Oracle® Fusion Middleware

Understanding OIG Configuration Utility

12c (12.2.1.4.0)
E98775-02
October 2019

Oracle Fusion Middleware Understanding OIG Configuration Utility, 12c (12.2.1.4.0)

E98775-02

Primary Author: Oracle Corporation

Contents

1

Introduction and Roadmap

Making manual configuration changes to a Oracle WebLogic Server using the UI can require multiple steps, and it can be difficult to verify that all of the necessary changes were made correctly. It's easy for input errors to occur, leading to incorrect system configurations, and the difficult task of determining which settings need to be adjusted to get the system to work properly. OIG Configuration Utility provides the ability to automate and verify configuration changes, and to restore a system to a known configuration.

About OIG Configuration Utility

The OIG Configuration Utility is designed to simplify configuration tasks. Rather than the manual process of changing dozens of values in the WebLogic Server UI, the configuration utility enables you to perform a complete configuration update in one step by applying a set of configuration values to the system. The utility makes a backup of the current system settings before applying any changes. If a problem arises due to a configuration change, you can use the utility to restore to a previous configuration.

Which attributes are modified in WebLogic Server are determined by the gestures and operations that are executed by the OIG Configuration Utility. A gesture consists of one or more operations, where an operation performs a set of configurations within the gesture's domain. For example, the "tune" gesture provides three operations to tune the performance of three aspects of WebLogic Server: database, JVM, and OIM. To tune the JVM performance of a WebLogic Server, you set the values related to JVM performance in the attributes file and call the utility, executing the JVM tuning operation from the tune gesture. You can implement your own gestures and use them with the utility; see Adding a Gesture for more information.

Document Scope and Audience

This document describes the Oracle Identity Governance (OIG) tool. It explains how to use the OIG utility command line interface to configure system settings, either in interactive mode with user input, or silent mode. It also explains how you can extend the framework behind the OIG utility to create custom configuration gestures that can be used by the OIG utility to perform configuration operations.

This document is written for WebLogic Server administrators and operators who configure and maintain WebLogic Server systems and provide support to users. It is assumed that readers are familiar with configuring and administrating WebLogic, Java, MBeans, JavaScript Object Notation (JSON), and the operating systems and platforms where WebLogic Server is installed.

Guide to this Document

This document is organized as follows:

- This chapter introduces the organization of this guide and lists related documentation.
- **About OIG Configuration Utility** describes the contents and structure of OIG Configuration Utility, including configuration and input files.
- **Using OIG Configuration Utility** describes how to use OIG Configuration Utility, the differences between silent and interactive modes, creating baselines and restoring a configuration from a baseline, gestures, and reports.
- Adding a Gesture describes how to extend OIG Configuration Utility's framework to provide additional gestures and operations to the utility.

Related Documentation

For information about tuning Oracle WebLogic Server, see:

- Top Tuning Recommendations for WebLogic Server in *Fusion Middleware Tuning Performance of Oracle WebLogic Server* for recommendations on tuning WebLogic Server and its application.
- Understanding WebLogic Server MBeans in *Fusion Middleware Developing Custom Management Utilities Using JMX for Oracle WebLogic Server* for information on using MBeans to configure WebLogic Server.
- Monitor and Tune documentation for Oracle WebLogic Server, which provides additional resources for monitoring, tuning, administering, and troubleshooting WebLogic Server.

2

About OIG Configuration Utility

The OIG Configuration Utility is designed to simplify configuration tasks. Rather than the manual process of changing dozens of values in the WebLogic Server UI, the configuration utility enables you to perform a complete configuration update in one step by applying a set of configuration values to the system. The utility makes a backup of the current system settings before applying any changes. If a problem arises due to a configuration change, you can use the utility to restore to a previous configuration.

Which attributes are modified in WebLogic Server are determined by the gestures and operations that are executed by the OIG Configuration Utility. A gesture consists of one or more operations, where an operation performs a set of configurations within the gesture's domain. For example, the "tune" gesture provides three operations to tune the performance of three aspects of WebLogic Server: database, JVM, and OIM. To tune the JVM performance of a WebLogic Server, you set the values related to JVM performance in the attributes file and call the utility, executing the JVM tuning operation from the tune gesture. You can implement your own gestures and use them with the utility; see Adding a Gesture for more information.

