Cummins – Integrity Java API Implementation

Table of Contents

[Jar Documentation 2](#_Toc400455372)

[createSandbox.jar 2](#_Toc400455373)

[Inputs 2](#_Toc400455374)

[Outputs 2](#_Toc400455375)

[Notes 2](#_Toc400455376)

[Example 2](#_Toc400455377)

[To-Do 3](#_Toc400455378)

[Addlabel.jar 3](#_Toc400455379)

[Inputs 3](#_Toc400455380)

[Outputs 3](#_Toc400455381)

[Notes 3](#_Toc400455382)

[Example 3](#_Toc400455383)

[To-Do 3](#_Toc400455384)

[Commented Code 4](#_Toc400455385)

[ArgParser.java 4](#_Toc400455386)

[CommandMan.java 7](#_Toc400455387)

[Main.java – (CreateSandbox.jar) 9](#_Toc400455388)

# Jar Documentation

## createSandbox.jar

### Inputs

#### **Required**

Project – Which project is the sandbox created from?

Path – Where will the sandbox be stored?

Label – Which label should all of the included files contain?

#### Optional

N/A

### Outputs

Sandbox is created.

No outputs via the command line

#### Fail Cases

0 – good!

1 – syntax error

2 – connection error

### Notes

User must specify all three of the required labels in order for the Java API to produce desired response

#### Assumptions

User is logging in as mk347 (Connor Boyle)

#### Classes Used

CommandMan.java

ArgParser.java

#### Libraries Used

mksapi.jar

### Example



### To-Do

What if label is not required?

What if a sandbox needs to be filtered on more than one label?

What if user isn’t mk347? – Should we create an “api” account?

## Addlabel.jar

### Inputs

#### **Required**

#### Optional

### Outputs

### Notes

#### Assumptions

#### Classes Used

#### Libraries Used

### Example

### To-Do

# Commented Code

## ArgParser.java

**package** com.mk347.jarArguments;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** com.mks.api.Option;

**import** com.mks.api.OptionList;

/\*\*

\*

\* ArgParser parses the arguments from the command line and creates meaningful Option Objects

\* for adding to Command object in Main.java prior to command execution

\* **@author** mk347, Connor Boyle

\* **@version** 10-7-2014

\*/

**public** **class** ArgParser {

//Default accepted switches. Anything else will throw an exception!!!!

**private** **static** **final** String *SWITCH\_LABEL* = "label";

**private** **static** **final** String *SWITCH\_PROJECT* = "project";

**private** **static** **final** String *SWITCH\_PATH* = "path";

//Arraylists to store Arguments and split-Arguments

**private** ArrayList<String> args;

**private** ArrayList<String[]> splitArgs;

//String path to store the sandbox's path

**public** String path;

/\*\*

\* Constructs an ArgParser object

\*

\* Called by Main.main()

\*

\* **@param** args - list of arguments from the command line

\*/

**public** ArgParser(String[] args)

{

//Set incoming args array to private variable

**this**.args = **new** ArrayList<String>(Arrays.*asList*(args));

}

/\*\*

\* Print the args private variable with formatting

\* (Used for debugging)

\*

\* Can be called by Main.main()

\*/

**public** **void** printArguments()

{

//Loop through ArrayList

**for**(**int** i = 0; i < args.size(); i++)

{

//Print the argument and the Index with formatting

System.*out*.println("Args:"+args.get(i)+ " Index:"+i);

}

}

/\*\*

\* Print the splitArgs private variable with formatting

\* (Used for debugging)

\*

\* Can be called by Main.main()

\*

\*/

**public** **void** printSplitArgs()

{

//Loop through splitArgs

**for** (**int** i = 0; i < splitArgs.size(); i++)

{

//Output key/value pairs

System.*out*.println("Key:"+splitArgs.get(i)[0]+" Value:"+splitArgs.get(i)[1]);

}

}

/\*\*

\* Print the passed-in splitArgs ArrayList with formatting

\* (Used for debugging)

\*

\* Can be called by Main.main()

\*

\* **@param** splitArgs - array of SplitArgs to print

\*/

**public** **void** printSplitArgs(ArrayList<String[]> splitArgs)

{

//Loop through passed-in splitArgs

**for** (**int** i = 0; i < splitArgs.size(); i++)

{

//Output key/value pairs

System.*out*.println("Key:"+splitArgs.get(i)[0]+" Value:"+splitArgs.get(i)[1]);

}

}

/\*\*

\* Makes sure that the passed in array of splitArgs will make

\* a set of valid option objects

\*

\* Called by this.getOptionsFromArgs()

\*

\* **@param** splitArgs - array to check

\* **@return** - checked array

\* **@throws** Exception - thrown if the key in key/value pair is not one of the default accepted

\* keys which are SWITCH\_LABEL, SWITCH\_PROJECT, and SWITCH\_PATH

\*/

**private** ArrayList<String[]> ensureValidOptions(ArrayList<String[]> splitArgs) **throws** Exception

{

//Loop through array of splitArgs

**for**(**int** i = 0; i < splitArgs.size(); i++)

{

//If key is invalid...

