Informatica B2B Data Transformation
(Version 9.6.0)

Engine Developer Guide

This product includes software licensed under the Academic Free License (http://www.opensource.org/licenses/afl-3.0.php), the Common Development and Distribution License (http://www.opensource.org/licenses/cddl1.php) the Common Public License (http://www.opensource.org/licenses/cpl1.0.php), the Sun Binary Code License Agreement Supplemental License Terms, the BSD License (http://www.opensource.org/licenses/bsd-license.php), the new BSD License (http://opensource.org/licenses/bsd-3-clause), the MIT License (http://www.opensource.org/licenses/mit-license.php), the Artistic License (http://www.opensource.org/licenses/artistic-license-1.0) and the Initial Developer's Public License Version 1.0 (http://www.firebirdsql.org/en/initial-developer-s-public-license-version-1-0/).

This product includes software copyright © 2003-2006 Joe Walnes, 2006-2007 XStream Committers. All rights reserved. Permissions and limitations regarding this software are subject to terms available at http://xstream.codehaus.org/license.html. This product includes software developed by the Indiana University Extreme! Lab. For further information please visit http://www.extreme.indiana.edu/.

This product includes software Copyright (c) 2013 Frank Balluffi and Markus Moeller. All rights reserved. Permissions and limitations regarding this software are subject to terms of the MIT license.

This Software is protected by U.S. Patent Numbers 5,794,246; 6,014,670; 6,016,501; 6,029,178; 6,032,158; 6,035,307; 6,044,374; 6,092,086; 6,208,990; 6,339,775; 6,640,226; 6,789,096; 6,823,373; 6,850,947; 6,895,471; 7,117,215; 7,162,643; 7,243,110; 7,254,590; 7,281,001; 7,421,458; 7,496,588; 7,523,121; 7,584,422; 7,676,516; 7,720,842; 7,721,270; 7,774,791; 8,065,266; 8,150,803; 8,166,048; 8,166,071; 8,200,622; 8,224,973; 8,271,477; 8,327,419; 8,386,435; 8,392,440; 8,453,159; 8,458,230; and RE44,478, International Patents and other Patents Pending.

DISCLAIMER: Informatica Corporation provides this documentation "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of noninfringement, merchantability, or use for a particular purpose. Informatica Corporation does not warrant that this software or documentation is error free. The information provided in this software or documentation may include technical inaccuracies or typographical errors. The information in this software and documentation is subject to change at any time without notice.

NOTICES

This Informatica product (the "Software") includes certain drivers (the "DataDirect Drivers") from DataDirect Technologies, an operating company of Progress Software Corporation ("DataDirect") which are subject to the following terms and conditions:

1. THE DATADIRECT DRIVERS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

2. IN NO EVENT WILL DATADIRECT OR ITS THIRD PARTY SUPPLIERS BE LIABLE TO THE END-USER CUSTOMER FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR OTHER DAMAGES ARISING OUT OF THE USE OF THE ODBC DRIVERS, WHETHER OR NOT INFORMED OF THE POSSIBILITIES OF DAMAGES IN ADVANCE. THESE LIMITATIONS APPLY TO ALL CAUSES OF ACTION, INCLUDING, WITHOUT LIMITATION, BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS.

Part Number: DT-EDG-96000-0001
Table of Contents

Preface ........................................................................ iii
Informatica Resources. ......................................................... iii
  Informatica My Support Portal. ............................................... iii
  Informatica Documentation. ................................................. iii
Informatica Web Site. ........................................................... iii
Informatica How-To Library. ................................................ iv
Informatica Knowledge Base. ............................................... iv
Informatica Support YouTube Channel. .................................... iv
Informatica Marketplace. .................................................... iv
Informatica Velocity. ........................................................... iv
Informatica Global Customer Support. ................................. iv

Chapter 1: Command-Line Interface. ................................. 1
Command-Line Interface Overview. ...................................... 1
CM_console. .................................................................... 1
Input Documents. ............................................................. 3
Web Server Authentication. ................................................ 4

Chapter 2: API Interfaces.................................................. 5
API Interfaces Overview. ..................................................... 5
  Supported Input/Output Locations. ...................................... 5
  Output Directories. ........................................................ 6
Guidelines for Java API. ....................................................... 6
Guidelines for .NET API. .................................................... 6
Guidelines for C and C++ API. ............................................. 6
  Guidelines for C and C++ API on AIX. ............................... 7
  Guidelines for C and C++ API on HP-UX. ............................ 7
  Guidelines for C and C++ API on Linux. ............................. 7
  Guidelines for C and C++ API on Solaris. ........................... 7
  Guidelines for C and C++ API on Windows. ........................ 8
Guidelines for Web Service API. .......................................... 8
  Configuring WebSphere Application Server. ......................... 8
  Configuring WebLogic Server. .......................................... 9
Sample Web Service API Client. ......................................... 9

Chapter 3: CGI Interface..................................................... 10
CGI Interface Overview. .................................................. 10
Installing the CGI Interface Component. ............................ 10
Using the CGI Interface. ................................................... 11
  Using the Parameters. .................................................... 12
<table>
<thead>
<tr>
<th>Chapter 4: Event Logs</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the Event Log for Troubleshooting</td>
<td>13</td>
</tr>
<tr>
<td>Log Generation</td>
<td>13</td>
</tr>
<tr>
<td>Log Configuration</td>
<td>14</td>
</tr>
<tr>
<td>Engine Initialization Event Log</td>
<td>15</td>
</tr>
<tr>
<td>User Logs</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 5: Data Transformation Server</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Transformation Server Overview</td>
<td>16</td>
</tr>
<tr>
<td>Configuring Data Transformation to Run Out of Process</td>
<td>16</td>
</tr>
<tr>
<td>Terminating the Server Threads</td>
<td>17</td>
</tr>
<tr>
<td>Troubleshooting Firewall Settings</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 6: Custom Script Components</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom Script Components Overview</td>
<td>18</td>
</tr>
<tr>
<td>Custom Component Example</td>
<td>18</td>
</tr>
<tr>
<td>Supported Property Types</td>
<td>19</td>
</tr>
<tr>
<td>Developing a Custom Component in Java</td>
<td>19</td>
</tr>
<tr>
<td>Interface Example</td>
<td>20</td>
</tr>
<tr>
<td>Sample Custom Components</td>
<td>20</td>
</tr>
<tr>
<td>Developing a Custom Component in C or C++</td>
<td>21</td>
</tr>
<tr>
<td>Limitation</td>
<td>22</td>
</tr>
<tr>
<td>Interface Example</td>
<td>22</td>
</tr>
<tr>
<td>Online Samples</td>
<td>22</td>
</tr>
<tr>
<td>Configuring an External Component</td>
<td>23</td>
</tr>
<tr>
<td>Online Samples</td>
<td>24</td>
</tr>
<tr>
<td>Other Ways to Run Custom Code</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>26</td>
</tr>
</tbody>
</table>
Preface

The Data Transformation Engine Developer Guide is written for software developers who are responsible for testing and activating transformations in a production environment. This guide explains how to run Data Transformation services from the command line, from applications that use an API, and from web applications. In addition, it explains how to program custom components that run within Data Transformation.