- Utility Structure and Contents
  The installation directory of the OIG Configuration Utility contains the utility itself, `oig-config-utility.sh`. This directory also contains the following directories used by the utility:
- Config Files
  The `config/` directory contains the files `oig-utility-config.json`, `oig-configuration-attributes.json`, and `inputs.properties`. These files are used to configure the utility, set the values to which the utility will configure WebLogic Server attributes, and pass inputs to the utility, respectively.

Utility Structure and Contents

The installation directory of the OIG Configuration Utility contains the utility itself, `oig-config-utility.sh`. This directory also contains the following directories used by the utility:

Directory	Description
`backup/`	Contains copies of the latest state, which are implicitly created after each operation.
`baseline/`	Contains baseline images created by the utility. These files contain the current values of the system attributes; that is, the current state of the system.
`config/`	Contains input files used by the utility (`inputs.properties`, `oig-configuration-attributes.json`, and `oig-utility-config.json`).

Directory	Description
`current_state/`	Contains files that store the current state of the system after a gesture operation is performed.
`lib/`	Contains the libraries used by the utility, including gesture implementation libraries, the utility library, and the service provider (SPI) library that is the backbone of the utility's framework.
`logs/`	Contains logs created by the utility.
`reports/`	Contains reports showing the pre- and post-operation values of attributes affected by operations executed by the utility.
`resource/`	Contains resource files used by the utility.

The utility is installed with the `config/`, `lib/`, and `resource/` directories already in place; the other directories are created at runtime by the utility as needed.

Every time one or more operations are executed, the utility creates a file with the current state of the attributes being modified and stores it in `backup/`. After the operation is executed, the final state is stored in the `current_state/` directory. Every time any operation is performed, files are created in both the `backup/` and `current_state/` directories.

A baseline contains the attributes for all the enabled gestures and their operations at the time the baseline is created. You can use any backup, baseline, or current state file to restore the state of the system. By default, these files are stored in the `backup/`, `baseline/`, and `current_state/` directories, respectively.

Config Files

The `config/` directory contains the files `oig-utility-config.json`, `oig-configuration-attributes.json`, and `inputs.properties`. These files are used to configure the utility, set the values to which the utility will configure WebLogic Server attributes, and pass inputs to the utility, respectively.

- oig-utility-config.json
  The file `oig-utility-config.json` is used to configure the OIG Configuration Utility. Its contents define a set of directories used by the utility, the inputs and attributes files to use, which gestures are available, and which gestures are enabled.
- oig-configuration-attributes.json
  The `oig-configuration-attributes.json` file contains the schema used by the utility to configure a WebLogic Server.
- inputs.properties
  The `inputs.properties` file contains an entry for each possible required input for all supported gestures and their operations. These are the inputs that you can specify on the command line when calling the utility, for example, WebLogic Server user name and password.

oig-utility-config.json

The file `oig-utility-config.json` is used to configure the OIG Configuration Utility. Its contents define a set of directories used by the utility, the inputs and attributes files to use, which gestures are available, and which gestures are enabled.

There are five directories whose locations you can configure for the OIG Configuration Utility. You can configure their values in the configuration file, or you can configure them using system/environment variables. In the case where a directory is defined both in the configuration file and in an environment variable, the value of the environment variable is used. The following table lists the directory variables in the configuration file, their corresponding environment variable names, and their default values:

Configuration File Variable Name	System/Environment Variable Name	Default Value
`backupDir`	`BACKUP_DIR`	backup
`baseLineDir`	`BASELINE_DIR`	baseline
`currentStateDir`	`CURRENT_STATE_DIR`	current_state
`logsDir`	`LOGS_DIR`	logs
`reportDir`	`REPORTS_DIR`	reports

The values of the attributes file is set by the `attributesFile` entry. Its default value is `config/oig-configuration-attributes.json`.

The `gesturesConfig` structure array contains information about each gesture that OIG Configuration Utility can use. Each entry contains a name for the gesture, the gesture class name, and whether the gesture is enabled. For example, the following is the entry for the tune gesture included with the utility:

```
{
    "name": "Tuning Gesture",
    "gestureClass": "com.oracle.oig.gestures.TuningGesture",
    "isEnabled": "true"
}
```

To disable a gesture from being used, set that gesture's `isEnabled` value to false.

The utility help provides a list of available gestures to the user. This list includes each gesture from the `oig-utility-config.json` file whose `isEnabled` value is true.

oig-configuration-attributes.json

The `oig-configuration-attributes.json` file contains the schema used by the utility to configure a WebLogic Server.

