor,vendor=net.java.slee.sip,version=1.2] \
--config=[]
```

4.5.6. remove

`remove` operation removes *`INACTIVE`* Resource Adaptor Entity. It does not support options. It requires Resource Adaptor Entity name as argument.

4.5.7. update-config

`update-config` operation updates configuration in specified entity. If new properties are verified by Resource Adaptor Object, old configuration properties are discarded and replaced with new set. This operation does not support argument, it requires options to specify entity name and new configuration properties. `update-config` supports following options:

--entity-name
: specifies entity name to be used for new Resource Adaptor Entity. Requires argument.

--config
: Specifies ConfigurationProperties passed to ResourceAdaptorObject during confgiuration part of its lifecycle. Requires string representation of ConfigProperties. Config property has general form of: (name:java.type=value) and array has different form, than in components, members are separated with `,` : [(cnf.prop),(cnf.prop)]

4.5.8. list

`list` operation queries container for certain information and lists output. It does not support argument. It requires one of following options to be present:

--ra-entities
: instructs operation to list Resource Adaptor Entities present in container. It supports optional argument. Argument is of ResourcedaptorID type, if its present, entities for this ID are listed.

Example 4.25. List entities for SIP RA

```
./twiddle.sh resource -l --ra-entities=\
ResourceAdaptorID[name=JainSipResourceAdaptor,vendor=net.java.slee.sip,version=1.2]
```

--ra-entities-in-state
: instructs operation to list entities in certain state. Requires string representation of entity state as argument. Resource Adaptor can be in one of following states:

  - INACTIVE
  - STOPPING
  - ACTIVE

--ra-entities-by-link
: instructs operation to list Resource Adaptor Entity name(s) for given link name(s). Requires link name as argument, it accepts array argument:

Example 4.26. List entities for link names

```
./twiddle.sh resource -l --ra-entities-by-link=SIP_30;SIP_40
```

--links
: instructs operation to list Resoruce Adaptor Entity links. It supports optional argument. Argument must be valid Resource Adaptor Entity name. If argument is specified, operation will list link names only for that specific entity.

Example 4.27. List link names for entity

```
./twiddle.sh resource -l --links=SipRA
```

--sbbs
: instructs operation to list SBB IDs which declare reference Resource Adaptor Entity link. Requires link name as argument:

Example 4.28. List SBBs for Sip RA

```
./twiddle.sh resource -l --sbbs=SIP_30
```

4.5.9. get

`get` operation fetches certain information from container. It does not support argument, it requires one option to be present:

--ra-id
: fetches ResoruceAdaptorID. Requires Resource Adaptor Entity name as argument.

--state
fetches state of certain entity. Requires Resource Adaptor Entity name as argument.

--config-by-id
fetches default ConfigurationProperties for Resource Adaptor Object. Requires ResourceAdaptorID as argument.

--config-by-name
fetches ConfigurationProperties for existing entity. Requires Resource Adaptor Entity name as argument.

4.6. service

`service` command interacts with `javax.slee.management:name=ServiceManagement` MBean. It is standard JSLEE MBean. `service` command allows to perform basic management operations on services , like activation, deactivation, state query . It supports following operations:

Table 4.5. Operations table for `service` command

Operation	Short form	Long form	Argument	Options
activate	-a	--activate	yes	No
deactivate	-d	--deactivate	Yes	No
deactivate-and-activate	-c	--deactivate-and-activate	No	Yes
services	-i	--services	Yes	No
state	-o	--state	Yes	No

4.6.1. activate

`activate` operation simply activates inactive service(Container will inform instance by emiting proper event). It does not support options. It requires ServiceID as argument, it accepts array of ServiceIDs as argument:

Example 4.29. Activate service

```
./twiddle.sh service \
-aServiceID[name=My_Service_Name,vendor=test_dummy_inc,version=1.0]
```

Example 4.30. Activate services

```
```

```
./twiddle.sh service \
-aServiceID[name=My_Service_Name,vendor=test_dummy_inc,version=1.0];\
ServiceID[name=My_Service_Name2,vendor=spartan_dummy_inc,version=1.0]
```

4.6.2. deactivate

`deactivate` operation simply deactivates active service(Container will inform instance by emiting proper event). It does not support options. It requires ServiceID as argument, it accepts array of ServiceIDs as argument:

Example 4.31. Deactivate service

