Talend ESB Container

Administration Guide

5.1

Talend ESB Container: Administration Guide

Publication date 3 May 2012
Copyright © 2011-2012 Talend Inc.

Copyleft

Notices

Table of Contents

List of Tables

Chapter 1. Introduction

Talend ESB provides an Apache Karaf-based ESB container preconfigured to support Apache Camel routing and Apache CXF-based services (both REST and SOAP-based). This container administration manual is intended to provide information on Karaf tooling commands and general administration. It is NOT a tutorial on getting started with Karaf - please see Talend ESB Getting Started User Guide for this.

> **Note**
>
> As the Talend ESB container is based on Apache Karaf, the two terms are used interchangeably within this document, except when explicitly stressed otherwise.

1.1. Structure of this manual

The following chapters are in this manual:

- Chapter 2, *Directory Structure* describes the directory layout of a Talend ESB installation.

- Chapter 3, *Starting/Stopping Talend Runtime* describes the various options that are available to start and stop Talend ESB.

- Chapter 4, *Starting/Stopping Talend ESB Infrastructure Components* describes how to start and stop the infrastructure components of Talend ESB.

- Chapter 5, *Using Console* gives a list of the console commands available.

- Chapter 6, *Deploying Multiple Karaf Containers* describes how to adapt the configuration to deploy multiple containers at the same machine.

- Chapter 7, *Remote Console* describes how to access Talend ESB using a remote console.

- Chapter 8, *Apache CXF and Camel, commands and tuning* describes commands provided by CXF and Camel.

- Chapter 9, *Security* describes how to configure security with Talend ESB.

- Chapter 11, *Logging System* describes configuring and using the logging system.

- Chapter 12, *Deployer* describes deploying bundles or re-defined groups of bundles (called features).

- Chapter 14, *Provisioning* describes how the provisioning system uses xml repositories that define sets of features (bundles). Thus when you install a feature and it will install all of its bundles defined under it.

- Chapter 15, *Web Applications* describes deploying WAR-based web applications into Talend Runtime.

- Chapter 16, *Monitoring and Administration using JMX* lists the MBeans available to monitor and administrate Karaf using any JMX client.

- Chapter 18, *Troubleshooting Talend ESB* describes how to fix problems related to JVM memory allocation parameters running Talend ESB.

Chapter 2. Directory Structure

The Talend ESB software may be downloaded from `http://www.talend.com/download.php`. The standard directory layout of a Talend ESB installation is as follows:

- /bin: startup scripts
- /etc: configuration files
  - /keystores: KeyStore files for JobServer SSL connection
- /data: working directory
  - /cache: OSGi framework bundle cache
  - /generated-bundles: temporary folder used by the deployer
  - /log: log files
- /deploy: hot deploy directory
- /instances: directory containing child instances
- /lib: contains the bootstrap libraries
  - /lib/ext: directory for JRE extensions
  - /lib/endorsed: directory for endorsed libraries
- /system: OSGi bundles repository, laid out as a Maven 2 repository

> **Note**
>
> The `data` folder contains all the working and temporary files for Karaf. If you want to restart from a clean state, you can wipe out this directory, which has the same effect as using the clean option [5].

Chapter 3. Starting/Stopping Talend Runtime

This chapter describes how to start and stop Talend Runtime and the various options that are available.

3.1. Starting Talend Runtime

Wait initally for commands to become available

After starting Talend Runtime, you need to wait a few seconds for initialization to complete before entering the commands. Karaf starts the non core bundles in the background. So even if the console is already available, the job commands may not.

3.1.1. On Windows

From a console window, change to the installation directory and run Talend Runtime. For the binary distribution, go to:

```
cd [karaf_install_dir]
```

where karaf_install_dir is the directory in which Karaf was installed, e.g., `${Talend-ESB-Version}\container`.

Then type:

```
bin\trun.bat
```

3.1.2. On Linux

From a console window, change to the installation directory and run Talend Runtime. For the binary distribution, do:

cd [karaf_install_dir]

where karaf_install_dir is the directory in which Karaf was installed, for example: `/usr/local/${Talend-ESB-Version}/container` .

Then type:

bin/trun

Warning

Do NOT close the console or shell in which Karaf was started, as that will terminate Karaf (unless Karaf was started using the **nohup** command).

3.1.3. Starting Talend Runtime without console

Karaf can be started without the console if you don't intend to use it (one can always connect using the remote ssh access) using the following command:

bin\trun.bat server (Windows)

bin\trun server (Linux).

3.1.4. Starting Talend Runtime in the background

Karaf can be easily started as a background process using the following command:

bin\start.bat (Windows)

bin\start (Linux).

Karaf can be reset to a clean state by simply deleting the [karaf_install_dir]/data folder. Alternatively, add `clean` to the above **start** command.

3.2. Stopping Talend Runtime

Within the Karaf console, you can perform a clean shutdown of Karaf by using the **osgi:shutdown** command, with an optional `-f` (force) flag to avoid a yes/no confirmation.

If you're running from the main console, exiting the shell using `logout` or `Ctrl+D` will also terminate the Karaf instance.

Further, from a command shell, you can run **bin\stop.bat** (Windows) or **bin/stop** (Linux).

For more advanced functionality, it's also possible to delay the shutdown using the time argument. The time argument can have different formats. First, it can be an absolute time in the format `hh:mm`, in which hh is the hour (1 or 2 digits) and mm is the minute of the hour (in two digits). Second, it can be in the format +m, in which m is the number of minutes to wait. The argument **now** is an alias for +0.

Examples: The following command will shutdown Karaf at 10:35am:

`osgi:shutdown 10:35`

The following command will shutdown Karaf in 10 minutes:

`osgi:shutdown +10`

Chapter 4. Starting/Stopping Talend ESB Infrastructure Components

This chapter describes how to start and stop the infrastructure components of Talend ESB.

4.1. Starting the Service Locator

After starting the Talend Runtime container, enter the following command at the console prompt:

```
tesb:start-locator
```

Standalone version of the Service Locator

Alternatively there is a standalone version of the Service Locator described in the **Talend ESB Getting Started User Guide**.

4.2. Stopping the Service Locator

Within the Talend Runtime container you can shutdown the Service Locator by typing:

```
tesb:stop-locator
```

4.3. Starting Service Activity Monitoring

After starting the Talend Runtime container, enter the following command at the console prompt:

```
tesb:start-sam
```

4.4. Stopping Service Activity Monitoring

At the console, you can shutdown Service Activity Monitoring by typing:

```
tesb:stop-sam
```

4.5. Starting the Security Token Service

After starting Talend Runtime container, enter the following command at the console prompt:

```
tesb:start-sts
```

4.6. Stopping the Security Token Service

At the console, you can shutdown the Security Token Service by typing:

```
tesb:stop-sts
```

4.7. Starting all three Talend ESB infrastructure services

After starting Talend Runtime container, enter the following command at the console prompt:

```
tesb:start-all
```

Port Conflicts

If there is another Talend Runtime container running at the same machine and you forgot to adapt the configuration of the used ports, a port conflict may arise while starting an infrastructure component. In this case **tesb:start-all** will not try to start the remaining infrastructure components. Stop all infrastructure services using **tesb:stop-all** and restart them after adapting the configuration. There are configuration scripts simplifying this task.

4.8. Stopping all three Talend ESB infrastructure services

At the console you can shutdown all three Talend ESB infrastructure services by typing:

```
tesb:stop-all
```

Chapter 5. Using Console

5.1. Viewing commands from the Karaf console

To see a list of the available commands in the console press the **tab** at the prompt. The **tab** key toggles completion anywhere on the line, so if you want to see the commands in the osgi group, type the first letters and hit **tab**. Depending on the commands, completion may be available on options and arguments too.

To view help on a particular command, type the command followed by --help or use the help command followed by the name of the command:

karaf@trun> features:list --help


```
DESCRIPTION
        features:list

        Lists all existing features available from the defined repositories.

SYNTAX
        features:list [options]

OPTIONS
        --help
                        Display this help message
        -i, --installed
                        Display a list of all installed features only
```

Help for each command is also available online at the Apache Karaf website.

5.2. Karaf Console command summary

The following tables summarize the out-of-the-box commands available with Karaf, grouped by scope (family of commands). View the command-line or online help as discussed in the previous section for information about the various arguments and options available with each command.

5.2.1. Admin Scope

The commands within the `admin:` scope involve administration tasks, including starting and stopping Karaf instances and connecting to them remotely.

Table 5.1. admin Scope commands

Command	Parameters	Description
change-opts	`[options] name javaOpts`	Changes the java options of an existing container instance.
change-rmi-registry-port	`[options] name port`	Changes the RMI registry port (used by management layer) of an existing container instance.
change-rmi-server-port	`[options] name port`	Changes the RMI server port (used by management layer) of an existing container instance.
change-ssh-port	`[options] name port`	Changes the secure shell port of an existing container instance.
connect	`[options] name [command]`	Connects to an existing container instance.
create	`[options] name`	Creates a new container instance.
destroy	`[options] name`	Destroys an existing container instance.
list	`[options]`	Lists all existing container instances.
rename	`[options] name new-name`	Renames an existing container instance.
start	`[options] name`	Starts an existing container instance.
stop	`[options] name`	Stops an existing container instance.

5.2.2. Config Scope

The commands within the `config:` scope are used to modify configuration data of Karaf instances (data stored in the `/etc` folder of the Karaf instance.) Updates take effect immediately without need of restarting Karaf. Note there is also a ConfigMBean that can be used to do this configuration within JMX.

Table 5.2. config Scope commands

Command	Parameters	Description
cancel	`[options]`	Cancels the changes to the configuration being edited.
delete	`[options] pid`	Delete a configuration.
edit	`[options] pid`	Creates or edits a configuration.

Command	Parameters	Description
list	`[options] [query]`	Lists existing configurations.
propappend	`[options] [query]`	Appends the given value to an existing property or creates the property with the specified name and value.
propdel	`[options] property`	Deletes a property from the edited configuration.
proplist	`[options]`	Lists properties from the currently edited configuration.
propset	`[options] property value`	Sets a property in the currently edited configuration.
update	`[options]`	Saves and propagates changes from the configuration being edited.

5.2.3. Dev Scope

The commands within the `dev:` scope are used to providing logging and other system output help to the administrator.

Table 5.3. dev Scope commands

Command	Parameters	Description
create-dump	`[options] [name]`	Creates zip archive with diagnostic info.
dynamic-import	`[options] id`	Enables/disables dynamic-import for a given bundle.
framework	`[options] [framework]`	OSGi Framework options.
print-stack-traces	`[options] [print]`	Prints the full stack trace in the console when the execution of a command throws an exception.
restart	`[options]`	Restart Karaf.
show-tree	`[options] id`	Shows the tree of bundles based on the wiring information.
watch	`[options] [urls]`	Watches and updates bundles.

5.2.4. Features Scope

The commands within the `features:` scope are used to provide support for Karaf features, which are predefined collections of bundles used to implement specific services.

Table 5.4. features Scope commands

Command	Parameters	Description
addUrl	`[options] urls`	Adds a list of repository URLs to the features service.
info	`[options] name [version]`	Shows information about selected information.
install	`[options] feature`	Installs a feature with the specified name and version.

Command	Parameters	Description
list	`[options]`	Lists all existing features available from the defined repositories.
listRepositories	`[options]`	Displays a list of all defined repositories.
listUrl	`[options]`	Displays a list of all defined repository URLs.
listVersions	`[options] feature`	Lists all versions of a feature available from the currently available repositories.
refreshUrl	`[options] urls`	Reloads the list of available features from the repositories.
removeRepository	`[options] repository`	Removes the specified repository features service.
removeUrl	`[options] urls`	Removes the given list of repository URLs from the features service.
uninstall	`[options] features`	Uninstalls a feature with the specified name and version.

5.2.5. JAAS Scope

The commands within the `jaas:` scope are used for management of JAAS users and realms.

Table 5.5. jaas Scope commands

Command	Parameters	Description
cancel	`[options]`	Cancel the modification of a JAAS realm.
realms	`[options]`	Lists the modification on the active realm/module.
manage	`[options] realm`	Manage user and roles of a Jaas Realm.
pending	`[options]`	Lists the modifications on the active realm/module.
roleadd	`[options] username role`	Add a role to a user.
roledel	`[options] username role`	Delete a role from a user.
update	`[options]`	Update JAAS realm.
useradd	`[options] username password`	Add a user.
userdel	`[options] username`	Delete a user.
users	`[options]`	Lists the users of the active realm/module.

5.2.6. Log Scope

The commands within the `log:` scope are used for management of system logs.

Table 5.6. log Scope commands

Command	Parameters	Description
clear	`[options]`	Clear log entries.
display	`[options]`	Displays log entries.

Command	Parameters	Description
display-exception	`[options]`	Displays the last occurred exception from the log.
get	`[options] [logger]`	Shows the currently set log level.
set	`[options] level [logger]`	Sets the log level.
tail	`[options]`	Continuously display log entries.

5.2.7. OBR Scope

The commands within the `obr:` scope are used to link to OSGi bundle repositories (OBRs) and install bundles defined within them.

