# .NET Integration SDK Readme

This is the OutSystems Database Integration SDK. The SDK allows you to create a plugin that can be used by the OutSystems Platform to integrate with external databases. It contains a test suite for your plugin, a sample plugin project and also a complete MySQL plugin project that you can use to guide your implementation.

## Requirements

- NUnit 2.4.8 (**use the installer for .NET 2.0**)
- Visual C# 2010+
- .NET Framework 4.5.1
- (Optional) MySQL 5.6.20+

## How to Install

- Download the Database Integration SDK from http://outsystems.com/network/dl-net-integration-sdk
- Unzip the archive
- Copy the DLLs located in the `NUnitPlugin` folder to NUnit addin folder
- Disable NUnit shadow copy feature
- Make sure the `nunit.exe.config` file has a `startup` section **uncommented** and the **first** entry is `<supportedRuntime version="v4.0" />`

## Getting Started

### Exploring the MySQL Plugin

The solution `MySQLDatabaseProvider.sln`, located in the `MySQLDatabaseProvider` folder, contains the full source code for the MySQL DatabaseProvider.
You can run the SDK tests suite against this plugin using the provided configurations and setup scripts. To do that:

- Go to the folder `MySQLDatabaseProvider`
- Open the `MySQLDatabaseProvider` solution and build it
- Go to the folder `MySQLDatabaseProvider.Tests`
- Use the script `MySQLDatabaseProvider.Tests.sql` to set up the test database
- Change the `bootstrap_NET.config` and `MySQL_NET.config` files under the folder `Default\MySQL\MySQL`. The connection string must point to your own database.
- Open the NUnit project `MySQLDatabaseProvider.Tests.nunit`
- Run the tests and verify they pass.

### The Sample Plugin Project

The sample plugin project contains a skeleton for a new plugin implementation. A database integration must implement five services:

- The Configuration service, responsible for holding and describing the data that is necessary to connect to a database and configure it

- The Execution service contains the logic that creates and processes objects that are used to query the database
- The Transaction service establishes connections and manages transactions
- The Introspection service implements logic to inspect the database model
- And finally the DML service holds the query building logic

For each of those services there is a base implementation that implements a great deal of logic for you. You can take advantage of it or implement the service interfaces yourself. These interfaces and base classes are fully documented.

In order to minimize issues with dependencies among services, implement the services in the following order:

- Configuration
- Execution
- Transaction
- DML
- Introspection

If your plugin implementation has dependencies, for example the database driver libraries, they should be copied to the folder `Libraries/plugins/database` together with the plugin implementation.

Use the MySQL plugin implementation to help you complete your plugin.

## Setting Up Tests for Your Plugin

The `SampleDatabaseProvider.Tests` folder contains placeholder configuration files, SQL scripts and NUnit projects that you can edit to fit your database engine setup and syntax.
An SDK tests runs in an environment defined by five things:

- A database initially setup by the script `SampleDatabaseProvider.Tests.sql`. You should adapt this script and run it once on your database.
- A configuration with admin privileges, located in `Default\Sample\Sample\bootstrap_NET.config`
- The tests themselves, implemented in code, run with a runtime user configuration, located in `Default\Sample\Sample\Sample_NET.config`
- A bootstrap script that sets up the state in the database for a particular test. This script runs with the admin configuration.
- A teardown script that deletes the state that was previously setup for the test. This script also runs with the admin configuration.

The bootstrap and teardown scripts are named `Sample_bootstrap.sql` and `Sample_teardown.sql`. They are located under the `SampleDatabaseProvider.Tests` folder and each service has its own.
If everything is setup correctly when you open the NUnit project you should see the SDK test suite configured with your plugin implementation.

### Changing the Sample Plugin Project Name

Of course your plugin won't be named *Sample*. To change your plugin name you must do the following steps:

- Open the SampleDatabaseProvider solution
- Change the plugin namespace
- Change the project name
- Change the plugin solution name
- Change the plugin assembly name. The name must contain "DatabaseProvider"-
- Change the plugin assembly information details
- Change the Database Provider Key in the `DatabaseProvider` class
- Change the paths inside the the file `SampleDatabaseProvider.Tests.config` to match your plugin name
- Change the Database Provider Key in the `DatabaseProvider` class
- Now the tricky part, under the `SampleDatabaseProvider.Tests`, change every folder named *Sample* and every file prefixed with *Sample* replacing the *Sample* word with the **Database Provider Key** you chose.
- Change the `SampleDatabaseProvider.Tests` folder name.
- Change the NUnit project name
- Check that the plugin project copies the `.config` file from the `SampleDatabaseProvider.Tests` folder to the `Libraries` folder.