```
./twiddle.sh service \
-dServiceID[name=My_Service_Name,vendor=test_dummy_inc,version=1.0]
```

Example 4.32. Deactivate services

```
./twiddle.sh service \
-dServiceID[name=My_Service_Name,vendor=test_dummy_inc,version=1.0];\
ServiceID[name=X_Service,vendor=test_dummy_inc,version=1.0]
```

4.6.3. deactivate-and-activate

`deactivate-and-activate` operation deactivates and activates certain set of services(order of options does not affect order of container operations). It does not support argument, it requires options to specify services sets:

--ta
: specifies set of services to activate. Requires ServiceID as argument(accepts array argument also).

--td
: specifies set of services to deactivate. Requires ServiceID as argument(accepts array argument also).

Example 4.33. Deactivate and activate services

```
./twiddle.sh service -c \
--ta=ServiceID[name=My_Service_Name,vendor=test_dummy_inc,version=1.0]\
;ServiceID[name=X_Service,vendor=test_dummy_inc,version=1.0]\
--td=ServiceID[name=My_Service_Name,vendor=test_dummy_inc,version=1.0]
```

4.6.4. services

`services` operation lists services in given state. Does not support options. Requires valid service state as argument. Service can be either *`Active`* or *`Inactive`*

4.6.5. state

`state` operation queries container about state of service. It does not support options. It requires ServiceID as argument.

4.7. profile

`profile` command interacts with `javax.slee.management:name=ProfileProvisioning` MBean. It is standard JSLEE MBean. This command allows to perform basic management operations on profile tables and profiles, like creation and removal . It supports following operations:

Table 4.6. Operations table for `profile` command

Operation	Short form	Long form	Argument	Options
list	-l	--list	No	Yes
create	-c	--create	No	Yes
remove	-r	--remove	No	Yes
rename	-n	--rename	No	Yes
get	-g	--get	No	Yes

4.7.1. list

`list` operation lists information based on passed options. It does not support argument. One of following options must be present:

--table
: instructs operation to list table names present in container. It supports optional argument. If ProfileSpecificationID is present as argument, table names are listed only for that specification:

Example 4.34. List table names for specific Profile Specification

```
./twiddle.sh profile -l --table=\
ProfileSpecificationID[name=CallControlProfileCMP,vendor=org.mobicents,version=0.1]
```

--profile
: instructs operation to list profile IDs. It requires ProfileTable name as argument.

4.7.2. create

`create` operation creates resource in container. Based on passed option it creates profile table or profile in table. It does not support argument, following options are supported:

--profile-name
: specifies profile name. It is used in conjunction with *`--table-name`*, it excludes *`--profile-spec`*. Requires profile name as argument.

--table-name
: specifies table name. Requires table name as argument.

--profile-spec
: specifies profile specification. It is used in conjunction with *`--table-name`*, it excludes *`--profile-name`*. Requires ProfielSpecificationID as argument.

Example 4.35. Create profile table

```
./twiddle.sh profile -c --table-name=NewCallControl --profile-spec=\
ProfileSpecificationID[name=CallControlProfileCMP,vendor=org.mobicents,version=0.1]
```

Example 4.36. Create profile in table

```
./twiddle.sh profile -c --table-name=NewCallControl --profile-name=newuser
```

4.7.3. remove

`remove` operation removes component based no options. It does not support argument. It supports following options:

--table-name
: specifies table name for operation. Requires table name as argument.

--profile-name
specifies profile name for operation. Requires profile name as argument.

Example 4.37. Remove profile table

```
./twiddle.sh profile -r --table-name=NewCallControl
```

Example 4.38. Remove profile from table

```
./twiddle.sh profile -r --table-name=NewCallControl --profile-name=newuser
```

4.7.4. rename

`rename` operation changes name of profile table, effectively moving profiles to new table in the process. It does not support argument. It supports following options:

--current-name
specifies current(old) name of profile table. Requires profile table name as argument.

--new-name
specifies new name for profile table. Requires profile table name as argument.

Example 4.39. Rename profile table