Table 5.7. OBR Scope commands

Command	Parameters	Description
addUrl	`[options] urls`	Adds a list of repository URLs to the OBR service.
deploy	`[options] bundles`	Deploys a list of bundles using OBR service.
find	`[options] requirements`	Find OBR bundles for a given filter.
info	`[options] bundles`	Prints information about OBR bundles.
list	`[options] [packages]`	Lists OBR bundles, optionally providing the given packages.
listUrl	`[options]`	Displays the repository URLs currently associated with the OBR service.
refreshUrl	`[options] [urls]`	Reloads the repositories to obtain a fresh list of bundles.
removeUrl	`[options] [urls]`	Removes a list of repository URLs from the OBR service.
resolve	`[options] requirements`	Shows the resolution output for a given set of requirements.
source	`[options] folder bundles`	Downloads the sources for an OBR bundle.
start	`[options] bundles`	Deploys and starts a list of bundles using OBR.

5.2.8. OSGi Scope

The commands within the `osgi:` scope handle bundle maintenance (installation, starting and stopping, etc.) as well as some container-level actions.

Table 5.8. OSGi Scope commands

Command	Parameters	Description
bundle-level	`[options] id [startLevel]`	Gets or sets the start level of a given bundle.
headers	`[options] [ids]`	Displays OSGi headers of a given bundle.
info	`[options] [ids]`	Displays detailed information of a given bundle.
install	`[options] [urls]`	Installs one or more bundles.
list	`[options]`	Lists all installed bundles.

Command	Parameters	Description
ls	`[options] [ids]`	Lists OSGi services.
refresh	`[options] [ids]`	Refresh a bundle.
resolve	`[options] [ids]`	Resolve bundle(s).
restart	`[options] [ids]`	Stops and restarts bundle(s).
shutdown	`[options] [time]`	Shuts the framework down.
start	`[options] ids`	Starts bundle(s).
start-level	`[options] [level]`	Gets or sets the system start level.
stop	`[options] ids`	Stop bundle(s).
uninstall	`[options] ids`	Uninstall bundle(s).
update	`[options] id [location]`	Update bundle.

5.2.9. Shell Scope

The commands within the `shell:` scope are used to provide terminal window commands in the OSGi shell.

Table 5.9. Shell Scope commands

Command	Parameters	Description
cat	`[options] paths or urls`	Displays the content of a file or URL.
clear	`[options]`	Clears the console buffer.
each	`[options] values function`	Execute a closure on a list of arguments.
echo	`[options] [arguments]`	Echoes or prints arguments to STDOUT.
exec	`[options] command`	Executes system processes.
grep	`[options] pattern`	Prints lines matching the given pattern.
head	`[options] [paths or urls]`	Displays the first lines of a file.
history	`[options]`	Prints command history.
if	`[options] condition ifTrue [ifFalse]`	If/Then/Else block.
info	`[options]`	Prints system information.
java	`[options] className [arguments]`	Executes a Java standard application.
logout	`[options]`	Disconnects shell from current session.
more	`[options]`	File pager.
new	`[options] class [args]`	Creates a new java object.
printf	`[options] format [args]`	Formats and prints arguments.
sleep	`[options] duration`	Sleeps for a bit then wakes up.
sort	`[options] [files]`	Writes sorted concatenation of all files to standard output.
source	`[options] script [args]`	Run a script
tac	`[options]`	Captures the STDIN and returns it as a string. Optionally writes the content to a file.
tail	`[path or url]`	Displays the last lines of a file.

5.2.10. Miscellaneous Scopes

There are a few scopes that offer just one or two commands each. They're listed in the below table.

Table 5.10. Miscellaneous Scope commands

Command	Parameters	Description
packages:exports	`[options] [ids]`	Displays exported packages.
packages:imports	`[options] [ids]`	Displays imported packages.
ssh:ssh	`[options] hostname [command]`	Connects to a remote SSH server.
ssh:sshd	`[options]`	Creates a SSH server
web:list	`[options]`	Lists details for war bundles.
wrapper:install	`[options]`	Install the container as a system service in the OS.

Chapter 6. Deploying Multiple Karaf Containers

This chapter describes how to adapt the configuration to deploy multiple containers at the same machine.

6.1. Deploying multiple containers using configuration adaption scripts

In order to avoid conflicts between multiple container instances, there are Karaf configuration adaption scripts which automate the adjustment of potentially conflicting parameters. They are based on the **edit** and **propset** commands described above.

First container state

Please make sure the first container is stopped (using the shutdown command or CTRL-D), and that its `data` directory has been deleted, before copying its directory and files to create the second container.

Otherwise the data in the first container (which contains absolute paths relating to the first container) will cause problems in the second one.

Also make sure this default container is stopped when using the Karaf configuration adaptation scripts below.

Start the second container with default settings, and then run the Karaf configuration adaption script at the Karaf prompt to update and save the new settings. All necessary parameters adjustments are done using a single script call. Changes performed by the Karaf configuration adaption scripts are persistent and reflected in the configuration files in `container/etc`.

In particular, after starting the container from the container directory, in the Karaf console, use:

source scripts/configureC1.sh for the first container copy

source scripts/configureC2.sh for the second container copy

source scripts/configureC3.sh for the third container copy

source scripts/configureC0.sh resets the parameters to the default values

The ports which are configured using the scripts are described in the table below:

Table 6.1. Ports configured by the configuration scripts

Parameter	configureC0.sh	configureC1.sh	configureC2.sh	configureC3.sh
HTTP Port	8040	8041	8042	8043
HTTPS Port	9001	9002	9003	9004
RMI Registry Port	1099	1100	1101	1102
RMI Server Port	44444	44445	44446	44447
SSH Port	8101	8102	8103	8104
Command Port	8000	8010	8020	8030
File Transfer Port	8001	8011	8021	8031
Monitoring Port	8888	8898	8908	8918

Restart Container

To make sure the new parameters are used, it is recommended to shut down and restart the container after applying a configuration adaption script. Most of the parameter changes will be adapted "on the fly", but for the Jobserver parameters in Talend Enterprise ESB this is not yet possible.

Troubleshooting

If you get the "Port already in use exception" when starting alternate-container, recheck that there is not already a container running using the default parameters.

If you are still getting the error, it may also be that the port is actually in use by an unrelated process, so change the ports in the Karaf configuration adaption scripts and rerun these to apply the changes.

Chapter 7. Remote Console

7.1. Configuring remote instances

It does not always make sense to manage an instance a Talend ESB instance using the local console. Talend ESB can be remotely managed using a remote console.

When you start Karaf, it enables a remote console that can be accessed over SSH from any other Karaf console or plain SSH client. The remote console provides all the features of the local console and gives a remote user complete control over the container and services running inside of it.

The SSH hostname and port number is configured in the `[karaf_install_dir]/etc/org.apache.karaf.shell.cfg` configuration file with the following defaults values:

```
sshPort=8101
sshHost=0.0.0.0
sshRealm=karaf
hostKey=${karaf.base}/etc/host.key
```

You can change this configuration using the config commands or by editing the above file, but you'll need to restart the ssh console in order for it to use the new parameters.

```
# define helper functions
bundle-by-sn = { bm = new java.util.HashMap ; //
      each (bundles) { $bm put ($it symbolicName) $it } ; $bm get $1 }
bundle-id-by-sn = { b = (bundle-by-sn $1) ;  //
      if { $b } { $b bundleId } { -1 } }
# edit config
config:edit org.apache.karaf.shell
```

```
config:propset sshPort 8102
config:update
# force a restart
osgi:restart --force (bundle-id-by-sn org.apache.karaf.shell.ssh)
```

7.2. Connecting and disconnecting remotely

7.2.1. Using the ssh:ssh command

You can connect to a remote Karaf's console using the ssh:ssh command.

karaf@trun> ssh:ssh -l tadmin -P tadmin -p 8101 hostname

Warning

The default password is tadmin but we recommend changing it. See Chapter 9, *Security* for more information.

To confirm that you have connected to the correct Karaf instance, type **shell:info** at the karaf> prompt. Information about the currently connected instance is returned, as shown.

```
Karaf
  Karaf home                /local/apache-karaf-2.0.0
  Karaf base                /local/apache-karaf-2.0.0
  OSGi Framework            org.eclipse.osgi - 3.5.1.R35x_v20090827
JVM
  Java Virtual Machine      Java HotSpot(TM) Server VM version 14.1-b02
  ...
```

7.2.2. Using the Karaf client

The Karaf client allows you to securely connect to a remote Karaf instance without having to launch a Karaf instance locally.

For example, to quickly connect to a Karaf instance running in server mode on the same machine, run the following command from the karaf-install-dir directory: **bin/client**. More usually, you would provide a hostname, port, username and password to connect to a remote instance. And, if you were using the client within a larger script, you could append console commands as follows:

bin/client -a 8101 -h hostname -u tadmin -p tadmin features:install wrapper

To display the available options for the client, type:

```
> bin/client --help
Apache Karaf client
  -a [port]     specify the port to connect to
  -h [host]     specify the host to connect to
```

```
   -u [user]      specify the user name
   -p [password] specify the password
   --help         shows this help message
   -v             raise verbosity
   -r [attempts] retry connection establishment (up to attempts times)
   -d [delay]     intra-retry delay (defaults to 2 seconds)
   [commands]     commands to run
If no commands are specified, the client will be put in an interactive mode
```

7.2.3. Using a plain SSH client

You can also connect using a plain SSH client from your *nix system or Windows SSH client like Putty.

```
~$ ssh -p 8101 tadmin@localhost
tadmin@localhost's password:
```

7.2.4. Disconnecting from a remote console

To disconnect from a remote console, press **Ctrl**+**D**, **shell:logout** or simply **logout** at the Karaf prompt.

7.3. Stopping a remote instance

7.3.1. Using the remote console

If you have connected to a remote console using **ssh:ssh** or the Karaf client, you can stop the remote instance using **osgi:shutdown**.

> **Note**
>
> Pressing Ctrl+D in a remote console simply closes the remote connection and returns you to the local shell without shutting off the remote instance.

7.3.2. Using the Karaf client

To stop a remote instance using the Karaf client, run the following from the `karaf-install-dir/lib` directory:

bin/client -u tadmin -p tadmin -a 8101 hostname osgi:shutdown

Chapter 8. Apache CXF and Camel, commands and tuning

These are commands that are related to Apache CXF and Camel functionality.

Note

CXF and Camel are already preconfigured in the Talend OSGi container, but you need to install Camel / CXF features in native Karaf to use these commands.

To view help on a particular command, type the command followed by --help or use the help command followed by the name of the command:

```
karaf@trun> cxf:list-endpoints --help
DESCRIPTION
        cxf:list-endpoints

        Lists all CXF Endpoints on a Bus.

SYNTAX
        cxf:list-endpoints [options] [bus]

ARGUMENTS
        bus     The CXF bus name where to look for the Endpoints

OPTIONS
        --help  Display this help message
```

8.1. Commands supplied by CXF

These commands related to CXF functionality. For more information on CXF, see `http://cxf.apache.org/`.

Table 8.1. CXF commands

Command	Parameters	Description
cxf:list-busses	`[options]`	Lists all CXF Busses
cxf:list-endpoints	`[options] [bus]`	bus is the optional CXF bus name where to look for the Endpoints
cxf:stop-endpoint busid endpointName	`[options] bus endpoint`	stops a CXF Endpoint on a Bus; **bus** is CXF bus name where to look for the Endpoint. **endpoint** is the Endpoint name to stop.
cxf:start-endpoint busid endpointName	`[options] bus endpoint`	starts a CXF Endpoint on a Bus; **bus** is CXF bus name where to look for the Endpoint. **endpoint** is the Endpoint name to start.

8.2. Commands supplied by Camel

These commands related to Camel functionality. Help for these commands is available at `http://camel.apache.org/karaf.html`. These are for version Camel 2.8.x, which is the current version used by Talend ESB.

> **Tip**
>
> Use TAB key for completion on the name parameters below.

Table 8.2. Camel commands

Command	Parameters	Description
camel:list-contexts	`[options]`	Lists the Camel contexts available in the current Karaf instance
camel:list-routes	`[options]`	Displays the list of Camel routes available in the current Karaf instance
camel:info-context	`[options] name`	Displays detail information about a given Camel context
camel:start-context	`[options] name`	Starts the given Camel context
camel:stop-context	`[options] name`	Stops the given Camel context
camel:info-route	`[options] name`	Provides detail information about a Camel route
camel:show-route	`[options] name`	Renders the route in XML
camel:start-route	`[options] name`	Starts the given route
camel:stop-route	`[options] name`	Stops the given route

8.3. Configuring CXF work queues

CXF work queues are used for queuing incoming work requests, using a thread pool.

The `etc/org.apache.cxf.workqueues.cfg` configuration file is used for workqueue configuration (without this file, the configuration would need to be done for each individual bundle using CXF services). This process can significantly optimize the performance of HTTP / CXF Service request handling in the Talend OSGi container.

This mechanism allows you to configure global work queues for use by all bundles that are created, and normally services would share a thread pool. However, an individual service can override this via local configuration, if they have specific requirements.

