GATE DAEMON REFERENCE

This document describes the "Gate Daemon" service ("gated"), a daemon managed by "systemd"/"systemctl".

Theory of Operation

Overview

GATE ("**G**eneral **A**rchitecture for **T**ext **E**ngineering") is a system written in Java that provides rich and robust text processing capabilities.

The `gated` service is a Java process that provides client access to a single GATE instance running in a single Java VM ("**JVM**").

NOTE: The `gated` service provides access to a single GATE instance on a single **JVM**. The `gated` service has no representation for a "user". It does not support parallel operations. The "GATE Developer" and "GATE Cloud" products from the University of Sheffield are more appropriate for team and parallel use.

The `gated` service has been exercised with clients written in `Python` and `NodeJS`, as well as command-line clients that use `wget` like described in this document.

System Architecture and Components

The components described here assume an Amazon Web Services ("AWS") EC2 instance running a current version of Rocky Linux.

- GATE (Java application)

  GATE itself is an instance of the standard `gate-core` installed and running on a JVM running Java 1.8

- `GateDaemon` (System service)

  `GateDaemon` is a system service named `gated` managed by `systemctl`. It listens on a dedicated TCP port (7499). All interactions with `GateDaemon` are performed by `https` exchanges using the specified port.

  `GateDaemon` is written in Java and directly uses GATE classes, methods, and behavior.

Lifecycle of a typical `gated` session

A client interaction with `gated` has three distinct phases:

1. Setup
2. Use
3. Cleanup

These occur in a single "session".

NOTE: The `gated` service provides no representation or behavior for a session. Each client of `gated` must manage their own session behavior.

During the "Setup" phase, a client loads one or more `plugin` instances.

During the "Use" phase, the client creates or loads needed resources and then performs operations on those resources.

During the "Cleanup" phase, the client invokes housekeeping endpoints that free resources in the backend and that reset the state of the `gate` service to discard any cached entities.

Each endpoint used during "Setup" and "Cleanup" is idempotent. It is therefore good practice to call the "Cleanup" endpoints during "Setup" to handle the unlikely event that any of `gate`, `gated`, or GATE were not properly cleaned up at the end of a prior session.

GATE (core) Resources

This section describes the GATE core resources supported and used by the `gate` service.

`Corpus`

A "Corpus" is a CREOLE resource that aggregates one or more instances of "Document" for use by a "Pipeline"

`Document`

A "Document" is a CREOLE resource that provides access to a specific text document.

`Pipeline`

A `gate` "Pipeline" is a "SerialAnalyserController" in GATE.

From the GATE documentation:

> A [Pipeline] opens each document in the corpus in turn, sets that document as a runtime parameter on each PR, runs all the PRs on the corpus, then closes the document.

`Plugin`

A "Plugin" is:

> a directory or JAR file containing an XML configuration file called creole.xml at its root

The `gate` service assumes the use of the `ANNIE` plugin.

`ProcessingResource`

A "ProcessingResource" is a CREOLE resource associated with a Plugin that provides specific behavior determined by run-time parameters defined for that ProcessingResource.

`gated` Resources

This section enumerates and briefly describes the Java classes that comprise the `gated` system service.

GateServer

The `GateServer` class directs the validation, dispatch, and response for each https interaction.

`GateServer` supports six https endpoints and two development endpoints, for a total of eight. The eight endpoints are as follows:

1. `hello`

   The `hello` development endpoint answers "Hello World" to any request. It is used to confirm the roundtrip path through `gated` itself.

2. `echo`

   The `echo` development endpoint its query string to any request. It is used to confirm the argument handling of `gated` itself.

3. `plugin`

   The `plugin` endpoint is used to load a specified Maven plugin.

4. `cleanup`

   The `cleanup` endpoint is used to empty the several caches of `gated` as well as invoke needed cleanup behavior in GATE.

5. `document`

   The `document` endpoint provides behavior for loading, accessing, and cleaning up instances of GATE `Document`.

6. `corpus`

   The `corpus` endpoint provides behavior for loading, accessing, and cleaning up instances of GATE `Corpus`.

7. `pr`

   The `pr` endpoint provides behavior for loading instances of ProcessingResource.

8. `pipeline`

   The `pipeline` endpoint provides behavior for creating, loading, configuring, running, and storing instances of Pipeline (`ConditionalSerialAnalyserController`).

Each endpoint is supported by a pair of classes -- a handler that extends `GateHandler` and a worker that extends `GateWorker`.

GateHandler

`GateHandler` is an abstract class that provides behavior shared by the handler of each endpoint.

The shared behavior includes methods that parse and dispatch specific operations, as well as catch and handle exceptions thrown by the worker.

The following classes extend GateHandler and provide access to the worker of each

- CleanupHandler
- CorpusHandler
- DocumentHandler
- EchoHandler
- HelloHandler
- PipelineHandler
- PRHandler

An instance of a GateHandler descendant catches instances of GateDaemonException, GateException, and IOError thrown by its worker while executing an operation. The information from each exception is gathered in the returnResponse of the worker so that it may be returned as a 200 response with an isSuccessful value of false.