To set the values the utility will use, open `oig-configuration-attributes.json`, and edit the values of the attributes used by the gesture operations you plan to execute. The attributes used by a gesture operation are in the section of the schema labeled with that operation's name (for example, "Tune OIM Operation").

For example, suppose you want to set the JVM minimum and maximum heap space values to 3072 and 4096 megabytes, respectively. Edit the values in the "Tune JVM Operation" section of the `oig-configuration-attributes.json` file to the following:

```
"Tune JVM Operation":
{
  "com.oracle.oig.config.attributes.JVMTuneAttributes":
  {
    "minHeapSpace": "3072m",
    "maxHeapSpace": "4096m"
  }
}
```

To configure the system with these values, save the attributes file, then run the utility and execute the tune JVM operation.

inputs.properties

The `inputs.properties` file contains an entry for each possible required input for all supported gestures and their operations. These are the inputs that you can specify on the command line when calling the utility, for example, WebLogic Server user name and password.

If you wish to supply a value without specifying it each time you call the utility, you can enter it into `inputs.properties`. For example, if your WebLogic Server host and port never change, enter those values in the file for their respective properties `weblogicHost` and `weblogicPort`. When you call the utility, the values from `inputs.properties` are used, and you do not need to enter them on the command line.

You can choose which values to supply via `inputs.properties`, if any. For example, you might choose to supply values for all of the required inputs except for passwords via `inputs.properties`. You'd then have to supply the passwords when calling the utility.

When you run the utility in interactive mode, you'll be prompted to enter a value for each required input whose value is not defined in `inputs.properties`.

When you run the utility in silent mode, you must provide a value on the command line for each required input that doesn't have a value defined in `inputs.properties`. If you specify a value for an input on the command line that's also defined in inputs.properties, the utility uses the value from the command line.

3

Using OIG Configuration Utility

Oracle WebLogic Server (WLS) has hundreds of configurable values. A specific group of values can affect how a component of the system operates. For example, you can tune database performance by configuring the database settings in WLS.

Oracle provides tuning guides for various components of WLS. Properly tuning a component can consist of performing dozens of steps, with many values to input. It's possible that entering these values manually can result in input errors, or one or more steps may be missed.

oig-configuration-attributes.json helps avoid manual configuration errors by using the values from an attributes file to configure the WLS system. The utility captures the state of the system before any changes are made, and creates a report that shows the system values before and after a gesture was executed. In the case where a set of changes made the system less stable, the utility allows you to restore the system to a previously captured state.

By default, the values to set are stored in the `oig-configuration-attributes.json` file in the `config/` directory. To make configuration changes: edit `oig-configuration-attributes.json`, entering the values you want to use for the attributes, and run the OIG Configuration Utility, specifying the gestures and operations to perform.

- Interactive Mode and Silent Mode
  OIG Configuration Utility can be run in two modes: interactive and silent.
- Baselines and Restoration
  OIG Configuration Utility provides the ability to create baseline images of the system configuration. A baseline contains the system state of all attributes that can be modified by all defined operations at the time the baseline is created. You can use a baseline to restore the system to a previous configuration, setting the system attributes to those stored in the specified baseline.
- Gestures and Operations
  The configuration operations that OIG Configuration Utility can perform are defined and implemented by gestures.
- Reports
  When you perform one or more operations using OIG Configuration Utility in interactive mode, before any changes are made to the system, the utility produces a report showing the initial and final attribute values.

Interactive Mode and Silent Mode

OIG Configuration Utility can be run in two modes: interactive and silent.

Interactive mode prompts the user to enter details required by the operation being performed, such as user names and passwords.

Silent mode requires that all necessary input be provided to the utility before being executed—no user interaction is required. This mode is ideal to be paired with a script,

which can enable a WebLogic Server admin to offer a configuration service to an end user via a button or link that they press, which in turn performs the configuration.

In both silent and interactive modes, you can specify values for inputs used by one or more operations in the `inputs.properties` file. In interactive mode, as the operations are executed, the user is prompted to enter any required input values that are not defined in the inputs file. In silent mode, if any required inputs are missing when the utility is called, the call fails.

- Interactive Mode
  Interactive mode prompts you to enter values for require inputs as the utility executes one or more operations. This mode can be convenient for an administrator working on configuring a system from a command prompt.
- Silent Mode
  Silent mode requires that you provide values for all the required inputs of the gesture operations that you want to execute when you launch the utility. If any inputs are missing, the call fails—the utility will not prompt you to enter any missing values. This mode is useful for providing the ability for an end-user to perform a system configuration task. For example, you might create a webpage that includes a button that, when clicked, launches the OIG Configuration Utility in silent mode and performs the desired configuration without the need for user input.

