

Oracle® Product Data Quality

Java Interface Guide

Version 5.5

June 2010

Copyright © 2001, 2010, Oracle and/or its affiliates. All rights reserved.

Primary Author: Lorna Vallad

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	vii
Audience	vii
Related Documents	vii
Conventions	viii
1 Overview	
Overview of the Oracle DataLens Server APIs	1-1
APIs	1-1
Platforms	1-2
Pre-Installation Requirements	1-2
Oracle DataLens Server Java Libraries	1-2
2 DSA API to the Oracle DataLens Server	
WfgClient	2-1
Updating Individual records and Data Lines	2-1
Transforming Data	2-1
Import	2-1
Initialize the Client	2-2
Create the List of Input Data	2-2
Transform a List of Data	2-3
Alternative Method of Transforming Data	2-3
Retrieve Results from the Server for Jobs with a Single Output Step	2-3
Synchronous Method	2-3
Asynchronous Method	2-4
Retrieve Results from the Server for Jobs with Multiple Output Steps	2-4
Pulling the Result Data from the List	2-4
List Data	2-4
Tab-Separated Data	2-5
Listing Multiple DSA Jobs	2-5
Listing a Single DSA Job	2-6
Using File Input and Output	2-6
Miscellaneous Settings for the WfgClient	2-6
Retry Count	2-6
Filter Data	2-7
Job Priority	2-7

Run-Time Locale	2-7
Separator Character	2-7
Client-Side Debugging Toggle	2-8
Email Output	2-8
FTP Output.....	2-8
Database Parameters.....	2-9
3 Server Information API to the Oracle DataLens Server	
InfoClient	3-1
Getting Transform Map and Data Lens Information.....	3-1
Import	3-1
Initialize the Client.....	3-1
Get a List of Deployed Data Lenses.....	3-1
Lists of Schemas and Translations.....	3-2
Get a List of Deployed DSAs	3-2
4 Server Availability API to the Oracle DataLens Server	
PingClient	4-1
Import	4-1
Simple Server Check.....	4-1
Round-Robin Server Check	4-1
5 Error Handling	
Client-Side Exceptions	5-1
Client-Side Log Messages	5-1
Log4j to Standard Output	5-1
Log4J to a File.....	5-2
Server-Side Faults	5-2
Server-Side Exceptions	5-2
Server-Side Log Messages	5-3
Debugging Client Requests and Responses	5-3
6 Compiling and Running with the API	
Compile the Application with the Oracle DataLens Libraries	6-1
Run the Application with the Oracle DataLens Libraries	6-1
7 Web Service Access to the Oracle DataLens Server Using Doc-Lit	
Generating a WSDL Document on Demand	7-1
Client Web Service Software	7-1
Overview of the DSA Interface	7-1
processListRequest and processOneLineRequest Operations.....	7-2
processDBRequest.....	7-2
SOAP Document-Literal One Line Request Example.....	7-2
SOAP Document-Literal One Line Response Example.....	7-3
SOAP Doc-Lit Multi-Line ProcessList Request Example	7-3

SOAP Doc-Lit Multi-Line ProcessList Response Example	7-3
SOAP Document-Literal ProcessDb Request Example	7-4
SOAP Document-Literal processDb Response Example.....	7-4

8 Customizing DSA Maps with Java Add-Ins and Algorithms

TMap Algorithms	8-1
Initial Configuration	8-1
Client Startup Changes	8-1
Creating a New TMap Algorithm.....	8-2
TMap Algorithm Debugging.....	8-4
Server	8-4
Client.....	8-4
TMap Add-In Transforms	8-4
Writing a TMap Add-In Transform.....	8-4
Defining the TMap Add-In Transform	8-5
Server	8-5
Defining the Input Parameters to the TMap Add-In Transform	8-6
Using the TMap Add-In Transform in the Client.....	8-6
DSA Add-In Outputters	8-7
Writing a DSA Add-In Outputter.....	8-7
Defining the DSA Add-In Outputter	8-8
Server	8-8
Defining the Input Parameters to the TMap Add-In Transform	8-9
Using the DSA Add-In Outputter in the Client.....	8-9
Use in the Application Studio	8-9

A Oracle DataLens Server JAVA API Reference

B Working Through a Proxy Server

Run-Time Java Proxy Parameters	B-1
RtClient Java Proxy Parameters	B-1

C Installing the Client Software

D Deprecated: Web Service Access to the Oracle DataLens Server using RPC

Generating a WSDL Document on Demand	D-1
Client Web Service Software.....	D-1
Overview of the DSA Interface	D-2
processListRequest and processOneLineRequest	D-2
processDBRequest.....	D-2
SOAP RPC One-Line Request Example.....	D-3
SOAP RPC One-Line Response Example	D-3

Preface

This guide is intended to explain the basic capabilities of the Oracle DataLens Server Java Interface.

To understand all of the features presented, you must use this guide in conjunction with the Oracle Product Data Quality documents listed in "[Related Documents](#)" on page -vii.

You must have Oracle Product Data Quality client software installed on your computer.

Audience

You should have a basic understanding of the DataLens Technology.

This document is intended for all users of the DataLens Technology, including:

- Subject Matter Experts (SMEs)
- IT Administrators

Related Documents

For more information, see the following documents in the documentation set:

- The *Oracle Product Data Quality Oracle DataLens Server Installation Guide* provides detailed Oracle Product Data Quality Oracle DataLens Server installation instructions.
- The *Oracle Product Data Quality Oracle DataLens Server Administration Guide* provides information about installing and managing an Oracle DataLens Server.
- The *Oracle Product Data Quality COM Interface Guide* provides information about installing and using the Oracle DataLens Server COM APIs.
- The *Oracle Product Data Quality Application Studio Reference Guide* provides information about creating and maintaining Data Service Applications (DSAs).
- The *Oracle Product Data Quality AutoBuild Reference Guide* provides information about creating initial data lens based on existing product information and data lens knowledge.
- The *Oracle Product Data Quality Knowledge Studio Reference Guide* provides information about creating and maintaining data lenses.
- The *Oracle Product Data Quality Glossary* provides definitions to commonly used Oracle Product Data Quality technology terms.

- The *Oracle Product Data Quality Services for Excel Reference Guide* provides information about creating a DSA based on data contained in a Microsoft Excel spreadsheet.
- The *Oracle Product Data Quality Task Manager Reference Guide* provides information about managing tasks created with the Task Manager or Governance Studio applications.

See the latest version of this and all documents listed at the Oracle Product Data Quality Documentation Web site at:

http://download.oracle.com/docs/cd/E17135_01/index.htm

Conventions

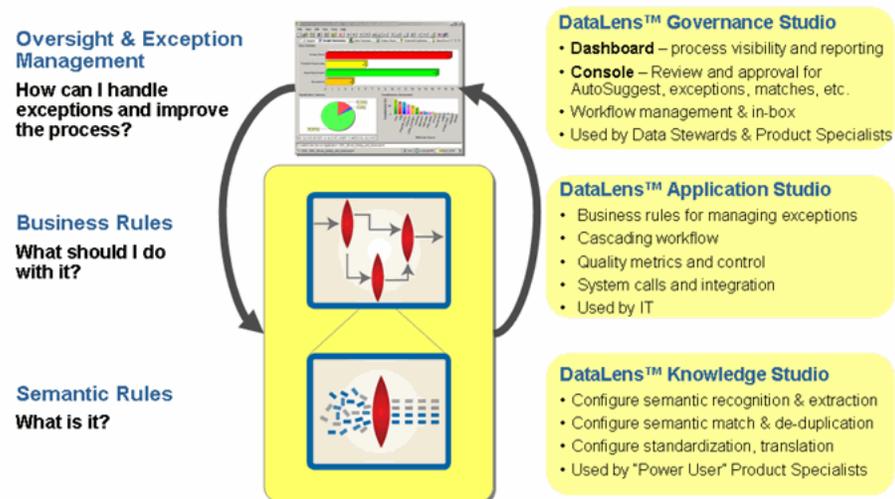
The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, text that you enter, or a file, directory, or path name.
monospace	Boldface, monospace type indicates commands or text that you enter.

Overview

Oracle DataLens Server is built on industry-leading DataLens™ Technology to standardize, match, enrich, and correct product data from different sources and systems. The core DataLens Technology uses patented semantic technology designed from the ground up to tackle the extreme variability typical of product data.

Oracle Product Data Quality uses three core DataLens Technology modules: Governance Studio, Knowledge Studio, and Application Studio. The following figure illustrates the process flow of these modules.



Overview of the Oracle DataLens Server APIs

APIs

There are three main Application Programming Interfaces to the Oracle DataLens Server platform.

