HP IT Executive Scorecard

For the Windows ® operating system

Software Version: 9.4

FBI Extractor SDK Guide

Document Release Date: June 2013

Software Release Date: June 2013

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

- Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.
- AMD and the AMD Arrow symbol are trademarks of Advanced Micro Devices, Inc.
- Google™ and Google Maps™ are trademarks of Google Inc.
- Intel®, Itanium®, Pentium®, and Intel® Xeon® are trademarks of Intel Corporation in the U.S. and other countries.
- Java is a registered trademark of Oracle and/or its affiliates.
- Microsoft®, Windows®, Windows NT®, Windows® XP, Windows Vista® and SQL Server® are U.S. registered trademarks of Microsoft Corporation.
- Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Disclaimer for PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format.

Note: Some topics do not convert properly to PDF, causing format problems. Some elements of online help are completely removed from the PDF version. Those problem topics can be successfully printed from within the online help.

Contents

Getting Started with Creating an FBI Extractor

The File Based Integration (FBI) extractor enables you to extract data from any data source with maximum agility and minimum requirements from the customer.

The FBI framework contains out-of-the-box extractors for MSSQL, Oracle and MySQL. A custom extractor can be created for a new database type or other protocols. When you have created new content and Content Packs, and the out-of-the-box extractors are not sufficient for your content integration, you can then develop an FBI extractor to extract the data for use in the Data Warehouse.

This document is a guide to extending the FBI framework with a new custom FBI extractor. This guide instructs you in the process of building an FBI extractor as follows:

1. Use the Source Model xml from the relevant Content Pack in order to develop and test the extractor. For details, see "Extractor Prerequisite" on page 12.

2. Write the Extractor, which includes implementing the extractor guidelines, developing the extractor using the Sample Extractor, and testing and modifying the extractor. For details, see "Develop a JAVA Based Extractor" on page 13.

3. Configure the Data Source Management UI which includes configuring and copying the xml file to the Content Pack for integrating to XS. For details, see "Configure the Data Source Management User Interface for XS Integration" on page 31.

4. Package and deploy the Extractor, which includes building and deploying the Extractor. For details, see "Package and Deploy the Extractor- JAVA Based Only" on page 35.

FBI Script Extractor

The FBI Script Extractor functions in conjunction with the current FBI framework. In this case the Script Extractor loads the script from the Data Source Content Pack to perform extraction. It therefore simplifies the customization for FBI Extractors. For details, see "Develop a Script Based Extractor" on page 26.

Learn More

A custom FBI extractor is required in the following situations:

- To support a new database type.
- To extract data from a protocol other than Oracle, MSSQL, or MySQL, for example Rest services, Web Services, or specific API.

For details on out-of-the-box FBI Extractors, see Learn About File Based Integration in the *Administrator Guide*.

To access:

You can create your extractor based on a modification of a Sample Project found in the **HP Live Network Portal: Home > Executive Scorecard > Integration Content for Executive Scorecard**.

For Script Extractor see "Develop a Script Based Extractor" on page 26.

Tasks

The following topics describe the various steps in the FBI SDK process:

Terms

The following describes the basic terms and concepts used in this guide. These terms are part of the FBI extraction framework, and are integral to understanding the extraction process.

BusinessObjects Data Services (BODS):

BODS is an operational and analytical data-driven tool it is used for data integration, data aggregation, data quality and text analysis. It is responsible for the execution of the ETL workflows.

Instruction Object

Consists of parameters that are set by the BODS workflow during the Source Extract. Parameters in the Instruction are made available to the Extractor during its runtime. Parameters in the Instruction object are accessable over the FBI framework.

Source Extract

The first stage of the ETL is the Source Extract. In this phase, the BODS workflow inserts an instruction to the INSTRUCTION table and then invokes the Extractor with the instruction ID as a parameter.

File Based Integration

File Based Integration (FBI) is a framework that manages the integration and extraction of data from data sources into the Data Warehouse local file system, as well as the runtime and activation of Extractors.

Extraction Methodology

The Extractors are components designed to extract data from external data sources into files which are compatible to the layout and structure of the extraction metadata. The Extractor validates the

structure of the data, and streamlines the data into the FBI framework. The framework in turn transforms the data into a file which complies to a naming and structure convention required for loading the data into the Staging area.

Content Packs:

A Content Pack is a set of XS Data warehouse ETL Workflows (BODS Workflows) and data model metadata artifacts. The Data Warehouse supports two types of Content Packs: CORE and Integration Content Pack (i-CP). The CORE content pack contains definitions of the target model while an i-CP contains the integration and source compliant data models.

Each content pack uses an extractor in order to extract the data from the specific data source and contain all the artifacts needed to connect to the relevant data source and gather data from that data source. It is comprised of folders which contain files of various processes and functions. Content packs are found in the following path: **<installation directory>\agora\ContentPacks.**

Data Source:

The integration of data into the data warehouse is done through the activation of data sources, or products that contain the relevant data in the form of a Content Pack.

Integrated Development Environment (IDE):

Enables you to add new content and extend existing content thereby creating the Source to Target model metadata for the FBI extractor.

Flat File:

The output txt file of the FBI extractor comprised of a well-defined format that is understood by the BODS EXT workflow.

Integration Architecture

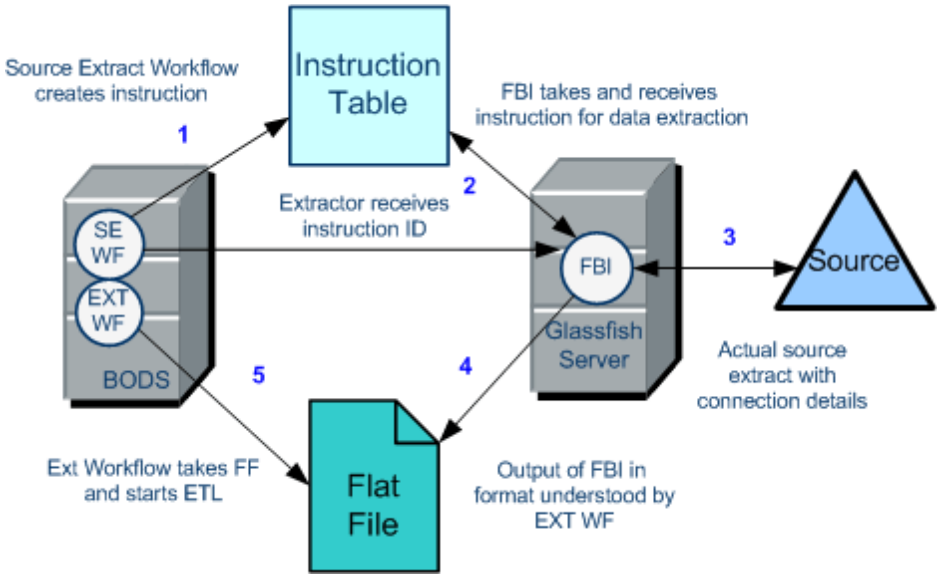
The following section describes the basic architecture and flow of the FBI extractor process which is necessary to create the custom extractor.