This guide assumes that you are familiar with programming languages such as C++ and Java.

Informatica Resources

Informatica My Support Portal


The site contains product information, user group information, newsletters, access to the Informatica customer support case management system (ATLAS), the Informatica How-To Library, the Informatica Knowledge Base, Informatica Product Documentation, and access to the Informatica user community.

Informatica Documentation

The Informatica Documentation team takes every effort to create accurate, usable documentation. If you have questions, comments, or ideas about this documentation, contact the Informatica Documentation team through email at infa_documentation@informatica.com. We will use your feedback to improve our documentation. Let us know if we can contact you regarding your comments.

The Documentation team updates documentation as needed. To get the latest documentation for your product, navigate to Product Documentation from http://mysupport.informatica.com.

Informatica Web Site

You can access the Informatica corporate web site at http://www.informatica.com. The site contains information about Informatica, its background, upcoming events, and sales offices. You will also find product and partner information. The services area of the site includes important information about technical support, training and education, and implementation services.

Informatica How-To Library

As an Informatica customer, you can access the Informatica How-To Library at http://mysupport.informatica.com. The How-To Library is a collection of resources to help you learn more
about Informatica products and features. It includes articles and interactive demonstrations that provide solutions to common problems, compare features and behaviors, and guide you through performing specific real-world tasks.

Informatica Knowledge Base

As an Informatica customer, you can access the Informatica Knowledge Base at http://mysupport.informatica.com. Use the Knowledge Base to search for documented solutions to known technical issues about Informatica products. You can also find answers to frequently asked questions, technical white papers, and technical tips. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team through email at KB_Feedback@informatica.com.

Informatica Support YouTube Channel

You can access the Informatica Support YouTube channel at http://www.youtube.com/user/INFASupport. The Informatica Support YouTube channel includes videos about solutions that guide you through performing specific tasks. If you have questions, comments, or ideas about the Informatica Support YouTube channel, contact the Support YouTube team through email at supportvideos@informatica.com or send a tweet to @INFASupport.

Informatica Marketplace

The Informatica Marketplace is a forum where developers and partners can share solutions that augment, extend, or enhance data integration implementations. By leveraging any of the hundreds of solutions available on the Marketplace, you can improve your productivity and speed up time to implementation on your projects. You can access Informatica Marketplace at http://www.informaticamarketplace.com.

Informatica Velocity

You can access Informatica Velocity at http://mysupport.informatica.com. Developed from the real-world experience of hundreds of data management projects, Informatica Velocity represents the collective knowledge of our consultants who have worked with organizations from around the world to plan, develop, deploy, and maintain successful data management solutions. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Global Customer Support

You can contact a Customer Support Center by telephone or through the Online Support. Online Support requires a user name and password. You can request a user name and password at http://mysupport.informatica.com.

CHAPTER 1

Command-Line Interface

This chapter includes the following topics:

- Command-Line Interface Overview, 1
- CM_console, 1
- Input Documents, 3
- Web Server Authentication, 4

Command-Line Interface Overview

You can use the command line to run a Data Transformation service on a Linux, UNIX, or Windows machine.

On a Linux or UNIX machine, open a command line.

On a Windows machine, open a command window.

CM_console

Runs a Data Transformation service.

The CM_console command uses the following syntax:

```
CM_console <ServiceName>
[< -f | -u | -t >InputDocument]
[ -aServiceParameter=InitialValue]
[ -o<[Path]FileName | FileName>]
[ -r<curr | res | spec=OutputDirectory | guid>]
[ -lUserName -pPassword]
[ -v]
[ -S]
[ -x<f | u | t>InputPortName=InputDocument]
[ -xoOutputPortName=OutputDocument]
[ -e]
```
Note: Do not include a space between an option and its argument.

The following table describes CM_console options and arguments:

<table>
<thead>
<tr>
<th>Option</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>ServiceName</td>
<td>Required. Specifies the name of the service.</td>
</tr>
<tr>
<td>-f</td>
<td>InputDocument</td>
<td>Optional. Specifies a path and file name on the local file system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>By default, the service uses the document defined in the example_source property of the startup component.</td>
</tr>
<tr>
<td>-t</td>
<td>InputDocument</td>
<td>Optional. Specifies a string surrounded by double quotes.</td>
</tr>
<tr>
<td>-u</td>
<td>InputDocument</td>
<td>Optional. Specifies a URL.</td>
</tr>
<tr>
<td>-a</td>
<td>ServiceParameter=InitialValue</td>
<td>Optional. Specifies an input parameter for the service. ServiceParameter is the name of a variable as defined in the service. InitialValue must be of a data type that is valid for the defined variable. You can enter multiple input parameters, separated by spaces.</td>
</tr>
<tr>
<td>-o</td>
<td>FileName</td>
<td>Optional. Directs output to Path/FileName. If you enter only FileName, you must define the Path with the -r option. By default, the CM_console command directs output to the screen.</td>
</tr>
<tr>
<td>-r</td>
<td>curr</td>
<td>Optional. Specifies the directory from which you ran the CM_console command.</td>
</tr>
<tr>
<td>-r</td>
<td>res</td>
<td>Optional. Specifies the results subdirectory under the directory that holds the service in the filesystem repository.</td>
</tr>
<tr>
<td>-r</td>
<td>spec=OutputDirectory</td>
<td>Optional. Specifies a directory on the local file system.</td>
</tr>
<tr>
<td>-r</td>
<td>guid</td>
<td>Optional. Specifies a directory with a unique name under the CMReports/tmp directory. You can use the configuration editor to change the location of this directory.</td>
</tr>
<tr>
<td>-l</td>
<td>UserName</td>
<td>Required when you use HTTP authentication. Specifies the user name for HTTP authentication. Note: This option is a lower-case L.</td>
</tr>
<tr>
<td>-p</td>
<td>Password</td>
<td>Required when you use HTTP authentication. Specifies the password for HTTP authentication.</td>
</tr>
<tr>
<td>-v</td>
<td>-</td>
<td>Optional. Displays verbose information about the Data Transformation version, the version of the Data Transformation syntax, the setup package identifier, the license, and other information.</td>
</tr>
<tr>
<td>-S</td>
<td>-</td>
<td>Required if the startup component of the service is a streamer. You must also use the -f option to define the input file.</td>
</tr>
<tr>
<td>-xf</td>
<td>InputPortName=InputDocument</td>
<td>Optional. InputPortName specifies the name of an AdditionalInputPort defined in the service. InputDocument specifies a path and file name on the local file system. You can enter multiple input ports, separated by spaces.</td>
</tr>
</tbody>
</table>
### Input Documents

An input document is a file on the local file system, a URL, or a hard-coded string.