- **DSA Client** - The Oracle DataLens Server DSA Client interface. This is used for direct access to the DSA loaded on the Oracle DataLens Servers for processing application/enterprise data. The DSA Client can process the following:
 - Tab-separated input data
 - Input data with any user-defined separator character

- Input data from a database query
- **Information Client** - The Oracle DataLens Server Information Client interface. This is used for access to information about the data lenses, Transform Maps and DSAs that are loaded on the Oracle DataLens Servers. This is useful to check what data lenses and maps are available from a client application.
- **Ping Client** - The Server Availability Client interface. This is used to check for a response from any Oracle DataLens Server in an Oracle DataLens Server Group.

There is also a low-level Application Programming Interface to the Oracle DataLens Server platform. This interface is not recommended for application developers. If this interface is needed, then Oracle Product Data Quality Professional Services should be contacted to get assistance on the best practices and use of this interface to the Oracle DataLens Servers.

- **Oracle DataLens Client** - The Real-Time Client interface. This is used for direct access to the data lenses loaded on the Oracle DataLens Servers for processing application data. The RT Client can process the following:
 - Single line of data
 - Array of data
 - List of data

Platforms

- The Java API can be used for integrating to the following.
 - o Java applications or Web pages
 - o Available on MS Windows or UNIX operating systems

Pre-Installation Requirements

Java JDK 1.4.2_02 - The API was compiled and built using this release.

Java JDK 1.5.0_01, 1.6.0_04, 1.6.0_11- The API is also compatible with these releases.

Oracle DataLens Server Java Libraries

This guide assumes that the reader is an experienced Java software developer.

All libraries and software needed for use by this API are loaded as part of the Oracle DataLens Knowledge Studio and Oracle DataLens Server installations.

The Libraries that are needed are included on the installation media. Two libraries are needed for creating new applications and integrating to existing applications.

ScsApi.jar

This library contains the Application interface classes.

ScsApilmports.jar

This is the library that contains the 3rd party components, needed by the API.

There is one 3rd party package used in this library.

Jdom 1.0

This is used for encoding/decoding the SOAP messages sent to the Oracle DataLens Server from the client application.

Note: If there are issues with using these versions of this software, please contact Oracle Product Data Quality Professional Services.

DSA API to the Oracle DataLens Server

WfgClient

Use of this interface is the preferred method to access any data transformations that need to be done on the Oracle DataLens Server. This interface will directly execute the DSAs that are deployed to the Oracle DataLens Server. Use of this interface should supplant use of the Oracle DataLens API.

The `WfgClient` class is used as an interface to the Oracle DataLens Server. This class provides methods to perform DSA transformations.

Updating Individual records and Data Lines

DSAs are not a one-to-one match of the input records and the output records. In some cases this may be true, depending on the map. More likely, there will be multiple output steps, and each step will only have a subset of the input data results. In some output steps, there may be no data returned, and in other cases there may be multiple output records returned for a single input record.

This means that the DSAs should pass the original Id into the processing, usually as the first data field. This provides a means for matching the output result data with the original input data.

In cases where data is just being processed, and there is no need to link the results back to each individual input record, then passing the ID through the DSA is not needed.

Transforming Data

Import

Import the `WfgClient` with the following lines:

```
import com.onerealm.solx.api.client.WfgClient;
import com.onerealm.solx.api.client.WfgResultLine;
import com.onerealm.solx.api.client.WfgRequestLine;
import com.onerealm.solx.api.bean.Fault;
import com.onerealm.solx.api.iface.ErrorIF;
import com.onerealm.solx.api.util.Priorities;
```

Initialize the Client

An instance of the WfgClient class needs to be created with the Oracle DataLens Server name and port.

Actual parameters:

- **Server Name** - This can be either a machine name (such as, "Production") or an IP address (such as "127.0.0.1").
- **Server Port** - This is the port number of the server. By default, the Oracle DataLens Server is installed on port 2229.
- **Encryption flag** - False uses normal HTTP communication; true uses the secure HTTPS. Use **false** unless instructed otherwise by Oracle Product Data Quality Professional Services.
- **Client Code** - This is the "secret code" that the Oracle DataLens Server provides with your server license to prevent unauthorized access to the Oracle DataLens Server via this API. This code is built into the Oracle DataLens Server license and is only active if requested as part of the license. This value can be left blank if the server license has no code.
- **Application** - This application name initiated the client request to the server. This name is used to accumulate server statistics on the Oracle DataLens Server Administration Web Pages.

```
// Create WfgClient object
WfgClient wfgClient;
    wfgClient = new WfgClient(serverName, SERVER_PORT, ENCRYPTION,
                            clientCode, APPLICATION);
```

Create the List of Input Data

This is a brief example of creating the input data list. First we will create the list from an array of static data as shown below.

```
private static final String m_inputData[][] = {
{"0", "Res, 20 Ohm"},
    {"1", "Res, Net 4 W"};
```

This data above is just an example. Your data will come from your application, from an input file or a database query.

In any case, the data needs to be put into an input list for the Java API to process the data. Below is an example of creating and populating the list using the example data. In this case, the input data needs to be separated using the character separator, in this case the Tab character. This interface is best used when there is only one field of input data to be processed.

```
// Setup this list of String Fields for the request
List list = new ArrayList();
    for (int i=0; i<m_inputData.length; i++) {
        List fields = new ArrayList(); // Create a List of Strings
        fields.add(new String(m_inputData[i][0])); // Add the ID Data Field
        fields.add(new String(m_inputData[i][1])); // Add the Description Field
        list.add(fields);
    }
```

Transform a List of Data

A list is passed to the `runRtJob` method and a single job ID is returned. The `runRtJob` Method is called just a single time with a single list of data.

Actual parameters:

- **Job ID** - The DSA Job ID obtained from the `runJob` call.
- **DSA Name** - The name of the DSA to run on the Oracle DataLens Server.
- **Description** - A description of this particular job.
- **List** - The list with a list of String input fields.

```
// Start the DSA job with our data
// NOTE: Input data with a List containing a list of string attributes.
//       This is useful for already separated data
m_wfgClient.setLinesFromFields(list);
int m_jobID = m_wfgClient.runJob(PMapName, "My Job");
```

The preceding call is using the following default values:

- Job Priority of medium
- Job run-time locale of USA English

Alternative Method of Transforming Data

In any case, the data needs to be put into an input list for the Java API to process the data. Following is an example of creating and populating the list using the example data. In this case, the input data needs to be separated using the character separator. The following example uses the Tab character. This interface is best used when there is only one field of input data to be processed.

```
List list = new ArrayList();
for (int i=0; i<m_inputData.length; i++) {
    list.add(new WfgRequestLine(m_inputData[i][0] + "\t" + m_inputData[i][1]));
}
```

Now process the data using the list you created:

List

The list of `WfgRequestLine` objects that have been initialized with the tab-separated input data.

```
// Start the DSA job with our data
wfgClient.setLines(list);
int m_jobID = wfgClient.runJob(PMapName, "Comment: API Test Job 1");
```

Retrieve Results from the Server for Jobs with a Single Output Step

When the job has finished, the transformed data can be retrieved from the Server back to the client application. The `getResultData` method is called just a single time and returns a list of `WfgResultLine` objects containing the result data.

Synchronous Method

The call will wait until the job has finished processing the data before control is returned to the program with the result data.

```
// Get the DSA Results!
```

```
boolean waitForResults = true;
resultData = wfgClient.getResultData(jobID, waitForResults);
```

Asynchronous Method

You can check the job status and do other processing while waiting for the job to complete. The `getResultData` method will throw a fault indicating that the job is still processing the input data.

```
try {
// Get the DSA Results!
    boolean waitForResults = true;
    resultData = wfgClient.getResultData(jobID, waitForResults);
} catch (Fault f) {
    // Check if the job has not completed yet
    if (f.getErrorCode() == ErrorIF.ERROR_NOT_COMPLETED)
        . . .
}
```

Server Faults that can be thrown from a call to `getResultData` include the following `ErrorIF` errors:

`ERROR_JOB_CANCELED`

`ERROR_CANCEL_FAILED`

`ERROR_COPY_FAILED`

`ERROR_JOB_FAILED`

`ERROR_NOT_COMPLETED`

Retrieve Results from the Server for Jobs with Multiple Output Steps

When the job has finished, the transformed data can be retrieved from the Server back to the client application. The `getResultData` method is called just a single time for each output step. Each call returns a list of `WfgResultLine` objects containing the result data, just as with the jobs with a single output step.

The call will wait until the job has finished processing the data before control is returned to the program with the result data.

```
// Get the DSA Results!
resultData = wfgClient.getResultData(jobID, stepName, waitForResults);
```

The call to `getResultData` can be made synchronously or asynchronously as demonstrated above.