In the extractor process the following information is gathered by the FBI framework:

- Instruction record: Created by the Source Extract Workflow.
- Connection details: The source credentials.
- Extraction Model Metadata
  - SourceModel.xml: Defines the data structure and how this information is extracted from the source (generated by the IDE).
  - Ext Workflow: Defines the BODS input structure that is the same structure as the flat file (generated by the IDE).

The following depicts the high level architecture of the FBI extraction process.

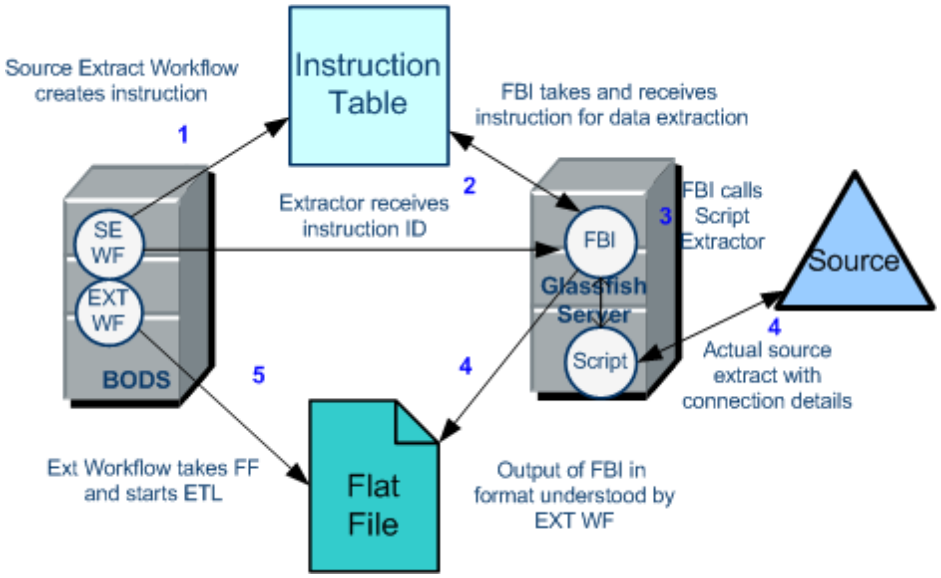
FBI Architecture Process - JAVA Based



- BODS Source Extract Workflow triggers the extractor by creating the Instruction table information.
- FBI receives instruction information for the Extraction process.
- FBI extracts the source model and connection details.
- The Extractor begins to extract the data from the data source into a local file.

- The flat file FBI output starts the Ext Workflow in BODS.
- Ext Workflow takes the flat file and starts the ETL.

FBI Architecture Process - Script Based



- BODS Source Extract Workflow triggers the extractor by creating the Instruction table information.
- FBI receives instruction information for the Extraction process.
- FBI calls the Script Extractor with instruction information.
- Script Extractor extracts the source model and connection details from a script in the Content Pack.
- The Extractor begins to extract the data from the data source into a local file.
- The flat file FBI output starts the Ext Workflow in BODS.
- Ext Workflow takes the flat file and starts the ETL.

  For details on a Script based extractor, see "Develop a Script Based Extractor" on page 26.

Extractor Prerequisite

The extractor developer uses the Source Model xml files from the Content Pack to enable the extractor functionality. This metadata component is required for the custom extractor development process. The Source Model xml is a mandatory component of each Content Pack.

To create a Source Model xml using the IDE:

1. For a new Content Pack, create your project in the IDE. For details, see Define New Content Pack Project in the *Content Extension Guide*.
2. Add content in the Engineer Stream Designer. For details, see Engineer Stream Designer Tasks in the *Content Extension Guide*.
3. Generate your content for use in the Data Warehouse. Select the **Generate Content Pack** checkbox to create a new Content Pack (not extending content). This creates a new content pack structure along with the generated ETL files. For details, see Generate Content in the *Content Extension Guide*.
4. The source entity xml is located in the CP output folder under **<installation directory>\agora\ContentPacks\<CP name>\dataModel\dataIntegration\source\<version>\<CP name>.**
5. The Source Model xml is then used in the development of the custom extractor. For details, see "Develop a JAVA Based Extractor" on page 13

Develop a JAVA Based Extractor

The next stage in the FBI Extractor process is to write the actual extractor. You can modify the Sample Extractor or write your own according to the guidelines.

This section contains the following:

Extractor Implementation Guidelines

You can create your own extractor, or adapt the Sample Project extractor. Note the following guidelines when adapting a sample or creating a new extractor.

The FBI Extractor implementation requires you to create three essential components. All of the components are packaged in a jar that you copy and deploy. For deployment, the jar must include the following META_INF\beans.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
</beans>
```

The extraction implementation product is a Flat File that is automatically saved and contains all relevant extraction information.

The following describes the best practices for implementation.

XXXExtractor

This component extends the BaseExtractor and requires the following implementation:

```
boolean testConnection()
```

```
extract
```

Before implementation, you must define a unique name of the Stateless EJB to enable remote method call for each instance. Define the name of the bean in annotation above the class as follows:

```
@Stateless(name = "extractorname")
```

Extractor Functionality

The Extractor is defined as an EJB and the lookup is done by a JNDI mechanism. The name of the Extractor must be defined , inside the stateless annotation (this name should be the exact match with the product type that is defined in the data source connection).

In order to implement a new Extractor you need to override two methods:

```
public boolean testConnection(DataSourceConnectionType connectionType, String pr
oductType) throws ExtractorException
```

```
public Status extract() throws ExtractorException
```

Activate:

- Test the connection.
- Verify that the relevant source is available and the Data Warehouse can connect to it.
- Use DataSourceConnectionType: The connection details of the specific source.

testConnection():

Reads the data source connection from the DataSourceConnectionType, connects to the source with this information and returns true if the connection succeeds, or throw exception if it fails.

The input parameter DataSourceConnectionType has all of the source connection details.

In the base there is a helper method for taking the connection details from the DataSourceConnectionType and placing into a key value map.

For example:

```
host -> <host of the source>
port -> <port of the source>
user -> <username for the source>
pass -> <password>
```

The test operation uses these details in order to validate that the source is accessible from the Data Warehouse.

SourceExtract:

- Extract the relevant data from the source.
- Use SourceModel:
  - Describes the required data and data structure of the source.
  - Defines the structure of the result output text file.

  The extract(), should connect to the source, read the desired data that is defined in the SourceModel, arrange it inside ResultIterator and call the writeTofile() method that is defined in the BaseExtractor. After, it should call the reportStatus and return the status.

  extract()

  Takes the connection details from the DataSourceConnectionType into key value map:

  ```
  HashMap<String, String> dataSourceGenericProperties = getDataSourceGenericProp
  erties(connectionType);
  ```

  SourceModel stores information about the data structure of the source. It defines the name of the entity and the fields related to this entity. In order to get the fields and their order, you can use fieldMapping (defined and initialized in the BaseExtractor).

- Key of fieldMapping is the datasourcetablenameoraliasname from the ColumnMapping in the sourceModel.

- Value of the fieldMapping is the Field Object that stores the rest of the ColumnMapping details.
- mappingType object stores the Entity name, the criteria and the group by details.

Checking extract status: ABC requires the status to know whether to retry the extract step.

- Return the operation status info to the invoker:

```
Status status = reportStatus("extract operation by
DelimitedFileExtractor", StatusEnum.SUCCESS);
return status
```

XXXResultIterator

This component extends the ResultIterator and requires the following implementation:

```
boolean hasNext()
```

```
dataRowMapping next()
```

ResultIterator Implementation:

Result Iterator is a representation of rows of data. It is best to keep the return data object in the ResultIterator instead of copying it to another data structure.

For example:

Construct this object with the resultSet of JDBC and use its next and hasNext method

```
hasNext()
```

Checks if there is another row.