You can define the input document in the script or in input parameters. The procedure for the main input document differs from the procedure for additional documents.

#### Main Input

In the script, you can define a main input document to read when you run the project in the Studio. You can also define a list of documents to read in the production environment and in the Studio.

Use the `example_source` property of the main parser, mapper, or serializer to define the main input document for the project when you run it in the Studio.

Use the `sources_to_extract` property of the main parser to define a list of input documents for both the Studio and the production environment.

**Note:** If you define the `sources_to_extract` property and you want to specify the input document with the input parameters, you must configure the `sources_to_extract > pre_processor` property to be the same as the `example_source > pre_processor` property.

When you use the CM_console command to run the service from the repository in the production environment, use the `-f`, `-u`, or `-t` option to specify the input document for the main input. When you use another API, use the applicable input parameters.
AdditionalInputPort

In the script, you can define an additional input document to read when you run the project in the Studio.

Use the example_source property of the AdditionalInputPort to define the input document for the port when you run the project in the Studio.

When you use the CM_console command to run the service from the repository in the production environment, use the -xf, -xu, or -xt option to specify the input document for the port. When you use another API, use the applicable input parameters.

Web Server Authentication

When the service accesses a Web server that requires authentication, specify the user credentials in the project properties or in the input parameters.

Project Properties

When you define the user credentials in the project properties, they are used in the following circumstances:

- When you use the Run command to run the project in the Studio.
- When you use an API to run the service from the repository in the production environment.

1. Click Project > Properties.
   The Properties for <ProjectName> window appears.
2. In the left panel, select Authentication, and then in the right panel, select Enable Authentication.
3. In the Login Information section, enter the user name and password.

Input Parameters

When you define the user credentials with the input parameters of the API, they override the definition in the project properties.

For the CM_console command, use the -l option to define the user name and the -p option to define the password.

For other APIs, use the relevant input parameters.
API Interfaces

This chapter includes the following topics:

• API Interfaces Overview, 5
• Guidelines for Java API, 6
• Guidelines for .NET API, 6
• Guidelines for C and C++ API, 6
• Guidelines for Web Service API, 8

API Interfaces Overview

An application can run a Data Transformation service by calling the Engine through an API.

You can use the following APIs to run the Engine in a standalone Data Transformation configuration:

• .NET API
• C/C++ API
• Java API
• Web Services API

The call to the Engine includes the following information:

• The name of the Data Transformation service to run.
• Names of service parameters, which must correspond to the names of the variables defined in the service.
• Values of service parameters, which must comply with the types defined in the service for the variables.
• Other parameters, as defined in the API.

For more information about the APIs, see the API documentation in the Data Transformation online help.

Supported Input/Output Locations

The Data Transformation APIs support input and output in the form of files, URLs, text strings, buffers, and streams.

If a transformation defines multiple input ports or output ports, the APIs can specify the locations of each port independently. In addition, you can use actions to obtain input from sources such as databases and message queues or to write output to such locations. For more information about ports and actions, see the Data Transformation Studio User Guide.
By combining these approaches, you can use an unlimited number of input/output locations in your applications.

Output Directories

When you run a Data Transformation service through an API, you can write output to a specified directory. You can also create an output directory with a unique, GUID-like name each time the service runs.

Guidelines for Java API

Java programs use the Java API to call the Engine. When you use the Java API, you can run Data Transformation in process or out of process.

The Java API is contained in the following JAR file:

\<INSTALL_DIR>\DataTransformation\api\lib\CM_JavaAPI.jar

For more information about configuring the Engine to run in process or out of process, see "Configuring Data Transformation to Run Out of Process" on page 16.

For more information about the Java classes, see the Data Transformation Java API Reference.

Guidelines for .NET API

Use C#, J#, or Visual Basic .NET with the .NET API to call Data Transformation Engine directly and run it in process.

The namespace of the .NET API is Itemfield.ContentMaster.DotNetAPI.

The 32-bit Data Transformation Studio installation includes .NET version 1.1.

The 64-bit Data Transformation Engine installation includes .NET version 2.

To build a .NET application, perform the following actions:

1. Reference the following DLL:

   \<INSTALL_DIR>\DataTransformation\api\lib\Itemfield.ContentMaster.DotNetAPI.dll

2. Compile the project with the current DLL version.

3. Store a copy of the DLL in the application directory, or register the DLL with the .NET regasm utility.

   Note: If you upgrade the Data Transformation version or install a service release (EBF), you must recompile the .NET project. Copy the new version of the DLL to the application directory, or re-register the DLL.

   For more information, see the Data Transformation .NET API Reference.

Guidelines for C and C++ API

C and C++ programs use the C API to call Data Transformation Engine directly and run it in process. The C++ API is an object-oriented wrapper for the C API.
The CApi.h file contains the C API. Include CApi.h in the C program.
The Api.h file contains the C++ API. Include Api.h in the C++ program.
For more information, see the Data Transformation C and C++ API Reference.