```
./twiddle.sh profile -n --current-name=CallControl --new-name=OldCCTable
```

4.7.5. get

`get` operation queries container for certain information. Type of retrieved information depends on sub-options. It does not support argument. Following options are supported:

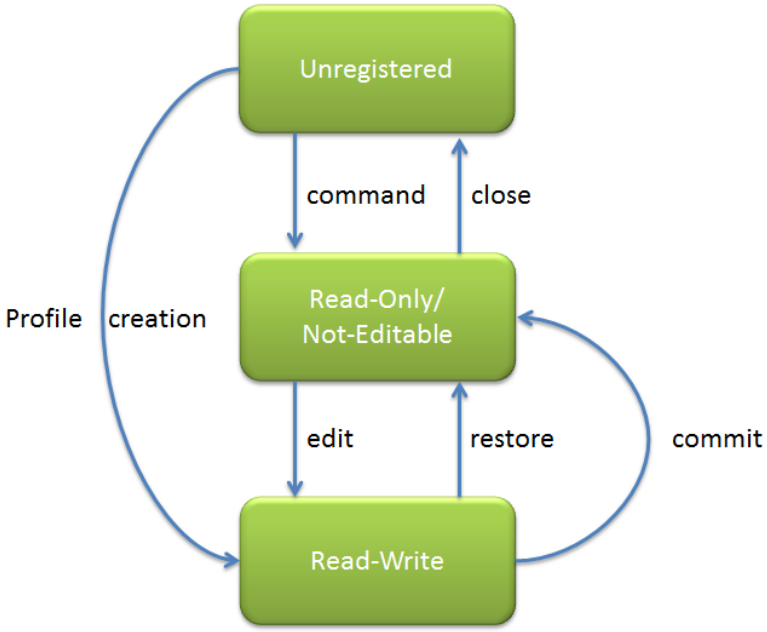
--profile-spec
retrieves profile ProfileSpecificationID for certain table. Requires profile table name as argument.

Example 4.40. Get specification id

```
./twiddle.sh profile -g --profile-spec=CallControl
```

4.8. profile.edit

`profile.edit` command interacts with specific profile MBean. This MBean is standard SLEE bean. It is generated based on profile specification. `profile.edit` allows to edit profile data(specific profile and default). Profile MBean follows state machine depicted on diagram below:



Profile Edit FSM

`profile.edit` command expects two non operation/option arguments. Form of invocation looks as follows:

```
profile.edit <profileTableName> <profileName> <-operation[[arg] | [--option[=arg]]]*>
```

It supports following operations:

Table 4.7. Operations table for `profile.edit` command

Operation	Short form	Long form	Argument	Options
list	-l	--list	No	No
dirty	-d	--dirty	No	No
write	-w	--write	No	No
edit	-e	--edit	No	No
commit	-c	--commit	No	No
restore	-r	--restore	No	No
close	-o	--close	No	No
get	-g	--get	Yes	No

Operation	Short form	Long form	Argument	Options
set	-s	--set	No	Yes

4.8.1. list

`list` operation lists available profile attributes and information associated with each attribute. Does nor support options not arguments.

4.8.2. dirty

`dirty` operation checks if profile data has been modified and not committed yet. Does not support options nor arguments.

4.8.3. write

`write` operation checks if profile is in edit mode, that is if data can be written into it. Does not support options nor arguments.

4.8.4. edit

`edit` operation marks profile as editable(writeable). Does not support options nor arguments.

4.8.5. restore

`restore` operation revokes all changes performed on profile and marks profile as not editable. Does not support options nor arguments.

4.8.6. commit

`commit` operation commits any changes performed on profile(makes them persistent) and marks profile as not editable. Does not support options nor arguments.

4.8.7. close

`close` operation deregister profile specific MBean. Does not support options nor arguments.

4.8.8. get

`get` operation operation fetches value of profile attribute. It requires attribute name as argument.

Example 4.41. Get value of *`voiceMailEnabled`* attribute