Interactive Mode

Interactive mode prompts you to enter values for require inputs as the utility executes one or more operations. This mode can be convenient for an administrator working on configuring a system from a command prompt.

Invoking OIG Configuration Utility in Interactive Mode

To invoke OIG Configuration Utility in interactive mode, execute the utility without any arguments, using the appropriate script for your environment.

On UNIX, Linux, and Mac OSX systems, run the utility in interactive mode by running the script:

```
cd $ORACLE_HOME/idm/server/bin/OIGConfigUtility
./oig-config-utility.sh
```

On Windows systems, run the utility in interactive mode by running the batch file:

```
cd $ORACLE_HOME\idm\server\bin\OIGConfigUtility
oig-config-utility.bat
```

For simplicity, other examples in this document demonstrate using the shell script to invoke the utility. On a Windows system, call the batch file, instead.

The basic utility usage information is displayed, and the interactive utility prompt is displayed as:

```
$oig>
```

From this prompt, you can enter commands. For example, to display gesture-specific help for the tune gesture, enter the following:

```
$oig> tune –h
```

The utility displays the help specific to the tune gesture. You can display the gesture-specific help for any gesture by entering the command *gestureName* `-h`. The names of the available gestures are listed in the Configured Gestures section of the utility help.

To run a gesture operation, enter the gesture name and operation, based on the gesture-specific help. For example, you can start the database tuning operation by entering:

```
$oig> tune –database
```

As the utility runs the operation, it retrieves any values used by the operation from `inputs.properties`. For each input where a value is not defined, the utility prompts you to enter a value.

Interactive Mode Options

You can use the following options in Interactive mode.

Option	Description
`-a|-all`	Use this option to execute all operations defined for all enabled gestures.
`-b|-baseline`	Use this option to create a baseline of the current system state for all enabled gestures.
`-rpt|-report`	Use this option to generate a pre-report for all enabled gestures. This report shows the pre- and post-operation values for all enabled gestures, but does not make any changes to the system.
`-r|-restore`	Use this option to restore the system from a baseline file. This restores the system configuration to that stored in the baseline file. This operation requires a valid file path as input.
`-h|-help`	Use this option to display help for the OIG Configuration Utility. This includes general instructions for using the utility, as well as which gestures are enabled.
gestureName `–h`	Use this option to display gesture-specific help for the specified gesture.

Exiting OIG Configuration Utility in Interactive Mode

To exit OIG Configuration Utility, enter `quit` or `q!` at the prompt. For example:

```
$oig> quit
```

or

```
$oig> q!
```

Silent Mode

Silent mode requires that you provide values for all the required inputs of the gesture operations that you want to execute when you launch the utility. If any inputs are missing, the call fails—the utility will not prompt you to enter any missing values. This mode is useful for providing the ability for an end-user to perform a system configuration task. For example, you might create a webpage that includes a button that, when clicked, launches the OIG Configuration Utility in silent mode and performs the desired configuration without the need for user input.

Invoking OIG Configuration Utility in Silent Mode

To invoke OIG Configuration Utility in silent mode, execute the utility with the `-S` switch, using the appropriate script for your environment.

On UNIX, Linux, and Mac OSX systems, run the utility in silent mode by running the script:

```
cd $ORACLE_HOME/idm/server/bin/OIGConfigUtility
./oig-config-utility.sh -S
```

On Windows systems, run the utility in silent mode by running the batch file:

```
cd $ORACLE_HOME\idm\server\bin\OIGConfigUtility
oig-config-utility.bat -S
```

For simplicity, other examples in this document demonstrate using the shell script to invoke the utility. On a Windows system, call the batch file, instead.

Running the utility in silent mode with no arguments displays the utility help text. This provides general instructions for using the utility, as well as which gestures are enabled.

When invoking the utility, you must specify which gestures and operations to perform, and provide values for all of the required inputs. These values can be provided as arguments on the command line, as part of `inputs.properties`, or a combination of both. If any input value is missing or invalid, the utility will exit without making any changes.

You can view the gesture-specific help for any available gesture listed in the utility's help text by entering the command:

```
./oig-config-utility.sh –S gestureName -h
```

For example, use the following command to display the gesture-specific help for the tune gesture:

```
./oig-config-utility.sh –S tune -h
```

Baselines and Restoration

OIG Configuration Utility provides the ability to create baseline images of the system configuration. A baseline contains the system state of all attributes that can be modified by all defined operations at the time the baseline is created. You can use a baseline to restore the system to a previous configuration, setting the system attributes to those stored in the specified baseline.