Guidelines for C and C++ API on AIX

On an AIX machine, you can find the C and C++ API in the libCM_Engine.a file in the following directory:

<INSTALL_DIR>/DataTransformation/api/include

Enter the following commands to compile the program and link to the C/C++ API:

xlc_r -qthreaded -DIFUNIX -I$[IFCONTENTMASTER_HOME]/include -c source.cc
xlc_r -qthreaded -brtl -bM:UR -o <application_name> source.o -L$ (IFCONTENTMASTER_HOME)/bin -lCM_Engine

If you omit the -bM:UR flag and the service runs a Java document processor, set the LDR_CNTRL environment variable with the following command:

setenv LDR_CNTRL USERREGS

Guidelines for C and C++ API on HP-UX

On an HP-UX machine, you can find the C and C++ API in the libCM_Engine.so file in the following directory:

<INSTALL_DIR>/DataTransformation/api/include

Enter the following commands to compile the program and link to the C/C++ API:

aCC -mt -AA -DIFUNIX -I$[IFCONTENTMASTER_HOME]/include -c source.cc
aCC -mt -lpthread -lstdv2 -1c sup -lunwind -lc -o <application_name> source.o -L$ (IFCONTENTMASTER_HOME)/bin -lCM_Engine

Guidelines for C and C++ API on Linux

On a Linux machine, you can find the C and C++ API in the libCM_Engine.so file in the following directory:

<INSTALL_DIR>/DataTransformation/api/include

Enter the following commands to compile the program and link to the C/C++ API:

g++ -pthread -DIFUNIX -I$[IFCONTENTMASTER_HOME]/include -c source.cc
g++ -pthread -o <application_name> source.o -L$[IFCONTENTMASTER_HOME]/bin -lCM_Engine

Guidelines for C and C++ API on Solaris

On a Solaris machine, you can find the C and C++ API in the libCM_Engine.so file in the following directory:

<INSTALL_DIR>/DataTransformation/api/include

Enter the following commands to compile the program and link to the C/C++ API:

CC -mt -DIFUNIX -xarch=v8plus -I$[IFCONTENTMASTER_HOME]/include -c source.cc
CC -mt -xarch=v8plus -o <application_name> source.o -L$[IFCONTENTMASTER_HOME]/bin -lCM_Engine
Guidelines for C and C++ API on Windows

On a Windows machine, you can find the C and C++ API in the CME directory in the following directory:

\<INSTALL_DIR\>\DataTransformation\api\include

Perform the following actions to use the C and C++ API in a Microsoft Visual Studio project:

1. In Microsoft Visual Studio, open the Project Properties window.
2. Expand the navigation tree and click Configuration Properties > C/C++ > Code Generation.
3. Set the Runtime Library property to Multi-threaded DLL (/MD).

Guidelines for Web Service API

A Web Service can run Data Transformation services through the Web Service API. The Web Service API calls Data Transformation Engine directly and runs it in process.

You can find the Web Service API in the following directory:

\<INSTALL_DIR\>/DataTransformation/api/lib

The Web Service API runs in the following application servers:

- IBM WebSphere Application Server version 6
- Red Hat JBoss version 4
- BEA WebLogic Server version 10

To use the Web Service API, load the CM_JavaAPI.ear file into the application server, and then restart the server.

For information about the object model of the API, see the Data Transformation Web Service API Reference.

Configuring WebSphere Application Server

To use the Web Service API in WebSphere Application Server, you must add a JAR file and an environment variable to the process definition.

1. In WebSphere Application Server, expand the navigation tree and select Servers > Application Servers.
2. Follow the link to the server name.
3. On the Configuration tab, follow the link to Process Definition.
4. In the Executable Arguments box, enter the full path of the CM_JavaAPI.jar file.

You can find the CM_JavaAPI.jar file in the following directory:

\<INSTALL_DIR\>/DataTransformation/api/lib

5. Follow the links to Environment Entries > New.
6. The following table describes the environment entries that you enter:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>IFCONTENTMASTER_HOME</td>
</tr>
<tr>
<td>Value</td>
<td>&lt;INSTALL_DIR&gt;/DataTransformation</td>
</tr>
</tbody>
</table>

7. Click **OK** to update the configuration.

### Configuring WebLogic Server

Install the EAR file on the WebLogic Server to use the Web Service API.

1. Open the WebLogic Server Console.
2. Log in with your user name and password.
3. Click **Console > Deployments > Install or Update an Application**.
4. Browse to the **CM_API.ear** file, and then click **Upload**.
   
   You can find the **CM_API.ear** file in the following directory:
   
   `<INSTALL_DIR>/DataTransformation/api/lib`

5. In the **Select the Archive for this Application** window, select **CM_API.ear**, and then click **Continue**.
6. In the **Review Your Choices and Deploy** window, click **Deploy**.
7. Click **Console > Deployments > CM_API > CM_API**.
8. On the **Testing** tab, click **index.html**.
   
   A web-service test page appears.
9. Click the WSDL link to download the WSDL file.
   
   If it opens, the installation is successful. The WSDL file is an XML description of the Web Service API.

### Sample Web Service API Client

You can find a sample Web Service API client in the following directory:

`<INSTALL_DIR>/DataTransformation\samples\API\WS_API_DEMO_CLIENT`

Use the sample client as a model for developing your own clients.
CHAPTER 3

CGI Interface

This chapter includes the following topics:

- CGI Interface Overview, 10
- Using the CGI Interface, 11

CGI Interface Overview

Note: The CGI interface is being phased out and is not included in the Data Transformation setup. To obtain a copy of the CGI interface component for use with existing applications, contact Informatica. In new applications, use the Web Service API, which provides improved greatly enhanced capabilities. For more information, see "Guidelines for Web Service API" on page 8.

The CGI interface enables web applications to run Data Transformation services. An application uses the HTTP protocol to access the CGI interface component, which is installed on a web server. The CGI component runs a service. Typically, it returns the output of the service as the HTTP response.

The CGI interface does not support streamers, service parameters, or multiple inputs and outputs.

Installing the CGI Interface Component

Before you can use the CGI interface, you must install the Data Transformation CGI interface component on a web server.

The component runs on any Linux, UNIX, or Windows platform where Data Transformation Engine is installed. It can run under any standard web server, such as IIS or Apache.

To install the CGI component on a web server:

1. Copy the file CM_CGI.exe from the Data Transformation installation directory to the CGI directory of the web server.
   Usually, the name of the CGI directory is cgi-bin. Confirm that the web application has permissions to execute CM_CGI.exe.

2. Confirm that the Data Transformation installation directory is in the system path.
   This is required for the CGI component to activate the Engine.

3. Confirm that the CGI user has the following permissions:
   - Read and execute permission for the Data Transformation installation directory.
   - Read permission for the Data Transformation repository directory.
Using the CGI Interface

To use the CGI interface, a web application can call the CM_CGI.exe component by HTTP, using the following syntax:

http://host/cgi-bin/CM_CGI.exe?parameters

Here, host / cgi-bin is the host name or IP address of the web server, and the CGI path. Insert your host name and path.

