### How to implement a QueryCommandParser

This section describes the [SampleQueryCommandParser.java](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/query/SampleQueryCommandParser.java) code and the [SampleSubnetGroupsCommandParser.java](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/query/SampleSubnetGroupsCommandParser.java) code.

#### Overview

A query in Logpresso is a combination of query commands separated by pipes. The Logpresso query engine splits the query string by pipe characters and then recognizes the first token separated by a space as the name of the query command. The query engine searches for a QueryCommandParser object with the name of the query command to request parsing of the query command.

A query command parser must implement the QueryCommandParser interface:

public interface QueryCommandParser { String getCommandName(); QueryCommand parse(QueryContext context, String commandString); Map<String, QueryErrorMessage> getErrorMessages(); QueryCommandHelp getCommandHelp(); QueryParserService getQueryParserService(); void setQueryParserService(QueryParserService queryParserService);}

Each method implements the following functions:

* getCommandName(): Returns the name of the query command, which is unique on the Logpresso platform. The query command name is automatically highlighted in the query input.
* parse(): Parses the query command statement and returns a QueryCommand object.
* getErrorMessages(): Provides localized messages about error codes encountered during query parsing.
* getCommandHelp(): Provides a description, options, and output field details for the query command. A default implementation is provided for this method.
* getQueryParserService(): Returns the QueryParserService object. Use this when you need to request parsing from the parser recursively, such as subqueries.
* setQueryParserService(): Sets the QueryParserService object. This method should not be called directly except in unit tests.

#### Parser with connect profile

Most of the query commands you extend in your app will interact with external systems, so they use connect profiles. A query command parser that uses a connect profile can be implemented simply by inheriting from the ConnectProfileQueryCommandParser class. The SampleQueryCommandParser provides a base implementation that the multiple query command parsers you implement in your app will inherit from.

public abstract class SampleQueryCommandParser extends ConnectProfileQueryCommandParser { protected static final String ERR\_SERVICE\_UNAVAILABLE = "204000"; protected static final String ERR\_PROFILE\_REQUIRED = "204001"; protected static final String ERR\_NAME\_REQUIRED = "204002"; public SampleQueryCommandParser() { super("sample", ERR\_SERVICE\_UNAVAILABLE, ERR\_PROFILE\_REQUIRED); }}

The constructor has the following parameters:

* profileType: The identifier of the type of connect profile used by the query command; that is, it must match the return value of getType() of ConnectProfileFactory.
* emptyProfileErrorCode: The error code to raise if there are no connect profiles allowed in the current session.
* missingProfileErrorCode: The error code to raise if the profile parameter of the query command is required and no profile value is specified.

protected List<String> getSupportedOptions() { return new ArrayList<String>(getCommandHelp().getOptions().keySet());}

The query command is defined in the form of COMMAND-NAME KEY1=VALUE1 KEY2=VALUE2. The getSupportedOptions() returns a list of available parameters. This code is implemented to return all parameters defined in the help.

protected abstract QueryCommand parse(QueryContext context, SampleParams params);

Query command parsers that inherit from SampleQueryCommandParser can use parameters that have already been parsed and passed to SampleParams. Each query command parser performs additional validation, including checking for required parameters.

protected ConnectProfileParams parseParams(QueryContext context, Map<String, String> opts) { SampleParams params = new SampleParams(); params.setName(opts.get("name")); return params;}protected QueryCommand parse(QueryContext context, ConnectProfileParams params, String commandString) { return parse(context, (SampleParams) params);}

* parseParams(): Parses the parameters of all extended query commands. You can parse parameters for each query command, but it's easier to parse common parameters if you parse them in one place.
* parse(): The ConnectProfileQueryCommandParser class parses the key=value parameter patterns from the query command to construct a map, then calls parseParams() and passes the resulting ConnectProfileParams object back to the parse() method. We simply utilize the parameters of the already parsed query command.

public Map<String, QueryErrorMessage> getErrorMessages() { Map<String, QueryErrorMessage> errors = new HashMap<>(); errors.put(ERR\_SERVICE\_UNAVAILABLE, newMsg("No available sample profile found.", "사용 가능한 샘플 프로파일이 없습니다.")); errors.put(ERR\_PROFILE\_REQUIRED, newMsg("Specify valid sample profile.", "샘플 프로파일 이름을 입력해주세요.")); errors.put(ERR\_NAME\_REQUIRED, newMsg("Specify name option in the sample-create-subnet-group command.", "sample-create-subnet-group 명령어에 name 옵션을 지정하세요.")); return errors;}

getErrorMessage() returns a pair of an error code and a localized message. The query comamnd parser specifies the error code when it throws a QueryParseException object. The Logpresso query engine references the return value of this method when converting the error code back into a localized message.

#### Parser details

The SampleSubnetGroupsCommandParser class inherits from the SampleQueryCommandParser class defined above.

public class SampleSubnetGroupsCommandParser extends SampleQueryCommandParser { public String getCommandName() { return "sample-subnet-groups"; } protected QueryCommand parse(QueryContext context, SampleParams params) { return new SampleSubnetGroupsCommand(params); }}

This parser defines a query command named sample-subnet-groups, which has no parameters, so parse() will create and return a SampleSubnetGroupsCommand object right away without additional parsing steps.

Now, pay attention to metadata of the constructor.

setDescription(Locale.ENGLISH, "Get subnet groups from the Logpresso server.");setOutput("guid", ValueType.STRING, Locale.ENGLISH, "GUID", "");

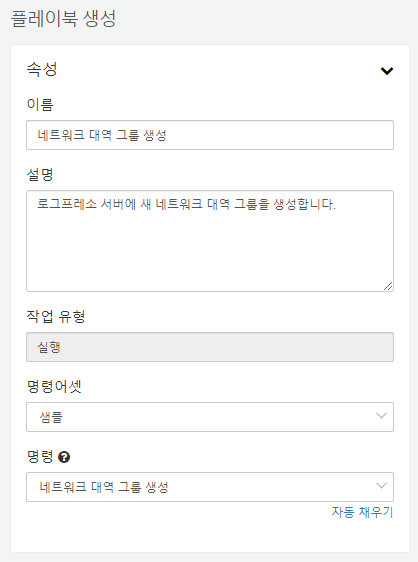
* setDescrpition(): Sets the localized description to display in the query help.
* setOutput(): Sets the type, localized name, and localized description of the output field to display in the query help.

Other metadata is also available in the SampleCreateSubnetGroupCommandParser class.

setDisplayGroup(Locale.ENGLISH, "Sample");setDisplayName(Locale.ENGLISH, "Create subnet group");setDescription(Locale.ENGLISH, "Create new subnet group in the Logpresso server.");setOption("name", REQUIRED, Locale.ENGLISH, "Name", "New name of the subnet group.");

* setDisplayGroup(): The display name of the query command group. Mapped to a command set in the playbook.
* setDisplayName(): The display name of the query command. Mapped to a command in the playbook.
* setOption(): Defines command parameters. Utilized for command help and autocomplete when pressing Ctrl+Space.

For example, if you select the Sample command set when creating an execution task in the Playbook Designer, you can see the Create subnet group item, as shown below.



Exceptions during query command parsing are raised as follows. The create-subnet-group command requires a name parameter, and if this value is not specified, it raises a QueryParseException exception, causing the parse to fail.

protected QueryCommand parse(QueryContext context, SampleParams params) { if (params.getName() == null) throw new QueryParseException(ERR\_NAME\_REQUIRED, -1); return new SampleCreateSubnetGroupCommand(params);}

In the next section, we'll look at registering this implemented query command parser.