```
next()
```

Returns the next row which is kept as DataRowMapping.

XXXDataRowMapping

This component extends the DataRowMapping and requires the following implementation:

```
String get(Field fieldObj)
```

The input argument should be Field (from the FieldsMapping List) and the output should be a String.

DataRowMapping Implementation:

```
get()
```

Gets the field detail as the argument and returns a value as a String.

API

Object Model: Contains the different models with metadata used in the source extract.

- **BaseExtractor:** The abstract implementation that performs the basic jobs for the extractor as well as writing the Flat File.

  The sample extractor displays the flow of the extractor from the trigger of the metadata to the writing of the Flat File and how the extractor pulls the data.

  For details on the Sample extractor, see "Use the Sample Extractor" on page 18

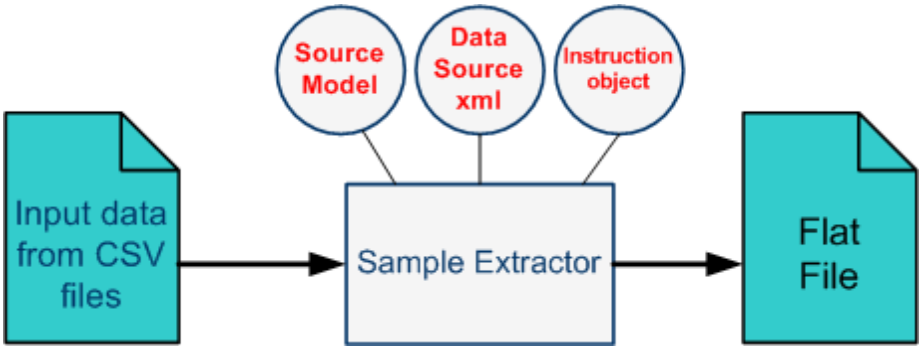
- **Instruction:** Parameters stored in the dwabc.INSTRUCTION table in the staging database. The parameters are set by the BODS workflow during the Source Extract phase. For details, see "Modify the Sample Extractor" on page 24.

- **Datamodel:** Source Model that describes the mapping of the Source fields to Flat File fields.

- **DataSourceConnection:** Source connection details used by the extractor.

Use the Sample Extractor

A sample extractor is provided for you to work with in order to extract source data in CSV format. The process consists of extracting the CSV data using the Sample Extractor which takes information from the Source Model, the Data Source xml connection details, and the Instruction table information. This produces a Flat File output used in running the ETL.



Note: Make sure you follow the extractor implementation guidelines when adapting a sample or creating a new extractor. For details, see "Extractor Implementation Guidelines" on page 14.

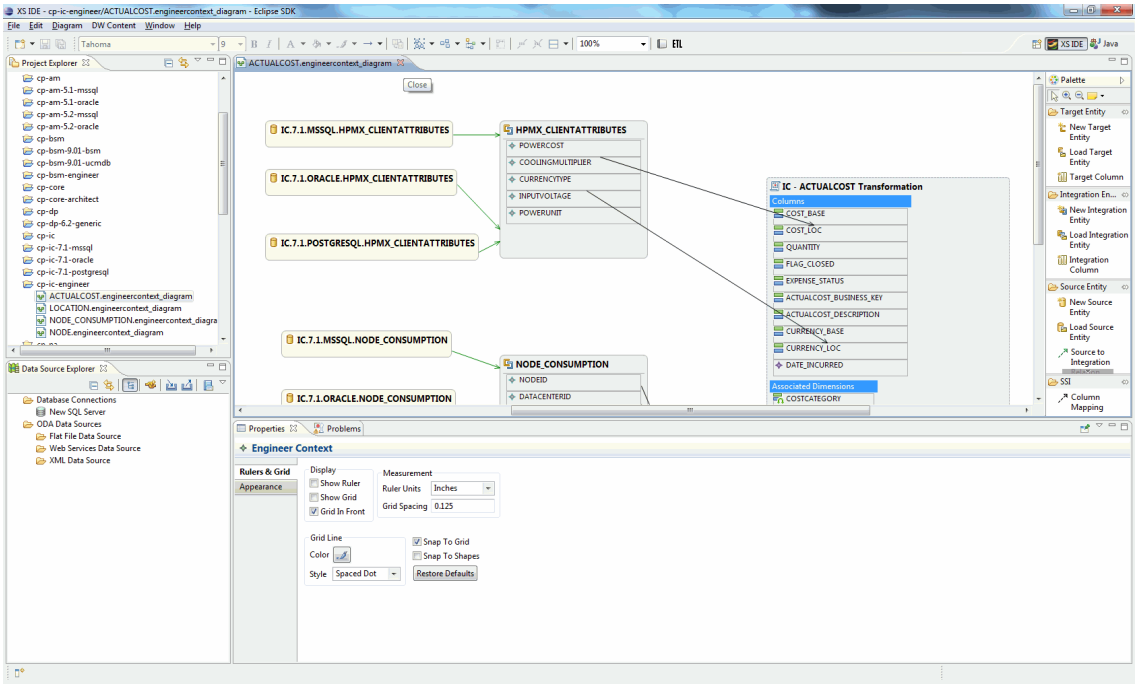
Tasks

Import the Extractor project into the Data Warehouse IDE

To import the Extractor project within the IDE, proceed as follows:

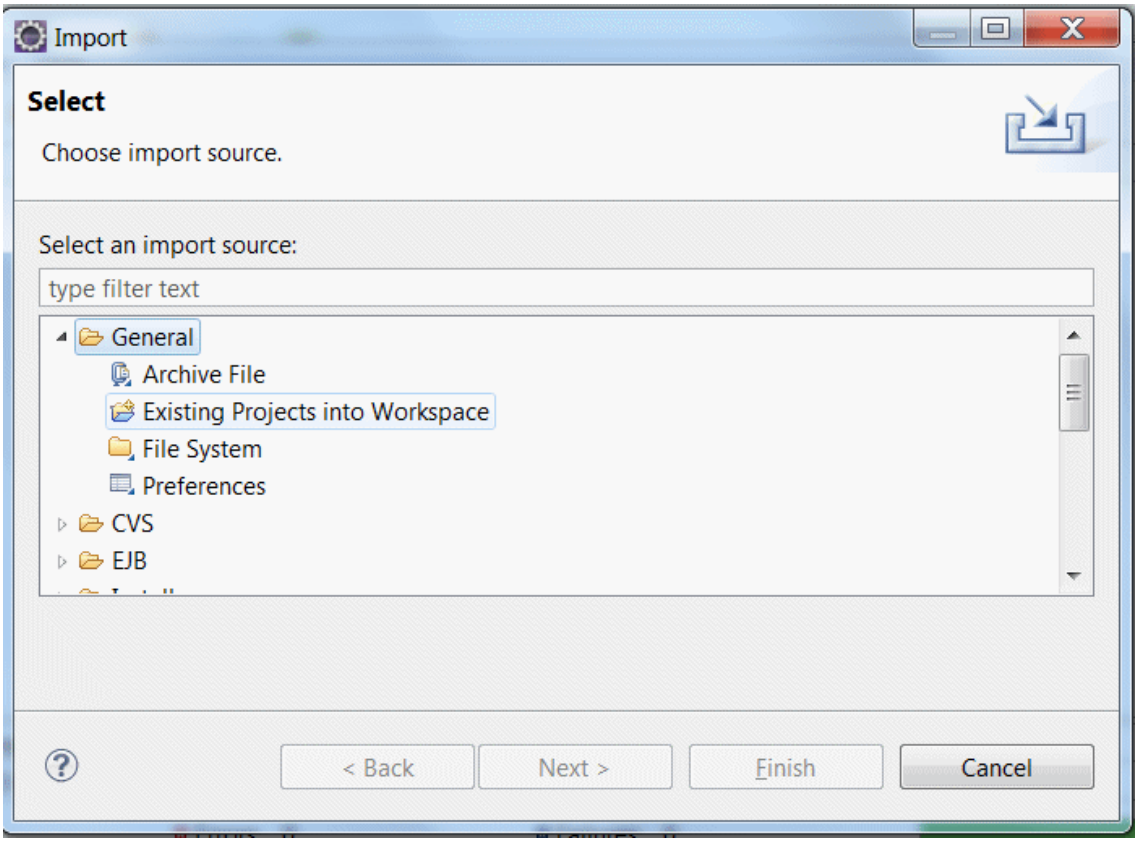
1. Download the Sample Project from the HP Live Network Portal: **Home** > **Executive Scorecard** > **Integration Content for Executive Scorecard**.

2. Open the IDE:

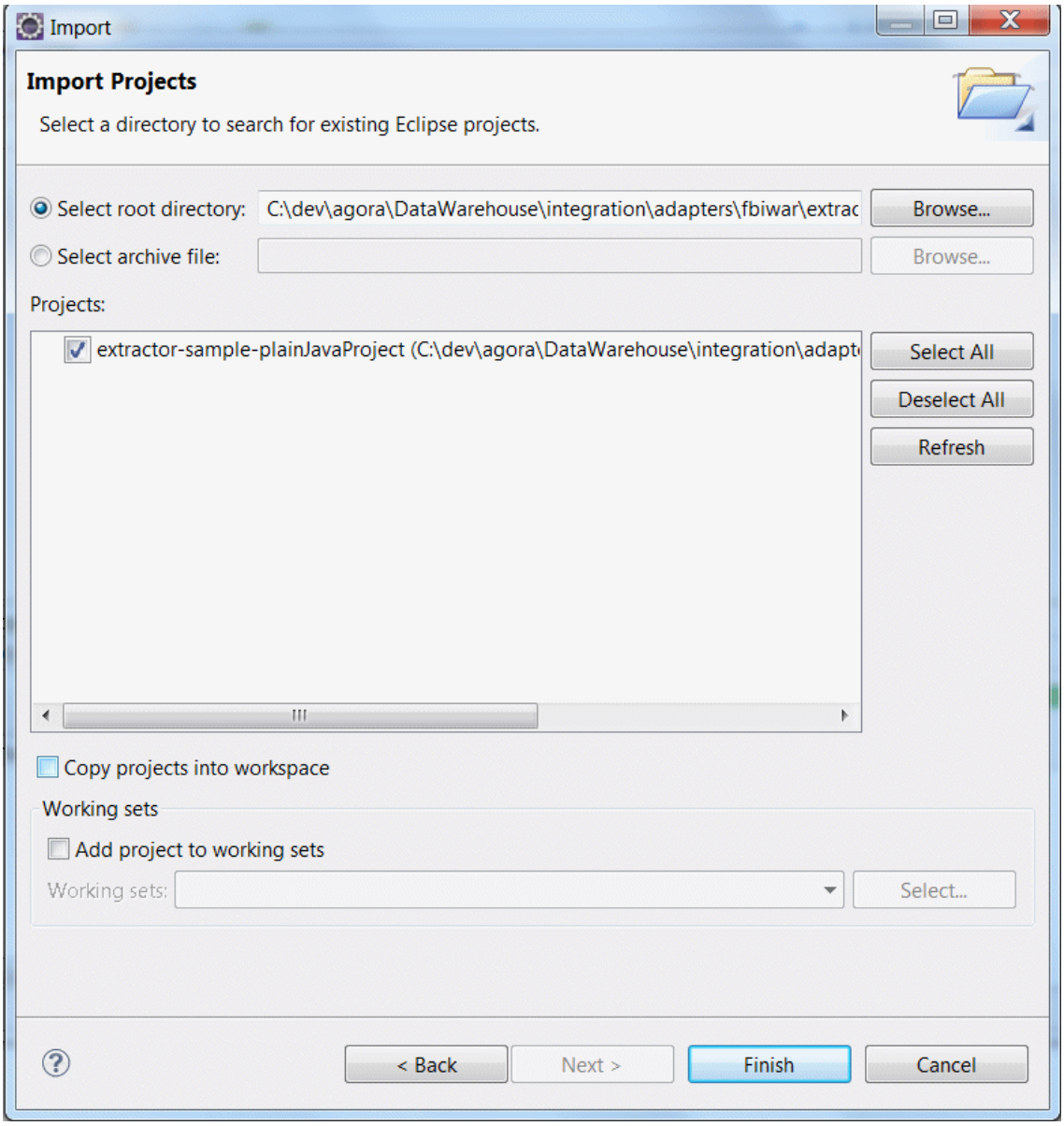


3. Import the downloaded Sample Project into the IDE:

a. Select **File** > **Import** > **Import** > **Existing Project into Workspace**.

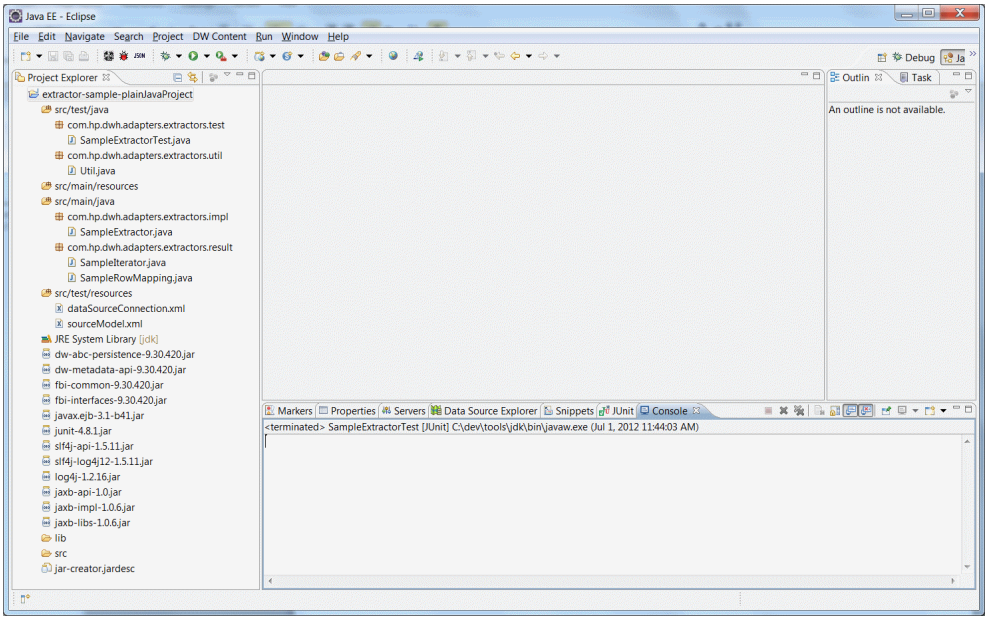


b. Browse to the **Downloaded Sample Project**.



c. Click **Finish** and the Sample project is added to the Workspace.

You can now see the development environment with all the sources.



4. Execute the Unit Test as described in "Test the Extractor (Unit Test)" on page 23. Using the Unit test, the Sample project can be adjusted as a starting point for your custom Extractor.

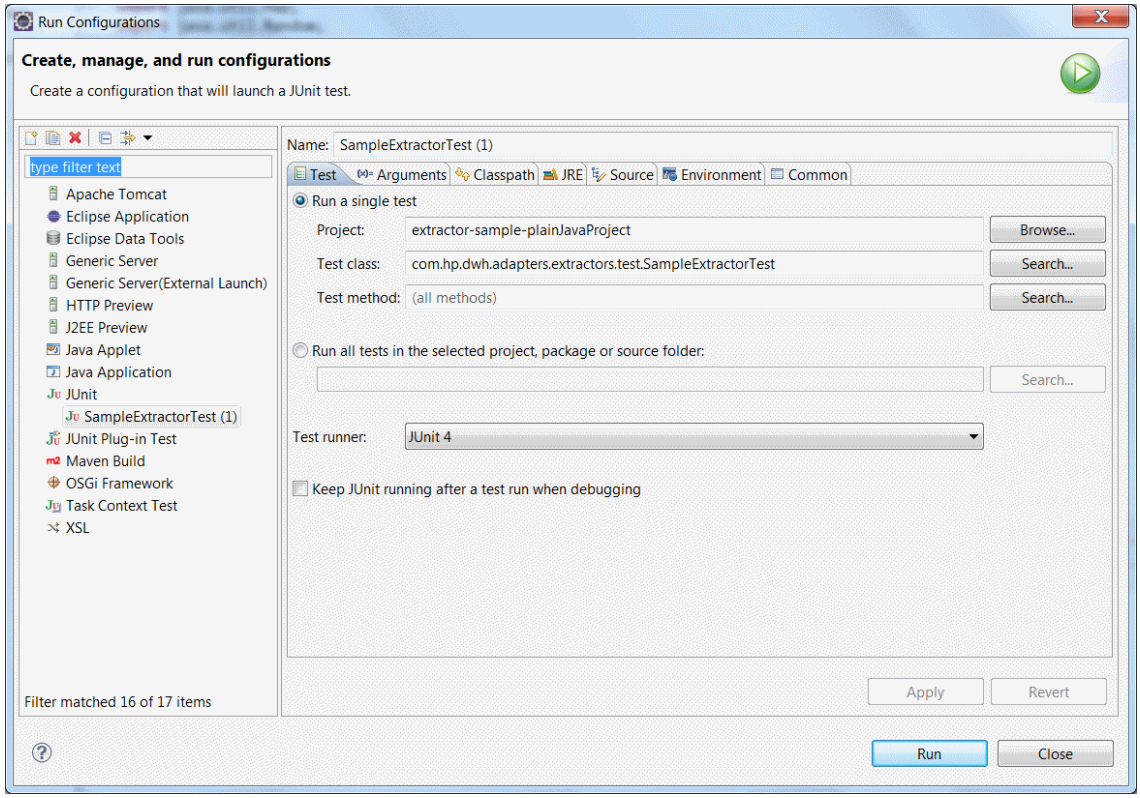
Test the Extractor (Unit Test)

You can access out-of-the-box resources and all IDE products that are required to test the Sample extractor. If you modify the Sample you can create your own unit testing by modifying the test files in order to check that your extractor works.

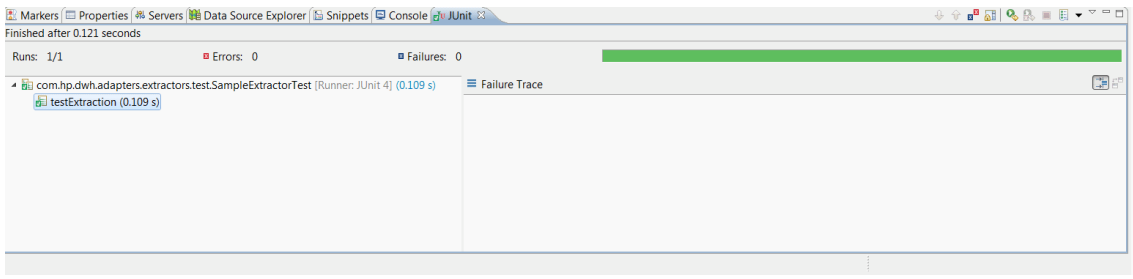
Tasks

Run the Unit Test on the sample Extractor, as follows.

1. Select **Run** > **Run Configurations** and define a new JUnit configuration:



2. Click **Run** and check that the test case completed successfully:



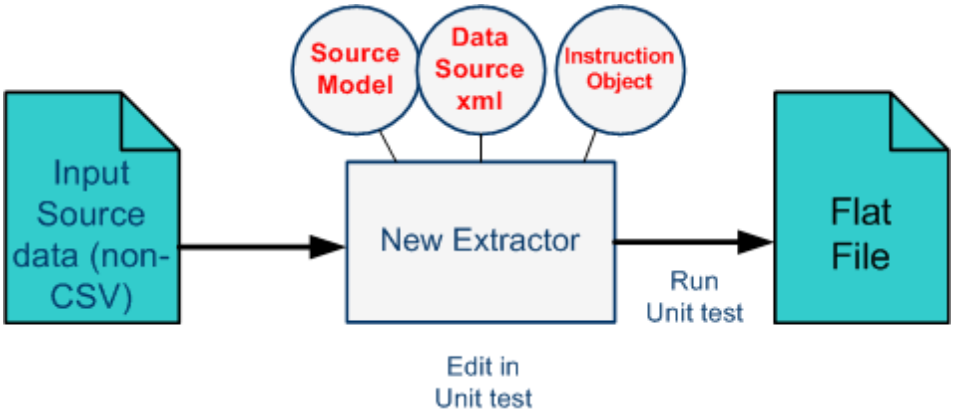
Modify the Sample Extractor

After you have successfully executed the Unit Test, you can now adjust the Sample project to develop a custom Extractor.

When the source data required to extract is from a protocol other than Oracle, MSSQL, or MySQL, for example Rest services, Web Services, or specific API, and not comprised of CSV files, you need to edit the Sample extractor project accordingly.

Each extractor receives the mandatory input required for the extraction process.

- **The Source Model:** Describes how the data should be extracted from the Source. This Configuration file is an output of the Content Pack development. For details, see the *Content Extension Guide*.
- **The Data Source xml:** A configuration file that provides the connection credentials to the source. For the Unit test, you must adjust this file according to the required source credentials.
- **The Instruction Object:** Contains the required information to perform the extraction.



You can modify the Sample in the Unit Test process, as follows:

- **Instruction:** The Instruction parameters are modified as required for your source entity. The Instruction record is automatically created in the IDE when you generate the ETL for new

content.