```
./twiddle.sh profile.edit CallControl gregory12 --get=voiceMailEnabled
```

4.8.9. set

`set` operation sets profile attribute value. It does not support argument. Requires options to be present to specify name and value:

--name
: specifies name of attribute to be changed.

--value
: specifies value to be set in profile attribute. Supported value types are:

- String
- java primitives
- JSLEE types - ComponentIDs and management classes, like Address

4.9. trace

`trace` command interacts with `javax.slee.management:name=Trace` MBean. It is standard JSLEE MBean. `trace` allows to manage trace level and tracers instances. Based on options passed to operations it performs operations on 1.1 and 1.0 logging facilities. It supports following operations:

Table 4.8. Operations table for `trace` command

Operation	Short form	Long form	Argument	Options
tracers-used	-a	--tracers-used	Yes	No
tracers-set	-f	--tracers-set	No	No
set-level	-s	--set-level	No	Yes
un-set-level	-u	--un-set-level	No	Yes
get-level	-g	--get-level	No	Yes

4.9.1. tracers-used

`tracers-used` operation returns names of tracers requested by notification source. Requires notification source as argument.

Example 4.42. Get list of tracer names

```
./twiddle.sh trace -aProfileTableNotificationSource[table=ExampleTable]
```

4.9.2. tracers-set

`tracers-set` operation fetches list of tracers for which level has been set explicitly via one of `setLevel` methods. Requires notification source as argument.

4.9.3. set-level

`set-level` operation sets level of tracer. Does not support argument. Supports following options:

--cid
: specifies ComponentID for 1.0 trace facility. Requires valid ComponentID as argument. It excludes *`--nsrc`* option. Example call:

Example 4.43. Set trace level in 1.0 facility

```
./twiddle.sh trace -s --cid=\
SbbID[name=ProxySbb,vendor=mobicents,version=1.1] --level=WARNING
```

--nsrc
: specifies notification source of which tracer(!) level should be set. Excludes *`--cid`* option. Requires valid notification source as argument.

Example 4.44. Set tracer level

```
./twiddle.sh trace -s --nsrc=ProfileTableNotification[table=xxx] \
--name=tracer.name --level=WARNING
```

--name
: specifies tracer name. Requires name as argument.

--level
: specifies level for 1.0 trace facility and 1.1 tracer. Requires valid level representation as argument. It accepts following values:

  - SEVERE
  - WARNING
  - INFO
  - CONFIG
  - FINE
  - FINER
  - FINEST

4.9.4. un-set-level

`un-set-level` operation unsets level of SLEE 1.1 tracer. Invoking this operation informs Trace facility to derive level for tracer from its parent. Does not support argument. Following options are supported:

--nsrc
: specifies notification source of which tracer(!) level should be unset. Requries valid notification source as argument.

--name
: specifies tracer name. Requires name as argument.

Example 4.45. Set tracer level

```
./twiddle.sh trace -u --nsrc=ProfileTableNotification[table=xxx] --name=tracer.name
```

4.9.5. get-level

`get-level` operation retrieves level of trace facility or tracer. Does not support arument. Following options are supported:

--cid
: specifies ComponentID for 1.0 trace facility. Requires valid ComponentID as argument. It excludes *`--nsrc`* option. Example call:

Example 4.46. Get trace level in 1.0 facility

```
./twiddle.sh trace -g --cid=SbbID[name=ProxySbb,vendor=mobicents,version=1.1]
```

--nsrc
: specifies notification source of which tracer(!) level should be fetched. Excludes *`--cid`* option. Requries valid notification source as argument.

Example 4.47. Set tracer level

```
./twiddle.sh trace -g --nsrc=ProfileTableNotification[table=xxx] --name=tracer.name
```

--name
: specifies tracer name. Requires name as argument.

4.10. usage.service

`usage.service` command interacts with service specific usage MBeans and Usage Notification manager MBean. `usage.service` allows to manage usage sets and parameters, for instance get value of parameter or turn on JMX notification for certain parameter. General form of command looks as follows:

```
usage.service <ServiceID> [SbbID] [SetName] <-operation[[arg] | [--option[=arg]]*]>
```

Note that access to default set does not require [SetName] to be present. It supports following operations:

Table 4.9. Operations table for `usage.service` command

Operation	Short form	Long form	Argument	Options
list	-l	--list	No	Yes
get	-g	--get	No	Yes
reset	-r	--reset	No	No
create	-c	--create	No	No
delete	-d	--delete	No	No
notify	-n	--notify	No	Yes

4.10.1. list

`list` operation lists certain information about parameter sets. It does not support argument. It support following options:

--sets
: instructs operation to list available sets for ServiceID and SbbID

Example 4.48. List available sets

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] -l --sets
```

--parameters
: instructs operation to list parameters from set and metadata associated with each.