You must create a baseline the first time you use OIG Configuration Utility. This creates a backup of your current system, ensuring that you can use the utility to restore your system to this known state, should you make a change that you want to revert.

OIG Configuration Utility includes a restore function that lets you revert the system configuration to that stored in a baseline file.

- Creating a Baseline
  You can use OIG Configuration Utility to create a baseline of your system in both the interactive and silent modes.
- Restoring System from a Baseline File
  You can use the restore feature of OIG Configuration Utility to configure the WebLogic Server with the attribute values from a baseline, backup, or current state file. By default, the files in the `backup/`, `baseline/`, and `current_state/` directories can all be used to restore a system. The restore option is performed by passing `-restore` or `-r` to the utility, followed by the baseline file to use.

Creating a Baseline

You can use OIG Configuration Utility to create a baseline of your system in both the interactive and silent modes.

In interactive mode, start the utility and enter the command `-b` or `-baseline`. The utility will prompt you to enter each value that the operation requires that isn't defined in `inputs.properties`.

In silent mode, start the utility in silent mode with the `-b` or `-baseline` switch. For example:

```
./oig-config-utility.sh –S -baseline
```

Values that the baseline operation requires must be provided via `inputs.properties` or as flags passed to the utility (for example, `-weblogicPort 1234`). The baseline operation fails if any inputs are missing. The baseline files that you create are stored in the `baseline/` directory, by default.

In addition to baseline files that you create explicitly, the utility creates a backup of system attributes associated with an operation before attempting the operation (for example, a copy of the database attributes before attempting a database tune operation), and similarly a backup of the current state of the system after an operation has been performed. By default, these files are stored in the `backup/` and `current_state/` directories, respectively.

Restoring System from a Baseline File

You can use the restore feature of OIG Configuration Utility to configure the WebLogic Server with the attribute values from a baseline, backup, or current state file. By default, the files in the `backup/`, `baseline/`, and `current_state/` directories can all be used to restore a system. The restore option is performed by passing `-restore` or `-r` to the utility, followed by the baseline file to use.

For example, suppose you perform a baseline of your system on July 1, 2018, creating the file `oig-baseline_2018-07-01_17_14_19.json` in the `baseline/` directory. You later make some configuration changes to your database and JVM tuning that have negatively affected your system's performance. You can revert to the old settings by running this command:

```
./oig-config-utility.sh –S –restore baseline/oig-
baseline_2018-07-01_17_14_19.json
```

Values that the restore operation requires must be provided via `inputs.properties` or as flags passed to the utility (for example, `-weblogicPort 1234`). The restore operation fails if any inputs are missing.

Gestures and Operations

The configuration operations that OIG Configuration Utility can perform are defined and implemented by gestures.

OIG Configuration Utility displays which gestures are available in the utility help. This information consists of a short name for the gesture and a description of the gesture. For example, help displays the following information for the tune gesture:

```
tune : Tune different components.
    Use 'tune -h' to display gesture-specific help
```

Here, *tune*' is the referred to as the short name of the gesture.

To display additional information about a gesture, including which operations the gesture can perform, invoke the utility and pass the gesture short name followed by `-h`.

To perform a gesture operation using the utility, call the utility specifying the gesture short name and either the operation's short or long CLI flag. The format of this call is:

`./oig-config-utility.sh` *`gestureName –<opr_long_flag | opr_short_flag>`*

For example, the CLI short and long flags (`-database` and `–d`) for the database tuning operation are defined in `TuneDBOperation.java`. To perform a JVM tuning operation in interactive mode, you invoke OIG Configuration Utility by entering one of the following commands:

```
./oig-config-utility.sh tune –database
```

using the CLI long flag, or:

```
./oig-config-utility.sh tune –d
```

using the CLI short flag.

You can create a gesture by extending the AbstractGesture class from the OIG Gesture framework, and operations for that gesture by extending AbstractGestureOperation. The tune sample gesture implementation is included with OIG Configuration Utility in the `TuningGesture.java` source file. See Adding a Gesture for more information about creating gestures.

To use a gesture that you create, copy your gesture jar file to the lib directory, and add your gesture to the `gesturesConfig` section of `oig-utility-config.json`. Provided the gesture is valid, it's shown as an available gesture the next time you launch the utility.