# Java Integration SDK Readme

This is the OutSystems Database Integration SDK. The SDK allows you to create a plugin that can be used by the OutSystems Platform to integrate with external databases. It contains a test suite for your plugin, a sample plugin project and also a complete MySQL plugin project that you can use to guide your implementation.

## Requirements

- Sun JDK 6u45
- Ant 1.8+
- (Optional) MySQL 5.6.20+

## How to Install

- Download the Database Integration SDK from http://outsystems.com/network/dl-java-integration-sdk
- Unzip the archive
- Make sure the your Ant installation is using Sun JDK 6u45

# Getting Started

## Exploring the MySQL Plugin

The the `MySQLDatabaseProvider` folder, contains the full source code for the MySQLDatabaseProvider.
You can also run the SDK tests suite against this plugin using the provided configurations and setup scripts. To do that:

- Use the script `MySQLDatabaseProvider.Tests.sql` to set up the test database
- Change the `bootstrap_JAVA.config` and `MySQL_JAVA.config` files connection string to connect to your own database
- Open the `MySQLDatabaseProvider` folder and run the command `ant jar`. This will build the MySQLDatabaseProvider plugin jar
- Open the `MySQLDatabaseProvider.Tests` folder and run the command `ant run`.
- Verify that the tests pass

## The Sample Plugin Project

The sample plugin project contains a skeleton for a new plugin implementation. A database integration must implement five services:

- The Configuration service, responsible for holding and describing the data that is necessary to connect to a database and configure it
- The Execution service contains the logic that creates and processes objects that are used to query the database
- The Transaction service establishes connections and manages transactions
- The Introspection service implements logic to inspect the database model
- And finally the DML service holds the query building logic

For each of those services there is a base implementation that implements a great deal of logic for you. You can take advantage of it or implement the service interfaces yourself. These interfaces and base classes are fully documented.

In order to minimize issues with dependencies among services, implement the services in the following order:

- Configuration
- Execution
- Transaction
- DML
- Introspection

If your plugin implementation has dependencies, for example the database driver libraries, they should be copied to the folder `Libraries` together with the plugin implementation.

Use the MySQL plugin implementation to help you complete your plugin.

## Setting Up Tests for Your Plugin

The `SampleDatabaseProvider.Tests` folder contains placeholder configuration files and SQL scripts that you can edit to fit your database engine setup and syntax.
An SDK tests runs in an environment defined by five things:

- A database initially setup by the script `SampleDatabaseProvider.Tests.sql`. You should adapt this script and run it once on your database.
- A configuration with admin privileges, located in `Default\Sample\Sample\bootstrap_JAVA.config`
- The tests themselves, implemented in code, run with a runtime user configuration, located in `Default\Sample\Sample\Sample_JAVA.config`
- A bootstrap script that sets up the state in the database for a particular test. This script runs with the admin configuration.
- A teardown script that deletes the state that was previously setup for the test. This script also runs with the admin configuration.

The bootstrap and teardown scripts are named `Sample_bootstrap.sql` and `Sample_teardown.sql`. They are located under the `SampleDatabaseProvider.Tests` folder and each service has its own.
If everything is setup correctly running the command `ant dump` this will create a file `tests.txt` with a list of tests available for your plugin.
Use the command `ant run` to run the SDK tests against your plugin.

## Changing the Sample Plugin Project Name

Of course your plugin won't be named *Sample*. To change your plugin name you must do the following steps:

- Go to the `SampleDatabaseProvider` folder
- Open the `build.xml` file and change
  - the project name
  - the jar destfile name
- Open the `.project` file and change the project name
- Open the `outsystems.hubedition.extensibility.data.IDatabaseProvider` file and change the namespace of the class name that implements the `IDatabaseProvider` interface.
- Open the `src` folder and change the package names together with the package directory names.
- Open the `DatabaseProvider` class and change the Database Provider Key
- Go to the `SampleDatabaseProvider.Tests` folder
- Open the `build.xml` and change the project name attribute
- Open the `integrationdatabaseprovider.tests.properties` file and change the file paths to match the Database Provider Key you picked for your database provider

- Now the tricky part, under the `SampleDatabaseProvider.Tests`, change every folder named *Sample* and every file prefixed with *Sample* replacing the *Sample* word with the provider **Database Provider Key**.