Example 4.49. List parameters in set

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] -l --parameters
```

4.10.2. get

`get` operation fetches value of parameter in certain set(default or named). It does not support argument. It supports following options:

--name
: specifies name of parameter. This is mandatory option. Parameter name starts with upper case character. Requires parameter name as argument.

--rst
: instructs operation that parameter should be reset after once value is obtained. Does not support argument, its a "flag" like option.

Example 4.50. Get parameter value from default set

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] -g --name=StreetName
```

Example 4.51. Get parameter value from named set

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] Drake -g --name=StreetName
```

Example 4.52. Get parameter value from named set and reset

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] Drake -g --name=StreetName --rst
```

4.10.3. reset

`reset` operation resets all parameters in certain set or sets. It does not support argument. It supports following options:

--all
: instructs operation to reset all usage sets. It does not require argument.

Example 4.53. Reset parameters in named set

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] Drake -r
```

Example 4.54. Reset parameters in all sets

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] -r --all
```

Example 4.55. Reset parameters in all sets

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] -r --all
```

4.10.4. create

`create` operation creates named usage parameter set. It does not support argument nor options.

Example 4.56. Create usage parameter set

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] Drake -c
```

4.10.5. delete

`delete` operation deletes named set from container. It does not support argument nor options.

Example 4.57. Delete usage parameter set

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] Drake -d
```

4.10.6. notify

`notify` operation performs actions on notification aspect of usage sets. It allows to to run on/off notifications for certain parameter and query about state of notification. It does not support argument. It supports following options:

--name
: specifies name of parameter. It is mandatory. It requires parameter name as argument. Parameter name starts with upper case character.

--value
: instructs operation to change state of notification. This options requires boolean argument(true - notification are enabled).

Example 4.58. Turn off notifications

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] Drake \
-n --name=Calls --value=false
```

--is
: instructs operation to check state of notification. This option does not support arument.

Example 4.59. Check state of notifications

```
./twiddle.sh usage.service ServiceID[name=XXX,vendor=mobicents,version=1.0] \
SbbID[name=example,vendor=mobicents,version=0.1] Drake \
-n --name=Calls --is
```

4.11. usage.profile

`usage.profile` command interacts with service specific usage MBeans and Usage Notification manager MBean. `usage.resource` allows to manage usage sets and parameters, for instance get

value of parameter or turn on JMX notification for certain parameter. General form of command looks as follows:

```
usage.ra <ProfileTableName> [SetName] <-operation[[arg] | [--option[=arg]]*]>
```

Note that access to default set does not require [SetName] to be present.

For description of operations and supported options please refer to *Section 4.10, “usage.service”*

4.12. usage.resource

`usage.resource` command interacts with service specific usage MBeans and Usage Notification manager MBean. `usage.resource` allows to manage usage sets and parameters, for instance get value of parameter or turn on JMX notification for certain parameter. General form of command looks as follows:

```
usage.ra <RAEntityName> [SetName] <-operation[[arg] | [--option[=arg]]*]>
```

Note that access to default set does not require [SetName] to be present.

For description of operations and supported options please refer to *Section 4.10, “usage.service”*

Commands Extended

This chapter describes commands that are specific to Mobicents JSLEE container. Following commands are not bound to work on any other implementation of JSLEE specification.

This set of commands is invoked exactly in the same way as commands interacting with standard JLSEE management interface.

5.1. activity

`activity` command interacts with `org.mobincets.slee:name=ActivityManagementMBean` Bean. It is non standard SLEE MBean. Please refer to its documentation for details. `activity` allows to access information about activities and perform management operations. It supports following operations:

Table 5.1. Operations table for `activity` command

Operation	Short form	Long form	Argument	Options
count	-c	--count	No	No
query	-q	--query	No	Yes
idle	-i	--idle	No	Yes
list	-l	--list	No	Yes

5.1.1. count

`count` operation simply returns current count of activities. It does not support argument nor options. Example call:

Example 5.1. List all deployed SBBs

```
twiddle.sh activity -c
```

5.1.2. query

`query` operation is capable of two things. First is to manage time between activity liveliness query(Refer to Mobicents JSLEE Guide for description of liveliness query). Second is triggering liveliness query. It does not support argument. Optionally it may be followed by one of following options:

--set
: specifies number of seconds between liveliness query. Requires number as argument.