8.3.1. Configuration parameters

A configuration file contains the following parameters:

Name	Description	Default value
org.apache.cxf.workqueue.names	One or more names of work queues, separated by commas	'default'
org.apache.cxf.workqueue.default. highWaterMark	Maximum number of threads to work on the queue	10
org.apache.cxf.workqueue.default. lowWaterMark	Minimum number of threads to work on the queue	5
org.apache.cxf.workqueue.default. initialSize	Initial number of threads to work on the queue	7
org.apache.cxf.workqueue.default. dequeueTimeout	this is the keep alive time for the threadpool executor. This allows excess threads to be terminated when idle for longer than this time, and they can be created again later if needed.	100 (optional)
org.apache.cxf.workqueue.default. queueSize	Maximum number of entries in the queue	100 (optional)

8.3.2. Configuration files

In the Talend Runtime container, the default configuration file is in `etc/org.apache.cxf.workqueues.cfg`.

Note this file is normally not used or edited directly; it gives work queue default values that each CXF-using bundle can choose to employ when it creates its work queues.

To configure a work queue, you need to create a specific file: `org.apache.cxf.workqueues-n.cfg`, where best practice would be for "n" to be the same as the work queue name; for example, `org.apache.cxf.workqueues-http-conduit.cfg` would configure the `http-conduit` work queue.

These is a list of pre-defined work queue names:

Name	Description
default	this means using the default values.

Name	Description
http-conduit	On the client side when using the asynchronous methods, the HTTP conduit must wait for and process the response on a background thread. This can control the queue that is used specifically for that purpose, to limit or expand the number of outstanding asynchronous requests.
jms-continuation	This is used by the JMS transport to handle continuations.
local-transport	The local transport being based on PipedInput/OutputStreams requires the use of separate threads; this workqueue can be used to configure the queue used exclusively for the local-transport.
ws-addressing	For decoupled cases, the ws-addressing layer may need to process the request on a background thread. This can control the work queue it uses.

Note that you can also update these variables using the standard Karaf configuration commands.

Chapter 9. Security

9.1. Managing users and passwords

The default security configuration uses a property file located at `karaf-install-dir/etc/users.properties` to store authorized users and their passwords. The default user name is `tadmin` and the associated password is `tadmin` too. We strongly encourage you to change the default password by editing the above file before moving Karaf into production.

The users are currently used in three different places in Karaf:

- access to the SSH console

- access to the JMX management layer

- access to the Web console

Those three ways all delegate to the same JAAS based security authentication.

The `users.properties` file contains one or more lines, each line defining a user, its password and the associated roles: **user=password[,role][,role]...**

9.2. Managing roles

JAAS roles can be used by various components. The three management layers (SSH, JMX and WebConsole) all use a global role based authorization system. The default role name is configured in the `etc/system.properties` using the `karaf.admin.role` system property and the default value is `admin`. All users authenticating for the management layer must have this role defined. The syntax for this value is the following:

[classname:]principal

where `classname` is the class name of the principal object (defaults to `org.apache.karaf.jaas.modules.RolePrincipal`) and principal is the name of the principal of that class (defaults to admin). Note that roles can be changed for a given layer using ConfigAdmin in the following configurations:

Layer	PID	Value
SSH	org.apache.karaf.shell	sshRole
JMX	org.apache.karaf.management	jmxRole
Web	org.apache.karaf.webconsole	role

9.3. Enabling password encryption

In order to not keep the passwords in plain text, the passwords can be stored encrypted in the configuration file. This can be easily enabled using the following commands:

```
# edit config
config:edit org.apache.karaf.jaas
config:propset encryption.enabled true
config:update
# force a restart
dev:restart
```

The passwords will be encrypted automatically in the etc/users.properties configuration file the first time the user logs in. Encrypted passwords are prepended with {CRYPT} so that are easy to recognize.

9.4. Managing realms

Karaf supports JAAS with some enhancements to allow JAAS to work nicely in an OSGi environment. This framework also features an OSGi keystore manager with the ability to deploy new keystores or truststores at runtime.

9.4.1. Overview

The Security framework feature of Karaf allows runtime deployment of JAAS based configuration for use in various parts of the application. This includes the remote console login, which uses the karaf realm, but which is configured with a dummy login module by default. These realms can also be used by the NMR, JBI components or the JMX server to authenticate users logging in or sending messages into the bus.

In addition to JAAS realms, you can also deploy keystores and truststores to secure the remote shell console, setting up HTTPS connectors or using certificates for WS-Security.

A very simple XML schema for spring has been defined, allowing the deployment of a new realm or a new keystore very easily.

9.4.2. Schema

To override or deploy a new realm, you can use the following XSD which is supported by a Spring namespace handler and can thus be defined in a Spring xml configuration file.

You can find the schema at the following location: http://karaf.apache.org/xmlns/jaas/v1.1.0.

Here are two examples using this schema:

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:jaas="http://karaf.apache.org/xmlns/jaas/v1.0.0"
           xmlns:ext="http://aries.apache.org/blueprint/xmlns/blueprint-ext/v1.0.0">

    <!-- Bean to allow the $[karaf.base] property to be correctly resolved -->
    <ext:property-placeholder placeholder-prefix="$[" placeholder-suffix="]"/>

    <jaas:config name="myrealm">
        <jaas:module className="org.apache.karaf.jaas.modules.properties.PropertiesLogi
                    flags="required">
            users = $[karaf.base]/etc/users.properties
        </jaas:module>
    </jaas:config>

</blueprint>
```

```
<jaas:keystore xmlns:jaas="http://karaf.apache.org/xmlns/jaas/v1.1.0"
               name="ks"
               rank="1"
               path="classpath:privatestore.jks"
               keystorePassword="keyStorePassword"
               keyPasswords="myalias=myAliasPassword">
</jaas:keystore>
```

The `id` attribute is the blueprint id of the bean, but it will be used by default as the name of the realm if no `name` attribute is specified. Additional attributes on the `config` elements are a `rank`, which is an integer. When the LoginContext looks for a realm for authenticating a given user, the realms registered in the OSGi registry are matched against the required name. If more than one realm is found, the one with the highest rank will be used, thus allowing the override of some realms with new values. The last attribute is `publish` which can be set to false to not publish the realm in the OSGi registry, thereby disabling the use of this realm.

Each realm can contain one or more module definitions. Each module identifies a LoginModule and the `className` attribute must be set to the class name of the login module to use. Note that this login module must be available from the bundle classloader, so either it has to be defined in the bundle itself, or the needed package needs to be correctly imported. The `flags` attribute can take one of four values that are explained on the JAAS documentation. The content of the `module` element is parsed as a properties file and will be used to further configure the login module.

Deploying such a code will lead to a JaasRealm object in the OSGi registry, which will then be used when using the JAAS login module.

9.4.2.1. Configuration override and use of the rank attribute

The `rank` attribute on the config element is tied to the ranking of the underlying OSGi service. When the JAAS framework performs an authentication, it will use the realm name to find a matching JAAS configuration. If multiple configurations are used, the one with the highest `rank` attribute will be used. So if you want to override the default security configuration in Karaf (which is used by the ssh shell, web console and JMX layer), you need to deploy a JAAS configuration with the name `name="karaf"` and `rank="1"`.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:jaas="http://karaf.apache.org/xmlns/jaas/v1.1.0"
           xmlns:ext="http://aries.apache.org/blueprint/xmlns/blueprint-ext/v1.0.0">

    <!-- Bean to allow the $[karaf.base] property to be correctly resolved -->
    <ext:property-placeholder placeholder-prefix="$[" placeholder-suffix="]"/>

    <jaas:config name="karaf" rank="1">
        <jaas:module className="org.apache.karaf.jaas.modules.properties.PropertiesLogi
                    flags="required">
            users = $[karaf.base]/etc/users.properties
            ...
        </jaas:module>
    </jaas:config>

</blueprint>
```

9.4.3. Available realms

Karaf comes with several login modules that can be used to integrate into your environment.

9.4.3.1. PropertiesLoginModule

This login module is the one configured by default. It uses a properties text file to load the users, passwords and roles from. This file uses the properties file format. The format of the properties are as follows, each line defining a user, its password and the associated roles: **user=password[,role][,role]...**

```
<jaas:config name="karaf">
    <jaas:module className=
    "org.apache.karaf.jaas.modules.properties.PropertiesLoginModule"
                flags="required">
        users = $[karaf.base]/etc/users.properties
    </jaas:module>
</jaas:config>
```

9.4.3.2. OsgiConfigLoginModule

The OsgiConfigLoginModule uses the OSGi ConfigurationAdmin service to provide the users, passwords and roles. Instead of `users` for the PropertiesLoginModule, this configuration uses a `pid` value for the process ID of the configuration containing user definitions.

9.4.3.3. JDBCLoginModule

The JDBCLoginModule uses a database to load the users, passwords and roles from, provided a data source (normal or XA). The data source and the queries for password and role retrieval are configurable, with the use of the following parameters.

Name	Description
datasource	The datasource as on OSGi ldap filter or as JDNI name
query.password	The SQL query that retries the password of the user
query.role	The SQL query that retries the roles of the user

Passing a data source as an OSGi ldap filter

To use an OSGi ldap filter, the prefix osgi: needs to be provided. See the example below:

```
<jaas:config name="karaf">
    <jaas:module
        className="org.apache.karaf.jaas.modules.jdbc.JDBCLoginModule"
        flags="required">
     datasource = osgi:javax.sql.DataSource/    \\
        (osgi.jndi.service.name=jdbc/karafdb)
     query.password = SELECT PASSWORD FROM USERS WHERE USERNAME=?
     query.role = SELECT ROLE FROM ROLES WHERE USERNAME=?
    </jaas:module>
</jaas:config>
```

Passing a data source as a JNDI name

To use an JNDI name, the prefix jndi: needs to be provided. The example below assumes the use of Aries JNDI to expose services via JNDI.

```
<jaas:config name="karaf">
    <jaas:module
        className="org.apache.karaf.jaas.modules.jdbc.JDBCLoginModule"
                flags="required">
        datasource = jndi:aries:services/javax.sql.DataSource/    \\
            (osgi.jndi.service.name=jdbc/karafdb)
        query.password = SELECT PASSWORD FROM USERS WHERE USERNAME=?
        query.role = SELECT ROLE FROM ROLES WHERE USERNAME=?
    </jaas:module>
</jaas:config>
```

9.4.3.4. LDAPLoginModule

The LDAPLoginModule uses a LDAP to load the users and roles, bind the users on the LDAP to check passwords. The LDAPLoginModule supports the following parameters:

Name	Description
connection.url	The LDAP connection URL, e.g. ldap://hostname
connection.username	Admin username to connect to the LDAP. This parameter is optional, if it's not provided, the LDAP connection will be anonymous.
connection.password	Admin password to connect to the LDAP. Only used if the connection.username is specified.
user.base.dn	The LDAP base DN used to looking for user, e.g. ou=user,dc=apache,dc=org

Name	Description
user.filter	The LDAP filter used to looking for user, e.g. (uid=%u) where %u will be replaced by the username.
user.search.subtree	If "true", the user lookup will be recursive (SUBTREE). If "false", the user lookup will be performed only at the first level (ONELEVEL).
role.base.dn	The LDAP base DN used to looking for roles, e.g. ou=role,dc=apache,dc=org
role.filter	The LDAP filter used to looking for user's role, e.g. (member:=uid=%u)
role.name.attribute	The LDAP role attribute containing the role string used by Karaf, e.g. cn
role.search.subtree	If "true", the role lookup will be recursive (SUBTREE). If "false", the role lookup will be performed only at the first level (ONELEVEL).
authentication	Define the authentication backend used on the LDAP server. The default is simple.
initial.context.factory	Define the initial context factory used to connect to the LDAP server. The default is com.sun.jndi.ldap.LdapCtxFactory
ssl	If "true" or if the protocol on the connection.url is ldaps, an SSL connection will be used
ssl.provider	The provider name to use for SSL
ssl.protocol	The protocol name to use for SSL (SSL for example)
ssl.algorithm	The algorithm to use for the KeyManagerFactory and TrustManagerFactory (PKIX for example)
ssl.keystore	The key store name to use for SSL. The key store must be deployed using a jaas:keystore configuration.
ssl.keyalias	The key alias to use for SSL
ssl.truststore	The trust store name to use for SSL. The trust store must be deployed using a jaas:keystore configuration.