Pulling the Result Data from the List

The result data is returned as a list of `WfgResultLine` objects.

List Data

This is how the result data fields should be pulled from the output lines. This list interface will always maintain all the columns of output data, even if there is no data for a particular output data field. In this case, the data field result will be a null value.

The following code excerpt demonstrates pulling out the individual data lines, with the individual data fields.

```

// Iterate through the result data lines
Iterator iter = resultData.iterator();
while (iter.hasNext()) {
    WfgResultLine resultLine = (WfgResultLine)iter.next();
    List outFields = resultLine.getDataFields();

    // Iterate through the result data fields
    Iterator i2 = outFields.iterator();
    while (i2.hasNext()) {
        String outField = (String)i2.next();
        System.out.print(outField);
        If (i2.hasNext())System.out.print(", ");
    }
    System.out.println(" ");
}

```

Tab-Separated Data

This is a simple way to get to the result data for testing. The following code excerpt demonstrates pulling out each line of tab-separated output data.

Note: This example works if you have specified an alternate separator character.

```

Iterator iter = resultData.iterator();
while(iter.hasNext()) {
    WfgResultLine resultLine = (WfgResultLine)iter.next();
    System.out.println(resultLine.getData());
}

```

Listing Multiple DSA Jobs

The DSA Client can list Jobs can be listed from the Oracle DataLens Server Administration Web Pages. The following types of lists can be retrieved from the server.

- All Jobs (also since a particular date)
- All jobs that have not completed
- All jobs for a particular submitter (also since a particular date)
- All not-completed jobs for a particular submitter
- All jobs for a particular approver

The following code shows the calls in the order listed in the preceding:

```

List list = wfgClient.listAllJobs(sinceTS);
List list = wfgClient.listNCJobs();
List list = wfgClient.listSubmitterJobs(submitter, sinceTS);
List list = wfgClient.listNCSubmitterJobs(submitter);
List list = wfgClient.listApproverJobs(approver);

```

These calls all return lists of WfgJobInfo objects.

Listing a Single DSA Job

Information can also be obtained from a single job given the Job ID. The following Java code example shows this:

```
WfgJobInfo jobInfo = wfgClient.listJob(jobID);
```

This call returns a single `WfgJobInfo` object with the job details.

Additionally, all the details on the steps are returned as well. To get the steps, use the `getSteps` method call as shown in the following example:

```
List steps = jobInfo.getSteps();
```

These steps are a list of `WfgJobStepInfo` objects with all the details on the individual job steps.

Using File Input and Output

The DSA API can use a text file as input and a text file as output. The complete path to the input file and the complete path to the output directory are needed. Use the setters to toggle on the input/output directory locations as in the following example:

```
// Setting the input file and output directory toggles on file processing
wfgClient.setOutputDirectory(outputLocation);
wfgClient.setInputFilePath(filePath);
jobID = wfgClient.runJob(transformProcess, desc);
```

These file input paths and the file output paths are sent directly to the Oracle DataLens Server. This means that the paths must be paths that are relative to the server. For example, if you give the path to an input file as:

```
C:/temp/raw_data.txt
```

This file is from the C drive on the server machine, not the C drive on the client machine. The output directory is also a relative path from the server machine as well.

The source path can be a UNC path to a file on a remote machine.

Here is an example:

```
//node_name/shared/test.txt
```

Miscellaneous Settings for the WfgClient

These are options that can be used by the `WfgClient`. In fact, these settings can be used by any of the Oracle DataLens Server Client API classes. For a complete list of methods in the `WfgClient` class and additional information, see the Javadoc documentation.

Retry Count

This is useful to control the amount of time that the client attempts to connect to the Oracle DataLens Server. The default is to retry 20 times. This could be a problem in an interactive user environment, where you do not have a couple of minutes while `WfgClient` is attempting to connect to the server. In these cases you could set the retry count to 1 or even 0. Look also at `PingClient`, which can be used to check if a particular server is responding.

```
// Just set the retry to one for starting the job, then use the default
```

```
wfgClient.setRetryCount(1);jobID = wfgClient.runRtJob(transformProcess,
jobPriority, desc, rtLocale, input);
wfgClient.setRetryCountToDefault();
```

Filter Data

By default, data filtering is turned on for all input data. This will filter out all inadvertent control characters that may be interspersed in your input data. This data can cause problems with processing and sometimes it can cause problems with sending the data from the client to the server via HTTP as XML Soap documents. Tab characters are never filtered out.

```
// By default filtering is turned on and nothing needs to be done
wfgClient.setFilterData(false);
jobID = wfgClient.runRtJob(transformProcess, jobPriority, desc, rtLocale, input);
```

In the preceding example, the parameter input (with the List of input data) will be filtered.

Where the filtering encounters control characters in the input data, they will be substituted with the "?" character. This facilitates you in tracking down the source and exact location of the control characters. The data lens can ignore the "?" character when processing the input lines.

Job Priority

By default, a job priority of medium is used for all jobs.

This is the priority the job will be given on the server for processing. Large batch overnight jobs should be given a priority of low. Small jobs with few input records, or requests that need a quick response, such as users waiting for a response should get a priority of High. All other jobs should use a priority of medium. The number of concurrent jobs that can be run on the server is also controlled by the priority of the job (For more information, use the Configuration link on the Oracle DataLens Server Administration Web Pages). These priority values can be used from the `Priorities` class in the `ScsApi.jar`.

- `Priorities.PRIORITY_LOW`
- `Priorities.PRIORITY_MEDIUM`
- `Priorities.PRIORITY_HIGH`

```
// Set the job priority
wfgClient.setPriority(Priorities.PRIORITY_HIGH);
```

Run-Time Locale

By default, a run-time locale of USA English ("en_US") is used.

Set the locale to use for output of this job.

```
// Set the run-time locale
wfgClient.setRuntimeLocale(RT_LOCALE);
```

Separator Character

By default, a field separator character of tab is used.

```
// Set the run-time locale
wfgClient.setFieldSeparator('|');
```

Note: If you are using a different separator character than the default, then the separator character must be specified when pulling the data fields from the `WfgResultLine` data object.

```
List fields = wfgResultLine.getDataFields(FIELD_SEPARATOR_CHAR);
```

Client-Side Debugging Toggle

By default, this is toggled off when a new `WfgClient` object is created

This will dump the client information out to standard output prior to sending the request to the server. This is only used for debugging and should never be toggled to on in a production environment.

```
// Toggle on client data to standard output
wfgClient.setTrace(true);
```

Email Output

If set, then an API request that would return a list or update a file, will email the results to the user specified instead.

```
// Send the results to the following user
wfgClient.setEmailAddress("ybodrak@systems.com");
```

A DSA that updates a database will continue to update the database.

A DSA can be defined to return the results to an email address. This will work regardless of this API Email setting. In fact, the email address in the DSA will take precedence over this email set in the API

FTP Output

If set, then an API request that would return a list or update a file, will send the results instead to the FTP location specified.

```
// Send the results to the following FTP site
wfgClient.setFtpName("internal");
```

Note: This should not be set if the `setEmailAddress` is being used. In addition, the FTP name being used "internal" is one that is setup on the Oracle DataLens Server as in the following figure.

Ftp Connections Currently Defined						
Name	Description	Directory	Host	Port	User	
internal	internal FTP Test	test	10.2.2.20	21	msjvm F	

Database Parameters

By default, database parameters are not used.

This is used where the input map is expecting input from a database query and the query requires parameters that must be passed in.

Create a list of Parameters and then set the Db parameters as shown in the following code excerpt:

```
// Set the database parameters
List dbParams = new ArrayList();
    dbParams.add("first_parameter");
    dbParams.add("second_parameter");
wfgClient.setDbParameters(dbParams);
```

Server Information API to the Oracle DataLens Server

InfoClient

Getting Transform Map and Data Lens Information

Getting the data lens or Transform Map information uses a Java List interface. This is used to find out what data lenses or Transform Maps are available (deployed) on a particular server for processing. Details about the data lenses are also returned.

Import

Import the InfoClient with the following lines:

```
import com.onerealm.solx.api.client.InfoClient;
import com.onerealm.solx.api.client.ProjectData;
import com.onerealm.solx.api.client.MapData;
```

Initialize the Client

For all the examples shown below, an instance of the InfoClient class needs to be created with the Oracle DataLens Server name and port.

The `SERVER_NAME` can be either a machine name such as "localhost" or an IP address "12.1.20.117".