Any other worker exception causes a 500 ("Server Error") response from GateServer.

This means that a non-200 response from GateServer indicates an error in the GateDaemon itself or an error in the JVM -- most GATE errors cause a GateException to be thrown by GATE (and caught by GateHandler).

GateWorker

GateWorker is an abstract class that provides behavior shared by the worker of each endpoint. GateWorker also maintains state that is shared by its descendants. This state is primarily the following five caches:

- PluginHash
- DocumentHash
- PipelineHash
- ProcessingResourceHash
- CorpusHash

Each of these binds a name (provided by the client) with a corresponding instance in GATE.

Each instance of a GateWorker descendant inherits a returnResponse. This is is a HashMap that contains an object that is ultimate returned to the client in the response of each successful https request.

CleanupWorker

An instance of CleanupWorker resets the shared caches of GateWorker and invokes cleanup methods in GATE for those instances that require it.

CorpusWorker

An instance of CorpusWorker provides behavior for creating, adding documents to, and clearing a GATE corpus.

DocumentWorker

An instance of `DocumentWorker` provides behavior for creating, loading, and accessing properties of a GATE document.

`DocumentWorker` uses a helper class named `DocumentWrapper`.

- `DocumentWrapper`

`DocumentWrapper` is a helper class with static methods that provide access to the GATE factory classes needed to create and load instances of GATE `Document`.

EchoWorker

An instance of `EchoWorker` echoes the value of its `text` query parameter.

HelloWorker

An instance of `HelloWorker` adds a binding with key `hello` and value `Hello World` to its `returnResponse`

PipelineWorker

An instance of `PipelineWorker` provides behavior for creating, loading, storing, configuring, and running a GATE pipeline.

PRWorker

An instance of `PRWorker` provides behavior for loading, reinitializing, and unloading a GATE `ProcessingResource`.

GateDaemonException

Descendants of `GateDaemonException` are thrown to report specific validation errors detected by `GateDaemon` itself. These are primarily thrown by instances of a `GateWorker` descendant.

Each instance of a `GateDaemonException` descendant thrown during an operation is caught by `GateHandler` and reported to the client as a 200 response whose `isSuccessful` key has a value of "false". The name, description, and stack for the specific failure is contained in the other bindings of the failed 200 response.

A client of `gated` is expected to test for and handle such failures.

The name of the `GateDaemonException` descendant reflects the meaning of the error being reported.

- `InvalidArtifactException`
- `InvalidCorpusNameException`
- `InvalidDocumentNameException`
- `InvalidDocumentPathException`
- `InvalidDocumentURLException`
- `InvalidFeaturePairException`
- `InvalidGroupException`
- `InvalidOperationException`

- `InvalidParameterNameException`
- `InvalidParameterTypeException`
- `InvalidParameterValueException`
- `InvalidPipelineNameException`
- `InvalidPipelinePathException`
- `InvalidPluginException`
- `InvalidPRNameException`
- `InvalidResourcePathException`
- `InvalidVersionException`
- `MissingCorpusException`
- `MissingDocumentException`
- `MissingPipelineException`
- `MissingPRException`
- `UnknownParameterTypeException`

Using `gated` from the command line

The same endpoints used by the `gate` service can be accessed from the command line using an SSH shell.

Many endpoints of `gated` throw exceptions or fail in unexpected ways if a plugin is not loaded. The current implementation and all examples in this document assume the use of the ANNIE plugin.

The `gated` daemon is a long-running service managed by `systemd` (`systemctl` on Rocky Linux). It is therefore good practice to invoke the `cleanup` endpoint at the conclusion of a manual session where other endpoints are exercised.

For convenience, the following `wget` command lines should be executed before and after exercising other `gated` endpoints:

- Setup (load the ANNIE plugin):

  ```
  wget "https://hoyo.zeetix.com:7899/v0/plugin/loadMavenPlugin?
  group=uk.ac.gate.plugins&artifact=annie&version=9.1"
  ```

- Cleanup:

  ```
  wget "https://hoyo.zeetix.com:7899/v0/cleanup"
  ```

  The cleanup endpoint is idempotent, so it never hurts to run it before starting a new manual session. This will cleanup if some earlier session was improperly terminated.