A example of LDAPLoginModule usage follows:

```
<jaas:config name="karaf">
  <jaas:module
     className="org.apache.karaf.jaas.modules.ldap.LDAPLoginModule"
     flags="required">
        connection.url = ldap://localhost:389
        user.base.dn = ou=user,dc=apache,dc=org
        user.filter = (cn=%u)
        user.search.subtree = true
        role.base.dn = ou=group,dc=apache,dc=org
        role.filter = (member:=uid=%u)
        role.name.attribute = cn
        role.search.subtree = true
        authentication = simple
  </jaas:module>
</jaas:config>
```

If you want to use an SSL connection, the following configuration can be used as an example:

```
<ext:property-placeholder />

<jaas:config name="karaf" rank="1">
    <jaas:module
      className="org.apache.karaf.jaas.modules.ldap.LDAPLoginModule"
      flags="required">
```

```
        connection.url = ldaps://localhost:10636
        user.base.dn = ou=users,ou=system
        user.filter = (uid=%u)
        user.search.subtree = true
        role.base.dn = ou=groups,ou=system
        role.filter = (uniqueMember=uid=%u)
        role.name.attribute = cn
        role.search.subtree = true
        authentication = simple
        ssl.protocol=SSL
        ssl.truststore=ks
        ssl.algorithm=PKIX
    </jaas:module>
</jaas:config>

<jaas:keystore name="ks"
      path="file:///${karaf.home}/etc/trusted.ks"
      keystorePassword="secret" />
```

9.5. Deploying security providers

Some applications require specific security providers to be available, such as BouncyCastle. The JVM imposes some restrictions about the use of such provider JAR files, namely, that they need to be signed and be available on the boot classpath. One way to deploy such providers is to put them in the JRE folder at `$JAVA_HOME/jre/lib/ext` and modify each provider's security policy configuration (`$JAVA_HOME/jre/lib/security/java.security`) in order to register them. While this approach works fine, it has a global effect and requires that all servers are configured accordingly.

However Talend ESB offers a simple way to configure additional security providers:

- put your provider jar in `[karaf-install-dir]/lib/ext`

- modify the `[karaf-install-dir]/etc/config.properties` configuration file to add the following property:

  ```
  org.apache.karaf.security.providers = xxx,yyy
  ```

  The value of this property is a comma separated list of the provider class names to register. For example:

  ```
  org.apache.karaf.security.providers = \\
          org.bouncycastle.jce.provider.BouncyCastleProvider
  ```

  In addition, you may want to provide access to the classes from those providers from the system bundle so that all bundles can access those. It can be done by modifying the org.osgi.framework.bootdelegation property in the same configuration file:

  ```
  org.osgi.framework.bootdelegation = ...,org.bouncycastle*
  ```

Chapter 10. HTTP Configuration

HTTP is a request and response protocol used to enable communications between clients and servers.

Basically, a client send a request to a server on a particular port. The server listening to that port is waiting for the client's request. And when it receives the request from the client, the server sends back the corresponding response.

To secure this communication, a SSL protocol can be used on top of HTTP to provide security. This will allow the data to be encrypted and safely transfered through a secure HTTP protocol: HTTPS.

10.1. Server HTTP Configuration

The Talend Runtime provides support for HTTP and HTTPS by default with the help of the pax web component. For more information, see the documentation of Pax Web on http://team.ops4j.org/wiki/display/paxweb/Documentation.

HTTP / HTTPS configuration for Talend Runtime is done in the following configuration file: `<Talend.runtime.dir>/container/etc/org.ops4j.pax.web.cfg`.

10.1.1. Basic configuration

To enable HTTP and HTTPS in Talend Runtime, configure the properties of the `org.ops4j.pax.web.cfg` file as follows:

Property	Default	Description
`org.osgi.service.http.port`	8080	This property specifies the port used for servlets and resources accessible via HTTP.

Property	Default	Description
		The default value for this property is 8080. You can specify a value of 0 (zero), if you wish to allow Pax Web to automatically determine a free port to use for HTTP access.
`org.osgi.service.http.port.secure`	8443	This property specifies the port used for servlets and resources accessible via HTTPS. The default value for this property is 8443. You can specify a value of 0 (zero), if you wish to allow Pax Web to automatically determine a free port to use for HTTPS access.
`org.osgi.service.http.enabled`	true	This property specifies if the HTTP is enabled or disabled. If the value is set to "true", the support for HTTP access is enabled. If the value is set to "false", the support for HTTP access is disabled. The default value is "true".
`org.osgi.service.http.secure.enabled`	false	This property specifies if the HTTPS is enabled. If the value is set to "true", the support for HTTPS access is enabled. If the value is set to "false", the support for HTTPS access is disabled. The default value is "false".

10.1.2. SSL configuration

To encrypt the communication and secure the identification of a server, you can use a HTTPS protocol. HTTPS is based on SSL, which supports the encryption of messages sent via HTTP.

To secure a communication, HTTPS uses key pairs containing one public key and one private key. Data is encrypted with one key and can only be decrypted with the other key of the key pair. This establishes trust and privacy in message transfers.

To authenticate the Talend Runtime, you need to configure its private key in the `org.ops4j.pax.web.cfg`.

Property	Default	Description
`org.ops4j.pax.web.ssl.keystore`		Path to the keystore file. See http://team.ops4j.org/wiki/display/paxweb/SSL+Configuration for details.
`org.ops4j.pax.web.ssl.keystore.type`	JKS	This property specifies the keystore type. By default, the value is JKS.
`org.ops4j.pax.web.ssl.password`		Password used for keystore integrity check.
`org.ops4j.pax.web.ssl.keypassword`		Password used for keystore.

10.1.3. Advanced configuration

For a complete list of all advanced configuration properties, see http://team.ops4j.org/wiki/display/paxweb/Configuration/.

10.1.4. Default configuration

The Talend Runtime is deployed with the following configuration of the `<Talend.runtime.dir>/container/etc/org.ops4j.pax.web.cfg` file.

```
org.osgi.service.http.port=8040
org.osgi.service.http.port.secure=9001
org.osgi.service.http.secure.enabled=true
org.ops4j.pax.web.ssl.keystore=./etc/keystore.jks
org.ops4j.pax.web.ssl.password=password
org.ops4j.pax.web.ssl.keypassword=password
```

> **Note**
>
> The Certificates deployed with the Talend Runtime by default must not be used for production, but only for demo purposes.

> **Note**
>
> And the key password corresponds to the password generated by the user when he/she generated the key.

10.2. Client HTTP Configuration

This section discusses configuring HTTPS using the OSGi Configuration Admin Service.

> **More information**
>
> For more background information, see CXF SSL configuration in http://cxf.apache.org/docs/client-http-transport-including-ssl-support.html, which has some sample configurations supported by CXF.

10.2.1. OSGi configuration files

If a Web service is deployed in Talend Runtime as an OSGi bundle, it is now possible to configure SSL via the OSGi Configuration Admin Service. This may be done using the following files (located at `<Talend.runtime.dir>/container/etc/`):

- a generic configuration `org.apache.cxf.http.conduits-common.cfg`, used by all HTTPS endpoints in the container,

- endpoint-specific files: `org.apache.cxf.http.conduits-<endpoint_name>.cfg`,

- there may also be additional endpoint-specific configuration files.

Note that instead of the tree-structured XML configurations, these configurations are flat property files. The properties are named after their XML equivalents and are described in detail in Section 10.2.2, "HTTP Conduit OSGi Configuration Parameters".

The non-OSGi style of SSL configuration is supported and if present will take precedence over the OSGi configuration.

10.2.2. HTTP Conduit OSGi Configuration Parameters

The configuration files described in this section are located in `<Talend.runtime.dir>/container/etc/org.apache.cxf.http.conduits-<endpoint_name>.cfg` in the Talend Runtime.

As an example of the syntax involved, here are the contents of the general `org.apache.cxf.http.conduits-common.cfg` configuration file:

```
url = https.*
tlsClientParameters.disableCNCheck = true
tlsClientParameters.trustManagers.keyStore.type = JKS
tlsClientParameters.trustManagers.keyStore.password = password
tlsClientParameters.trustManagers.keyStore.file = ./etc/keystore.jks
tlsClientParameters.keyManagers.keyStore.type = JKS
tlsClientParameters.keyManagers.keyStore.password = password
tlsClientParameters.keyManagers.keyStore.file = ./etc/keystore.jks
tlsClientParameters.cipherSuitesFilter.include = *_EXPORT_.*,.*_EXPORT\
1024_.*,.*_WITH_DES_.*,.*_WITH_AES_.*,.*_WITH_NULL_.*,.*_DH_anon_.*
```

10.2.2.1. The `url` parameter

The `url` parameter is one of the main parameters. In the configuration files, the `url` parameter defines the list of matching client endpoints for which the contained configuration parameters are applied. (The client endpoint address is retrieved using `HTTPConduit.getAddress()`).

`url` may be a full endpoint address or may be a regular expression containing wild cards - for example:

• `".*"` matches all endpoints,

• `https.*` matches all client addresses starting with "https".

All parameters contained in all matching configuration files are collected:

1. in the order defined by the `order` parameter (see the table in the Section 10.2.2.2, "The `order` parameter"),

2. then by an exact match,

3. then by a configuration with a matching conduit bean name.

If a parameter is defined in multiple matching configuration files, then the last parameter definition found is the one that is used.

10.2.2.2. The `order` parameter

This parameter defines the order in which the parameters in configuration files are applied. Each file has a unique value of `order`. For example:

`abc.cfg`:

```
order = 1
url = .*
client.ReceiveTimeout = 60000;

xyz.cfg:

order = 2
url = .*busy.*
client.ReceiveTimeout = 120000;
```

If the endpoint address contains "busy", then both config files match as applicable, according to the rules in Section 10.2.2.1, "The url parameter".

In this case, client.ReceiveTimeout will have the longer timeout value 120000 because the order parameters stipulate that xyz.cfg is applied after abc.cfg.

10.2.2.3. Configuration properties

In this table, we look at the complete list of possible properties:

Property	Default	Description
url		The endpoint URL - either defined as exact string or as regular expression pattern (see Section 10.2.2.1, "The url parameter")
order	50	Defines the order in which parameters are applied (see Section 10.2.2.2, "The order parameter").
name		If name is defined and is equal to the conduit bean name, HTTPConduit.getBeanName(), the parameter definitions have highest priority, overwriting and extending other matching configurations.
tlsClientParameters.secureSocketProtocol	TLS	Protocol Name. Most common examples are "SSL", "TLS" or "TLSv1".
tlsClientParameters.sslCacheTimeout	JDK default	Sets the SSL Session Cache timeout value for client sessions handled by CXF.
tlsClientParameters.jsseProvider		JSSE provider name.
tlsClientParameters.disableCNCheck	false	Indicates whether that the hostname given in the HTTPS URL will be checked against the service's Common Name (CN) given in its certificate during SOAP client requests - it fails if there is a mismatch. If set to true (not recommended for production use), such checks will be bypassed. That will allow you, for example, to use a URL such as localhost during development.
tlsClientParameters. useHttpsURLConnectionDefaultHostname-Verifier	false	This attribute specifies if HttpsURLConnection. getDefaultHostnameVerifier() should be used to create HTTPS connections. If 'true', the 'disableCNCheck' configuration parameter is ignored.
tlsClientParameters. useHttpsURLConnectionDefaultSsl-SocketFactory	false	Specifies if HttpsURLConnection. getDefaultSSLSocketFactory() should be used to create HTTPS connections.

Property	Default	Description
		If 'true', 'jsseProvider', 'secureSocketProtocol', 'trustManagers', 'keyManagers', 'secureRandom', 'cipherSuites' and 'cipherSuitesFilter' configuration parameters are ignored.
tlsClientParameters.certConstraints. SubjectDNConstraints.combinator		SubjectDN certificate constraints specification as combinator.
tlsClientParameters.certConstraints. SubjectDNConstraints.RegularExpression		SubjectDN certificate constraints specification as regular expression.
tlsClientParameters.certConstraints. IssuerDNConstraints.combinator		IssuerDN certificate constraints specification as combinator.
tlsClientParameters.certConstraints. IssuerDNConstraints.RegularExpression		IssuerDN certificate constraints specification as regular expression.
tlsClientParameters. secureRandomParameters.algorithm	JVM default	algorithm parameter of the SecureRandom specification.
tlsClientParameters. secureRandomParameters.provider	JVM default	provider parameter of the SecureRandom specification.
tlsClientParameters.cipherSuitesFilter.include		filters the supported CipherSuites, list of CipherSuites that will be supported and used if available.
tlsClientParameters.cipherSuitesFilter.exclude		filters the supported CipherSuites, list of CipherSuites that will be excluded.
tlsClientParameters.cipherSuites	default sslContext cipher suites	CipherSuites that will be supported.
tlsClientParameters. trustManagers.provider	JVM default	Provider of the trust manager.
tlsClientParameters. trustManagers.factoryAlgorithm	JVM default	factory algorithm of the trust manager.
tlsClientParameters. trustManagers.keyPassword	JVM default	Key password of the trust manager.
tlsClientParameters. trustManagers.keyStore.type	JVM default	Keystore type of the trust manager.
tlsClientParameters. trustManagers.keyStore.password	JVM default	Keystore password of the trust manager.
tlsClientParameters. trustManagers.keyStore.provider	JVM default	Keystore provider of the trust manager.
tlsClientParameters. trustManagers.keyStore.url	JVM default	Trust Managers URL to hold X509 certificates.
tlsClientParameters. trustManagers.keyStore.file	JVM default	Trust Managers file to hold X509 certificates.
tlsClientParameters. trustManagers.keyStore.resource	JVM default	Trust Managers resource to hold X509 certificates.
tlsClientParameters. keyManagers.provider	JVM default	Provider of the key manager.
tlsClientParameters. keyManagers.factoryAlgorithm	JVM default	factory algorithm of the key manager.