The `SERVER_PORT` is the port number of the server. By default, the Oracle DataLens Server is installed on port 2229.

```
// Create the Server Api object and point to the server
InfoClient projectApi = new InfoClient(SERVER_NAME, SERVER_PORT);
```

Get a List of Deployed Data Lenses

With the `getDeployedProjectList` method, you get a List of ProjectData objects that contains all the Data Lenses that are **deployed and loaded** on the Oracle DataLens Server.

Each ProjectData object contains a:

- data lens name
- description of the data lens
- list of the Standardizations used by the data lens
- list of the Classification schemas used by the data lens
- list of the Unit Conversions used by the data lens
- single Source Locale used by the data lens
- list of the target Translation locales used by the data lens

This is a simple example of pulling the data from the returned List:

```
List prjList = infoApi.getDeployedProjectList();
Iterator itr = prjList.iterator();
    while (itr.hasNext()) {
        // Get the data lens information
        ProjectData prjData = (ProjectData)itr.next();
        String projectName = prjData.getProject();
        String projectDesc = prjData.getDescription();
        List standardizations = prjData.getStandardizations();
        List schemas = prjData.getClassifications();
        List unitConversions = prjData.getUnitConversions();
        String sourceLocale = prjData.getSourceLocale();
        List targetLocales = prjData.getTargetLocales();
    }
```

Lists of Schemas and Translations

The `ProjectData` object contains lists of classification and translation data for each data lens as shown above. These are just lists of String data. These lists include:

- Classification Schemas used (UNSPSC, eCl@ss, or user-defined)
- Target Translation locales supported.

In addition, the `ProjectData` object contains lists of input and output data for each data lens, as in the preceding. These are just lists of string data. These lists include:

- Input data list
- Output data list

Get a List of Deployed DSAs

With the `getDeployedWorkflowList` method, you get a List of `WorkflowData` objects that contains all the DSAs that are **deployed and loaded** on the Oracle DataLens Server.

Each `WorkflowData` object contains:

- DSA name
- Description of the DSA
- List of input fields
- List of output fields
- A list of Transform Maps used by this DSA
- A list of the Database Connections used by this DSA

Following is a simple example of pulling the data from the returned list:

```
List workflowList = infoClientApi.getDeployedWorkflowList();
Iterator itr = workflowList.iterator();
while (itr.hasNext()) {
    // List the DSAs
    WorkflowData workflowData = (WorkflowData)itr.next();
    String name                = workflowData.getWorkflowName();
    String desc                = workflowData.getDescription();
    List inputFields           = workflowData.getInputFields();
    List outputFields          = workflowData.getOutputFields();
    List transformMaps         = workflowData.getTransformMaps();
    List dbConnections         = workflowData.getDbConnections();
}
```

Server Availability API to the Oracle DataLens Server

PingClient

This interface is provided to access the availability of the Oracle DataLens Servers.

The `PingClient` class is used as an interface to the Oracle DataLens Server. This class provides a simple method that returns true if an Oracle DataLens Server is available for processing data and false if it is not.

Import

Import the `PingClient` with the following line:

```
import com.onerealm.solx.api.client.PingClient;
```

Simple Server Check

This is a simple test to check the server availability. This can be used prior to sending data to the server for processing.

```
// Create the Server Api object and point to the server
PingClient pingApi = new PingClient(serverName, serverPort);
boolean available = pingApi.pingServer("JavaAPI");
```

Round-Robin Server Check

This is essentially the same code as in the previous section, just that we are going through a list of Oracle DataLens Servers in a Production Oracle DataLens Server Group and returning the first server that responds.

```
Iterator iter = serverList.iterator();
while (iter.hasNext()) {
    String serverNameStr = (String)iter.next();
    if (new PingClient(serverName, serverPort).pingServer(userName)) {
        return serverNameStr;
    }
}
```


Client-Side Exceptions

Client-side exceptions are caught via the standard java Exception catching mechanism. These faults are typically Connection exceptions, such as request/response timeouts or failure to connect to the server.

Client-Side Log Messages

The client side messages are output using standard output.

The client side messages can be output using a logging package called log4j. The API library uses Java reflection to determine if log4j is available. If it is already in your client application, log4j is used to output messages.

For more information about log4j, see the Apache log4j Web site:

<http://logging.apache.org/log4j/>

Log4j to Standard Output

By default, client-side error messages will go to standard output.

These same log messages are sent to a log-file if log4j is used in your client application.

If you want to add log4j for use in your client application, then the logger needs to be initialized. Otherwise, you will get the following error message in the standard output window or log file if there is a client-side problem.

```
log4j:WARN No appenders could be found for logger
(com.onerealm.solx.api.client.ClientBase).
log4j:WARN Please initialize the log4j system properly.
```

Add the following line of code to initialize the client-side logger, enabling logging output to standard output.

```
import org.apache.log4j.BasicConfigurator;
. . .
// Initialize Log4J to get client-side logging to standard output
BasicConfigurator.configure();
```

The output in this log is usually a connection re-try attempt or some other client-side processing. Instead of the warning messages, you will now see the following types of messages:

- Attempt 2 to connect to http://lrivas-xp-a31:2229/datalens/Workflow
- Attempt 3 to connect to http://lrivas-xp-a31:2229/datalens/Workflow
- Attempt 4 to connect to http://lrivas-xp-a31:2229/datalens/Workflow

Log4J to a File

If log4j is being use in the client application, then the following will apply.

The client-side logging messages can be sent to a file as well. The following example is a very simple logging configuration that will log all the messages to a log file in the /tmp directory.

```
import org.apache.log4j.*;
. . .
// Initialize Log4J to get client-side logging to the /tmp directory
Logger logger =
Logger.getLogger(com.onerealm.solx.api.client.ClientBase.class);
SimpleLayout layout = new SimpleLayout();
FileAppender appender = null;
try {
    appender = new FileAppender(layout, "/tmp/SCS_Log.txt", false);
} catch(Exception e) {}
logger.addAppender(appender);
logger.setLevel((Level) Level.WARN);
```

Note: The level for messages can be changed from WARN to DEBUG to get additional information if needed.

Server-Side Faults

There are also server-side exceptions that are propagated back to the client via the SOAP interface.

Here is how those exceptions are caught:

```
try {
m_client.getResultData (...
    } catch (Fault f) {
        System.out.println(f.getErrorCode());
    } catch (Exception e) {
        System.out.println("Error in Test: " + e.getMessage());
    }
}
```

Server-Side Exceptions

The Fault Exception object will provide a listing of error codes of the problem and status of your request to the Oracle DataLens Server.

Use the macros in the com.onerealm.solx.api.iface.ErrorIf class to check for specific errors.

For example:

```
try {
resultData = wfgRtApi.getResultData(m_jobID, waitForResults);
    } catch (Fault f) {
```

```
        if (f.getErrorCode() == ErrorIF.ERROR_NOT_COMPLETED) ...  
    }
```

Server-Side Log Messages

Go to the Oracle DataLens Server Administration Web Pages and examine the log file from the home page. This will have a listing of any errors that were encountered in the server-side processing of your request.

Debugging Client Requests and Responses

The Oracle DataLens Server API communicates with the Oracle DataLens Servers by sending HTTP SOAP requests to the server and receiving HTTP SOAP responses back from the server. The content of these XML messages can be sent to standard output for debugging by the application programmer. This is useful if you want to verify that the data being send and received by the application program is exactly what is being communicated to the server.

This is turned on from the Oracle DataLens Server Administration Web Pages as shown in the following:

Packet Tracing Administration

Trace Level	Toggle packet tracing on or off
Trace DataLenses** (Real-Time)	<input type="radio"/> on <input type="radio"/> off
Trace Transform Maps	<input type="radio"/> on <input type="radio"/> off
Trace Process Maps	<input checked="" type="radio"/> on <input type="radio"/> off
Trace General Packets	<input type="radio"/> on <input type="radio"/> off

Compiling and Running with the API

Compile the Application with the Oracle DataLens Libraries

To compile your class with the Oracle DataLens Server client calls, the Oracle DataLens Server client libraries (`ScsApi.jar` and `ScsApiImports.jar`) will need to be part of your `CLASSPATH`.

These jar files are located in the `/Interfaces` directory on your installation CD.

Put this into either the `CLASSPATH` environment variable or use in the command-line Java compile as shown follows:

```
javac -classpath " ScsApi.jar; ScsApiImports.jar" WfgClientTest.java
```

This creates the `WfgClientTest.class` file that is part of your application.

Run the Application with the Oracle DataLens Libraries

The Oracle DataLens Server libraries need to be referenced when running an application that accesses the Oracle DataLens Server.

The `ScsApi.jar` and `ScsApiImports.jar` file is located in the `/Interfaces` directory on your installation CD.

The following is an example of running the program compiled in the previous example.

```
java -cp " ScsApi.jar; ScsApiImports.jar;%;" WfgClientTest
```

In this case, we are using the API and the `ScsApiImports` libraries, running the Java class file that we just compiled.

