Y A D E

YADE - Managed File Transfer

Application Program Interface Reference

November 2015

Contact Information

Software- und Organisations-Service GmbH

Giesebrechtstr. 15
D-10629 Berlin
Germany

Telephone +49 (0)30 86 47 90-0
Telefax +49 (0)30 8 61 33 35
Mail info@sos-berlin.com
Web http://www.sos-berlin.com

Last Updated: 11/04/2015 05:33 PM

This documentation is based on JobScheduler Version 1.11.0-SNAPSHOT.

This product includes software developed by the Apache Software Foundation (http://apache.org/)

We would appreciate any feedback you have, or suggestions for changes and improvements; please forward your comments to info@sos-berlin.com.

Table of Contents

1 Introduction 4

2 The Object Model 5
2.1 The Engine 5
2.2 The Options Class 5
2.2.1 Initializing the Parameters 6
2.2.2 Methods and Properties 6
2.2.3 Using the Configuration File 6
2.2.4 Using a hashmap for options 7
2.2.5 Using command line parameters 8
2.3 The Virtual Filesystem 9
2.3.1 Instantiating the VFS 9
2.3.2 Getting the Data Provider 9
2.4 The Callback Interface 9
2.5 The Adapter Classes 10
2.5.1 Command line 10
2.5.2 JobScheduler 10

3 Examples 11
3.1 Simple Transfer 11
3.1.1 Instantiating the Engine 11
3.1.2 Defining the Parameter 11
3.1.3 Starting the Transfer 11
3.2 Sending 11
3.3 Receiving 11
3.4 Site to Site Transfer 11

4 More Examples 12
4.1 JUnit Tests 12
4.2 Source Code 12
4.2.1 YADE Engine 12
4.2.2 The VFS 12
4.2.3 JITL Jobs 12

5 YADE API Installation 13

6 Acknowledgements 14

1 Introduction

YADE (JobScheduler Advanced Data Exchange; formerly SOSFtp) is a solution for Managed File Transfer (MFT). It supports atomic and transactional data transfer, consistent logging, a transfer history, automatic alerting and reporting and much more.

With YADE API it is possible to use the full functionality of YADE in any other application.

YADE has a strong component oriented architecure. This makes it possible to use YADE in different environments, from a command line client up to the usage in any customer applications. YADE consists of the components Engine, Virtual File System (VFS), Options Engine and Adapters or Bridges.

YADE is implemented using pure Java®. That makes it possible that YADE can be used by nearly any language which is running in a Java® Virtual Machine (JVM). And can run on any operating system on which a Java® Virtual Machine (JVM) is able to run.

Java® Version 6 and higher is required to run YADE.

This handbook describes how to use the YADE API in a Java® environment. Knowledge of Java® and the concepts of object oriented programming (OOP) is required.

2 The Object Model

YADE has a strong component oriented architecure. This makes it possible to use YADE in different environments, from a command line client up to the usage in any customer applications. YADE consists of the components Engine, Virtual File System (VFS), Options Engine and Adapters or Bridges.

2.1 The Engine

"The Engine" is the core of YADE. Here the logic of the application is implemented. This relates to the transfer at an abstract level. Also various transformations that are performed by YADE are here implemented.

The transfer operations itself are performed by various componente of the Virtual File System (VFS).

The Engine is able to communicate via interfaces with the adapter-classe(s).

```
import com.sos.DataExchange.JadeEngine;
import com.sos.VirtualFileSystem.FTP.SOSFTPOptions;
 ...
JadeEngine objJadeEngine = new JadeEngine();
objOptions = objJadeEngine.Options();
// set the options ...
objJadeEngine.Execute();

objJadeEngine.Logout();
```

Example: get an instance of the engine

To get an instance of the YADE engine you have to create a new instance of the JadeEngine class. The next step is to get the instance of the options class from the Engine, set all the needed parameters. The Method Execute with start the execution of the transfer. And at last the Method Logout will logout the user and disconnect the connection to the data source and to the data target, if possible.

How to set options is described in one of the next chapters.

2.2 The Options Class

The options class is a container for all options (or settings) which are needed to perform the transfer process. Options are implemented as objects and all of this options can be set by the application which is using YADE.

The architecture of the options class is not specific for YADE, but are used by all JITL jobs. In general, an options class is generated from the sosdoc-template, which is available for all JITL jobs.

Independently of the source of parameters, e.g. profile or cmd-line or anything else, there is no differentially spelling.