The operations defined by these gestures can update the WebLogic Server configuration by applying the values from `oig-configuration-attributes.json` to the system.

Reports

When you perform one or more operations using OIG Configuration Utility in interactive mode, before any changes are made to the system, the utility produces a report showing the initial and final attribute values.

In both interactive and silent modes, you can direct the utility to produce a pre-execution report for the gesture operations you pass to the utility. This will produce a report with the initial and final attribute values, based on the operations that you selected. No changes will be made to the system.

To create a pre-report, use the `–r` or `–report` options. For example, in silent mode, you might want a pre-report for the tune database operation, in which case you'd call the utility similar to:

```
./oig-config-utility.sh –S –rpt tune –d [inputs]
```

Whether generated automatically in interactive mode, or explicitly using the `–r` or `-report` options, reports are stored in the `reports/` directory by default. The reports are in HTML and can be viewed in any web browser.

4
Adding a Gesture

You can add your own gestures and operations to OIG Configuration Utility. The utility includes GesturesSPI, which is a Service Provider Interface that you can use to implement new gestures. To do so, create your own gesture by extending the `AbstractGesture` class. Each gesture can define one or more operations. Add support for operations by extending `AbstractGestureOperation` for each operation. You can use MBeans operations to make changes to WebLogic Server. After your code is complete, copy your gesture class to the `lib/` directory, add the gesture to the `gesturesConfig` section of `oig-utility-config.json` and set its `isEnabled` value to `true`. Provided the gesture is valid, when you launch the utility, your gesture appears in the "Configured Gestures" section of the utility help.

- Extending AbstractGesture
  To create a gesture, you must extend the `AbstractGesture` class, implementing abstract methods to add gesture-specific information. The `TuningGesture.java` source code provides an example of how to extend `AbstractGesture` to implement a gesture. The following sections provide some additional details on creating a gesture class.
- Extending AbstractGestureOperation
  To create an operation, you must extend the `AbstractGestureOperation` class, implementing abstract methods to provide operation-specific information. For example, the tune gesture provides a "tune DB" operation, which is implemented by `TuneDBOperation.java`. The code for the operations that come with OIG Configuration Utility demonstrate how to extend `AbstractGestureOperation` to implement an operation. Each gesture can define one or more operations. The following sections offer additional information about implementing operations to work in the OIG Configuration Utility framework:
- Deploying a Gesture

Extending AbstractGesture

To create a gesture, you must extend the `AbstractGesture` class, implementing abstract methods to add gesture-specific information. The `TuningGesture.java` source code provides an example of how to extend `AbstractGesture` to implement a gesture. The following sections provide some additional details on creating a gesture class.

- Overriding Gesture Methods
  You must override the following methods in your gesture class for your gesture to be usable in OIG Configuration Utility:

Overriding Gesture Methods

You must override the following methods in your gesture class for your gesture to be usable in OIG Configuration Utility:

Method	Description	Sample Return Value
`public String getDescription()`	Returns a string that contains a description for the gesture.	`Tune different components.`
`public List<GestureOperation> getOperations()`	Returns a list of operations supported by this gesture. Operations are created by extending the `AbstractGestureOperation` class.	
`public String getShortName()`	Returns the short name of the gesture.	`tune`

The utility automatically generates help for enabled gestures by using the values returned by these methods. You can enable gestures in `oig-utility-config.json`.

Extending AbstractGestureOperation

To create an operation, you must extend the `AbstractGestureOperation` class, implementing abstract methods to provide operation-specific information. For example, the tune gesture provides a "tune DB" operation, which is implemented by `TuneDBOperation.java`. The code for the operations that come with OIG Configuration Utility demonstrate how to extend `AbstractGestureOperation` to implement an operation. Each gesture can define one or more operations. The following sections offer additional information about implementing operations to work in the OIG Configuration Utility framework:

- Overriding GestureOperation Methods
- Implementing Attributes
- Defining Options
- Defining Required Parameters

Overriding GestureOperation Methods

You must override the following methods in your gesture operation class for your gesture to be usable in OIG Configuration Utility:

Method	Description	Sample Return Value
`public int execute()`	This method defines the execution logic of the operation. The framework calls this method when it receives a request to execute this operation.	
`public GestureOperationReport generateReport()`	Generates a report for the attributes affected by this operation.	
`public String getDescription()`	This method returns a description of the operation.	`Tune Database Operation`