Property	Default	Description
tlsClientParameters. keyManagers.keyPassword	JVM default	Key password of the key manager.
tlsClientParameters. keyManagers.keyStore.type	JVM default	Keystore type of the key manager.
tlsClientParameters. keyManagers.keyStore.password	JVM default	Keystore password of the key manager.
tlsClientParameters. keyManagers.keyStore.provider	JVM default	Keystore provider of the key manager.
tlsClientParameters. keyManagers.keyStore.url	JVM default	Key managers URL to hold X509 certificates.
tlsClientParameters. keyManagers.keyStore.file	JVM default	Key managers file to hold X509 certificates.
tlsClientParameters. keyManagers.keyStore.resource	JVM default	Key managers resource to hold X509 certificates.
authorization.UserName		Specifies the UserName parameter for configuring the basic authentication method that the endpoint uses preemptively.
authorization.Password		Specifies the Password parameter for configuring the basic authentication method that the endpoint uses preemptively.
authorization.Authorization		Corresponds to the authentication specified in the SPNEGO/Kerberos login.conf.
authorization.AuthorizationType		Authorization type: "Basic", "Digest" or "Negotiation"
proxyAuthorization.UserName		Specifies the UserName parameter for configuring basic authentication against outgoing HTTP proxy servers.
proxyAuthorization.Password		Specifies the Password parameter for configuring basic authentication against outgoing HTTP proxy servers.
proxyAuthorization.Authorization		Proxy authorization type: "Basic", "Digest" or "Negotiation"
proxyAuthorization.AuthorizationType		Corresponds to the proxy authentication specified in the SPNEGO/Kerberos login.conf.
client.ConnectionTimeout	30000	Specifies the amount of time, in milliseconds, that the client will attempt to establish a connection before it times out. 0 specifies that the client will continue to attempt to open a connection indefinitely.
client.ReceiveTimeout	60000	Specifies the amount of time, in milliseconds, that the client will wait for a response before it times out. 0 specifies that the client will wait indefinitely.
client.AutoRedirect	false	Specifies if the client will automatically follow a server issued redirection. The default is false.
client.MaxRetransmits	-1	Specifies the maximum number of times a client will retransmit a request to satisfy a redirect.

Property	Default	Description
		The default of -1 specifies that unlimited retransmissions are allowed.
client.AllowChunking	true	Specifies whether the client will send requests using chunking. The default is true which specifies that the client will use chunking when sending requests. Chunking cannot be used if either • http-conf:basicAuthSupplier is configured to provide credentials preemptively or • AutoRedirect is set to true. In both cases the value of `AllowChunking` is ignored and chunking is disallowed. See note about chunking below.
client.ChunkingThreshold	4000	Specifies the threshold at which CXF will switch from non-chunking to chunking. By default, messages less than 4K are buffered and sent non-chunked. Once this threshold is reached, the message is chunked.
client.Connection	Keep-Alive	Specifies whether a particular connection is to be kept open or closed after each request/response dialog. There are two valid values: • Keep-Alive specifies that the client wants to keep its connection open after the initial request/response sequence. If the server honors it, the connection is kept open until the consumer closes it. • close specifies that the connection to the server is closed after each request/response sequence.
client.DecoupledEndpoint		Specifies the URL of a decoupled endpoint for the receipt of responses over a separate server-client connection. Warning: You must configure both the client and server to use WS-Addressing for the decoupled endpoint to work.
client.ProxyServer		Specifies the URL of the proxy server through which requests are routed.
client.ProxyServerPort		Specifies the port number of the proxy server through which requests are routed.
client.ProxyServerType	HTTP	Specifies the type of proxy server used to route requests. Valid values are: HTTP (default), SOCKS
client.NonProxyHosts		a (possibly empty) list of hosts which should be connected directly and not through the proxy server; it may contain wild card expressions.

10.2.3. Chunking

There are a number of parameters related to chunking, so we discuss this here in more detail. There are two ways of putting the body of a message into an HTTP stream:

1. Standard scheme: this is used by most browsers. It consists in specifying a Content-Length header in the HTTP headers. This allows the receiver to know how much data is coming and when to stop reading.

   The problem with this approach is that the length needs to be pre-determined. The data cannot be streamed as generated as the length needs to be calculated upfront.

   Thus, if chunking is turned off, we need to buffer the data in a byte buffer (or temp file if it is too large) so that the Content-Length can be calculated.

2. Chunked scheme: with this mode, the data is sent to the receiver in chunks. Each chunk is preceded by a hexadecimal chunk size. When a chunk size is 0, the receiver knows that all the data has been received.

   This mode allows better streaming, as we just need to buffer a small amount, up to 8K by default. When the buffer fills, the chunk is written out.

Some parameters in Section 10.2.2.3, "Configuration properties" allow you to specify the details of the chunking.

In general, the Chunked scheme will perform better as the streaming can take place directly. However, there are some problems with chunking:

- Many proxy servers don't understand it, especially older proxy servers. Many proxy servers want the Content-Length up front so they can allocate a buffer to store the request before passing it onto the real server.

- Some of the older Web Services stacks also have problems with Chunking - specifically, older versions of .NET.

Tip

If you are getting strange errors (generally not SOAP faults, but other HTTP-type errors) when trying to interact with a service, try turning off chunking to see if that helps.

Chapter 11. Logging System

Karaf provides a powerful logging system based on OPS4j Pax Logging. In addition to being a standard OSGi Log service, it supports the following APIs:

- Apache Commons Logging

- SLF4J

- Apache Log4j

- Java Util Logging

Karaf also comes with a set of console commands that can be used to display, view and change the log levels.

11.1. Configuration

11.1.1. Configuration file

The configuration of the logging system uses a standard Log4j configuration file at the following location: `[karaf_install_dir]/etc/org.ops4j.pax.logging.cfg`

You can edit this file at runtime and any change will be reloaded and be effective immediately.

11.1.2. Configuring the appenders

The default logging configuration defines three appenders:

- the `stdout` console appender is disabled by default. If you plan to run Karaf in server mode only (i.e. with the locale console disabled), you can turn on this appender on by adding it to the list of configured appenders using the `log4j.rootLogger` property

- the `out` appender is the one enabled by default. It logs events to a number of rotating log files of a fixed size. You can easily change the parameters to control the number of files using `maxBackupIndex` and their size `maxFileSize`.

- the `sift` appender can be used instead to provide a per-bundle log file. The default configuration uses the bundle symbolic name as the file name to log to

11.1.3. Changing the log levels

The default logging configuration sets the logging levels so that the log file will provide enough information to monitor the behavior of the runtime and provide clues about what caused a problem. However, the default configuration will not provide enough information to debug most problems. The most useful logger to change when trying to debug an issue with Karaf is the root logger. You will want to set its logging level to DEBUG in the `org.ops4j.pax.logging.cfg` file.

```
log4j.rootLogger=DEBUG, out, osgi:VmLogAppender
...
```

When debugging a problem in Karaf you may want to change the level of logging information that is displayed on the console. The example below shows how to set the root logger to DEBUG but limiting the information displayed on the console to >WARN.

```
log4j.rootLogger=DEBUG, out, stdout, osgi:VmLogAppender
log4j.appender.stdout.threshold=WARN
...
```

11.2. Console Log Commands

The log scope comes with several commands -- see Section 5.2.6, "Log Scope" for a full list.

For example, if you want to debug something, you might want to run the following commands:

```
<log:set DEBUG
... do something ...
< log:display
```

Note that the log levels set using the log:set commands are not persistent and will be lost upon restart. To configure those in a persistent way, you should edit the configuration file mentioned above using the config commands or directly using a text editor of your choice. The log commands have a separate configuration file: `[karaf_install_dir]/etc/org.apache.karaf.log.cfg`

11.3. Advanced Configuration

The logging backend uses Log4j, but offers a number of additional features.

11.3.1. Filters

Appender filters can be added using the following syntax:

```
log4j.appender.[appender-name].filter.[filter-name]=[filter-class]
log4j.appender.[appender-name].filter.[filter-name].[option]=[value]
```

Below is a real example:

```
log4j.appender.out.filter.f1=org.apache.log4j.varia.LevelRangeFilter
log4j.appender.out.filter.f1.LevelMax=FATAL
log4j.appender.out.filter.f1.LevelMin=DEBUG
```

11.3.2. Nested appenders

Nested appenders can be added using the following syntax:

```
log4j.appender.[appender-name].appenders=    //
     [comma-separated-list-of-appender-names]
```

Below is a real example:

```
log4j.appender.async=org.apache.log4j.AsyncAppender
log4j.appender.async.appenders=jms

log4j.appender.jms=org.apache.log4j.net.JMSAppender
...
```

11.3.3. Error handlers

Error handlers can be added using the following syntax:

```
log4j.appender.[appender-name].errorhandler=[error-handler-class]
log4j.appender.[appender-name].errorhandler.root-ref=[true|false]
log4j.appender.[appender-name].errorhandler.logger-ref=[logger-ref]
log4j.appender.[appender-name].errorhandler.appender-ref=[appender-ref]
```

11.3.4. OSGi specific MDC attributes

Pax-Logging provides the following attributes by default:

- `bundle.id`: the id of the bundle from which the class is loaded

- `bundle.name`: the symbolic-name of the bundle

- `bundle.version`: the version of the bundle

11.3.5. MDC sifting appender

An MDC sifting appender is available to split the log events based on MDC attributes. Below is a configuration example for this appender:

```
log4j.appender.sift=org.apache.log4j.sift.MDCSiftingAppender
log4j.appender.sift.key=bundle.name
log4j.appender.sift.default=karaf
log4j.appender.sift.appender=org.apache.log4j.FileAppender
log4j.appender.sift.appender.layout=org.apache.log4j.PatternLayout
log4j.appender.sift.appender.layout.ConversionPattern=          \\
      %d{ABSOLUTE} | %-5.5p | %-16.16t | %-32.32c{1} | %-32.32C %4L | %m%n
log4j.appender.sift.appender.file=${karaf.data}/log/$\\{bundle.name\\}.log
log4j.appender.sift.appender.append=true
```

11.3.6. Enhanced OSGi stack trace renderer

This renderer is configured by default in Karaf and will give additional informations when printing stack traces. For each line of the stack trace, it will display OSGi specific informations related to the class on that line: the bundle id, the bundle symbolic name and the bundle version. This information can greatly help diagnosing problems in some cases. The information is appended at the end of each line in the following format id:name:version as shown below:

```
java.lang.IllegalArgumentException: Command not found:  *:foo
    at org.apache.felix.gogo.runtime.shell.Closure.execute
       (Closure.java:225)[21:org.apache.karaf.shell.console:2.1.0]
    at org.apache.felix.gogo.runtime.shell.Closure.executeStatement
       (Closure.java:162)[21:org.apache.karaf.shell.console:2.1.0]
    at org.apache.felix.gogo.runtime.shell.Pipe.run
       (Pipe.java:101)[21:org.apache.karaf.shell.console:2.1.0]
    at org.apache.felix.gogo.runtime.shell.Closure.execute
       (Closure.java:79)[21:org.apache.karaf.shell.console:2.1.0]
    at org.apache.felix.gogo.runtime.shell.CommandSessionImpl.execute
       (CommandSessionImpl.java:71)[21:org.apache.karaf.shell.console:2.1.0]
    at org.apache.karaf.shell.console.jline.Console.run
       (Console.java:169)[21:org.apache.karaf.shell.console:2.1.0]
    at java.lang.Thread.run(Thread.java:637)[:1.6.0_20]
```

11.3.7. Using your own appenders

If you plan to use your own appenders, you need to create an OSGi bundle and attach it as a fragment to the bundle with a symbolic name of org.ops4j.pax.logging.pax-logging-service. This way, the underlying logging system will be able to see and use your appenders. So for example you write a log4j appender:

```
class MyAppender extends AppenderSkeleton {
...
}
```

Then you need to package the appender in a jar with a Manifest like this:

```
Manifest:
Bundle-SymbolicName: org.mydomain.myappender
Fragment-Host: org.ops4j.pax.logging.pax-logging-service
...
```

Now you can use the appender in your log4j config file like shown in the config examples above.