```
private Instruction getInstruction(DataSourceConnectionType dataSource, S
    Instruction instruction = new Instruction();
    instruction.setEncoding("UTF-8");
    instruction.setEntity(sourceModel.getDwid());
    instruction.setInstructionId(99);
    instruction.setProductInstance(new Long(dataSource.getCPID()).intValu
    instruction.setProductName(sourceModel.getProductname());
    instruction.setProductType(dataSource.getProductType());
    instruction.setFieldsDelimiter("  __|__  ");
    instruction.setProductVersion(dataSource.getProductVersion());
    instruction.setResultFileName("out.txt");
    return instruction;
}
```

- **sourceModel.xml:** Replace the sourceModel.xml (location: <sample_ extractor>\src\test\resources) with the metadata provided by the IDE.
- **dataSource.xml:** Modify the dataSource.xml (location: <sample_extractor>\src\test\resources) with the connection details for your extractor.

Develop a Script Based Extractor

In a script based FBI Extractor, the Script Extractor loads a script to perform extraction. Using this method does not require packaging or deployment and the script is stored in and released as part of the Content Pack. You can make changes to the script at run time.

Learn More

- A script is a small program written for a command interpreter. The Script Extractor mechanism has no limit on the type of script languages. Any script language that supports Java Script API will work in the FBI Extractor.The script should be named after the entity, except for TEST_ CONNECTION script file. The script file extension should be the script type, such as Groovy or Ruby.
- Script Extractor will extend Base Extractor and reuse the tools of the FBI.
- **Script Location:**
  - Script should be saved inside the script folder created under the content pack. If there are versions related to the script, create a sub folder for the specific version.

    For example: C:\<Installation Directory>\agora\ContentPacks\VCM\script\entity4.ruby

  - After the modification of the script, no other action is required to run an ETL process. The script is released as a part of the content pack.

    Note: The script must be saved on the same machine as the FBI Extractor.

- **Script Components**
  - Test connection: There must be a method called TEST_CONNECTION() that will throw an exception if the connection failed.