The parameters are a standard HTTP query string, in the format

parameter=value&parameter=value...

The query string must use the standard URL encoding. For example, encode space characters as %20.

The following table describes the parameters of the CGI interface:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>The Data Transformation service to run.</td>
</tr>
<tr>
<td>file</td>
<td>Path and filename of the input.</td>
</tr>
<tr>
<td>URL</td>
<td>URL of the input.</td>
</tr>
<tr>
<td>text</td>
<td>A text string that is the input of the service.</td>
</tr>
<tr>
<td>output</td>
<td>An output filename. Optional, the name may contain an absolute or relative path. A relative path is resolved relative to the directory that you specify in the outputlocation option. If you omit this parameter, the CGI interface returns the output as the HTTP response.</td>
</tr>
<tr>
<td>outputlocation</td>
<td>The directory for file output. Assign one of the following values: curr: The current directory. res: The Results subdirectory of the Data Transformation service directory. spec: A specified path. Use outputlocation path to define the path. guid: The output is stored, by default, in a system-generated directory, CMReports\Tmp &lt;Unique_Name&gt;, where CMReports is the Data Transformation log location and &lt;Unique_Name&gt; is a unique identifier constructed from the server name and a GUID. The output of each execution is stored in a separate &lt;Unique_Name&gt; directory.</td>
</tr>
<tr>
<td>outputlocationpath</td>
<td>If outputlocation = spec or guid, the output path.</td>
</tr>
<tr>
<td>user</td>
<td>A user name that the service should use for HTTP authentication, if needed. It overrides the user name in the project properties.</td>
</tr>
<tr>
<td>password</td>
<td>A password that the service should use for HTTP authentication, if needed. It overrides the password in the project properties.</td>
</tr>
</tbody>
</table>
Using the Parameters

Specify at most one of the input parameters: file, URL, or text. If the service runs a serializer, mapper, or transformer, an input parameter is required. If the service runs a parser, the behavior is as follows:

- If you specify an input parameter, it overrides the sources_to_extract property of the parser. If the example_source property of the parser defines a document processor, the service applies it to the input.
- If you omit the input parameters, the parser processes the input that is defined in the sources_to_extract property of the parser. If the sources_to_extract property defines a document processor, the service applies it to the input.

The user and password options are useful if the service accesses a web server that requires authentication. They override the user name and password that are assigned in the project settings.

Example

The following example runs a Data Transformation parsing service called Parse1. The input to the service is a text string "hello world". The XML output is returned as the HTTP response.

http://example.com/cgi-bin/CM_CGI.exe?service=Parse1&text=hello%20world

In the following example, the output is stored in the file c:\temp\result.xml.

http://example.com/cgi-bin/CM_CGI.exe?service=Parse1&text=hello%20world&output=result.xml&outputlocation=spec&outputlocationpath=c:\temp

Testing

Test the CGI interface by entering the HTTP syntax on the URL line of a browser. If you are working on the same computer as the web server, you can enter localhost as the host name, for example:

http://localhost/cgi-bin/CM_CGI.exe?service=Parse1&text=hello%20world
Event Logs

This chapter includes the following topics:

- Using the Event Log for Troubleshooting, 13
- User Logs, 15

Using the Event Log for Troubleshooting

The main troubleshooting tool of Data Transformation Engine is the event log. The event log contains information about the operations that the software performs while processing a document.

Event logs can be generated both when you test a project in the Studio and when you run a service in the Engine. This chapter discusses some specific points about the Engine event logs.

For more information about using the event log for testing and debugging, see the Data Transformation Studio User Guide.

Log Generation

Data Transformation Engine generates event logs when it encounters an error or failure while running a service.

The logs are stored in the CMReports directory.

- On Windows platforms, the default log location is the following directory:
  `<INSTALL_DIR>\DataTransformation\CMReports\Logs`
- On UNIX platforms, the default location is the following directory:
  `<INSTALL_DIR>/CMReports/Logs`

You can customize the location with the configuration editor.

Within this directory, the logs are organized by date and service name:

`19Jan2004\Service1_A115656584-14611-19812-12403\events.cme`

The string `A115656584-14611-19812-12403` is a GUID identifier.

To view a log, drag the `.cme` file to the Events view of Data Transformation Studio.

In addition to the event log, Data Transformation Engine saves a copy of the source document in a subdirectory of the log directory. If you double-click an event in the log, the text that caused the event appears in the IntelliScript editor.

**Note:** If you use the CM_console command to run the Engine or if you test a project in the Studio, the logs are stored in the project Results directory, and not in the CMReports directory.
Disabling Very Large Logs

Data Transformation Studio can display event logs of up to approximately 300 MB. If a service generates logs that are larger than 300 MB, you can disable the log generation.

1. In Data Transformation Studio, open the original project from which you deployed the service.
2. Click Project > Properties > Output Control.
3. Deselect the option to Create Event Log.
4. Redeploy the service.

Log Configuration

You can configure the Engine event logs in the Data Transformation Configuration Editor. For more information on using the Configuration Editor, see the Data Transformation Installation and Configuration Guide.

Log Location

The following table describes the settings in the Configuration Editor that you edit to change the log location:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM Configuration/General/CM Reports directory</td>
<td>The path of the reports directory. If the Configuration Editor does not display this setting, you can add it by right-clicking the CM Configuration/General node, and click Add &gt; CM Reports Directory. You can then enter the setting value.</td>
</tr>
<tr>
<td>CM Configuration/General/CM log files directory</td>
<td>Subdirectory of the reports directory, where Data Transformation stores the event logs. The default value is Logs.</td>
</tr>
</tbody>
</table>

Days to Keep History

Data Transformation purges old Engine event logs and old copies of source documents periodically. The following table describes the parameter that configures log behavior:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM Configuration/CM Engine/ Days to keep history</td>
<td>The number of days that Data Transformation saves event logs and user logs before purging. Default is 4. Only the log locations defined in the Configuration Editor are purged.</td>
</tr>
</tbody>
</table>

Multiple Users

If Data Transformation Engine runs under multiple user accounts, the users' logs may overwrite each other, or it may be difficult to identify the logs belonging to a particular user. You can prevent this by configuring the users with different log locations. For more information about setting up multiple configurations, see the Data Transformation Installation and Configuration Guide.
Engine Initialization Event Log

In addition to the logs of service events, there is an Engine initialization event log. This log records problems that occur when Data Transformation Engine starts, without reference to any service or input data. View this log to diagnose installation problems such as missing environment variables.