**if**(!isValidKey(splitArgs.get(i)[0]))

{

//Yell at user

System.*out*.println(splitArgs.get(i)[0]);

**throw** **new** Exception("Invalid key/value pair! " +

"Did you enter your arguments correctly?");

//System.exit() is called in this.getOptionsFromArgs()

}

//If key is label

**if**(splitArgs.get(i)[0].equals(*SWITCH\_LABEL*))

{

//change to scope and anyrevlabellike:

//(correct integrity command syntax)

splitArgs.get(i)[0] = "scope";

splitArgs.get(i)[1] = "anyrevlabellike:"+splitArgs.get(i)[1];

}

//If the argument is path...

**if**(splitArgs.get(i)[0].equals(*SWITCH\_PATH*))

{

//Assign path to instance variable so that it can be returned to the main program

//(Because path is a selection not an Option object)

path = splitArgs.get(i)[1];

}

}

**return** splitArgs;

}

/\*\*

\* Checks to see if the incoming string is a valid key

\*

\* Called by this.ensureValidOptions()

\*

\* **@param** string - key to be checked

\* **@return** - true if key is one of the default accepted

\* keys which are SWITCH\_LABEL, SWITCH\_PROJECT, and SWITCH\_PATH; false otherwise

\*/

**private** **boolean** isValidKey(String string) {

//Better way to write this logic

**if**(string.equals(*SWITCH\_LABEL*) || string.equals(*SWITCH\_PROJECT*)|| string.equals(*SWITCH\_PATH*)) **return** **true**;

**else** **return** **false**;

}

/\*\*

\* Overarching method in this class - calls other methods and outputs Integrity OptionList

\*

\* Called by Main.main()

\*

\* **@return** - OptionList - Integrity OptionList

\*/

**public** OptionList getOptionsFromArgs()

{

//Creates splitArgs from Args

//(splitArgs are just Args that are split into key/value pairs)

ArrayList<String[]> argPairs = splitListIntoArgs();

//The following may cause an exception!

**try** {

//Ensure that the splitArgs will make a set of valid Option objects

argPairs = ensureValidOptions(argPairs);

} **catch** (Exception e) {

//If the splitArgs don't make valid objects...

System.*exit*(1); // Exit code 1 = syntax error

e.printStackTrace();

}

//Return coveted prize - the OptionList!!!

**return** createOptionList(argPairs);

}

/\*\*

\* Takes a list of valid splitArgs and turns them into an Integrity OptionList

\*

\* Called by this.getOptionsFromArgs()

\*

\* **@param** argPairs - array of valid splitArgs

\* **@return** - Integrity OptionList

\*/

**private** OptionList createOptionList(ArrayList<String[]> argPairs) {

//Create new OptionList

OptionList ol = **new** OptionList();

//Loop through array of splitArgs

**for**(**int** i = 0; i < argPairs.size(); i++)

{

//If the key/value pair is not the path...

**if**(!argPairs.get(i)[0].equals(*SWITCH\_PATH*))

{

//Create new Integrity Option with arg pair at index "i"

Option o = **new** Option(argPairs.get(i)[0], argPairs.get(i)[1]);

//Add Integrity Option to Integrity OptionList

ol.add(o);

}

}

//Return the coveted prize - the OptionList!!!

**return** ol;

}

/\*\*

\* Creates ArrayList of splitArgs from ArrayList of args

\*

\* Called by this.getOptionsFromArgs()

\*

\* **@return** - ArrayList of splitArgs

\*/

**private** ArrayList<String[]> splitListIntoArgs()

{

//Create new ArrayList of splitArgs

splitArgs = **new** ArrayList<String[]>();

//Loop through args array

**for** (**int** i = 0; i < args.size(); i++)

{

//If it starts with a -- it must be a key/value pair!

**if**(args.get(i).startsWith("--"))

{

//Add the key/value pair to the list of splitArgs

splitArgs.add(**new** String[]{args.get(i).substring(2),args.get(i+1)});

}

}

//Return the ArrayList of splitArgs

**return** splitArgs;

}

/\*\*

\* Returns the value of path

\* (because path is a selection and not an Option object)

\*

\* Called by Main.main()

\*

\* **@return** - path

\*/

**public** String getPath()

{

**return** path;

}

}

## CommandMan.java

**package** com.mk347.CreateSandbox;

**import** java.io.IOException;

**import** java.util.Iterator;

**import** com.mks.api.CmdRunner;

**import** com.mks.api.Command;

**import** com.mks.api.IntegrationPoint;

**import** com.mks.api.IntegrationPointFactory;

**import** com.mks.api.Session;

**import** com.mks.api.response.APIException;

**import** com.mks.api.response.Field;

**import** com.mks.api.response.Response;

**import** com.mks.api.response.WorkItem;

**import** com.mks.api.response.WorkItemIterator;

/\*\*

\* Command man helps to manage commands and connect to the server.

