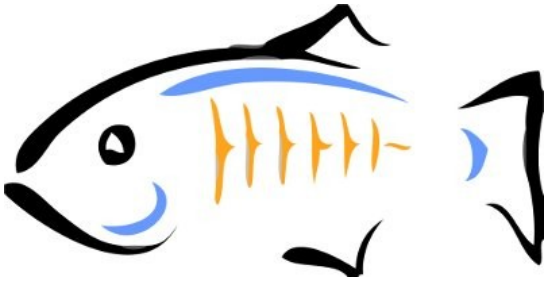

APPLICATION VERSIONING

Basic Commands & Scenarios

SERLI

Romain GRECOURT romain.grecourt@serli.com
Hervé SOUCHAUD herve.souchaud@serli.com
Mathieu ANCELIN mathieu.ancelin@serli.com
Pierre-Henri DEZANNEAU pierrehenri.dezanneau@serli.com

April 15, 2010

1 Summary

2 Requirements

2.1 Definitions

There are two kinds of syntax defined to handle versioning within GlassFish server :

Version identifier :

A version identifier satisfies the following regular expression :

*[A-Z|a-z|0-9]+([_|.|-][A-Z|a-z|0-9]+)**

This expression allows the use of maven/OSGi version values (major.minor.micro-QUALIFIER or QUALIFIER-major.minor.micro) as identifiers.

Version expression :

A version expression is either a version identifier or a wildcard expression that contains one or more « * » characters as the wildcard. GF applies the associated operation to all versions of the application which match the expression.

Version identifier and expression are used as suffixes to application names, introduced by a semi-colon « ; ». For example, « myApp;Beta » or « inventory;RC* ».

Untagged version :

The untagged version is a version having an empty string as identifier and that follows the same rules as all others version. This concept was introduced to maintain backward compatibility with GlassFish v2 users.

2.2 Context of commands use

An identifier is bound to an unique version while an expression matches a set of versions. Consequently there are two different command groups using either an identifier or an expression as argument :

Identifier aware operations	Expression aware operations
•*“deploy”* •*“enable”* •*“get-client-stubs”*	•*“undeploy”* •*“disable”*

3 Basic commands list

3.1 Version deployment

The deployment process is performed by the deploy command which is an identifier aware operation. The syntaxes presented above are used as name option, therefore there is no relationship between the archive name and the versioning system at all.

Moreover deploying an enabled version will automatically disable the previous enabled version.

• A user wants to deploy the application “foo” (untagged version)

```
$ asadmin deploy foo.war
```

• A user wants to deploy an enabled version :

```
$ asadmin deploy --name='foo;1' foo_1.war
```

• A user wants to deploy a new disabled version :

```
$ asadmin deploy --enable=false --name='foo;2' foo_2.war
```

• A user wants to deploy a version named “RC-1” :

```
$ asadmin deploy --name='foo;RC-1' foo_RC-1.war
```

• A user wants to deploy a version named “RC-2” from a directory :

```
$ asadmin deploy –-name='foo;RC-2' /home/user/applications/foo_RC-2/
```

3.2 Version redeployment

The redeployment process is accomplished by the redeploy command which simply invokes the deploy command correctly. The use and the behavior of this command is similar to deploy command :

```
$ asadmin deploy --name='foo;1' foo_1.war (1st deployment)
$ asadmin redeploy --name='foo;1' foo_1.war (redeployment)
$ asadmin deploy --force=true --name='foo;1' foo_1.war (redeployment)
```

The last two commands are equivalent. The force option is needed for redeployment when using deploy command so the user does not accidentally overwrite the version.

3.3 Version switch

The user can perform switch operation by using the enable command which is an identifier aware operation because only one application can be enabled at runtime. The enable process disables the currently enabled version before enabling the one specified by the user.

The primary argument of this command is the application name, it's used to specify the version to enable (the targeted version must be previously deployed).

- A user wants to activate the untagged version :

```
$ asadmin enable foo
```

- A user wants to activate a version with a simple identifier :

```
$ asadmin enable 'foo;2'
```

- A user wants to activate a version with a more complex identifier :

```
$ asadmin enable 'foo;RC-1'
```

3.4 Version deactivation

The disable command is an expression aware operation which allows the user to disable any version.