```
public TEST_CONNECTION() {
        System.out.println("SM dummy test connection ....");
        System.out.println("Always return true...");
        return true;
}
```

  - Entities:

- hasNext(). Indicates whether there is a next row.
- Next(). Get the next row. The row should be a map. The key is the column name and the value is the content formatted according to the request.

  The framework will call hasNext() and next() one by one to fetch all the information from the script.

- **Parameters for Script:** Placed in a map passed to the script by the framework. You can access the map through JAVA_PARAMETER_MAP.get(paramenter_name). Constants are defined for parameters to fulfill the extraction.
  - public static final String CONNECTION_TYPE = "CONNECTION_TYPE";
  - public static final String CONNECTION_TYPE_GENERIC_PROPERTIES = "CONNECTION_TYPE_GENERIC_PROPERTIES";
  - public static final String LOGGER = "LOGGER";
  - public static final String INSTRUCTION = "INSTRUCTION";
  - public static final String SOURCE_MODEL = "SOURCE_MODEL";
  - public static final String SCRIPT_PATH = "SCRIPT_PATH";

  The instruction works with the script when the product type is specified.

SM Devices Example using Groovy

```
import com.hp.dwh.adapters.extractors.impl.ScriptConstants
import java.util.Map;
import java.util.HashMap;

initialized = false;
extractionCompleted = false;

def DATA_SIZE = -1;
def index = -1;
def data;
def logger;

//This can be overriden in unit test for mocking data.
protected HashMap<String, Object> getJavaParameterMap(){
	return JAVA_PARAMETER_MAP;
}

public void init(){
	def javaParametersMap = getJavaParameterMap();
```

```
	logger = getJavaParameterMap().get(ScriptConstants.Parmeters.LOGGER);
	def scriptPath = javaParametersMap.get(ScriptConstants.Parmeters.SCRIPT_P
ATH);
	if(logger.isInfoEnabled()){
		logger.info('Script path: ' + scriptPath);
	}
	instruction = javaParametersMap.get(ScriptConstants.Parmeters.INSTRUCTIO
N);
	if(logger.isInfoEnabled()){
		logger.info('Extract type: ' + instruction.getProductType());
	}

	data = initClass(scriptPath + '/SMDevicesData.groovy');
	data.init(javaParametersMap);
	index = 0;
	DATA_SIZE = 50;
	initialized = true;
	if(logger.isInfoEnabled()){
		logger.info('Groovy script iniliazed successfully.');
	}
}

protected initClass(scriptName){
	File sourceFile = new File(scriptName);
	Class groovyClass = new GroovyClassLoader(getClass().getClassLoader()).pa
rseClass(sourceFile);
	GroovyObject object = (GroovyObject) groovyClass.newInstance();
	return object;
}

//This should firstly be invoked by java code or external sources, then al
lowed to invoke hasNext()/next().
public void extract(){
	if (!initialized){
		init();
	}

	extractionCompleted = true;
}

public hasNext() {

	//To-do: Change java code to invoke extraction() method firstly before ex
ecuting hasNext().
	//It should have thrown an exception(Please extract firstly.) instead of
invoking extract() here.
```