Web Service Access to the Oracle DataLens Server Using Doc-Lit

Access is provided to the Oracle DataLens Server as a Document-Literal Web Service.

Generating a WSDL Document on Demand

To integrate with an Oracle Product Data Quality DSA as a Web Service, you need software that will talk to the specific Oracle DataLens Web Services. Many vendors provide tools to generate this software from a **Web Services Description Language** (WSDL) document, which is an XML format for describing network services. You can view the WSDL for the Oracle DataLens Web Services by using a browser.

Enter the following into a browser:

Note: The host name and port number may differ.

<http://localhost:2229/datalens/ws/Processor?wsdl> (Document-Literal)

This displays the WSDL document, which can be saved by right-clicking in the document in the browser, selecting **View Source**, and then saving the file from within your browser. For instance, the file can be saved as `Processor.wsdl`.

Note: Internet Explorer displays the WSDL document; Netscape Navigator displays a blank web page for the returned document.

Client Web Service Software

For your Web Service clients, client-side software can be generated from this WSDL document to access the Oracle DataLens Server.

Overview of the DSA Interface

There is a single Service called "ProcessorService". This uses a Port called "Processor".

Three Oracle DataLens Web Services Operations can be used to process data as follows:

ProcessorList

This takes an input array of strings and returns an output array of strings.

ProcessorOneLine

This takes a single string of input and returns a single string of output.

ProcessorDB

This takes a database query (defined in the Transform Map) and returns a job Id of the DSA Job that handled the request. The output is assumed to be a database update, email, or FTP.

processListRequest and processOneLineRequest Operations

The difference between these two operations is that `processListRequest` takes an array of lines and `processOneLineRequest` takes a single line of data as a string. The transformed data is returned. This call is synchronous.

Parameters are as follows:

```
dsaName
lines/line
dbParameters
priority
runtimeLocale
fieldSeparatorChar
application
description
```

processDBRequest

This call takes the database parameters as input and returns the DSA Job ID. This call is asynchronous.

Parameters are as follows:

```
dsaName
dbParameters
priority
runtimeLocale
fieldSeparatorChar
application
description
```

For additional information about these parameters, see [Chapter 2, "DSA API to the Oracle DataLens Server."](#)

SOAP Document-Literal One Line Request Example

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://www.silvercreeksystems.com/ws">
<soapenv:Header/>
<soapenv:Body>
  <ws:processOneLine>
    <dsaName>sampleDSA</dsaName>
    <!--Optional:-->
    <line>l^res, 17ohm, 19watt, 20%</line>
    <!--Zero or more repetitions:-->
    <dbParameters>?</dbParameters>
    <!--Optional:-->
    <priority>1</priority>
    <!--Optional:-->
```

```

        <runtimeLocale>en_US</runtimeLocale>
        <!--Optional:-->
        <fieldSeparator>^</fieldSeparator>
        <!--Optional:-->
        <application>ClientCall</application>
        <!--Optional:-->
        <description>Example Doc-Lit client call</description>
    </ws:processOneLine>
</soapenv:Body>
</soapenv:Envelope>

```

SOAP Document-Literal One Line Response Example

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
    <ns2:processOneLineResponse
xmlns:ns2="http://www.silvercreeksystems.com/ws">
        <return>1^Resistor, 17 Ohm, 20%, 19 Watt^32121609^Fixed
resistors^Resistor^Item_Name^RESISTOR^Item_Type^^Resistance^17
OHM^Power^19^Tolerance^20%^Package_Size^^Construction^^Mounting^^Pin_Count
^^For_sale_packaging^</return>
    </ns2:processOneLineResponse>
</S:Body>
</S:Envelope>

```

SOAP Doc-Lit Multi-Line ProcessList Request Example

```

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://www.silvercreeksystems.com/ws">
<soapenv:Header/>
    <soapenv:Body>
        <ws:processList>
            <dsaName>sampleDSA</dsaName>
            <!--Zero or more repetitions:-->
            <linesOfData>1^res, 17ohm, 19watt, 10%</linesOfData>
            <b>linesOfData</b>2^res, 27ohm, 29watt, 20%</linesOfData>
            <linesOfData>3^res, 37ohm, 39watt, 30%</linesOfData>
            <!--Zero or more repetitions:-->
            <dbParameters>?</dbParameters>
            <!--Optional:-->
            <priority>1</priority>
            <!--Optional:-->
            <runtimeLocale>en_US</runtimeLocale>
            <!--Optional:-->
            <fieldSeparator>^</fieldSeparator>
            <!--Optional:-->
            <application>ClientCall</application>
            <!--Optional:-->
            <description>Example list Doc-Lit client call</description>
        </ws:processList>
    </soapenv:Body>
</soapenv:Envelope>

```

SOAP Doc-Lit Multi-Line ProcessList Response Example

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">

```

```

<S:Body>
  <ns2:processListResponse xmlns:ns2="http://www.silvercreeksystems.com/ws">
    <return>1^Resistor, 17 Ohm, 10%, 19 Watt^32121609^Fixed
resistors^Resistor^Item_Name^RESISTOR^Item_Type^^Resistance^17
OHM^Power^19^Tolerance^10%^Package_Size^^Construction^^Mounting^^Pin_Count
^^For_sale_packaging^</return>
    <return>2^Resistor, 27 Ohm, 20%, 29 Watt^32121609^Fixed
resistors^Resistor^Item_Name^RESISTOR^Item_Type^^Resistance^27
OHM^Power^29^Tolerance^20%^Package_Size^^Construction^^Mounting^^Pin_Count
^^For_sale_packaging^</return>
    <return>3^Resistor, 37 Ohm, 30%, 39 Watt^32121609^Fixed
resistors^Resistor^Item_Name^RESISTOR^Item_Type^^Resistance^37
OHM^Power^39^Tolerance^30%^Package_Size^^Construction^^Mounting^^Pin_Count
^^For_sale_packaging^</return>
  </ns2:processListResponse>
</S:Body>
</S:Envelope>

```

SOAP Document-Literal ProcessDb Request Example

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://www.silvercreeksystems.com/ws">
<soapenv:Header/>
  <soapenv:Body>
    <ws:processDB>
      <dsaName>SampleDSADbInput</dsaName>
      <!--Zero or more repetitions:-->
      <dbParameters>1</dbParameters>
      <dbParameters>2</dbParameters>
      <!--Optional:-->
      <priority>2</priority>
      <!--Optional:-->
      <runtimeLocale>en_US</runtimeLocale>
      <!--Optional:-->
      <fieldSeparator>|</fieldSeparator>
      <!--Optional:-->
      <application>ClientCall</application>
      <!--Optional:-->
      <description>Example Db Input Doc-Lit client call</description>
    </ws:processDB>
  </soapenv:Body>
</soapenv:Envelope>

```

Note: The field separator will be used when the output from the database job is a text file.