Chapter 12. Deployer

The Karaf deployer is used for deploying bundles or groups of bundles (called features) into the Karaf container. The following diagram describes the architecture of the deployer.
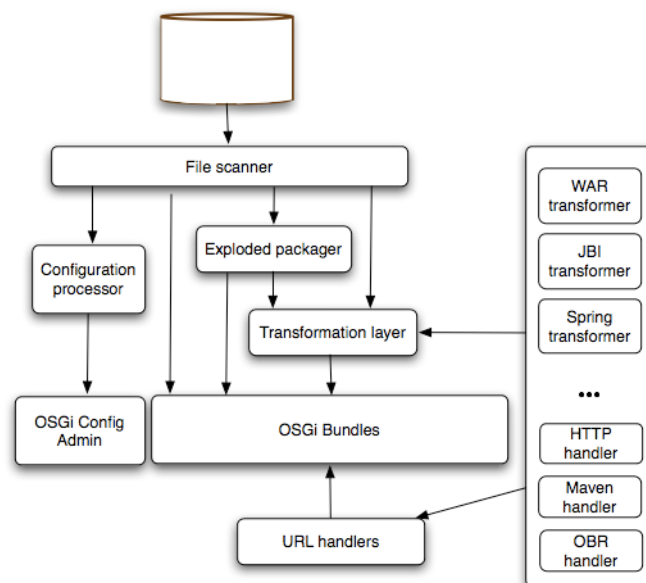
12.1. Features deployer

To be able to hot deploy features from the deploy folder, you can simply drop a Feature descriptor on that folder. A bundle will be created and its installation (automatic) will trigger the installation of all features contained in the descriptor. Removing the file from the deploy folder will uninstall the features. If you want to install a single feature, you can do so by writing a feature descriptor like the following:

```
<features>
    <repository>mvn:org.apache.servicemix.nmr/apache-servicemix-nmr/   \\
        1.0.0/xml/features</repository>
    <feature name="nmr-only">
        <feature>nmr</feature>
    </feature>
</features>
```

For more informations about features, see the Chapter 14, *Provisioning*.

12.2. Spring deployer

Karaf includes a deployer that is able to deploy plain blueprint or spring-dm configuration files. The deployer will transform on the fly any spring configuration file dropped into the deploy folder into a valid OSGi bundle. The generated OSGi manifest will contain the following headers:

```
Manifest-Version: 2
Bundle-SymbolicName: [name of the file]
Bundle-Version: [version of the file]
Spring-Context: *;publish-context:=false;create-asynchronously:=true
Import-Package: [required packages]
DynamicImport-Package: *
```

The name and version of the file are extracted using a heuristic that will match common patterns. For example `my-config-1.0.1.xml` will lead to name = my-config and version = 1.0.1. The default imported packages are extracted from the Spring file definition and includes all classes referenced directly. If you need to customize the generated manifest, you can do so by including an xml element in your Spring configuration:

```
<spring:beans ...>
  <manifest>
    Require-Bundle= my-bundle
  </manifest>
```

12.3. Wrap deployer

The wrap deployer allows you to hot deploy non-OSGi jar files ("classical" jar files) from the deploy folder. It's a standard deployer (you don't need to install additional Karaf features):

```
karaf@trun> la|grep -i wrap
[ 1] [Active ] [          ] [  5] OPS4J Pax Url - wrap: (1.2.5)
[32] [Active ] [Created ] [ 30] Apache Karaf ::  Deployer :: Wrap Non OSGi Jar
```

Karaf wrap deployer looks for jar files in the deploy folder. The jar files is considered as non-OSGi if the MANIFEST doesn't contain the Bundle-SymbolicName and Bundle-Version attributes, or if there is no MANIFEST at all. The non-OSGi jar file is transformed into an OSGi bundle. The deployer tries to populate the Bundle-SymbolicName and Bundle-Version extracted from the jar file path. For example, if you simply copy commons-lang-2.3.jar (which is not an OSGi bundle) into the deploy folder, you will see:

```
karaf@trun> la|grep -i commons-lang
[ 41] [Active ] [          ] [ 60] commons-lang (2.3)
```

If you take a look on the commons-lang headers, you can see that the bundle exports all packages with optional resolution and that Bundle-SymbolicName and Bundle-Version have been populated:

```
karaf@trun> osgi:headers 41
```

```
commons-lang (41)
-----------------
Specification-Title = Commons Lang
Tool = Bnd-0.0.357
Specification-Version = 2.3
Specification-Vendor = Apache Software Foundation
Implementation-Version = 2.3
Generated-By-Ops4j-Pax-From = wrap:file:/home/onofreje/workspace/karaf/
    assembly/target/apache-karaf-2.99.99-SNAPSHOT/deploy/commons-lang-2.3.jar$
    Bundle-SymbolicName=commons-lang&Bundle-Version=2.3
Implementation-Vendor-Id = org.apache
Created-By = 1.6.0_21 (Sun Microsystems Inc.)
Implementation-Title = Commons Lang
Manifest-Version = 1.0
Bnd-LastModified = 1297248243231
X-Compile-Target-JDK = 1.1
Originally-Created-By = 1.3.1_09-85 ("Apple Computer, Inc.")
Ant-Version = Apache Ant 1.6.5
Package = org.apache.commons.lang
X-Compile-Source-JDK = 1.3
Extension-Name = commons-lang
Implementation-Vendor = Apache Software Foundation

Bundle-Name = commons-lang
Bundle-SymbolicName = commons-lang
Bundle-Version = 2.3
Bundle-ManifestVersion = 2

Import-Package =
        org.apache.commons.lang;resolution:=optional,
        org.apache.commons.lang.builder;resolution:=optional,
        org.apache.commons.lang.enum;resolution:=optional,
        org.apache.commons.lang.enums;resolution:=optional,
        org.apache.commons.lang.exception;resolution:=optional,
        org.apache.commons.lang.math;resolution:=optional,
        org.apache.commons.lang.mutable;resolution:=optional,
        org.apache.commons.lang.text;resolution:=optional,
        org.apache.commons.lang.time;resolution:=optional
Export-Package =
        org.apache.commons.lang;uses:="org.apache.commons.lang.builder,
            org.apache.commons.lang.math,org.apache.commons.lang.exception",
        org.apache.commons.lang.builder;
            uses:="org.apache.commons.lang.math,org.apache.commons.lang",
        org.apache.commons.lang.enum;uses:=org.apache.commons.lang,
        org.apache.commons.lang.enums;uses:=org.apache.commons.lang,
        org.apache.commons.lang.exception;uses:=org.apache.commons.lang,
        org.apache.commons.lang.math;uses:=org.apache.commons.lang,
        org.apache.commons.lang.mutable;uses:="org.apache.commons.lang,
            org.apache.commons.lang.math",
        org.apache.commons.lang.text;uses:=org.apache.commons.lang,
        org.apache.commons.lang.time;uses:=org.apache.commons.lang
```

12.4. War deployer

See Chapter 15, *Web Applications* for information on web application (war) deployment.

Chapter 13. Servlet Context

A servlet context defines a set of methods which allows a servlet to communicate with its servlet container. Karaf and CXF provide servlet context custom configuration, for building services. For example, you can deploy into a servlet container, using a servlet transport, `CXFServlet`. The following section explains servlet context configuration in more detail.

Please read the Servlet Transport on the Apache CXF website for additional information about this servlet context.

13.1. Configuration

> **Note**
>
> The paths `[karaf_install_dir]/` and `[Talend-ESB-Version]/container` are equivalent.

First we look at the configuration files. The configuration of the servlet context for the OSGi HTTP Service is specified in a file at the following location: `[karaf_install_dir]/etc/org.apache.cxf.osgi.cfg`

The configuration of the port for the OSGi HTTP Service is specified in: `[karaf_install_dir]/etc/org.ops4j.pax.web.cfg`

You can edit these files at runtime and any change will be reloaded and be effective immediately.

13.2. Configuring the context

When editing the files, the `org.apache.cxf.osgi.cfg` file specified prefix for the services is:

```
org.apache.cxf.servlet.context=/services
```

The `org.ops4j.pax.web.cfg` file specified port for the services is:

```
org.osgi.service.http.port=8040
```

13.2.1. Relative endpoint address

The CXFServlet uses a relative address for the endpoint rather than a full http address. For example, given an implementation class called `GreeterImpl` with endpoints `greeter` and `greeterRest`, the relative endpoint addresses would be configured as:

```
<jaxws:endpoint id="greeter"
    implementor="org.apache.hello_soap_http.GreeterImpl"
    address="/Greeter1"/>

<jaxrs:server id="greeterRest"
    serviceClass="org.apache.hello_soap_http.GreeterImpl"
    address="/GreeterRest"/>
```

The cumulative result of these changes is that the endpoint address for the servlet will be: `http://{server}:8040/services/Greeter1` and `http://{server}:8040/services/GreeterRest`

Chapter 14. Provisioning

Karaf provides a simple, yet flexible, way to provision applications or "features". Such a mechanism is mainly provided by a set of commands available in the features shell. The provisioning system uses xml "repositories" that define a set of features.

14.1. Example: Deploying a sample feature

In the rest of this chapter, we describe in detail the process involved in provisioning. But as a quick demonstration, we'll run a sample Apache Camel feature already present in the Talend ESB distribution. In the console, run the following commands:

```
features:addUrl mvn:org.apache.camel/camel-example-osgi/2.5.0/xml/features
features:install camel-example-osgi
```

The example installed uses Camel to start a timer every 2 seconds and output a message on the console. These **features:addUrl** and **features:install** commands download the Camel features descriptor and install this example. The output is as follows:

```
>>>> SpringDSL set body:  Fri Jan 07 11:59:51 CET 2011
>>>> SpringDSL set body:  Fri Jan 07 11:59:53 CET 2011
>>>> SpringDSL set body:  Fri Jan 07 11:59:55 CET 2011
```

14.1.1. Stopping and uninstalling the sample application

To stop this demo, run the following command:

```
features:uninstall camel-example-osgi
```

14.2. Repositories

So, first we look at feature repositories. The complete xml schema for feature descriptor are available on Features XML Schema page. We recommend using this XML schema. It will allow Karaf to validate your repository before parsing. You may also verify your descriptor before adding it to Karaf by simply validation, even from IDE level. Here is an example of such a repository:

```
<features xmlns="http://karaf.apache.org/xmlns/features/v1.0.0">
    <feature name="spring" version="3.0.4.RELEASE">
        <bundle>mvn:org.apache.servicemix.bundles/   \\
            org.apache.servicemix.bundles.aopalliance/1.0_1</bundle>
        <bundle>mvn:org.springframework/spring-core/3.0.4.RELEASE</bundle>
        <bundle>mvn:org.springframework/spring-beans/3.0.4.RELEASE</bundle>
        <bundle>mvn:org.springframework/spring-aop/3.0.4.RELEASE</bundle>
        <bundle>mvn:org.springframework/spring-context/3.0.4.RELEASE</bundle>
        <bundle>mvn:org.springframework/spring-context-support/3.0.4.RELEASE
            </bundle>
    </feature>
</features>
```

A repository includes a list of feature elements, each one representing an application that can be installed. The feature is identified by its name which must be unique amongst all the repositories used and consists of a set of bundles that need to be installed along with some optional dependencies on other features and some optional configurations for the Configuration Admin OSGi service.

References to features define in other repositories are allow and can be achieved by adding a list of repository.

```
<features xmlns="http://karaf.apache.org/xmlns/features/v1.0.0">
  <repository>mvn:org.apache.servicemix.nmr/apache-servicemix-nmr/   \\
      1.3.0/xml/features</repository>
  <repository>mvn:org.apache.camel.karaf/apache-camel/2.5.0/xml/features
      </repository>
  <repository>mvn:org.apache.karaf/apache-karaf/2.1.2/xml/features
      </repository>
  ...
```

Be careful when you define them as there is a risk of 'cycling' dependencies. Note: By default, all the features defined in a repository are not installed at the launch of Apache Karaf (see Section 14.4, "Service configuration" for more information).

14.2.1. Bundles

The main information provided by a feature is the set of OSGi bundles that defines the application. Such bundles are URLs pointing to the actual bundle jars. For example, one would write the following definition:

```
<bundle>http://repo1.maven.org/maven2/org/apache/servicemix/nmr/   \\
    org.apache.servicemix.nmr.api/1.0.0-m2/   \\
    org.apache.servicemix.nmr.api-1.0.0-m2.jar</bundle>
```

Doing this will make sure the above bundle is installed while installing the feature. However, Karaf provides several URL handlers, in addition to the usual ones (file, http, etc...). One of these is the maven URL handler, which allow reusing maven repositories to point to the bundles.

14.2.1.1. Maven URL Handler

The equivalent of the above bundle would be:

```
<bundle>
    mvn:org.apache.servicemix.nmr/org.apache.servicemix.nmr.api/1.0.0-m2
</bundle>
```

In addition to being less verbose, the maven url handlers can also resolve snapshots and can use a local copy of the jar if one is available in your maven local repository.

The org.ops4j.pax.url.mvn bundle resolves mvn URLs. This flexible tool can be configured through the configuration service. For example, to find the current repositories type: **karaf@trun:/> config:list** and the following will display:

```
Pid:            org.ops4j.pax.url.mvn
BundleLocation: mvn:org.ops4j.pax.url/pax-url-mvn/0.3.3
Properties:
   service.pid = org.ops4j.pax.url.mvn
   org.ops4j.pax.url.mvn.defaultRepositories = file:/opt/development/  \\
       karaf/assembly/target/apache-felix-karaf-1.2.0-SNAPSHOT/    \\
       system@snapshots
   org.ops4j.pax.url.mvn.repositories = http://repo1.maven.org/maven2,
       http://svn.apache.org/repos/asf/servicemix/m2-repo
   below = list of repositories and even before the local repository
```

The repositories checked are controlled by these configuration properties. For example, org.ops4j.pax.url.mvn.repositories is a comma separated list of repository URLs specifying those remote repositories to be checked. So, to replace the defaults with a new repository at http://www.example.org/repo on the local machine:

```
karaf@trun:/> config:edit org.ops4j.pax.url.mvn
karaf@trun:/> config:proplist
   service.pid = org.ops4j.pax.url.mvn
   org.ops4j.pax.url.mvn.defaultRepositories = file:/opt/development/karaf/
       assembly/target/apache-felix-karaf-1.2.0-SNAPSHOT/system@snapshots
   org.ops4j.pax.url.mvn.repositories = http://repo1.maven.org/maven2,
       http://svn.apache.org/repos/asf/servicemix/m2-repo
   below = list of repositories and even before the local repository
karaf@trun:/> config:propset org.ops4j.pax.url.mvn.repositories
       http://www.example.org/repo
karaf@trun:/> config:update
```

By default, snapshots are disabled. To enable an URL for snapshots append @snapshots. For example: http://www.example.org/repo@snapshots. Repositories on the local are supported through file:/ URLs.