```
	if(false == extractionCompleted){
		extract();
	}

	return index < DATA_SIZE;
}


public next() {
	index++;
	Map<String, String> map = new HashMap<String, String>();
	def data = data.generateDummyData();
	for(Parameter parameter: data){
		map.put(parameter.getName().toUpperCase(), parameter.getValue());
//		logger.info(parameter.getName() + "-->" + parameter.getValue());
	}
	if(logger.isInfoEnabled()){
		logger.info("Parameters size=" + map.size());
		logger.info("Paramerters: " + map.toString());
	}
	return map;
}


class Parameter{

	private String name;
	private String value;
	private String type;
	private long length;

	public Parameter(String name, String type, long length) {
		super();
		this.name = name;
		this.type = type;
		this.length = length;
	}

	public Parameter(String name, String value, String type, long length) {
		super();
		this.name = name;
		this.value = value;
		this.type = type;
		this.length = length;
	}
```

```
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getValue() {
                return value;
        }
        public void setValue(String value) {
                this.value = value;
        }
        public String getType() {
                return type;
        }
        public void setType(String type) {
                this.type = type;
        }
        public long getLength() {
                return length;
        }
        public void setLength(long length) {
                this.length = length;
        }

        @Override
        public String toString() {
                return "Parameter [name=" + name + ", value=" + value + ", type="
                                + type + ", length=" + length + "]";
        }

}
```

Tasks

To configure script based extraction:

1. Place the script in the Content Pack folder under: **C:\<Installation Directory>\agora\ContentPacks\<CP name>\script\<entity name>.<script extension>**

2. Specify **script** in the **productType** field of the xml file in the Content Pack folder (C:\<Installation Directory>\agora\ContentPacks\VCM\conf\dataSources.xml).

3. BODS sends a request to the FBI for the Content Pack and script extractor details.

4. The extractor mechanism starts extraction while the Script Extractor locates the script in the Content Pack.

Configure the Data Source Management User Interface for XS Integration

After the integration test, you must define the end point settings for the data source required for integration activation. The Data Source Management User Interface enables you to activate, deactivate and configure data source parameters based on the Content Pack data source xml file. For new content you can customize the datasourceDef.xml file and configure according to the requirements of your Content Pack. Once you have created a new Content Pack folder, it contains the xml configuration file under **<installation directory>\agora\ContentPacks\<CP name>\conf\datasourceDef.**

Any configuration changes made in the xml file can then be viewed in the Data Source Management UI. For details, see Perform Tasks for Data Source Management, in the *Administrator Guide*.

Tasks

Deploy the Configuration

The Data Source Management UI supports new data source types that can be extended at runtime. You deploy the configuration using the xml file listed below. It provides a dynamic list of attributes that are required to connect to a new source type.

The Data Source Activation process uses all of these attributes to connect to the data source. When you want to integrate a new data source data into Executive Scorecard, you create a directory for the data source.

For example: **C:\<Installation Directory>\agora\ContentPacks\<data_source_acronym>**

Each Content Pack includes a custom Data Source Management User Interface definition file in xml format (**dataSourceDef.xml**). It is located by default at: **C:\<Installation Directory>\agora\ContentPacks\<data_source_acronym>\conf**.

1. Configure the dataSourceDef.xml file that is used to create the Data Source Management UI page for the corresponding data source. The page is used by the administrator to enter the details of the data source.

   The following is a sample dataSourceDef xml file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dataSourceType typeId="AM">
       <fields>
              <field fieldType="LIST" required="false" fieldId="version">
                     <localizedLabel>
                            <key>AMVersion</key>
                            <defaultString>AM Version</defaultString>
                     </localizedLabel>
                     <options>
                            <option key="5.1" label="5.1"/>
                            <option key="5.2" label="5.2/9.3"/>
```

```
                        </options>
                        <preselectedValue>5.2</preselectedValue>
                </field>
                <field fieldType="LIST" required="false" fieldId="timezone">
                        <generatorClass>com.hp.btoe.dw.dsm.server.TimeZonesGenerator</generator
ass>
                        <localizedLabel>
                                <key>wizardTimeZone</key>
                                <defaultString>Time Zone</defaultString>
                        </localizedLabel>
                </field>
                <field fieldType="SPACER" >
                </field>
                <field fieldType="LIST" required="false" fieldId="productType">
                        <localizedLabel>
                                <key>wizardDBProductName</key>
                                <defaultString>Data Source Type</defaultString>
                        </localizedLabel>
                        <options>
                                <option key="mssql" label="MSSQL"/>
                                <option key="oracle" label="Oracle"/>
                        </options>
                        <preselectedValue>mssql</preselectedValue>
                </field>
                <field fieldType="SPACER" >
                </field>
                <field fieldType="TEXT" required="true" fieldId="user">
                        <localizedHint>
                                <key>wizardUsernameHint</key>
                                <defaultString>&lt;&lt;Enter username&gt;&gt;</defaultString>
                        </localizedHint>
                        <localizedLabel>
                                <key>wizardUsername</key>
                                <defaultString>Username</defaultString>
                        </localizedLabel>
                </field>
```

a. The xml contains, per Content Pack, the definition of the different properties. Each Property field can have the following types:

   - **LIST.** The user must select from a list of parameters.
   - **TEXT.** The user must enter the relevant text in the field.
   - **SPACER.** Configures a space between fields in the UI.
   - **PASSWORD.** The user enters an encrypted password in the field.
   - **BOOLEAN.** The user selects a checkbox.

b. The following options can be configured for field values:

- `preselectedValue.`

  The displayed default value.

  For example, for the **AM Version** field the default value is **5.2**.

```
<key>AMVersion</key>
<defaultString>AM Version</defaultString>
</localizedLabel>
<options>
<option key="5.1" label="5.1"/>
<option key="5.2" label="5.2/9.3"/>
</options>
<preselectedValue>5.2</preselectedValue>
</field>
```

- **`required.`**

  A mandatory field. Mandatory fields are indicated by a red asterisk in the UI.

  For example, the **Username** field is mandatory:

```
</field>
<field fieldType="TEXT" required="true" fieldId="user">
<localizedHint>
<key>wizardUsernameHint</key>
<defaultString>&lt;&lt;Enter username&gt;&gt;</defaultString>
</localizedHint>
<localizedLabel>
<key>wizardUsername</key>
<defaultString>Username</defaultString>
</localizedLabel>
</field>
```

- **`dependentField/Value.`**

  A field that is displayed only when a specific value is selected.

  For example: The string **SID** is only enabled when the selected value in the dependent **productType** field is **Oracle**.

```
<localizedLabel>
<key>wizardSid</key>
```

```
<defaultString>SID</defaultString>
</localizedLabel>
<dependentField>productType</dependentField>
<dependentValues>oracle</dependentValues>
</field>
```

- **localizedHint.**

  Text that is displayed to assist the user in entering or selecting a value.

  For example, the **Username** field displays a hint for the user: "Enter username"

```
<field fieldType="TEXT" required="true" fieldId="user">
<localizedHint>
<key>wizardUsernameHint</key>
<defaultString>&lt;&lt;Enter username&gt;&gt;</defaultString>
</localizedHint>
<localizedLabel>
<key>wizardUsername</key>
```

2. **Deploy the Configuration Files:** When you have completed the xml file, save it as a dataSourceDef.xml in the conf directory of the Content Pack directory of the Content Pack you are creating for the data source you want to integrate with Executive Scorecard.