SOAP Document-Literal processDb Response Example

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
  <ns2:processDBResponse xmlns:ns2="http://www.silvercreeksystems.com/ws">
    <return>784</return>
  </ns2:processDBResponse>
</S:Body>
</S:Envelope>

```

The return value of the preceding `processDb` call is the DSA Job ID, in this case 784. This job reads data from the database and updates other fields in the database as in the following example:

Property		Value
Job ID		784
Status		Completed
Definition		samplePMapIDefDBInput
Description		Example Db Input Doc-Lit client call
Start Time		January 27, 2010 1:14:56 PM MST
Finish Time		January 27, 2010 1:14:56 PM MST
Duration		H:0 M:0 S:0
Created by		ClientCall
Input Line Count		2
Output Line Count (Good)		0
Output Line Count (Not Processed)		0
Output Path/File		Not Used
Run-time Locale		en_US

Step Name	Type	Status	Description	Start Time	End Time	Duration	Input Line Count	Output Line Count	Comments
DB Input	DB Input	Completed	Input from Database	2010-01-27 13:14:56.177	2010-01-27 13:14:56.183	H:0 M:0 S:0	0	0	
ProcessResistors	Processing	Completed	null	2010-01-27 13:14:56.203	2010-01-27 13:14:56.413	H:0 M:0 S:0	2	2	
updateValue	Processing	Completed	null	2010-01-27 13:14:56.425	2010-01-27 13:14:56.478	H:0 M:0 S:0	2	2	
add_resistors	DB Output	Completed	update className in Db	2010-01-27 13:14:56.483	2010-01-27 13:14:56.491	H:0 M:0 S:0	2	0	

Customizing DSA Maps with Java Add-Ins and Algorithms

There are three types of customizations that can be done in a DSA and Transform Map (TMap).

- TMap Algorithms (The easiest to use)
- TMap Add-in Transforms (Usually used by Oracle Product Data Quality Professional Services)
- DSA Add-in Outputters (Usually used by Oracle Product Data Quality Professional Services)

The TMap Algorithms are the easiest to use and require no special work outside of the Application Studio. The Java code is written directly in the Transform Map Builder in Application Studio and can be tested and run from here. The limitation is that all the Java code needs to be part of the single transform method.

The TMap Add-in Transforms and DSA Add-in Outputters are really for use by Oracle Product Data Quality Professional Services to create powerful custom widgets for use in the DSAs. This Java code can contain entire packages of classes and needs to be written in an external Java API and imported into the Oracle DataLens Server.

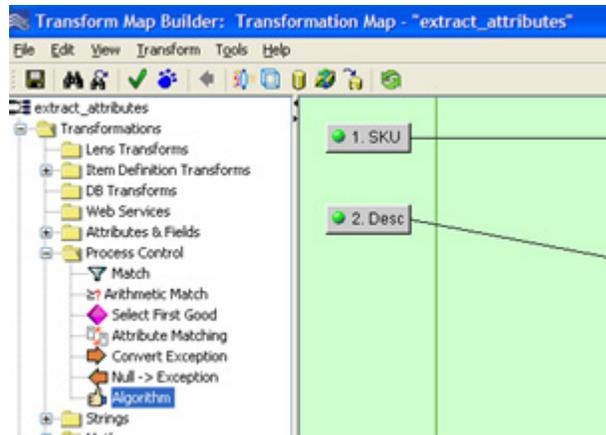
TMap Algorithms

Initial Configuration

Java TMap Algorithms, allow Java code to be embedded into Transform Maps and have the code be executed when the parent DSA is run. The Oracle DataLens Client and Server software is configured "out of the box" to support this with no configuration changes needed.

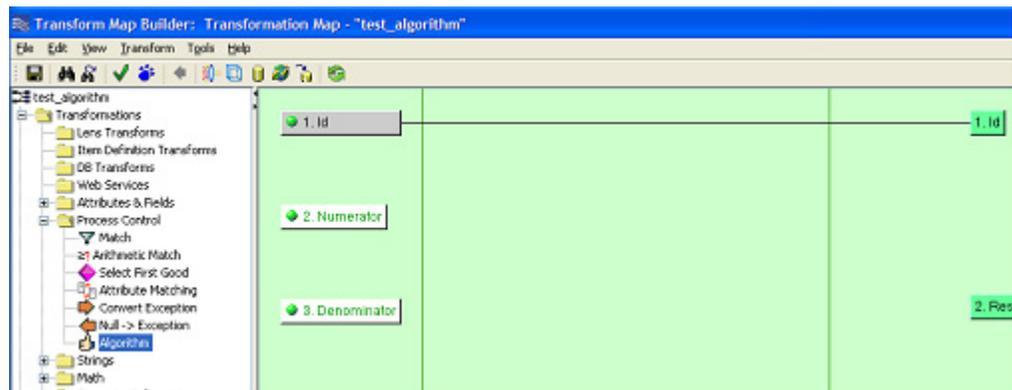
Client Startup Changes

The Algorithm widget is available in the Process Control Transformation menu in the Transform Map Builder as follows:

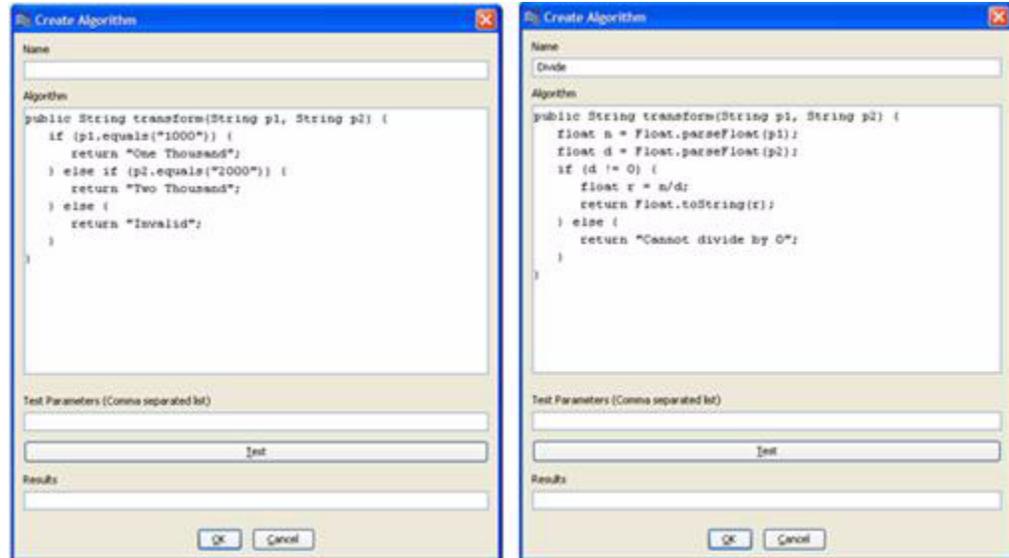


Creating a New TMap Algorithm

Drag the Algorithm object into your Transform Map as follows:



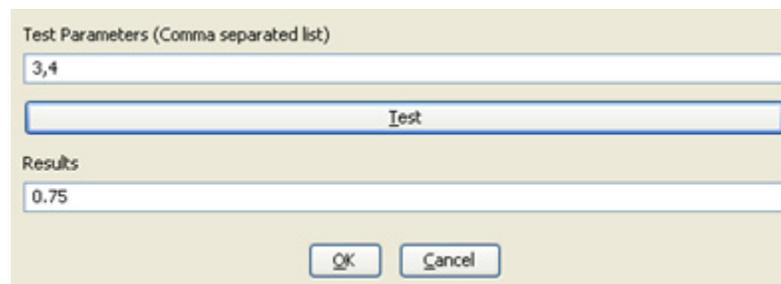
The following dialog will appear. Name and create a new algorithm object as shown on the left. In this example, we modified the template Java code to divide two numbers as follows:



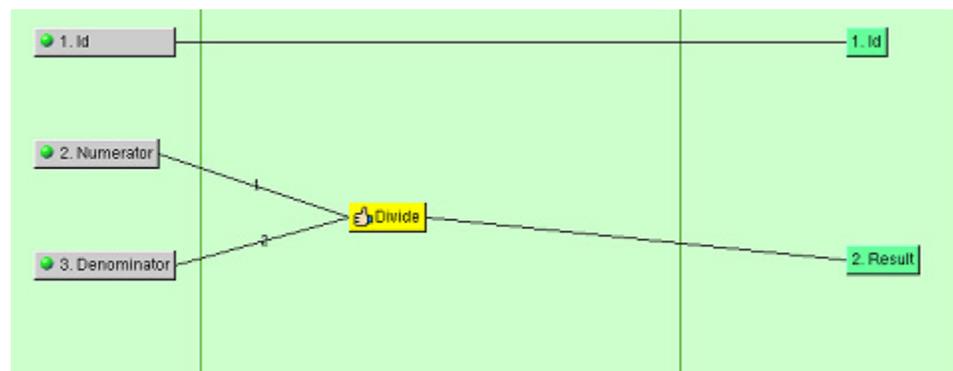
Notes on the Algorithm Java method:

- The name of the Algorithm can be any valid Java Class name, but the method needs to be declared as public String transform.
- The method may contain one or more string parameters. In the previous example, the method is taking two string parameters.

Test the new code as in the following figure:



Select **OK** to save the new custom Algorithm. The Transform Map looks like the following:



The new Custom Algorithm is ready to check-in and deploy to the Oracle DataLens Server and start using in your client applications.

TMap Algorithm Debugging

All these Customizations require that the classes directory be part of the Java `classpath`. This has already been done for you as part of the standard Oracle DataLens Client and Server installations.

Server

If you encounter any problems running the TMap Algorithm on the server, check that the environment variable `CATALINA_HOME` is set on the Oracle DataLens Server with value `C:\Program Files\Apache Software Foundation\Tomcat 5.5`.

The classes directory is located in the `.../Tomcat/webapps/datalens/WEB-INF/classes` directory already and nothing needs to be done.

Client

The script file that starts the Application Studio needs to have the `$SOLX_HOME/classes` directory added to the `classpath` as follows:

```
set CP6=%SOLX_HOME%/classes
set classpath=%CP1%;%CP2%;%CP3%;%CP4%;%CP5%;%CP6%
```

Note: This is already done for you in during the Oracle DataLens client software installation.

TMap Add-In Transforms

Java TMap Add-in Transforms are only created by Oracle Product Data Quality Professional Services.

Java TMap Add-in Transforms allow user-defined widgets to be available for use in DSAs.

For additional information on these classes, see the file, `Add2Int.java`.

Writing a TMap Add-In Transform

The class may be in any Java package of your choosing.

The class name may be any valid Java Class name.

In the following example, we are using a TMap add-in transform class that is shipped with the Oracle DataLens Server installation called `GetField`.

The class must implement a constructor with a single string argument (the name).