\* It makes everything with the Java API for PTC Integrity easier to work with

\*

\* **@author** MK347, Connor Boyle

\* **@version** 10-6-2014

\*

\*/

**public** **class** CommandMan {

**private** String username;

**private** String password;

**private** CmdRunner cr = **null**;

**private** Session session = **null**;

/\*\*

\* Connects to the Integration Point and creates Session and CmdRunner

\*

\* Called by Main.main()

\*

\* **@param** hostname - hostname for integrity server

\* **@param** port - port of integrity server

\* **@param** username - username used to connect to server

\* **@param** password - password used to connect to server

\* **@return** - true if connected; false if error

\*/

**public** **boolean** connect(String hostname, **int** port)

{

//Should be changed to "server user" or "api" account

**this**.username = "username";

**this**.password = "password";

//The following creates the integration point, session, and command runner

**try** {

**boolean** useClientIP = **true**;

**if** (useClientIP) {

// Client Integration Point

IntegrationPoint ip = IntegrationPointFactory.*getInstance*().createLocalIntegrationPoint(4, 12);

ip.setAutoStartIntegrityClient(**true**);

session = ip.getCommonSession(username, password);

session.setAutoReconnect(**true**);

session.setDefaultHostname(hostname);

session.setDefaultPort(port);

session.setDefaultUsername(username);

session.setDefaultPassword(password);

} **else** {

// Server Integration Point

session = IntegrationPointFactory.*getInstance*().createIntegrationPoint(hostname, port, 4, 12).createSession(username, password);

session.setAutoReconnect(**true**);

}

//Command runner

cr = session.createCmdRunner();

}

**catch** (APIException apie) {

System.*out*.println("Error occurred during initialization: "

+ apie.getMessage());

apie.printStackTrace();

//System.exit() is called by Main.main() if exception is thrown

**return** **false**;

}

**return** **true**;

}

/\*\*

\* Executes the passed-in command and returns the response object

\*

\* Called by Main.main()

\*

\* **@param** cmd - command object to be executed

\* **@return** - Response to be parsed by this.printResponse()

\*/

**public** Response execute(Command cmd)

{

**try** {

**return** cr.execute(cmd);

} **catch** (APIException e) {

System.*out*.println(e.getMessage());

e.printStackTrace();

**return** **null**;

}

}

/\*\*

\* Prints the Response work items to the console

\*

\* Called by Main.main()

\*

\* **@param** response - from the executed command

\*/

**public** **void** printResponse(Response response)

{

**try**{

**if** (response != **null**) {

System.*out*.println("Response not null");

WorkItemIterator wii = response.getWorkItems();

**while** (wii.hasNext()) {

WorkItem wi = wii.next();

@SuppressWarnings("unchecked")

Iterator<Field> iterator = wi.getFields();

**while** (iterator.hasNext()) {

Field field = iterator.next();

System.*out*.println(field.getName() + " : "

+ field.getValueAsString());

}

}

}

}**catch**(APIException e)

{

System.*out*.println(e.getMessage());

e.printStackTrace();

}

}

/\*\*

\* Closes the server connection

\*

\* Called by Main.main()

\*/

**public** **void** close()

{

**try** {

session.release();

} **catch** (IOException e) {

e.printStackTrace();

} **catch** (APIException e) {

e.printStackTrace();

}

}

}

## Main.java – (CreateSandbox.jar)

**package** com.mk347.CreateSandbox;

**import** com.mk347.jarArguments.ArgParser;

**import** com.mks.api.Command;

**import** com.mks.api.Option;

**import** com.mks.api.OptionList;

**public** **class** Main {

**public** **static** **void** main(String[] args)

{

ArgParser ap = **new** ArgParser(args);

OptionList ol = ap.getOptionsFromArgs();

//Some of the following code may throw an exeption

**try**{

//Command manager - helps manage the server connection

CommandMan man = **new** CommandMan();

man.connect("integrity.cummins.com", 7002);//uses username and password defined in CommandMan

//Create Sandbox

Command cmd = **new** Command(Command.*SI*, "createsandbox");//si createsandbox

cmd.addSelection(ap.getPath());//with path

ol.add(**new** Option("recurse"));//with recursion

cmd.setOptionList(ol);//with other options

//Execute command

man.execute(cmd);

//Close the Connection

man.close();

System.*exit*(0);// Exit code 0 = all good!

}**catch**(Exception e){

System.*exit*(2);//Exit code 2 = connection error

}

}

}