   <installation directory>\agora\ContentPacks\<CP name>\conf\datasource.xml

Package and Deploy the Extractor- JAVA Based Only

The next stage in the FBI Extractor process is to package and deploy the extractor. You can perform the build,and then deploy it on the server.

This section contains the following:

"Package and Build the Extractor" on page 36

"Deploy the Extractor" on page 39

Package and Build the Extractor

The next step after testing is to perform the compilation and packaging process of the FBI extractor project. You can perform the build on the sample project or modify and make changes to create your own extractor.

Learn More

Dependencies

The following is a list of jars required when performing a build. If you are not modifying a sample project, make sure your extractor contains this list.

To build an Extractor you need the following jars:

- dw-abc-persistence-<version>.jar
- dw-metadata-api-<version>.jar
- fbi-common-<version>.jar
- fbi-interfaces-<version>.jar
- javax.ejb-3.1-b41.jar
- jaxb-api-1.0.jar
- jaxb-impl-1.0.6.jar
- jaxb-libs-1.0.6.jar
- junit-4.8.1.jar
- log4j-1.2.16.jar
- slf4j-api-1.5.11.jar
- slf4j-log4j12-1.5.11.jar

Dependencies Location

You can find the dependencies in the lib folder of the Sample project.

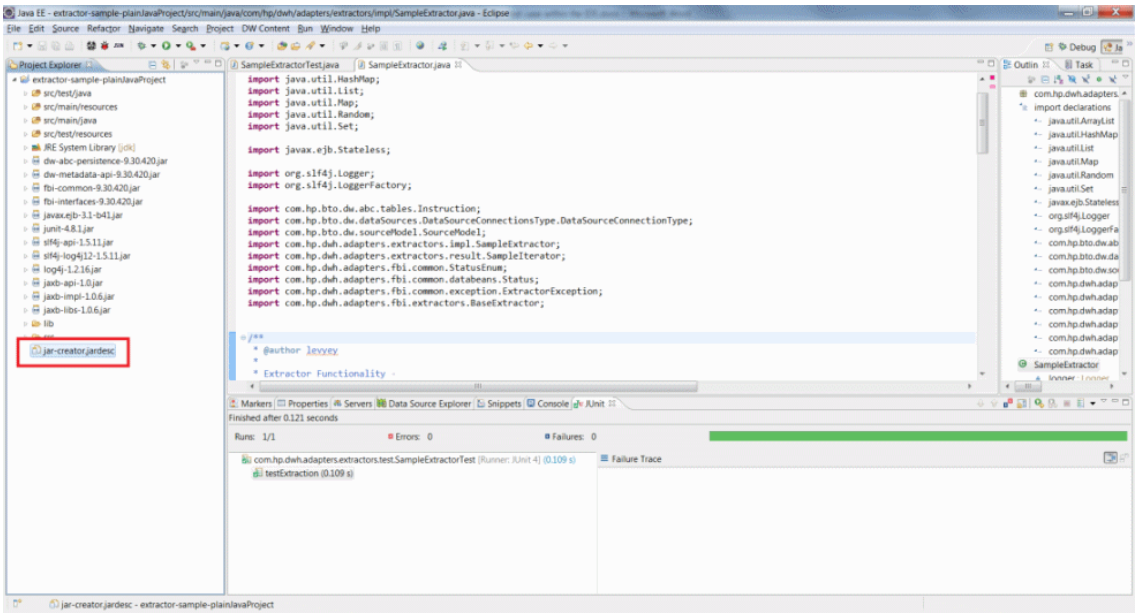
Tasks

To package and build the Sample Project in the Data Warehouse IDE:

> **Note:** To change or edit the build process, all jar properties must be adapted.

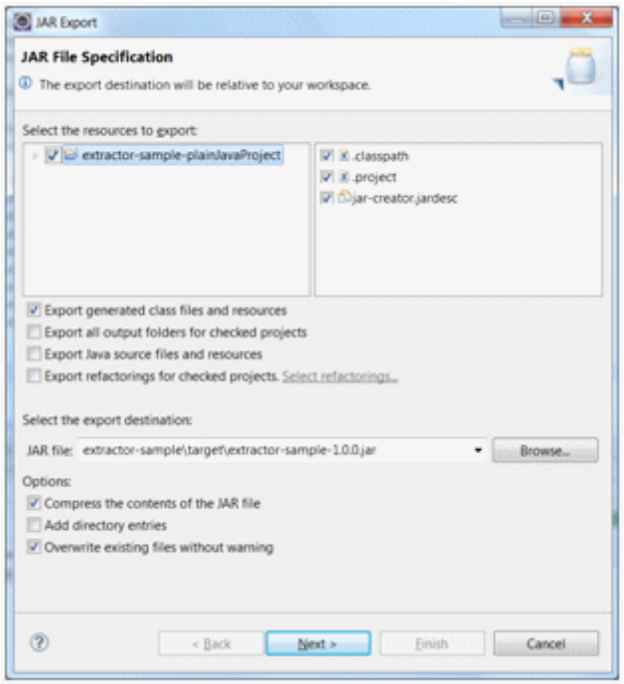
1. Build a sample project in the Data Warehouse IDE:

   a. Double-click **jar-creator.jardesc** in the Project Explorer. This jar descriptor is part of the sample project.



   The Jar File Specification page opens.

   b. Select and enter the relevant export destination parameters.



c. Check that you can find the packaged **JAR -extractor-sample-1.0.0.jar** in the predefined exported destination. If you have modified the sample extractor, make sure the name reflects your changes and click **Next**.

d. Click **Finish** to confirm your configuration. The Jar is saved in the configured location.

Deploy the Extractor

After packing the extractor into a Jar file, you can take the jar from the Sample project and deploy it on the server. If you have modified the sample, make sure that is reflected in the jar details. Perform the following steps to deploy the compiled custom extractor.

Tasks

To deploy and run the Sample extractor.

1. Build the extractor Sample Project: **extractor-sample-1.0.0.jar**.
2. On the Data Warehouse server, undeploy **fbi-war** using the following command: **%BTOA_HOME%\bin\fbi_undeploy.bat**.
3. On the Data Warehouse server, copy the **extractor-sample-1.0.0.jar** file to the following location: **%BTOA_HOME%\apps\fbi-web\WEB-INF\lib**.
4. On the Data Warehouse server, redeploy the fbi-war web application using the following command: **%BTOA_HOME%\bin\fbi_deploy.bat**.

Troubleshooting

Troubleshooting Logs

- **<installation directory>\agora\glassfish\glassfish\domains\BTOA\logs\fbi.log**: This log describes all of the current activity of the file based framework as well as the activity of the extractors themselves.
- **<installation directory>\agora\DataWarehouse\log\dw_fbi_client.log**: This log describes the activity of calling the FBI url from a batch file (which is invoked by a BODS workflow during the ETL source extract).

We appreciate your feedback!

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on IT Executive Scorecard, 9.4 FBI Extractor SDK Guide

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to SW-Doc@hp.com.


Java™
COMPATIBLE