Method	Description	Sample Return Value
`public String getLongCliFlag()`	This method returns the long CLI flag that can be used to execute this operation.	`-database`
`public String getShortCliFlag()`	This method returns the short CLI flag that can be used to execute this operation.	`-d`
`public Class getOperationAttributeClass()`	This method returns the attributes class used by this operation to define attributes.	`DBTuneAttributes.class`
`public List<Option> getOptions()`	This method returns a list of supported options that can be specified on the command line for this operation.	
`public List<RequiredParam> getRequiredParams()`	This method returns the list of parameters that are require to execute this operation. They can be defined in `inputs.properties` or provided via the CLI.	
`public List<String> getWarnings()`	This method returns a list of warnings that you want to display in an HTML report and in the logs. These are generic warnings, such as any manual steps required after operation execution.	
`public void validate()`	This method contains any validation that you want to perform for your gesture operation.	

Implementing Attributes

For each operation, you must create a class that implements the `com.oracle.oig.gesture.spi.Attributes` marker interface. This interface is defined in `Attributes.java` as part of the `GesturesSPI` framework. Each class that implements `Attributes` contains the attributes that the operation modifies.

An example implementation of `Attributes` is `DBTuneAttributes.java`, which models all the attributes used by the database tuning operation. The framework uses your implementation to create objects to use when executing the operation associated with that set of attributes.

Update oig-utility-config.json

After creating an attributes model for an operation, you must update `oig-utility-config.json` by adding a section with your operation's attributes and their values. For example, the attributes for the DB Tuning operation, which correspond to the

DBTuneAttributes.java implementation of the Attributes marker interface, are included in oig-utility-config.json as follows:

```
"Tune Database Operation":
{
  "com.oracle.oig.attributes.DBTuneAttributes":
  {
    "queries":
    [
      "java.util.ArrayList",
      [
        {
          "com.oracle.oig.attributes.QueryAttributes":
          {
            "parameter": "DB_KEEP_CACHE_SIZE=200M"
          }
        },

        ...

        {
          "com.oracle.oig.attributes.StoredProcedures":
          {
            "parameter": "OPEN_CURSORS=600"
          }
        }
      ]
    ]
  }
},
```

Users of the OIG Configuration Utility can modify the values of the attributes of the structure that you add for your operation to set new values in the configuration.

Defining Options

An operation has a set of associated options, each of which is defined by an Option object.

Each operation defines an array to hold its options. TuneDBOperation defines its array as follows:

```
private static final List<Option> options = new ArrayList<Option>();
```

An Option consists of six values:

- description - a description of the operation, used in the utility help.
- hasValue - a flag that indicates whether the option has an associated value. If set to true, the framework expects a value for this option, and validates whether a value was provided when the operation is executed.
- isOptional - a flag that indicates whether the option is optional. Set to true if optional; false, if the option is required.

- `longCliFlag` - the option's CLI long flag.
- `shortCliFlag` - the option's CLI short flag.
- `validValues` - the set of valid values for this option.

The Class Variables demonstrates how to define the `longCliFlag` and `shortCliFlag` members of an `Option`. The values for `hasValue` and `isOptional` are defined when the `Option` is created.

Adding Options to an Operation

You add options to an operation by implementing the `getOptions()` method. In it, you create the options for the operation, and add them to the `options` list.

When creating an `Option`, there are two available constructors—one where the option defaults to being optional (no value is passed for `isOptional`), and a second where you must specify whether the option is optional. For example, `TuneDBOperation` uses the first method, as demonstrated in the following:

```
  @Override
  public List<Option> getOptions() throws OIGException {
    if (options == null || options.isEmpty()) {
      final Option dbOption = new Option(LONG_CLI_FLAG, SHORT_CLI_FLAG,
"Tune DB params", false);
      options.add(dbOption);
    }
    return options;
  }
```

The `getOptions()` method is also where you specify the valid values for an option. For example, the following `getOptions()` method adds the option to enable some function "XYZ":

```
  @Override
  public List<Option> getOptions() throws OIGException {
    if (options == null || options.isEmpty()) {
      final Option enableOption =
          new Option(LONG_CLI_FLAG, SHORT_CLI_FLAG, "Enable XYZ for OIM
servers", false);

      for (final PostInstallConstants.SupportedComponents s :
PostInstallConstants.SupportedComponents
          .values()) {

        enableOption.addValidValue(s.getSupportedComponent());
      }

      options.add(enableOption);
    }
    return options;
  }
```

The above example demonstrates adding the valid values to an `Option` (here, `enableOption`), before adding the `Option` to the operation's options list. The

framework checks input against these values, and returns an error if an invalid value is provided.