- Restart `gated`

  In the unlikely event that the underlying **JVM** running GATE exhibits unexpected behavior, the following command line kills and restarts the **JVM**:

  ```
  ```

```
sudo systemctl restart gated
```

- Display `gated` status

  The following command line shows the current status of `gated`:

  ```
  sudo systemctl status gated
  ```

  Here is the response (formatted for clarity):

  ```
  ● gated.service - The GATE daemon service
  Loaded: loaded (/etc/systemd/system/gated.service; enabled; vendor
  preset: disabled)
  Active: active (running) since Thu 2023-06-15 15:34:45 UTC; 1min 1s
  ago
  Main PID: 2833 (java)
      Tasks: 17 (limit: 99621)
  Memory: 323.6M
  CGroup: /system.slice/gated.service
          └─2833 /usr/bin/java
              -classpath /opt/maven/boot/plexus-classworlds-2.6.0.jar
              -Dclassworlds.conf=/opt/maven/bin/m2.conf
              -Dmaven.home=/opt/maven
              -Dlibrary.jansi.path=/opt/maven/lib/jansi-native
              -
  Dmaven.multiModuleProjectDirectory=/home/tms/backend/gate-daemon
              org.codehaus.plexus.classworlds.launcher.Launcher
              exec:java -
  Dexec.mainClass=com.zeetix.gate.daemon.server.GateServer
              -Dexec.args=7899

  Jun 15 15:34:45 hoyo.zeetix.com systemd[1]: Started The GATE daemon
  service.
  Jun 15 15:34:47 hoyo.zeetix.com mvn[2833]: [INFO] Scanning for
  projects...
  Jun 15 15:34:48 hoyo.zeetix.com mvn[2833]: [INFO]
  Jun 15 15:34:48 hoyo.zeetix.com mvn[2833]: [INFO] -------------------<
  com.zeetix.gate.daemon:server >--------------------
  Jun 15 15:34:48 hoyo.zeetix.com mvn[2833]: [INFO] Building
  ZeetixGateDaemon 0.0.0-SNAPSHOT
  Jun 15 15:34:48 hoyo.zeetix.com mvn[2833]: [INFO]   from pom.xml
  Jun 15 15:34:48 hoyo.zeetix.com mvn[2833]: [INFO] --------------------
  ------------[ jar ]---------------------------------
  Jun 15 15:34:48 hoyo.zeetix.com mvn[2833]: [INFO]
  Jun 15 15:34:48 hoyo.zeetix.com mvn[2833]: [INFO] --- exec:3.1.0:java
  (default-cli) @ server ---
  ```

The following example endpoint is used to run a pipeline that has already been created and configured:

```
https://hoyo.zeetix.com:7899/v0/pipeline/runPipeline?
pipelineName=testpipeline
```

The following table uses the above example to illustrate the several parts of a `gated` endpoint, using the above example.

Part	Example	Description
base	https://hoyo.zeetix.com	The https URL of the target system
port	7899	The designated port of the `gated` service
version	v0	Version specifier for the endpoint
path	pipeline	A version designator followed by the name of the resource to be used
operation	runPipeline	The operation to perform on the designated resource
query	pipelineName=testpipeline	A sequence of one or more key-value pairs

Table 1: gated Endpoint Anatomy

For convenience, every endpoint in the `gate` project is a GET endpoint. Each successful response is a JSON-encoded object containing the response from the GATE instance.

This means that any endpoint can be exercised using the "wget" command. Here is a command line string that invokes the example endpoint:

```
wget "https://hoyo.zeetix.com:7899/v0/pipeline/runPipeline?
pipelineName=testpipeline"
```

NOTE: The URL is enclosed in a double-quote pair (`"`) because many endpoints contain reserved characters interpreted by the command line shell (such as `bash`).

Here is a command line that loads the ANNIE plugin, together with its response:

Command line:

```
wget "https://hoyo.zeetix.com:7899/v0/plugin/loadMavenPlugin?
group=uk.ac.gate.plugins&artifact=annie&version=9.1"
```

Response (pretty printed and reordered for clarity):

```
{
    "isSuccessful":"true",
    "operation":"loadMavenPlugin",
    "pluginGroup":"uk.ac.gate.plugins"
    "pluginArtifact":"annie",
    "pluginVersion":"9.1",
}
```

Path	Operations	Parameters (*=optional)
plugin		
	loadMavenPlugin	group, artifact, version
cleanup		
	n.a.	*n.a.*
document		
	loadFromURL	documentURL
	loadFromPath	documentPath
	getDocumentContent	documentName
	getAnnotationSetNames	documentName
	getAnnotationsForName	documentName, annotationSetName*
	cleanupDocument	documentName
corpus		
	createCorpus	corpusName
	clearCorpus	corpusName
	addDocument	corpusName, documentName
pr		
	loadPR	prName, resourcePath
	reinitPR	prName
	unloadPR	prName
pipeline`		
	createPipeline	pipelineName
	addPR	pipelineName, prName
	setCorpus	pipelineName, corpusName
	getParameterValue	pipelineName, prName, parameterName

Path	Operations	Parameters (*=optional)
	setParameterValue	pipelineName, prName, parameterName, parameterValue*, parameterType
	runPipeline	pipelineName
	storePipelineToFile	pipelineName, pipelinePath
	loadPipelineFromFile	pipelineName, pipelinePath

Table 2: gated Endpoints

NOTE: The `parameterValue` is optional for the `v0/pipeline` endpoint so that a parameter value can be set to `null`. This is accomplished by providing values for the `pipelineName`, `prName`, `parameterName`, and `parameterType`.