Although only one version of an application can be enabled at runtime, this command accepts version expressions to be more flexible for users. Only one application can be disabled by the command process even if the targeted expression matches more than one version.

If the currently enabled version isn't matched by the expression, the command will result in a no-operation.

• A user wants to disable the untagged version :

```
$ asadmin disable foo
```

• A user wants to disable a version with a simple identifier :

```
$ asadmin disable 'foo;2'
```

• A user wants to disable the currently enabled version, but he doesn't know the version identifier :

```
$ asadmin disable 'foo;*'
```

The expression “foo;*” matches all versions of “foo” application and so the enabled version.

• A user wants to disable the currently enabled version if its identifier begins with “RC” :

```
$ asadmin disable 'foo;RC*'
```

3.5 Version(s) undeployment

The undeployment process is performed by the undeploy command which is an expression aware operation because several versions can be undeployed at the same time unlike the disable command that process over one version at most whatever expression supplied.

Indeed the undeployment process deletes matched version bits in repository and disables the currently enabled version if the expression matches it.

• A user wants to undeploy the untagged version :

```
$ asadmin undeploy foo
```

• A user wants to undeploy a version with a simple identifier :

```
$ asadmin undeploy 'foo;2'
```

• A user wants to undeploy all versions (including the untagged) of “foo” application :

```
$ asadmin undeploy 'foo;*'
```

• A user wants to undeploy all the version with an identifier begining with “RC” :

```
$ asadmin undeploy 'foo;RC*'
```

3.6 Version client- stubs

The client-stubs generation process is performed by the get-client-stubs command which is an identifier aware operation because the generated files can only match one version.

The targeted version is specified as identifier.

• A user wants to retrieve the stubs of an application with a specific version :

```
$ asadmin get-client-stubs --appname='foo;RC-1.0.0' [path]/stubs/
```

• A user wants to retrieve the stubs of an application with the untagged version :

```
$ asadmin get-client-stubs --appname='foo' [path]/stubs/
```

3.7 Version list

The “list-applications” and “list-components” commands allow users to display the versions of each deployed application, no identifier will be shown for untagged applications. The interest of this command in the context of versioning is that in addition to list all versions, it allows users to quickly find which version of an application is currently enabled.

This command has two modes of use, a default and a verbose mode.

The default mode will display the versions in a compact way, using the version identifier format used in the commands. This mode allows v2 users to keep using GlassFish as they are used to.

The verbose mode will be activated with the –-verbose option, it will add the enabled state of each version in addition with the information of the default mode.

- A user wants to list versions after some deployments :

```
$ asadmin deploy foo.war
$ asadmin deploy --name='foo;RC-1' foo_RC-1.war
$ asadmin deploy --enabled=false --name='foo;RC-2' foo_RC-2.war
$ asadmin list-applications
foo <web>
foo;RC-1 <web>
foo;RC-2 <web>
```

- After some time, the user might not remember which version is enabled, he can then get the version list again, but this time with the enabled state information :

```
$ asadmin list-applications --verbose=true
foo <web>         (disabed)
foo;RC-1 <web>    (enabled)
foo;RC-2 <web>    (disabled)
```

4 Versioned / unversioned

Users not using versioning as in GlassFish v2 are implicitly processing on the untagged version. Actually when people refer to an application “foo”, GlassFish perform the task over a version with an empty string as identifier.

The user can start to use versions and go back to the untagged one when he wants, he can also operate on the untagged version as he would with all others basic commands.

A user wants to use v2 unversioned commands :

```
$ asadmin deploy foo.war
$ asadmin enable foo
$ asadmin undeploy foo
```

A user starts to use versions and then go back to use the untagged version (without versioning) :

```
$ asadmin deploy –-name='foo;1' foo_1.war
$ asadmin deploy foo.war
```

A user starts to use GlassFish without versioning and then use the versioning :

```
$ asadmin deploy foo.war
$ asadmin deploy –-name='foo;1' foo_1.war
```

5 Samples scenarios

This part attends to present the life-cycle of an application from a user perspective through command sequences.

A user is developing a simple Java EE application named “foo” and uses GlassFish to deploy it. He deploys his application the first time as follows :

```
$ asadmin deploy foo.war
```

After some code updates, he compiles again the application and wants to re-deploy it this way :

```
$ asadmin deploy --force=true foo.war
```

At this point the user thinks he has reached the milestones he fixed, but he wants to go further in the development of its application by proposing new features. Consequently he plans new milestones and decides to version his application.