14.2.1.2. Bundle start-level

By default, the bundles deployed through the feature mechanism will have a start-level equals to the value defined in the configuration file config.properties with the variable karaf.startlevel.bundle=60. This value can be changed using the xml attribute start-level.

```
<feature name='my-project' version='1.0.0'>
<feature version='2.4.0'>camel-spring</feature>
   <bundle start-level='80'>mvn:com.mycompany.myproject/   \\
```

```
        myproject-dao</bundle>
    <bundle start-level='85'>mvn:com.mycompany.myproject/    \\
        myproject-service</bundle>
    <bundle start-level='85'>mvn:com.mycompany.myproject/    \\
        myproject-camel-routing</bundle>
</feature>
```

The advantage to define the start-level of a bundle is that you can deploy all your bundles including those of the project with the 'infrastructure' bundles required (e.g : camel, activemq) at the same time and you will have the guaranty when you use Spring Dynamic Module (to register service through OSGI service layer), Blueprint that by example Spring context will not be created without all the required services installed.

14.2.1.3. Bundle stop/start

The OSGI specification allows to install a bundle without starting it. To use this functionality, simply add the following attribute in your <bundle> definition

```
<feature name='my-project' version='1.0.0'>
    <feature version='2.4.0'>camel-spring</feature>
    <bundle start-level='80' start='false'>mvn:com.mycompany.myproject/    \\
        myproject-dao</bundle>
    <bundle start-level='85' start='false'>mvn:com.mycompany.myproject/    \\
        myproject-service</bundle>
    <bundle start-level='85' start='false'>mvn:com.mycompany.myproject/    \\
        myproject-camel-routing</bundle>
</feature>
```

14.2.1.4. Bundle dependency

A bundle can be flagged as being a dependency. Such information can be used by resolvers to compute the final list of bundles to be installed.

14.2.2. Dependent Features

Dependent features are useful when a given feature depends on another feature to be installed. Such a dependency can be expressed easily in the feature definition:

```
<feature name="jbi">
  <feature>nmr</feature>
  ...
</feature>
```

The effect of such a dependency is to automatically install the required nmr feature when the jbi feature will be installed. A version range can be specified on the feature dependency:

```
<feature name="spring-dm">
  <feature version="[2.5.6,4)">spring</feature>
  ...
</feature>
```

In such a case, if no matching feature is already installed, the feature with the highest version available in the range will be installed. If a single version is specified, this version will be chosen. If nothing is specified, the highest available will be installed.

14.2.3. Configurations

The configuration section allows to deploy configuration for the OSGi Configuration Admin service along a set of bundles. Here is an example of such a configuration:

```
<config name="com.foo.bar">
  myProperty = myValue
</config>
```

The name attribute of the configuration element will be used as the ManagedService PID for the configuration set in the Configuration Admin service. When using a ManagedServiceFactory, the name attribute is servicePid-_aliasId_, where servicePid is the PID of the ManagedServiceFactory and aliasId is a label used to uniquely identify a particular service (an alias to the factory generated service PID). Deploying such a configuration has the same effect than dropping a file named com.foo.bar.cfg into the etc folder.

The content of the configuration element is set of properties parsed using the standard java property mechanism. Such configuration as usually used with Spring-DM or Blueprint support for the Configuration Admin service, as in the following example, but using plain OSGi APIs will of course work the same way:

```
<bean ...>
    <property name="propertyName" value="${myProperty}" />
</bean>

<osgix:cm-properties id="cmProps" persistent-id="com.foo.bar">
    <prop key="myProperty">myValue</prop>
</osgix:cm-properties>
<ctx:property-placeholder properties-ref="cmProps" />
```

There may also be cases where you want to make the properties from multiple configuration files available to your bundle context. This is something you may want to do if you have a multi-bundle application where there are application properties used by multiple bundles, and each bundle has its own specific properties. In that case, <ctx:property-placeholder> won't work as it was designed to make only one configuration file available to a bundle context. To make more than one configuration file available to your bundle-context you would do something like this:

```
<beans:bean id="myBundleConfigurer" class=
   "org.springframework.beans.factory.config.PropertyPlaceholderConfig">
   <beans:property name="ignoreUnresolvablePlaceholders" value="true"/>
   <beans:property name="propertiesArray">
       <osgix:cm-properties id="myAppProps" persistent-id="myApp.props"/>
       <osgix:cm-properties id="myBundleProps"
           persistent-id="my.bundle.props"/>
   </beans:property>
</beans:bean>
```

In this example, we are using SpringDM with osgi as the primary namespace. Instead of using ctx:context-placeholder we are using the "PropertyPlaceholderConfig" class. Then we are passing in a beans array and inside of that array is where we set our osgix:cm-properties elements. This element "returns" a properties bean.

For more informations about using the Configuration Admin service in Spring-DM, see the Spring-DM documentation.

14.2.4. Configuration files

In certain cases it is needed not only to provide configurations for the configuration admin service but to add additional configuration files e.g. a configuration file for jetty (jetty.xml). It even might be help full to deploy

a configuration file instead of a configuration for the config admin service since. To achieve this the attribute finalname shows the final destination of the configfile, while the value references the Maven artifact to deploy.

```
<configfile finalname="/etc/jetty.xml">mvn:org.apache.karaf/apache-karaf/   \\
    ${project.version}/xml/jettyconfig</configfile>
```

14.2.5. Feature resolver

The resolver attribute on a feature can be set to force the use of a given resolver instead of the default resolution process. A resolver will be use to obtain the list of bundles to actually install for a given feature. The default resolver will simply return the list of bundles provided in the feature description. The obr resolver can be installed and used instead of the standard one. In that case, the resolver will use the OBR service to determine the list of bundles to install (bundles flagged as dependency will only be used as possible candidates to solve various constraints).

14.3. Commands

14.3.1. Repository management

The following commands can be used to manage the list of descriptors known by Karaf. They use URLs pointing to features descriptors. These URLs can use any protocol known to Apache Karaf, the most common ones being http, file and mvn.

- **features:addUrl**: Add a list of repository URLs to the features service

- **features:removeUrl**: Remove a list of repository URLs from the features service

- **features:listUrl**: Display the repository URLs currently associated with the features service.

- **features:refreshUrl**: Reload the repositories to obtain a fresh list of features

Karaf maintains a persistent list of these repositories so that if you add one URL and restart Karaf, the features will still be available. The **refreshUrl** command is mostly used when developing features descriptors: when changing the descriptor, it can be handy to reload it in the Kernel without having to restart it or to remove then add again this URL.

14.3.2. Features management

Common features: scope commands used in features management include **features:install**, **features:uninstall**, and **features:list**. See Section 5.2.4, "Features Scope" for more information on these commands.

14.3.3. Examples

To install features using mvn handler:

```
features:addUrl mvn:org.apache.servicemix.nmr/apache-servicemix-nmr/   \\
    1.0.0-m2/xml/features
features:install nmr
```

To use a file handler to deploy a features file (note the path is relative to the Apache Karaf installation directory):

```
features:addUrl file:base/features/features.xml
```

To deploy bundles from file system without using Maven: As we can use file:// as protocol handler to deploy bundles, you can use the following syntax to deploy bundles when they are located in a directory which is not available using Maven:

```
<features xmlns="http://karaf.apache.org/xmlns/features/v1.0.0">
   <feature name="spring-web" version="2.5.6.SEC01">
       <bundle>file:base/bundles/spring-web-2.5.6.SEC01.jar</bundle>
   </feature>
</features>
```

Note the path again is relative to Apache Karaf installation directory.

14.4. Service configuration

A simple configuration file located in `[FELIX:karaf]/etc/org.apache.karaf.features.cfg` can be modified to customize the behavior when starting the Kernel for the first time. This configuration file contains two properties:

- featuresBoot: a comma separated list of features to install at startup

- featuresRepositories: a comma separated list of feature repositories to load at startup

This configuration file is of interest if you plan to distribute a Apache Karaf distribution which includes pre-installed features.

Chapter 15. Web Applications

Karaf provides an ability to deploying WAR-based web applications within the Jetty server contained in the Karaf instance.

15.1. Installing WAR support

The following steps will install the "war" feature (support for deploying WAR files with Servlet and JSPs into a Jetty server) into your Karaf instance.

1. List the available features -

    ```
    karaf@trun> features:list
    State          Name
    . . .
    [uninstalled] [2.2.0] obr        karaf-2.2.0
    [uninstalled] [2.2.0] config     karaf-2.2.0
    [uninstalled] [2.2.0] http       karaf-2.2.0
    [uninstalled] [2.2.0] war        karaf-2.2.0
    [uninstalled] [2.2.0] webconsole karaf-2.2.0
    [installed  ] [2.2.0] ssh        karaf-2.2.0
    . . .
    ```

2. Install the war feature (and the sub-features it requires) - **karaf@trun> features:install war**

    Note: you can use the -v or --verbose to see exactly what Karaf does

    ```
    karaf@trun> features:install -v war
    Installing feature war 2.1.99-SNAPSHOT
    ```

_____

```
      Installing feature http 2.1.99-SNAPSHOT
      Installing feature jetty 7.1.6.v20100715
      Installing bundle mvn:org.apache.geronimo.specs/
          geronimo-servlet_2.5_spec/1.1.2
      Found installed bundle: org.apache.servicemix.bundles.asm [10]
      Installing bundle mvn:org.eclipse.jetty/jetty-util/7.1.6.v20100715
      Installing bundle mvn:org.eclipse.jetty/jetty-io/7.1.6.v20100715
      Installing bundle mvn:org.eclipse.jetty/jetty-http/7.1.6.v20100715
      Installing bundle mvn:org.eclipse.jetty/jetty-continuation/7.1.6.v20100715
      Installing bundle mvn:org.eclipse.jetty/jetty-server/7.1.6.v20100715
      Installing bundle mvn:org.eclipse.jetty/jetty-security/7.1.6.v20100715
      Installing bundle mvn:org.eclipse.jetty/jetty-servlet/7.1.6.v20100715
      Installing bundle mvn:org.eclipse.jetty/jetty-xml/7.1.6.v20100715
      Checking configuration file mvn:org.apache.karaf/apache-karaf/
          2.1.99-SNAPSHOT/xml/jettyconfig
      Installing bundle mvn:org.ops4j.pax.web/pax-web-api/0.8.2-SNAPSHOT
      Installing bundle mvn:org.ops4j.pax.web/pax-web-spi/0.8.2-SNAPSHOT
      Installing bundle mvn:org.ops4j.pax.web/pax-web-runtime/0.8.2-SNAPSHOT
      Installing bundle mvn:org.ops4j.pax.web/pax-web-jetty/0.8.2-SNAPSHOT
      Installing bundle mvn:org.ops4j.pax.web/pax-web-jsp/0.8.2-SNAPSHOT
      Installing bundle mvn:org.ops4j.pax.web/pax-web-extender-war/
          0.8.2-SNAPSHOT
      Installing bundle mvn:org.ops4j.pax.web/pax-web-extender-whiteboard/
          0.8.2-SNAPSHOT
      Installing bundle mvn:org.ops4j.pax.web/pax-web-deployer/0.8.2-SNAPSHOT
      Installing bundle mvn:org.ops4j.pax.url/pax-url-war/1.2.4
```

3. Verify the features were installed

```
      servicemix> features/list
      State           Name
      . . .
      [installed ] [2.2.0] http karaf-2.2.0
      [installed ] [2.2.0] war  karaf-2.2.0
      . . .
```

4. Verify the installed bundles were started

```
      karaf@trun> osgi:list
      START LEVEL 100
       ID    State           Level  Name
      . . .
      [ 32] [Active ] [ ] [ 60] geronimo-servlet_2.5_spec (1.1.2)
      [ 33] [Active ] [ ] [ 60] Apache ServiceMix :: Bundles ::
          jetty (6.1.22.2)
      [ 34] [Active ] [ ] [ 60] OPS4J Pax Web - API (1.0.0)
      [ 35] [Active ] [ ] [ 60] OPS4J Pax Web - Service SPI (1.0.0)
      [ 36] [Active ] [ ] [ 60] OPS4J Pax Web - Runtime (1.0.0)
      [ 37] [Active ] [ ] [ 60] OPS4J Pax Web - Jetty (1.0.0)
      [ 38] [Active ] [ ] [ 60] OPS4J Pax Web - Jsp Support (1.0.0)
      [ 39] [Active ] [ ] [ 60] OPS4J Pax Web - Extender - WAR (1.0.0)
      [ 40] [Active ] [ ] [ 60] OPS4J Pax Web -
          Extender - Whiteboard (1.0.0)
      [ 42] [Active ] [ ] [ 60] OPS4J Pax Web - FileInstall Deployer (1.0.0)
      [ 41] [Active ] [ ] [ 60] OPS4J Pax Url - war:, war-i: (1.2.4)
      . . .
```

5. The Jetty server should now be listening on http://localhost:8181/, but with no published applications available.

```
HTTP ERROR: 404
NOT_FOUND
RequestURI=/
Powered by jetty://
```

15.2. Deploying a WAR to the installed web feature

The following steps will describe how to install a simple WAR file (with JSPs or Servlets) to the just installed web feature.