Example 5.2. Set time between query to 60 seconds

```
./twiddle.sh activity -q --set=60
```

--get
: Retrieves number of seconds between liveliness queries.

Example 5.3. Trigger query

```
./twiddle.sh activity -q
```

5.1.3. idle

`idle` operation allows to manage maximum idle time for activity before it is eligible for garbage collection. It does not support argument. It supports following options:

--set
: specifies number of seconds which indicates time allowed for activity to be not accessed. If this values is exceeded, next liveliness query will try to free resource.

--get
: fetches number of seconds indicating how long activity may be not accessed before liveliness query will reclaim it.

5.1.4. list

`list` operation lists certain information based on passed options. It does not support argument. It requires one option to be present. Following options are supported:

--factories
: instructs operation to list activity contexts factories registered in container. Does not support argument. Example call:

Example 5.4. List factories

```
twiddle.sh activity -l --factories
```

--contexts
: instructs operation to list activity contexts and data associated with them. It optionally may take boolean argument. If argument is present and set to *true* operation will list details in text form, otherwise only statistic is printed for activity context.

--id-by-activity-type
: instructs operation to list ID of activity contexts based on activity type. Requires FQN of activity class as argument.

--id-by-ra-entity
: instructs operation to list ID of activity contexts based on Resource Adaptor Entity name. Requires entity name as argument.

--id-by-sbb-id
: instructs operation to list ID of activity contexts based on SbbID. Requires SbbID as argument.

5.2. router.stats

`activity` command interacts with `org.mobincets.slee:name=EventRouterStatistics` Bean. It is non standard SLEE MBean. `router.stats` allows to access statistics information of router and container executors. Note time statistics are measured in milliseconds. It supports following operations:

Table 5.2. Operations table for `router.stats` command

Operation	Short form	Long form	Argument	Options
mapped-activities	-m	--mapped-activities	Yes	No
avg-time	-a	--avg-time	No	Yes
events-routed	-r	--events-routed	No	Yes
executed-tasks	-e	--executed-tasks	Yes	No
idle-time	-i	--idle-time	Yes	No
misc-tasks	-c	--misc-tasks	Yes	No
executing-time	-t	--executing-time	Yes	No
misc-executing-time	-x	--misc-executing-time	Yes	No
routing-time	-o	--routing-time	No	Yes
print-all	-p	--print-all	No	No

5.2.1. mapped-activities

`mapped-activities` operation checks how many activities are mapped to certain executor. It requires integer number as argument, representing executor index(starting from 0).

5.2.2. avg-time

`avg-time` operation return average time spent on routing certain event type. It does not support argument. It requires `--eventTypeId` option to be present. It support following options:

--eventTypeId
: specifies event type for which average time should be fetched. It is mandatory. It requires valid EventTypeID as argument.

Example 5.5. Get average time in all executors

```
./twiddle.sh router.stats --avg-time --eventTypeId=\
EventTypeID[name=javax.slee.profile.ProfileUpdatedEvent,vendor=javax.slee,version=1.0]
```

--executor
: optional option. Indicates executor index for which statistic should be fetched.

5.2.3. events-routed

`events-routed` operation allows to retrieve information about number of routed events. It does not support argument. It requires `--eventTypeId` option to be present. It support following options:

--eventTypeId
: specifies event type for which number should be fetched. It is mandatory. It requires valid EventTypeID as argument.

Example 5.6. Get number of routed events in all executors

```
./twiddle.sh router.stats --events-routed --eventTypeId=\
EventTypeID[name=javax.slee.profile.ProfileUpdatedEvent,vendor=javax.slee,version=1.0]
```

--executor
: optional option. Indicates executor index for which statistic should be fetched.

