### How to implement a ConnectProfileFactory

This section describes the code for [SampleConnectProfileFactory.java](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/impl/SampleConnectProfileFactory.java).

#### Overview

Logpresso's query commands and collectors often communicate with external systems. For example, the [dbquery](https://docs.logpresso.com/en/query/dbquery-command) command retrieves the results of SQL query on an external database. The JDBC connection string, account, and password are required to connect to the external database.

Therefore, the dbquery query command receives the connect profile name as the PROFILE parameter. To execute the dbquery query command, you must preset the database connect profile as shown below.



On the other hand, even if a connect profile is set up, you shouldn't allow just anyone to run SQL queries against your database. You must grant an account or group of accounts permission to use the connect profile before they can use it to communicate with external systems.

Most external systems, not just databases, require authentication. The sample app calls Logpresso Sonar's REST API, which requires you to pass an API key in the Authorization HTTP header to call the REST API.

Since the settings are all different depending on the authentication method required by the external system, the app needs to implement a proprietary ConnectProfileFactory to extend the functionality of the Logpresso platform.

#### Factory registration

The sample app implements a SampleConnectProfileFactory to register with the Logpresso platform.

@Component(name = "sample-connect-profile-factory")@Instantiatepublic class SampleConnectProfileFactory extends AbstractConnectProfileFactory { @Requires private ConnectProfileFactoryRegistry connectProfileFactoryRegistry; @Validate public void start() { connectProfileFactoryRegistry.addFactory(this); } @Invalidate public void stop() { if (connectProfileFactoryRegistry != null) connectProfileFactoryRegistry.removeFactory(this); }}

The code above defines a SampleConnectProfileFactory component that inherits from AbstractConnectProfileFactory. When the component starts, it registers itself with the ConnectProfileFactoryRegistry OSGi service by calling addFactory(), and when the component ends, it removes itself by calling removeFactory().

The @Component, @Instantiate, @Requires, @Validate, and @Invalidate are all iPOJO annotations. Each annotation is described below:

* @Component: Declares that this class is an iPOJO component. An iPOJO component must have a unique name.
* @Instantiate: Instructs to create an instance of the iPOJO component. If you don't declare @Instantiate, no component instance is created when you start the app bundle.
* @Requires: Declare a service interface dependency. There is no separate assignment, but when the component starts, the iPOJO framework will automatically inject an OSGi service object that provides the ConnectProfileFactoryRegistry interface. We refer to this as an injected POJO (Plain Old Java Object), which is where the framework name iPOJO comes from.
* @Validate: Declares a callback that will be called when the component starts. The component will only start if all dependencies declared with @Requires are satisfied.
* @Invalidate: Declares a callback to be called when the component is stopped. If any of the dependencies declared as @Requires are not satisfied, the component stops. When you stop or delete an OSGi bundle, the service is removed and the components that depended on it are also stopped.

#### Factory details

**Identifier**

getType() returns the identifier needed to distinguish each ConnectProfileFactory object registered in the ConnectProfileFactoryRegistry. This is typically defined using lowercase letters of the alphabet and hyphens.

public String getType() { return "sample";}

**Display Name**

getDisplayName() returns the name of the connect profile type to display in the user interface. It should return a localized string literal.

public String getDisplayName(Locale locale) { if (locale != null && locale.equals(Locale.KOREAN)) return "샘플"; return "Sample";}

**Description**

getDescription() returns a description of the connect profile type to display in the user interface. It should return a localized string literal.

public String getDescription(Locale locale) { if (locale != null && locale.equals(Locale.KOREAN)) return "샘플 접속 설정을 관리합니다."; return "Manage connection properties of sample app.";}

**Protected Config Key**

getProtectedConfigKeys() returns a list of identifiers for configuration items that should not be displayed in the user interface. For example, passwords or API keys should not be viewable on screen, even as an administrator. Protected configuration items can only be reset, not looked up, and are stored encrypted within the Logpresso platform.

public Set<String> getProtectedConfigKeys() { return Set.of("api\_key");}

**Config Option**

The getConfigOptions() defines the configuration form for the connect profile. While it may seem a bit awkward to use a type named LoggerConfigOption for a connect profile's configuration specification, we use the same type to define the logger's configuration form dynamically.

public List<LoggerConfigOption> getConfigOptions() { LoggerConfigOption endpoint = new MutableStringConfigType("endpoint", t("Endpoint", "엔드포인트"), t("REST API URL", "REST API 주소"), true); LoggerConfigOption apiKey = new MutableStringConfigType("api\_key", t("API key", "API 키"), t("GUID format", "GUID 형식"), true); return Arrays.asList(endpoint, apiKey);}

The following three types are available

* MutableStringConfigType: Mutable string configuration item.
* MutableIntegerConfigType: Mutable 32-bit integer configuration item.
* MutableBooleanConfigType: Mutable boolean configuration item.

The configuration items returned in the list are displayed on the screen in the order they appear.

**Config Summary**

Returns a summary of the core settings so that you can quickly see the details of the settings in the list of connect profiles. You should return the localized string literal.

public String describeConfigs(ConnectProfile profile, Locale locale) { String endpoint = profile.getString("endpoint"); if (locale != null && locale.equals(Locale.KOREAN)) return "엔드포인트: " + endpoint; return "Endpoint: " + endpoint;}