1. To deploy a WAR (JSP or Servlet) to Jetty, update its MANIFEST.MF to include the required OSGi headers as described here: http://team.ops4j.org/wiki/display/paxweb/WAR+Extender

2. Copy the updated WAR (archive or extracted files) to the deploy directory.

If you want to deploy a sample web application into Karaf, you could use the following command:

**karaf@trun> osgi:install -s webbundle:http://tomcat.apache.org/ **
**tomcat-5.5-doc/appdev/sample/sample.war?Bundle-SymbolicName= **
tomcat-sample&Webapp-Context=/sample

Then open your web browser and point to http://localhost:8181/sample/index.html.

Chapter 16. Monitoring and Administration using JMX

Apache Karaf provides a large set of MBeans that allow you to fully monitor and administrate Karaf using any JMX client (for example, JConsole provided in the Oracle or IBM JDK). They provide more or less the same actions that you can do using the Karaf shell commands. The list of MBeans available:

Table 16.1. Karaf Management MBeans

MBean (org.apache.karaf:type value)	Description
admin	administrates the child instances
bundles	manipulates the OSGi bundles
config	manipulates the Karaf configuration files (in the /etc folder) and the ConfigAdmin layer
dev	provides information and administration of the OSGi framework
diagnostic	used to create information files (dumps) about Karaf activity
features	manipulate the Karaf features
log	manipulate the logging layer
packages	manipulate the PackageAdmin layer and get information about exported and imported packages
services	to get information about the OSGi services
system	to shutdown the Karaf container itself
web	to get information about the Web bundles (installed with the war feature)
obr	to manipulate the OBR layer (installed with the obr feature)

Chapter 17. Installing the Talend Runtime container as a service

17.1. Introduction

The Talend Runtime container is based on Apache Karaf. Karaf Wrapper (for service wrapper) makes it possible to install the Talend Runtime container as a Windows Service. Likewise, the scripts shipped with Karaf also make it very easy to install the Talend Runtime container as a daemon process on Unix systems.

To install Talend Runtime container as a service, you first have to install the wrapper, which is an optional feature.

The Wrapper correctly handles "user log outs" under Windows, service dependencies, and the ability to run services which interact with the desktop.

17.2. Supported platforms

The following platforms are supported by the Wrapper:

- AIX

- FreeBSD

- HP-UX, 32-bit and 64-bit versions

- SGI Irix

- Linux kernels 2.2.x, 2.4.x, 2.6.x. Known to work with Debian, Ubuntu, and Red Hat, but should work with any distribution. Currently supported on both 32-bit and 64-bit x86, Itanium, and PPC systems.

- Macintosh OS X

- Sun OS, Solaris 9 and 10. Currently supported on both 32-bit and 64-bit sparc, and x86 systems.

- Windows - Windows 2000, XP, 2003, Vista, 2008 and Windows 7. Currently supported on both 32-bit and 64-bit x86 and Itanium systems. Also known to run on Windows 98 and ME, however due the lack of support for services in the OS, the Wrapper can be run only in console mode.

17.3. Installing the wrapper

First, to install the wrapper, simply:

1. Browse to the `bin` folder of the Talend Runtime container directory, then launch:

   - `trun.bat` in Administrator mode on Windows

   - `trun` as root user on Linux

2. To install the wrapper feature, simply type:

   - **karaf@trun> features:install wrapper** on Windows.

   - **trun@root> features:install wrapper** on Linux.

   Once installed, wrapper feature will provide **wrapper:install** new command in the trun:

```
trun@root> wrapper:install --help

DESCRIPTION
        wrapper:install

        Install the container as a system service in the OS.

SYNTAX
        wrapper:install [options]

OPTIONS
        -d, --display
                The display name of the service.
        --help
                Display this help message

-s, --start-type
                Mode in which the service is installed. AUTO_START or
                DEMAND_START (Default: AUTO_START)
                (defaults to AUTO_START)

        -n, --name
                The service name that will be used when installing the
                service. (Default: Karaf)
                (defaults to karaf)

-D, --description
                The description of the service.
                (defaults to )
```

3. To set up the installation of the service, type in the following command:

   - **karaf@trun> wrapper:install** on Windows.

_____

- **trun@root> wrapper:install** on Linux.

For instance, to register Talend Runtime container as a service (depending on the running OS), in automatic start mode, simply type:

- For Windows:

  **karaf@trun> wrapper:install -s AUTO_START -n TALEND-ESB-CONTAINER -d Talend-ESB-Container
    -D "Talend ESB Container Service"**

- For Linux:

  **trun@root> wrapper:install -s AUTO_START -n TALEND-ESB-CONTAINER -d Talend-ESB-Container
    -D "Talend ESB Container Service"**

Here is an example of **wrapper:install** command executing on Windows:

```
karaf@trun> wrapper:install -s AUTO_START -n TALEND-ESB-CONTAINER
-d Talend-ESB-Container  -D "Talend ESB Container Service"

Creating file: C:\work\5.0.1-release\Talend-ESB-V5.0.1\container\bin\
TALEND-ESB-CONTAINER-wrapper.exe
Creating file: C:\work\5.0.1-release\Talend-ESB-V5.0.1\container\etc\
TALEND-ESB-CONTAINER-wrapper.conf
Creating file: C:\work\5.0.1-release\Talend-ESB-V5.0.1\container\bin\
TALEND-ESB-CONTAINER-service.bat
Creating file: C:\work\5.0.1-release\Talend-ESB-V5.0.1\container\lib\
wrapper.dll
Creating file: C:\work\5.0.1-release\Talend-ESB-V5.0.1\container\lib\
karaf-wrapper.jar
Creating file: C:\work\5.0.1-release\Talend-ESB-V5.0.1\container\lib\
karaf-wrapper-main.jar


Setup complete.  You may wish to tweak the JVM properties in the
wrapper configuration file:
        C:\work\5.0.1-release\Talend-ESB-V5.0.1\container\etc\TALEND-
ESB-CONTAINER-wrapper.conf
before installing and starting the service.

To install the service, run:
  C:> C:\work\5.0.1-release\Talend-ESB-V5.0.1\container\bin\
TALEND-ESB-CONTAINER-service.bat install

Once installed, to start the service run:
  C:> net start "TALEND-ESB-CONTAINER"

Once running, to stop the service run:
  C:> net stop "TALEND-ESB-CONTAINER"

Once stopped, to remove the installed the service run:
  C:> C:\work\5.0.1-release\Talend-ESB-V5.0.1\container\bin\
TALEND-ESB-CONTAINER-service.bat remove
```

Here is an example of **wrapper:install** command executing on Linux:

```
trun@root> wrapper:install -s AUTO_START -n TALEND-ESB-CONTAINER  \
        -d Talend-ESB-Container -D "Talend ESB Container Service"
Creating file: /home/onofreje/5.0.1-release/Talend-ESB-V5.0.1/container/
bin/KARAF-wrapper
```

```
Creating file: /home/onofreje/5.0.1-release/Talend-ESB-V5.0.1/container/
bin/KARAF-service
Creating file: /home/onofreje/5.0.1-release/Talend-ESB-V5.0.1/container/
etc/KARAF-wrapper.conf
Creating file: /home/onofreje/5.0.1-release/Talend-ESB-V5.0.1/container/
lib/libwrapper.so
Creating file: /home/onofreje/5.0.1-release/Talend-ESB-V5.0.1/container/
lib/karaf-wrapper.jar
Creating file: /home/onofreje/5.0.1-release/Talend-ESB-V5.0.1/container/
lib/karaf-wrapper-main.jar

Setup complete.  You may want to tweak the JVM properties in the wrapper
configuration file:
        /home/onofreje/apache-karaf-2.1.3/etc/KARAF-wrapper.conf
before installing and starting the service.
```

17.4. Installing the service

17.4.1. On Windows

1. Open a CMD window in Administrator mode.

2. Browse to the `bin` folder of the Talend Runtime installation directory, then type in the following command:

    TALEND-ESB-CONTAINER-service install


    ```
    Here is an example of the installation of Talend Runtime container as a service on W

    C:\Builds\Talend-Runtime\bin>TALEND-ESB-CONTAINER-service.bat install

    wrapper | Talend ESB Container installed.

    C:\Builds\Talend-Runtime\bin>
    ```


    The Talend Runtime service is created and can be viewed by selecting Control Panel > Administrative Tools > Services in the Start menu of Windows.

You can then run the **net start "TALEND-CONTAINER"** and **net stop "TALEND-ESB-CONTAINER"** commands to manage the service.

To remove the service, type in the following command in the command window:

TALEND-ESB-CONTAINER-service.bat remove

17.4.2. On Linux

The way the service is installed depends upon your flavor of Linux:

17.4.2.1. On Redhat/Fedora/CentOS Systems

To install the service:

$ ln -s /home/onofreje/5.0.1-release/Talend-ESB-V5.0.1/container/bin/TALEND-ESB-CONTAINER-service /etc/init.d/
$ chkconfig TALEND-ESB-CONTAINER-service --add

To start the service when the machine is rebooted:

$ chkconfig TALEND-ESB-CONTAINER-service on

To disable starting the service when the machine is rebooted:

$ chkconfig TALEND-ESB-CONTAINER-service off

To start the service:

$ service TALEND-ESB-CONTAINER-service start

To stop the service:

$ service TALEND-ESB-CONTAINER-service stop

To uninstall the service :

$ chkconfig TALEND-ESB-CONTAINER-service --del
$ rm /etc/init.d/TALEND-ESB-CONTAINER-service

17.4.2.2. On Ubuntu/Debian Systems

To install the service:

$ ln -s /home/onofreje/5.0.1-release/Talend-ESB-V5.0.1/container/bin/
TALEND-ESB-CONTAINER-service /etc/init.d/

To start the service when the machine is rebooted:

$ update-rc.d TALEND-ESB-CONTAINER-service defaults

To disable starting the service when the machine is rebooted:

$ update-rc.d -f TALEND-ESB-CONTAINER-service remove

To start the service:

$ /etc/init.d/TALEND-ESB-CONTAINER-service start

To stop the service:

$ /etc/init.d/TALEND-ESB-CONTAINER-service stop

To uninstall the service :

$ rm /etc/init.d/TALEND-ESB-CONTAINER-service

Chapter 18. Troubleshooting Talend ESB

This chapter describes how to fix problems related to JVM memory allocation.

18.1. Memory Allocation Parameters

The Talend ESB start scripts define the JVM memory allocation parameters used for running Talend ESB.

JAVA_MIN_MEM. Determines the start size of the Java heap memory. Higher values may reduce garbage collection and improve performance. Corresponds to the JVM parameter -Xms.

JAVA_MAX_MEM. Determines the maximum size of the Java heap memory. Higher values may reduce garbage collection, improve performance and avoid "Out of memory" exceptions. Corresponds to the JVM parameter -Xmx.

JAVA_PERM_MEM. Determines the start size of permanent generation. This is a separate heap space that is normally not garbage collected. Java classes are loaded in permanent generation. Using Talend ESB many classes are loaded, specially when using security features and deploying multiple services to the same container. Increase if a "PermGen space" exception is thrown. Corresponds to the JVM parameter -XX:PermSize

JAVA_MAX_PERM_MEM. Determines the maximum size of permanent generation. This is a separate heap space that is normally not garbage collected. Increase if a "PermGen space" exception is thrown. Corresponds to the JVM parameter -XX:MaxPermSize

Additionally the following parameters are used as default to enable class unloading from permanent generation which reduces the overall permanent generation space usage: -XX:+CMSClassUnloadingEnabled and -XX:+UseConcMarkSweepGC.

18.2. On Windows

Adapt the memory allocation parameters in the karaf start script at the following location: `[karaf_install_dir]/bin/trun.bat`.

Determined by the script variable

`%Bit_64%`

you can set different values depending whether your JVM is 64-bit. 64-bit-JVMs need significantly more memory.

18.3. On Linux

Adapt the memory allocation parameters in the karaf start script at the following location: `[karaf_install_dir]/bin/trun`.

Determined by the script variable

`$JAVA_SIXTY_FOUR`

you can set different values depending whether your JVM is 64-bit. 64-bit-JVMs need significantly more memory.