2.2.1 Initializing the Parameters

```
SOSFTPOptions objOptions = new SOSFTPOptions();
```

Example: Initializing the Parameters

This code line create a new instance of the options class.

2.2.2 Methods and Properties

Every option, which is documented in the parameter documentation, is available as a method. The option itself is implemented by a class. The classes are packaged in the com.sos.JSHelper package. (see here and the unit tests here)

```
SOSFTPOptions objOptions = new SOSFTPOptions();
         ...
         objOptions.host.Value(conHostNameWILMA_SOS);
         objOptions.protocol.Value(enuTransferTypes.sftp);
         objOptions.port.value(SOSOptionPortNumber.conPort4SSH);
         objOptions.user.Value("test");
         objOptions.password.Value("12345");
         ...
         Assert.assertEquals("Source.Host", conHostNameWILMA_SOS, objOptions.host.Value());
         Assert.assertEquals("file_path", "test.txt", objOptions.file_path.Value());
```

Example: Read and Write options by methods

Because an option is implemented as an instance of a class you can direct access this instance (yes, it is not the hardcore jave style with getters and setters, but i like it). Every class has properties, e.g. to set a value or get a value or to do some other things which are implemented in the class.

Classes have sometimes, depending on the datatype, more than one possibility to set a value. e.g. the "port" can be set as a string or can be set as an integer.

The source code and the doxygen based documentation for the option classes can be found here here and the unit tests here.

2.2.3 Using the Configuration File

The configuration file is an ini file with the well known structure of sections and entries. This ini-file can be used by the options class as well.

Using the ini file is the most common way to configure YADE. An ini file can be used by any kind of adapter which is calling the YADE engine.

```
public void testIniFileWithSourceAndTarget() throws Exception {
    objOptions.settings.Value("./sosdex_settings.ini");
    objOptions.profile.Value("ftp_server_2_server");
    objOptions.ReadSettingsFile();

    SOSConnection2Options objConn = objOptions.getConnectionOptions();

    String strComputerName = System.getenv("computername");
    Assert.assertEquals("Source.Host", conHostNameWILMA_SOS, objConn.Source().host.Value());
    Assert.assertEquals("Target.Host", conHostName8OF9_SOS, objConn.Target().host.Value());
    Assert.assertEquals("file_path", "test.txt", objOptions.file_path.Value());

    objOptions.CheckMandatory();
    // logger.info(objOptions.toString());

    JadeEngine objJadeEngine = new JadeEngine(objOptions);
    objJadeEngine.Execute();
    objJadeEngine.Logout();
}
```

Example: Initialize the options class by ini file

In this example a profile (aka ini file section) is used to configure the options. The name of the ini file is defined by the option "settings" and the profile by the option "profile". Actually it is required to use the method "ReadSettingsFile()" to signal that a settings file will be used and the content has to be processed.

```
[ftp_server_2_server]
ssh_auth_method=password

source_user=kb
source_password=kb
source_ssh_auth_method=password
source_host=wilma.sos
source_protocol=sftp
source_port=22
;source_dir=${local_dir}

target_user=kb
target_password=kb
target_host=8of9.sos
target_protocol=ftp
target_port=21
;target_dir=${remote_dir}

file_path=test.txt
local_dir=/home/kb/
remote_dir=/kb/
operation=copy

log_filename=${TEMP}/sosftphistory.log
```

Example: The section and entries

The section and entries

2.2.4 Using a hashmap for options

```
                HashMap<String, String> objHsh = new HashMap<String, String>();

objHsh.put("operation", "copy");

objHsh.put("source_host", conHostNameWILMA_SOS);
objHsh.put("alternative_source_user", "sos");
objHsh.put("alternative_source_password", "sos");

objHsh.put("source_user", "sos");
objHsh.put("source_port", "22");
objHsh.put("source_protocol", "sftp");
objHsh.put("source_password", "****");
objHsh.put("source_dir", "/home/sos/setup.scheduler/releases");
objHsh.put("source_ssh_auth_method", "password");

objHsh.put("target_host", "tux.sos");
objHsh.put("target_protocol", "sftp");
objHsh.put("target_port", "22");
objHsh.put("target_password", "sos");
objHsh.put("target_user", "sos");
objHsh.put("alternative_target_user", "abcdef");

objHsh.put("target_dir", "/srv/www/htdocs/test");
objHsh.put("target_ssh_auth_method", "password");

objHsh.put("overwrite_files", "true");
objHsh.put("check_size", "true");
objHsh.put("file_spec", "^scheduler_(win32|linux)_joe\\.[0-9]+\\.[0-9]+\\.[0-9]+\\.[0-9]{4}\\.(tar\\.gz|zip)$");
objHsh.put("recursive", "false");

objHsh.put("verbose", "9");
objHsh.put("buffer_size", "32000");
objHsh.put("SendTransferHistory", "false");
objHsh.put("log_filename", "c:/temp/test.log");

objOptions = new SOSFTPOptions();
objOptions.setAllOptions(objHsh);

JadeEngine objJadeEngine = new JadeEngine(objOptions);

assertEquals("replacing", "^renamed_", objOptions.replacing.Value());
assertEquals("replacement", "", objOptions.replacement.Value());

objJadeEngine.Execute();
objJadeEngine.Logout();
```