Once the development of the next targeted version finished, the user would deploy it like :

```
$ asadmin deploy --name='foo;BETA-1.0' foo_beta-1-0.war
```

Then the user has decided to plan a new “BETA” version containing a special feature not needed at this moment but potentially required later. Therefore once the version developed, he deploys it but doesn't enable it (he will switch later) :

```
$ asadmin deploy --name='foo;BETA-1.1' --enable=false foo_beta-1-1.war
```

The 'BETA-1.1' additional feature is needed now, and the user has to switch for this version :

```
$ asadmin enable 'foo;BETA-1.1'
```

The user has now reached an other version and he deploys it :

```
$ asadmin deploy --name='foo;RC-1.0' foo_rc-1-0.war
```

Before doing anything else the user want to refresh his memory about the deployed versions :

```
$ asadmin list-applications --verbose=true
foo <web>               (disabled)
foo;BETA-1.0 <web>      (disabled)
foo;BETA-1.1 <web>      (disabled)
foo;RC-1.0 <web>        (enabled)
```

At this moment the current enabled version is “RC-1.0”, and the user assumes that the “BETA” versions are no longer useful. Therefore he can undeploy all the “BETA” versions this way :

```
$ asadmin undeploy 'foo;BETA*'
```

A moment later, a major bug is found in version RC-1.0. He knows that the application before versioning does not include the bug, so he decided to return to the untagged version :

```
$ asadmin enable foo
```

A famous company released an open source Java EE application named “magicFOO” performing exactly the “foo” process in a more powerful way. Therefore the user decides to drop his application and use the new one instead :

```
$ asadmin undeploy 'foo;*'
$ asadmin deploy magicFOO.war
```

6 Rejected scenarios

The scenarios presented below show what the user cannot do in accordance with the definitions.

6.1 Bad use of commands

The identifier aware operations can't process if the expected argument is a version expression :

```
$ asadmin deploy --name='foo;*' foo.war

      ==> ERROR '*' wildcard not accepted
```

or

```
$ asadmin enable 'foo;*'

      ==> ERROR '*' wildcard not accepted
```

6.2 Syntax errors

The only parts concerned by syntax error are version identifier and so version expression :

• A version identifier with rejected characters : all special characters different than dot, underscore and hyphen :

```
$ asadmin deploy --name='foo;RC(1)' foo_1.war

      ==> ERROR '(' ')' not valid
```

or

```
$ asadmin deploy --name='foo;RC<1>' foo_1.war

      ==> ERROR '<>' not valid
```

• A version identifier with authorized characters not matching the regular expression of the definition :

```
$ asadmin deploy --name='foo;-._.-' foo_1.war
```

6.3 Bad scenarios

This part deals with the forbidden sequences causing a GlassFish error. The rejected commands cause no change in which version (if any) of an application is enabled.

6.3.1 Switch to a non-deployed version

It is not possible to enable a version not previously deployed :

```
$ asadmin deploy --name='foo;1' foo_1.war
$ asadmin deploy --enable=false --name='foo;3' foo_3.war
$ asadmin enable 'foo;2'

      ==> ERROR Application foo;2 not registered
```

6.3.2 Undeployment of a non-deployed version

Same case for undeploy command, it's not possible to undeployed a non-existing version :

```
$ asadmin deploy foo.war
$ asadmin undeploy 'foo;1'

      ==> ERROR : Application foo;ALPHA-1 not registered
```

6.3.3 Deployment of an existing version

If an application is deployed, the user must supply the 'force' argument to specify that he wants to re-deploy the application :

```
$ asadmin deploy --name='foo;1' foo_1.war
$ asadmin deploy --name='foo;1' foo_2.war

      ==> ERROR : Application with name foo;1 is already registered
```

6.3.4 In-place directory deployment

A directory can't be deployed twice with each time a different version identifier. The versioning for an in-place directory must be manage by the user. For each new version the user must create a new directory.

```
$ asadmin deploy --name='foo;1' /home/user/applications/foo/
$ asadmin deploy --name='foo;2' /home/user/applications/foo/

==> ERROR : '/home/user/applications/foo/' directory is already used by
'foo;1'
```