```
public GetField(String name) {
    super(name);
}
```

The class must implement a method called `getResults` as follows:

```
/**
 * This is the main method called by the Add-In Transform server code.
```

```

*
* @param linesOfInputData is an Array of data for each line being
* processed,
* where the data is an array of the inputs to a single computation.
*
* @param parameters are the input parameters for this TMap Add-in function.
*
* @return XfmInfo[] of the results of the transformation, one element
* for each line of input data.
*/
public XfmInfo[] getResults(String[][] linesOfInputData,
                             Map<String, String>
fixedParameters) {
    int inputLength = linesOfInputData.length;
    XfmInfo[] results = new XfmInfo[inputLength];
    // Get the parameters here...
    // Processing happens here...
    return results;
}

```

1. `linesOfInputData` parameter - An array of arrays of Strings. The 1-D level array has one element for each line of input that must be processed. Each element of the 1-D array has a 2-D `String[]` containing the column data needed for the transformation. Thus the array looks like:

```
String[numberOfLines][numberOfColumnsOfInputData]
```

2. `fixedParameters` - These are the parameters to this add-in transform, passed in from the DSA.
3. Return an `XfmInfo[]` of the results of the transformation, one element for each line of input data.

Defining the TMap Add-In Transform

In the Application Studio, the add-in classes will be toggled on if the system finds the `AddInClasses.xml` file in the shared config directory on the Oracle DataLens Server.

Server

The class needs to be added to the `AddInClasses.xml` file, in the `C:\Datalens\server\data\shared\common\config` directory, as follows:

```

<AddInClasses>
<Transforms>
  <class>
    <name>Get Field</name>

    <className>com.onerealm.solx.maps.xfm.code.transform.GetField</className>
    <description>Gets the specified field from a string. The field index,
    field separator, and default value are specified in the fixed
parameters.</description>
  </class>
</Transforms>
</AddInClasses>

```

Defining the Input Parameters to the TMap Add-In Transform

This step can be skipped if the TMap Add-in transform does not use any initialization parameters.

The parameters to the new TMap add-in transform need to be added to the `AddInTransformParameters.xml` configuration file. This file is located in the same configuration directory as the `AddInClasses.xml`. In this case, our `GetField` add-in takes three parameters and the `AddInTransformParameters.xml` file will look like similar to the following:

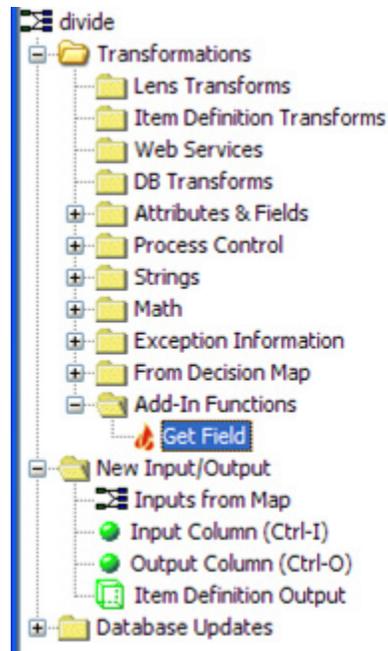
```
<TransformParameters>
<AddIn>
  <name>Get Field</name>
  <parameters>
    <parameter>
      <name>separator</name>
      <default>|</default>
      <desc>Separator for splitting string into fields</desc>
      <editable>true</editable>
    </parameter>
    <parameter>
      <name>index</name>
      <default>0</default>
      <desc>Index of field to extract (1-based)</desc>
      <editable>true</editable>
    </parameter>
    <parameter>
      <name>default</name>
      <default></default>
      <desc>Default value to return if field not found</desc>
      <editable>true</editable>
    </parameter>
  </parameters>
</AddIn>
</TransformParameters>
```

Using the TMap Add-In Transform in the Client

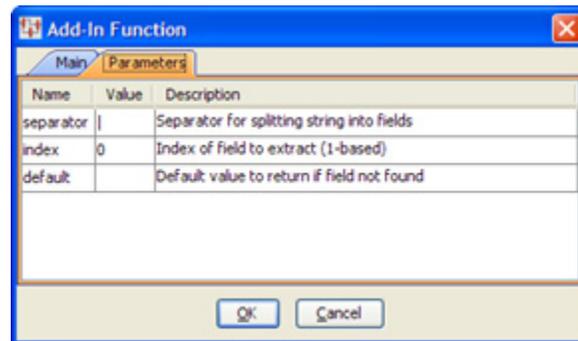
No changes are need for the client configuration to pick up your new Tmap Add-In Transformation. The Oracle DataLens Server just needs to be restarted whenever the `CustomClasses.xml` file is updated (because the server reads this file on startup).

Note: You will be able to add the new Tmap Add-in to your DSA map, but this cannot be tested on the client, it can only be tested by running a job on the server.

Now start the Application Studio and the new add-in class will be available in the TMap interface as follows:



Drag the Get Field Add-in Function into the TMap and the parameters will be displayed for editing in a table as follows:



DSA Add-In Outputters

Java DSA Add-in Outputters can write data out in any user-defined format. Since this is an output step, there is no routing of data past this step in the DSA Map, so this should be used only by Java code that will not be throwing any exceptions that need to be caught and processed by the DSA Map.

Writing a DSA Add-In Outputter

The class may be in any Java package of your choosing.

The class name may be any valid Java Class name.

In the following example, we are using a TMap add-in transform class that is shipped with the Oracle DataLens Server installation called **SCS XML**.

The main method that is called is **writeOutput**. This returns a `WfgCustomOutputReturn` object, which contains information needed to forward the

result data to an email address or an FTP site. If this returns null, then there will be no email or FTP.

Note: Even if this object is returned, the email and/or FTP is only sent if it is defined in the DSA or the DSA job has defined email or FTP output.

Following is an example of the structure of the Output Adapter Class:

```
package com.onerealm.solx.maps.wfg.code.output;
/**
 * @param job PMap job information
 * @param data Wfg Job data
 * @param outputDir The directory to write the xml file
 * @param parameters The input parameters to the Add-in Outputter.
 * @return Custom output
 * @throws SaException Superclass for all SCS exceptions
 * @throws IOException Signals that an I/O exception of some sort has occurred
 */
public WfgCustomOutputReturn writeOutput(WfgJob job, WfgInputData data, String
outputDir,
                                     Map<String, String> parameters) {
    List<WfgDataLine> lines;
    while ((lines =
data.getNextGoodLines(WfgConstants.MAX_MEMORY_LINES)) != null) {
        for (WfgDataLine line : lines) {
            System.out.println(line.getData());
        }
    }
    return null;
}
```

Defining the DSA Add-In Outputter

In the Application Studio, the add-in classes will be toggled on if the system finds the `AddInClasses.xml` file in the shared config directory on the Oracle DataLens Server.

Server

The class needs to be added to the `AddInClasses.xml` file as follows:

```
<AddInClasses>
  <Outputs>
    <class>
      <name>SCS XML</name>

      <className>com.onerealm.solx.maps.wfg.code.output.scspim.ScsStepPimProducts</class
Name>
      <description>This will output a STEP PIM Product XML document
with SCS processed data; The default file is
/tmp/ScsStepPimProductData_jobId.xml</description>
    </class>
  </Outputs>
</AddInClasses>
```

Note: There is *no* client file needs to be updated for use with the Application Studio.

The server file needs to be updated for use running DSAs on the Oracle DataLens Server and to make this available to the Application Studio Clients.

Follow the instructions in the TMap Add-in Transforms section for changing the startup scripts and adding the new classes to the classpath.

Defining the Input Parameters to the TMap Add-In Transform

This step can be skipped if the TMap Add-in transform does not use any initialization parameters. In this example, we are not using any input parameters.

Using the DSA Add-In Outputter in the Client

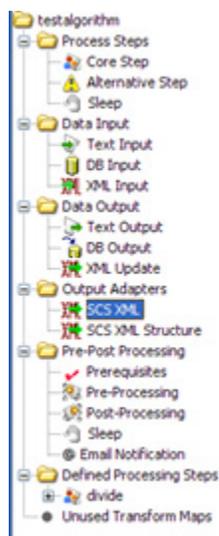
No changes are need for the client configuration to pick up your new DSA Add-In Outputter. The Oracle DataLens Server just needs to be restarted whenever the `CustomClasses.xml` file is updated (because the server reads this file on startup).

Note: You will be able to add the new DSA Add-in Outputter to your DSA map, but this cannot be tested on the client, it can only be tested by running a job on the server.

Now start the Application Studio and the new add-in class will be available in the TMap interface s shown below.

Use in the Application Studio

Once the above steps are finished, the new DSA Output Adapter is now available for use in the Application Studio as follows:



Oracle DataLens Server JAVA API Reference

Please contact technical support for the complete JavaDoc HTML reference pages to all the Java APIs described in this document.

- Tree representation (both sparse and full trees supported)

Working Through a Proxy Server

Sometimes a Java program needs to call the Oracle DataLens Java API to an Oracle DataLens Server outside of your firewall. Normally this is not a problem, but sometimes there is a proxy server that must be negotiated to get to the outside world.

There are two solutions to this problem.

- Use the Oracle DataLens Web Services interface and your WSDL connection software to negotiate through the proxy server.
- Use the Java Proxy arguments to the java command.

Run-Time Java Proxy Parameters

Three parameters are used with the java command to set the proxy information:

- `DproxySet=true`
- `DproxyHost=hostname or IP Address`
- `DproxyPort=8080`

This is shown in the following example java call to a program called `WfgProgram`:

```
java -cp "./ScsApi.jar;./ScsApiImports.jar;." -DproxySet=true  
-DproxyHost=10.1.60.116 -DproxyPort=8080 WfgProgram
```

RtClient Java Proxy Parameters

There are four additional parameters to the `RtClient` overloaded constructor for use going through a proxy. For additional information, see the JavaDoc.

- `@param proxyHost` - the name of the proxy server
- `@param proxyPort` - the port of the proxy server
- `@param proxyUser` - the user name to login to the proxy server
- `@param proxyPassword` - the password to login to the proxy server

These are shown in the following between the `ServerPort` and the `ENCRYPTION` parameters:

```
m_wfgClient = new WfgClient(serverName, serverPort,  
    "10.1.60.106",2229, "cbidwell", "secret1",  
    ENCRYPTION, clientCode, APPLICATION);
```

Installing the Client Software

Oracle Product Data Quality uses a concept called Java Web Start to initially install and maintain the current version of the software on your client desktop. The process requires you to access the Oracle DataLens Server to initiate the connection and download the software.

You download and install the Oracle Product Data Quality client applications using Java Web Start by browsing to the installation page for your Oracle DataLens Server as follows:

1. Using Microsoft Internet Explorer, browse to one of the following URLs as appropriate for your server:

Note: If you setup a different port number for your application server other than 2229, you must use that port number in the following URL when browsing to the Oracle DataLens Server to download the client applications.

32-bit

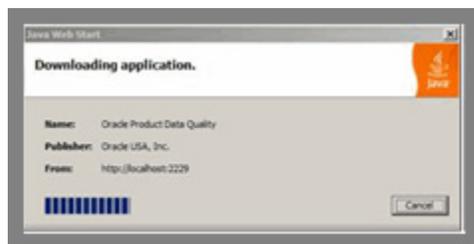
<http://<server>:2229/datalens/datalens.html>

64-bit

<http://<server>:2229/datalens/datalens64.html>

Where *<server>* is the hostname of the Oracle DataLens Server

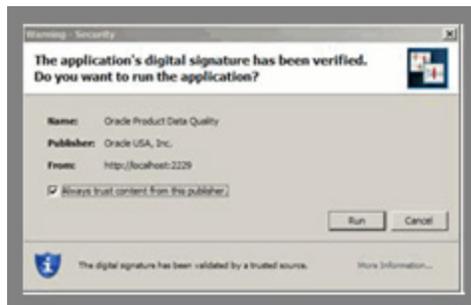
The application download and installation begins. If you do not have a supported Java environment on the target installation machine the Java Web Start program automatically redirects you to a Java download site and begins a Java Runtime installation.



2. If the preceding Java Web Start message is not displayed, you must initiate a connection and download the software by browsing to:

<http://<server>:2229/datalens/datalens.jnlp>

Oracle Product Data Quality files are digitally signed by a trusted source so the following security warning is displayed.



3. To avoid the security dialogue in the future you can select the **Always trust content from this publisher** check box.
4. Click **Run** to continue and complete the installation.
The Oracle Product Data Quality log on dialog is displayed.



Deprecated: Web Service Access to the Oracle DataLens Server using RPC

Important: This web service has been deprecated and is no longer supported.

Access is provided to the Oracle DataLens Server as a Web Service.

Generating a WSDL Document on Demand

Note: The `wsdl4j` library is needed to enable the RPC Web Services interface. Contact Professional Services to get the library that needs to be put in your Oracle DataLens Server `lib` directory. You will get a `ClassNotFoundException` if the `wsdl4j.jar` is not there.

To integrate with an Oracle Product Data Quality DSA as a Web Service, you need software that will talk to the specific Oracle DataLens Web Services. Many vendors provide tools to generate this software from a WSDL document. You can view the WSDL for the Oracle DataLens Web Services by using a browser. Enter the following in a browser (the host and port may differ).

<http://localhost:2229/datalens/services/Processor?wsdl> (RPC)

This displays the WSDL document, which can be saved by doing View Source and saving the file from within your browser. For instance, the file can be saved as `Processor.wsdl`.

Note: Internet Explorer displays the WSDL document; Netscape Navigator displays a blank web page for the returned document.

Client Web Service Software

For your Web Service clients, client-side software can be generated from this WSDL document to access the Oracle DataLens Server. If the client uses Apache Axis the `WSDL2Java` program can be run to generate client-side Java files. For other types of Web Service clients, use the Web Service generator supplied by the vendor.

Overview of the DSA Interface

There is a single Service called `ProcessorService`, which uses a port called `Processor`.

There are three Oracle DataLens Web Services Operations that can be used to process data.

ProcessorList

This takes an input array of strings and returns an output array of strings.

ProcessorOneLine

This takes a single string of input and returns a single string of output.

ProcessorDB

This takes a database query (defined in the Transform Map) and returns a job Id of the DSA Job that handled the request. The output is assumed to be a database update, email, or FTP.

processListRequest and processOneLineRequest

The difference between these two is that `processListRequest` takes an array of lines and `processOneLineRequest` takes a single line of data as a string. The transformed data is returned. This call is synchronous.

Parameters are as follows:

```
dsaName  
lines/line  
dbParameters  
priority  
runtimeLocale  
fieldSeparatorChar  
application  
description
```

processDBRequest

This call takes the database parameters as input and returns the DSA Job ID. This call is asynchronous.

Parameters are as follows:

```
dsaName  
dbParameters  
priority  
runtimeLocale  
fieldSeparatorChar  
application  
description
```

For additional information about these parameters, see [Chapter 2, "DSA API to the Oracle DataLens Server."](#)

SOAP RPC One-Line Request Example

```

<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsr="http://wsrpc.svr.solx.onerealm.com">
<soapenv:Header/>
  <soapenv:Body>
    <wsr:processOneLine
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      <dsaName xsi:type="soapenc:string"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">sampleDSA</dsaName>
      <line xsi:type="soapenc:string"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">17^res, 1ohm, 2w,
3%</line>
      <dbParameters xsi:type="xsd:ArrayOfstring"
soapenc:arrayType="soapenc:string[]" xmlns:xsd="http://soapinterop.org/xsd"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        <priority xsi:type="soapenc:string"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">1</priority>
        <runtimeLocale xsi:type="soapenc:string"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">en_US</runtimeLocale>
        <fieldSeparatorChar xsi:type="soapenc:string"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">^</fieldSeparatorChar>
        <application xsi:type="soapenc:string"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">ClientRPCCall</applicati
on>
        <description xsi:type="soapenc:string"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">Web Services RPC One
Line Client Call</description>
        <clientCode xsi:type="soapenc:string"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">NotUsed</clientCode>
      </wsr:processOneLine>
    </soapenv:Body>
  </soapenv:Envelope>

```

SOAP RPC One-Line Response Example

```

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
  <ns1:processOneLineResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://wsrpc.svr.solx.onerealm.com">
    <processOneLineReturn xsi:type="soapenc:string"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">17^Resistor, 1 Ohm, 3%,
2 Watt^32121609^Fixed
resistors^Resistor^Item_Name^RESISTOR^Item_Type^^Resistance^1
OHM^Power^2^Tolerance^3%^Package_Size^^Construction^^Mounting^^Pin_Count^^
For_sale_packaging^</processOneLineReturn>
  </ns1:processOneLineResponse>
</soapenv:Body>
</soapenv:Envelope>

```