Defining Required Parameters

Your operation may require certain inputs. For example, an operation might require a WebLogic Server username and password. These types of inputs are referred to as required parameters.

Each operation defines a class variable to store its required parameters as follows:

```
private static final List<RequiredParam> requiredParams = new
LinkedList<RequiredParam>();
```

You define a required input by creating a `RequiredParam` object. Each `RequiredParam` has five attributes: the name of the parameter, the text to display when prompting a user to enter a value for the parameter in interactive mode, a description of the parameter, whether the input is secure and needs to be masked in interactive mode, and whether the parameter is optional. If the parameter is secure (set to true), then input entered for that value is masked on the command line. Creating a `RequiredParam` without specifying whether it's optional, results in it being required.

You set the required parameters for an operation by overriding `AbstractGestureOperation`'s `getRequiredParams()` method. The following is the implementation from `TuneDBOperation.java`:

```
@Override
public List<RequiredParam> getRequiredParams() throws OIGException {
  if (requiredParams.isEmpty()) {

    final RequiredParam dbSysUrl = new
RequiredParam(DBParams.DB_SYS_URL.paramName,
      "Enter Database JDBC URL for sys dba:
Ex:jdbc:oracle:thin:@dbhostname:5521(:orclsid or /serviceName)",
      "Jdbc url is used to connect to the db ", false);

    final RequiredParam dbSysDbaUser = new
RequiredParam(DBParams.DB_SYS_DBA_USER.getParamName(),
      "Enter Database Sys User", "Database Sys User", false);

    final RequiredParam dbSysDbaPassword =
      new RequiredParam(DBParams.DB_SYS_DBA_PASSWORD.getParamName(),
      "Enter Database Sys Password", "Database Sys Password", true);

    final RequiredParam dbOimUrl = new
RequiredParam(DBParams.DB_OIM_URL.paramName,
      "Enter Database JDBC URL for oim user:
Ex:jdbc:oracle:thin:@dbhostname:5521(:orclsid or /serviceName)",
      "Jdbc url is used to connect to the oim user. (Example:
jdbc:oracle:thin:@dbhostname:5521(:orclsid or /serviceName)) ",
      false);

    final RequiredParam dbOimUser = new
RequiredParam(DBParams.DB_OIM_USER.getParamName(),
```

```
      "Enter Database Oim User", "Database Oim User", false);

    final RequiredParam dbOimUserPassword =
      new RequiredParam(DBParams.DB_OIM_USER_PASSWORD.getParamName(),
      "Enter Database Oim Password", "Database Oim Password", true);

    requiredParams.add(dbSysUrl);
    requiredParams.add(dbSysDbaUser);
    requiredParams.add(dbSysDbaPassword);
    requiredParams.add(dbOimUrl);
    requiredParams.add(dbOimUser);
    requiredParams.add(dbOimUserPassword);
  }
  return requiredParams;

}
```

When an operation is executed, the framework checks the inputs provided to the operation, and returns an error if any of the required parameters were not provided.

Deploying a Gesture

After you created a gesture by writing an `AbstractGesture` class and `AbstractGestureOperation` classes for each operation that the gesture supports, you need to perform a few steps to add the gesture to the utility.

After your code is complete, create a JAR for your gesture, copy the JAR to the lib/ directory, add the gesture to the gesturesConfig section of `oig-utility-config.json`, and set its isEnabled value to true.

Build and Copy a JAR

To prepare your gesture for the utility, build your Java code, generate a JAR for your gesture, copy the JAR to the utility's `lib/` directory, and set the JAR class path environment variable, if not already set. You can set the class path in oig-config-utility.sh by appending the path to the line that begins with "`CLASSPATH=`".

Add and Enable the Gesture

The framework can access all of the gesture code that exists in the `lib/` directory. However, for OIG Configuration Utility to offer a gesture to its users, the gesture must be added to the utility configuration.

To add a gesture to the utility configuration, edit `oig-utility-config.json` and add an entry for your gesture to the `gesturesConfig` array. For example, the entry for our sample `MyGestureOperation` gesture would be as follows:

```
  {
    "name": "My Gesture",
    "gestureClass": "com.yourcompany.oig.gestures.MyGesture",
    "isEnabled": "true"
  }
```

To enable your gesture, set the value of `isEnabled` to `true` in its `gesturesConfig` entry. To disable a gesture, set its `isEnabled` value to `false`.