Example: Initialize the options by a hashmap

This example show a site to site transfer using sftp.

The Metho "setAllOptions(hashMap)" is used to set the options via a HashMap. This Method is used for example by the JobScheduler Adapter.

2.2.5 Using command line parameters

```
                application.sh -settings=./jade-settings.ini -profile=test4711
....
public final static void main(final String[] pstrArgs) {

    JadeEngine objEngine = new JadeEngine();
    SOSFTPOptions objO = objEngine.Options();
    objO.CommandLineArgs(pstrArgs);

    objO.CheckMandatory();
    objEngine.Execute();
    objEngine.Logout();
    System.exit(0);
}
```

Example: Initialize the options by command line parameters

This example shows how to initialize an options class using command line parameters. On the command line the name and the value of a parameter must be specified like "-name=value". If the parameter "-settings=" and "-profile=" is specified all other parameters can be defined in the profile.

The method "CheckMandatory()" will check whether all mandatory options at least has a value assigned. If not, an error message will raised and the execution will aborted with an exit-code.

The command line parameters can be used directly:

```
    public void testDeleteZipFile() throws Exception {

        String[] strCmdLineParameters = new String[] { "-settings=" + strSettingsFileName, "-profile=zip_local_files"
};

        objOptions.CommandLineArgs(strCmdLineParameters);

        File fleFile = new File(objOptions.remote_dir.Value());
        fleFile.delete();

    } // private void testDeleteZipFile
```

Example: command line as a string

command line as a string

2.3 The Virtual Filesystem

2.3.1 Instantiating the VFS

2.3.2 Getting the Data Provider

2.4 The Callback Interface

The callback interface was primary developed for the communication of an JobScheduler API worker class and the JobScheduler adapter. But nevertheless it is very usefull even for other adapters, just to communicate with the engine.

```
public interface JSJobUtilities {
           public String myReplaceAll(String source, String what, String replacement) ;
           public String replaceSchedulerVars(boolean isWindows, final String pstrString2Modify);
           public void setJSParam(final String pstrKey, final String pstrValue);
           public void setJSParam(final String pstrKey, final StringBuffer pstrValue);
           public String getCurrentNodeName ();
           public void setJSJobUtilites (JSJobUtilities pobjJSJobUtilities);
           public void setStateText(final String pstrStateText);
      }
```

Example: The callback interface

This interface is used to establish a communication between the YADE engine and the calling application. For example, via "setStateText" the engine is ongoing reporting the status of a transfer. Via the JobScheduler adapter the state of a job, during the job is running, is updated using this way.

The method "getCurrentNodeName" is used by the JobScheduler during execution of a job chain to indicate which step (state) of the chain is actually running.

2.5 The Adapter Classes

2.5.1 Command line

2.5.2 JobScheduler

3 Examples

3.1 Simple Transfer

3.1.1 Instantiating the Engine

3.1.2 Defining the Parameter

3.1.3 Starting the Transfer

3.2 Sending

3.3 Receiving

3.4 Site to Site Transfer

4 More Examples

4.1 JUnit Tests

4.2 Source Code

4.2.1 YADE Engine

The source code and the doxygen based documentation can be found here here . The source code and the doxygen based documentation for the JUnit test can be found here here .

4.2.2 The VFS

The source code and the doxygen based documentation can be found here here. The source code and the doxygen based documentation for the JUnit test can be found here here.

4.2.3 JITL Jobs

The source code and the doxygen based documentation can be found here SOSScheduler. The source code and the doxygen based documentation for the JUnit test can be found here here.

5 YADE API Installation

The setup for the client and/or the JobScheduler setup is sufficient for to use the YADE API. No further installation steps are need.

YADE requires the jars listed below during runtime.

6 Acknowledgements