The initialization log is located in the `CMReports\Init` directory.

User Logs

A transformation can output failure events to a user-defined log.

If an anchor fails to find text in the source document, it can write a message in the user log. This can occur even if the anchor is defined as optional, so that the failure does not terminate the transformation processing.

The user log can contain the following information:

- Failure level: Information, Warning, or Error
- Name of the component that failed
- Failure description
- Location of the failed component in the script
- Additional information about the transformation status, such as the values of data holders.

To define the user-log output, assign the `on_fail` property of the appropriate transformation components. For more information about the `on_fail` property and failure handling, see the `Data Transformation Studio User Guide`.

On Windows platforms, you can find the user log in the following directory:

```<INSTALL_DIR>\DataTransformation\UserLogs```

On Linux and UNIX platforms, you can find the user log in the following directory:

```<INSTALL_DIR>/UserLogs```

You can customize the location with the configuration editor.

By default, each execution of a transformation generates a user log having a unique name:

```<service_name>+<unique_string>.log```

A transformation can set the user-log location at runtime by using `SetValue` actions to assign the following system variables. Set the phase property of `SetValue` to `initial`, ensuring that `SetValue` runs before any component that writes to the user log. For more information about `SetValue`, see the `Data Transformation Studio User Guide`.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VarServiceInfo/StandardError/StandardErrorDir</code></td>
<td>Directory path of the user log</td>
</tr>
<tr>
<td><code>VarServiceInfo/StandardError/StandardErrorName</code></td>
<td>File name of the user log</td>
</tr>
</tbody>
</table>

Data Transformation purges old user logs stored in the location that is defined in the Configuration Editor. If you store user logs in another location, Data Transformation does not purge them. For more information, see “Days to Keep History” on page 14.
CHAPTER 5

Data Transformation Server

This chapter includes the following topics:

- **Data Transformation Server Overview, 16**
- **Configuring Data Transformation to Run Out of Process, 16**
- **Terminating the Server Threads, 17**
- **Troubleshooting Firewall Settings, 17**

Data Transformation Server Overview

Data Transformation Server is a component of Data Transformation Engine that enables Java programs to call the Engine out of process.

Note the following characteristics of out-of-process invocation:

- 64-bit Java applications can call a 32-bit version of the Engine.
- An Engine failure is less likely to disrupt the calling application, as compared with in-process invocation.

**Note:** For more information about running the Engine out of process, see the *Data Transformation Installation and Configuration Guide*.

Configuring Data Transformation to Run Out of Process

The Data Transformation configuration controls the method of calling the Engine through the Java API.

On a Windows machine, use the configuration editor to set the value of **CM Configuration > CM Engine > Invocation to Server**.

On a Linux or UNIX machine, edit the configuration file to set the value of **CMConfig/CMEngine/Invocation to Server**.

**Note:** For more information about configuration, see the *Data Transformation Installation and Configuration Guide*. 
Terminating the Server Threads

When a Java application invokes the Data Transformation Server, the API creates two new threads, which run indefinitely and read the logging information produced by the Server process.

When the invoking process ends, it must call `System.exit` to terminate the threads.

Troubleshooting Firewall Settings

Data Transformation Server uses TCP/IP to communicate with Java client applications. If an application fails to activate the Server, the communication might fail because of a firewall on the computer that hosts the Server.

Configure the firewall to allow Data Transformation Server and the client application to access the local network.

The firewall must allow Data Transformation Server to receive and initiate connections.

On Windows, the firewall must be open to `cm_server.exe` in the following directory:

`<INSTALL_DIR>\DataTransformation\bin`

On Linux and UNIX, the firewall must be open to `cm_server` and `cm_server.sh` in the following directory:

`<INSTALL_DIR>/DataTransformation/bin`
Custom Script Components

This chapter includes the following topics:

- Custom Script Components Overview, 18
- Developing a Custom Component in Java, 19
- Developing a Custom Component in C or C++, 21
- Configuring an External Component, 23
- Other Ways to Run Custom Code, 24

Custom Script Components Overview

When you design and configure a Data Transformation service, you can use a large number of built-in script components. You can also program custom components, such as document processors or transformers, and insert them into a script. When you export the Data Processor transformation as a Data Transformation service, the service runs the custom components.

You can implement the custom components in Java, C, or C++. This chapter provides programming and configuration guidelines. For more information about the interfaces that you must implement, see the External-Component Java Interface Reference or the External-Component C and C++ Interface Reference.

Custom Component Example

Suppose you need to parse a proprietary binary data format. Rather than parse the binary data directly, you prefer to convert the data to a text representation that is easier to parse.

To do this, you can program a custom document processor, which you might call MyBinaryToText. The following table describes processor properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeepLineBreaks</td>
<td>A Boolean property. When true, the processor preserves the line-break characters in the binary data.</td>
</tr>
<tr>
<td>MaxLineLength</td>
<td>An integer property. Specifies the maximum length of the text lines to output.</td>
</tr>
<tr>
<td>Ignore</td>
<td>A string property. Tells the processor to ignore data fields beginning with the specified string.</td>
</tr>
</tbody>
</table>

After you develop the processor, you can install it and use it in scripts.
Supported Property Types

The properties of a custom component can have integer, Boolean, string, or list-of-string data types. You can assign either a constant property value or the name of a data holder that contains the value.

You can hide some of the properties in the IntelliScript editor. For example, a custom component might support four properties. In its TGP configuration file, you can configure it to display only the first two properties. The script passes only the displayed properties to the component. The component can assign its own default values to the hidden properties.

The maximum number of properties depends on the component type and the language of implementation. The following table describes the component properties:

<table>
<thead>
<tr>
<th>Component type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document processor</td>
<td>Language: Java</td>
</tr>
<tr>
<td></td>
<td>Maximum properties: 4</td>
</tr>
<tr>
<td>Document processor</td>
<td>Language: C or C++</td>
</tr>
<tr>
<td></td>
<td>Maximum properties: 5</td>
</tr>
<tr>
<td>Transformer</td>
<td>Language: Java</td>
</tr>
<tr>
<td></td>
<td>Maximum properties: 10</td>
</tr>
<tr>
<td>Transformer</td>
<td>Language: C or C++</td>
</tr>
<tr>
<td></td>
<td>Maximum properties: 10</td>
</tr>
</tbody>
</table>

Developing a Custom Component in Java

1. Create a class that implements interfaces. The following table describes the interfaces that can be implemented:

<table>
<thead>
<tr>
<th>Component Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document processor</td>
<td>Input: File</td>
</tr>
<tr>
<td></td>
<td>Interface: CMXFileProcessor</td>
</tr>
<tr>
<td>Document processor</td>
<td>Input: Buffer</td>
</tr>
<tr>
<td></td>
<td>Interface: CMXByteArrayProcessor</td>
</tr>
<tr>
<td>Transformer</td>
<td>Input: String</td>
</tr>
<tr>
<td></td>
<td>Interface: CMXStringTransformer</td>
</tr>
<tr>
<td>Transformer</td>
<td>Input: Buffer</td>
</tr>
<tr>
<td></td>
<td>Interface: CMXByteArrayTransform</td>
</tr>
</tbody>
</table>

For more information about these interfaces, see the External-Component Java Interface Reference.

2. Compile the project to a JAR file.
3. Store the JAR in the `externLibs\user` subdirectory of the installation directory of every computer where you plan to use the component.

4. Create a script file that defines the display name of the component and its properties. Store the file in the `autoInclude\user` subdirectory of the installation directory.
   
   For more information about this step, see "Configuring an External Component" on page 23.
   
   You can then use the custom component in transformations.

## Interface Example

As an example, consider a document processor that accepts file input. The processor must implement the `CMXFileProcessor` class, which has the following method:

```java
public String process(
    CMXContext context,
    String in,
    String additionalFilesDir,
    CMXEventReporter reporter)
    throws Exception
```

The following table describes the `CMXFileProcessor` parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>In: An object containing the properties that the script passes to the component. The parameters method of the object returns a vector containing the property values.</td>
</tr>
<tr>
<td>in</td>
<td>In: The full path of the file upon which the component should operate.</td>
</tr>
<tr>
<td>additionalFilesDir</td>
<td>Out: Optionally, the path of a temporary directory where the component writes files. At the end of processing, the script deletes the entire directory content.</td>
</tr>
<tr>
<td>reporter</td>
<td>In: An object providing the report method, which the component can use to write events to the event log.</td>
</tr>
<tr>
<td>reporter</td>
<td>Out: The full path of a file that contains the output of the component.</td>
</tr>
</tbody>
</table>

## Sample Custom Components

For samples of the implementation of the custom components, see the following subdirectory of the installation directory:

`samples\SDK\ExternalParameters\Java_SDK\Java`

The following table describes the samples in the directory:

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FilePP.java</td>
<td>A document processor accepting file input.</td>
</tr>
<tr>
<td>ByteArrayPP.java</td>
<td>A document processor accepting buffer input.</td>
</tr>
<tr>
<td>StringTT.java</td>
<td>A transformer accepting string input.</td>
</tr>
<tr>
<td>ByteArrayTT.java</td>
<td>A transformer accepting buffer input.</td>
</tr>
</tbody>
</table>
Developing a Custom Component in C or C++

1. Create a C or C++ project.
2. Add the following files to the project:
   ```c
   General.c
   Utils.c
   ```
   You can find the files in the following directory:
   ```
   <INSTALL_DIR>/DataTransformation/samples/SDK/ExternalParameters/Cpp_SDK/Cpp
   ```
3. Include all *.h files from the following directories:
   ```
   <INSTALL_DIR>/DataTransformation/samples/SDK/ExternalParameters/Cpp_SDK/Cpp/include
   ```
   ```
   api/include
   ```
4. Set the linker option to add the following subdirectory:
   ```
   <INSTALL_DIR>/DataTransformation/api/lib
   ```
5. Create a module that implements the appropriate functions. The following table describes the components that you can implement:

<table>
<thead>
<tr>
<th>Component Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Document processor | Input Type: File  
                       Interface: CMXProcessFile |
| Document processor | Input type: Multiple files  
                       Interface: CMXProcessMultipleFiles |
| Document processor | Input type: Buffer  
                       Interface: CMXProcessBuffer |
| Document processor | Input type: C++ stream  
                       Interface: CMXProcessStream |
| Transformer     | Input type: Null-terminated string  
                       Interface: CMXTransformBuffer |
| Transformer     | Input type: Buffer input that is not null-terminated  
                       Interface: CMXTransformBinaryBuffer |

There are some restrictions on whether a single module can implement more than one of the above interfaces. For more information about the interfaces, see the External-Component C and C++ Interface Reference.

6. For use on Windows platforms, compile the project to a DLL. For use on Linux or UNIX platforms, compile to a shared object.
7. Store the DLL or the shared object in the `<INSTALL_DIR>/DataTransformation/externLibs/user` directory on every computer where you plan to use the component in a service.
8. Create a script file that defines the display name of the component and its properties. Store the file in the `autoInclude/user` directory.
   For more information about this step, see "Configuring an External Component" on page 23.
You can then use the external component in transformations.
Limitation

The property values that the script can pass to a C or C++ custom component can have lengths of up to 4000 characters.

Interface Example

As an example of one of the C/C++ interfaces, consider a document processor that accepts file input. The processor must implement the `CMXProcessFile` function, which has the following syntax:

```c
int CMXProcessFile(
    void * sessionToken,
    const CMXContext *params,
    const IFfile_char_t * in_file,
    int in_len,
    IFfile_char_t ** out_file,
    int * out_len,
    CMXEventReporter * reporter)
```

The following table describes the `CMXProcessFile` parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sessionToken</td>
<td>In: A pointer to the current session.</td>
</tr>
<tr>
<td>params</td>
<td>In: A structure containing the properties that the script passes to the component.</td>
</tr>
<tr>
<td>in_file</td>
<td>In: The full path of the file upon which the component should operate.</td>
</tr>
<tr>
<td>in_len</td>
<td>In: The length, in bytes, of the input file path.</td>
</tr>
<tr>
<td>out_file</td>
<td>Out: The full path of a file that contains the output of the component.</td>
</tr>
<tr>
<td>out_len</td>
<td>Out: The length, in bytes, of the output file path.</td>
</tr>
<tr>
<td>reporter</td>
<td>In: Provides the <code>report</code> method, which the component can use to write events to the event log.</td>
</tr>
<tr>
<td>Return value</td>
<td>Out: 1 if successful, 0 if unsuccessful.</td>
</tr>
</tbody>
</table>

Online Samples

For online samples of the implementation, see the following subdirectory of the installation directory:

```
samples\SDK\ExternalParameters\Cpp_SDK\Cpp
```

The following tables describes the samples in the directory:

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor.c</td>
<td>A document processor accepting either file or buffer input.</td>
</tr>
<tr>
<td>Transformer.c</td>
<td>A transformer accepting null-terminated string input.</td>
</tr>
</tbody>
</table>
Configuring an External Component

After you develop an external component, you must prepare a script file that defines the component. You cannot prepare the TGP file in the IntelliScript editor. Instead, you must prepare it in a text editor.

After you install the component and the TGP file, you can configure the custom component in the IntelliScript editor.

1. Create a text file and save it with a *.tgp extension.

   **Note:** You can define more than one external component in a single TGP file.

2. For each property that your external component supports, add lines such as the following to the TGP file:
   
   ```
   profile <CustomPropertyName1> ofPT <DataType>
   {
     paramName = "<CustomPropertyName1>" ;
   }
   
   <CustomPropertyName1> is the name of a property that you want to display in the IntelliScript editor.
   <DataType> is the data type of the property. The supported data types are NamedParamIntT for an integer property, NamedParamBoolT for a boolean property, NamedParamStringT for a string property, or NamedParamListT for a property that is a list of strings.
   
   3. For each external component that you wish to define, add lines such as the following to the TGP file. For a Java component:
      
      ```
      profile <ExternalComponentName> ofPT <ComponentType>
      {
        jclass = "<ClassName>" ;
        param1 = <CustomPropertyName1>();
        param2 = <CustomPropertyName2>();
      }
      
      ```
      
      For a C or C++ component:
      
      ```
      profile <ExternalComponentName> ofPT <ComponentType>
      {
        import_dll = DllPath("<DllName>");
        param1 = <CustomPropertyName1>();
        param2 = <CustomPropertyName2>();
      }
      
      ```
      
      <ExternalComponentName> is the name of the external component that you want to display in the IntelliScript editor. The following table describes the <ComponentType> values:

<table>
<thead>
<tr>
<th>For</th>
<th>ComponentType</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Java document processor with 0 to 4 properties</td>
<td>ExternalJavaProcessorNoParamsT</td>
</tr>
<tr>
<td></td>
<td>ExternalJavaProcessor1ParamsT</td>
</tr>
<tr>
<td></td>
<td>ExternalJavaProcessor2ParamsT</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>A C or C++ document processor with 0 to 5 properties</td>
<td>ExternalProcessorNoParamsT</td>
</tr>
<tr>
<td></td>
<td>ExternalProcessor1ParamsT</td>
</tr>
<tr>
<td></td>
<td>ExternalProcessor2ParamsT</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Save the code.

Open a project and insert the custom component in the script. The custom component name, which you store in the project file, appears in the IntelliScript drop-down list. The IntelliScript editor displays its properties.

5. Store the file in the DataTransformation\autoInclude\user subdirectory of the installation directory of every computer where you want to use the component.
6. If the Developer tool is open, close it and re-open it.
7. If an autoInclude error is displayed, review the TGP file for syntax errors or naming inconsistencies, and open the Developer tool again.
8. Open a project and insert the custom component in the script. The custom component name, which you assigned in step 3 above, appears in the IntelliScript drop-down list. The IntelliScript editor displays its properties.

### Online Samples

For online samples of the script files, see the following subdirectories of the installation directory:

- samples\SDK\ExternalParameters\Java_SDK\autoInclude
- samples\SDK\ExternalParameters\Cpp_SDK\autoInclude

### Other Ways to Run Custom Code

The custom components described above in this chapter are not the only way to run custom code within a transformation. You can use components such as document processors and transformers to run custom code.

<table>
<thead>
<tr>
<th>Type</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>XSLTTransformer</td>
<td>Applies an XSLT transformation to XML input text.</td>
</tr>
<tr>
<td>Action</td>
<td>CalculateValue</td>
<td>Performs a computation defined in a JavaScript expression.</td>
</tr>
<tr>
<td>Action</td>
<td>JavaScriptFunction</td>
<td>Runs a JavaScript function.</td>
</tr>
<tr>
<td>Type</td>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Action</td>
<td>WriteValue</td>
<td>Writes data to an external location. Among other options, you can use custom code to write the data.</td>
</tr>
<tr>
<td>Action</td>
<td>XSLTMap</td>
<td>Runs an XSLT transformation on a branch of an XML document.</td>
</tr>
</tbody>
</table>
INDEX

A
AdditionalInputPort
   input documents 3
API interfaces
   description 5
APIs
   web service instructions 8
   autoInclude
      external components 23

C
C and C++ API
   programming guidelines 7
C/C++
   custom components 21
CGI interface
   Engine 10
command-line interface
   CM_console command 1
components, custom
   C/C++ 21
components, custom script
   description 18
   properties 19
configuration
   running Engine out of process 16
connections
   Server firewall 17
custom code
   how to run 24
   custom component
      example 18
custom components
   C/C++ 21
   Java 19
custom script components
   description 18
   properties 19

D
Data Transformation Server
   description 16
Data Transformation service
   running from the command line 1
days to keep history
   event log 14
document, input
   defined 3

E
Engine
   configuring to run out of process 16
   engine event log
   troubleshooting 13
   event log
   days to keep 14
   Engine initialization 15
   for multiple users 14
   troubleshooting 13
   event logs
   configuration 14
   Engine 13
   when generated 13
   example_source property
      defined 3

F
Failure logs
   user-defined 15
firewall
   Server connections 17

H
HTTP interface
   Engine 10

I
initialization
   event log 15
input document
   defined 3
   example_source property 3
   sources_to_extract property 3
input port, additional
input documents 3
input ports
   passing data in APIs 5
input, main
   input documents 3

J
Java
   custom components 19
Java API
   programming instructions 6
log files
  user-defined 15
  logs
    event 13

main input
  input documents 3
  multiple inputs and outputs
    in APIs 5

.NET
  .NET API programming instructions 6

out-of-process invocation
  Engine 16
  output ports
    passing locations in APIs 5

port, additional input
  input documents 3
  properties
    custom script components 19

query string
  CGI interface 11

Red Hat JBoss
  running Web Service API 8

script components, custom
description 18
  properties 19
Server, Data Transformation
  configuring 16
  service, Data Transformation
    running from the command line 1
  sources_to_extract property
defined 3

TGP file
  for external components 23
  threads
    created by Server 17

user logs
  failures 15

Web Service API
  sample client 9
WebLogic Server
  configuring Web Service API 9
WebSphere Application Server
  configuring Web Service API 8