5.2.4. executed-tasks

`executed-tasks` operation returns number of executed tasks. If no argument is specified it will return total count. If executor index is specified it will return task count for particular executor.

5.2.5. idle-time

`idle-time` operation return time in which executor was idle. Requires executor index as argument(starting from 0).

5.2.6. misc-tasks

`misc-tasks` operation returns number of non routing tasks performed by container. If optional argument is present(executor index), operation returns number for particular executor.

5.2.7. routing-time

`routing-time` operation returns time spent on routing of events. If optional argument is specified(executor index), operation returns time spent on routing by particular executor. It support following options:

--eventTypeId
: specifies event type for which number should be fetched. It is mandatory. It requires valid EventTypeID as argument.

Example 5.7. Get number of routed events in all executors

```
./twiddle.sh router.stats --routing-time --eventTypeId=\
EventTypeID[name=javax.slee.profile.ProfileUpdatedEvent,vendor=javax.slee,version=1.0]
```

--executor
: optional option. Indicates executor index for which statistic should be fetched.

5.2.8. executing-time

`executing-time` operation returns time spent on executing tasks. If optional argument is specified(executor index), operation returns time spent on routing by particular executor.

5.2.9. misc-executing-time

`misc-executing-time` operation returns time spent on executing non routing tasks. If optional argument is specified(executor index), operation returns time spent on routing by particular executor.

5.2.10. print-all

`print-all` operation prints all statistics. Does not support argument nor options.

5.3. deployer

deployer command interacts with Mobicents JSLEE container deployer. It supports following operations:

Table 5.3. Operations table for `deployer` command

Operation	Short form	Long form	Argument	Options
status	-s	--status	No	No

5.3.1. status

`status` operation is simple. It does not support argument nor options. It lists status of deployment.

5.4. congestion

`congestion` command interacts with `org.mobincets.slee:name=CongestionControlConfiguration` Bean. It is non standard SLEE MBean. Please refer to its documentation for details. `congestion` allows to change how configuration of congestion mechanism. for instance free memory level which will disable congestion mechanism. It supports following operations:

Table 5.4. Operations table for `congestion` command

Operation	Short form	Long form	Argument	Options
period	-p	--period	No	Yes
disable-level	-d	--disable-level	No	Yes
enable-level	-e	--enable-level	No	Yes
refuse-event	-f	--refuse-event	No	Yes
refuse-activity	-a	--refuse-activity	No	Yes

5.4.1. period

`period` operation allows to manage period between congestion checks. Period is expressed in seconds. This operation does not support argument. It supports following options:

--get
: instructs operation to fetch period between congestion checks. Does not support argument.

--set
: instructs operation to set period between congestion checks. Requires number of seconds as argument.

5.4.2.

`disable-level` operation allows to manage minimum level of free memory for congestion mechanism to stop limiting container. It does not support argument. It supports following options

--get
: instructs operation to fetch number MBs. Does not support argument.

--set
: instructs operation to set number of MBs to stop congestion mechanism. Requires number of MB (MegaBytes).

5.4.3. enable-level

`enable-level` operation allows to manage minimum level of free memory to trigger congestion mechanism. It does not support argument. It support following options:

--get
: instructs operation to fetch number MBs. Does not support argument.

--set
: instructs operation to set number of MBs to start congestion mechanism. Requires number of MB (MegaBytes).

5.4.4. refuse-event

`refuse-event` operation allows to manage how congestion mechanism behaves. This operation allows to manage whether events are allowed to be fired into container or not. It does not support argument. It supports following options:

--get
: instructs operation to check if events will be refused to be fired in case of congestion. Does not support argument.

--set
: instructs operation to disallow events to be fired. It requires boolean argument(true - disallow).

5.4.5. refuse-activity

`refuse-activity` operation allows to manage how congestion mechanism behaves. This operation allows to manage whether new activities can be created in container or not. It does not support argument. It supports following options:

--get
: instructs operation to check if new activities will be refused in case of congestion. Does not support argument.

--set

instructs operation to disallow new activities creation. It requires boolean argument(true - disallow).

Appendix A. Revision History

Revision History
Revision 1.0 Tue Aug 21 2010 BartoszBaranowski
Creation of the Mobicents JSLEE Twiddle User Guide.

Index

F