Informatica® B2B Data Transformation
10.4.0

# Engine Developer Guide

Informatica B2B Data Transformation Engine Developer Guide
10.4.0
December 2019

Publication Date: 2019-12-12

# Table of Contents

# Preface

Use the *Data Transformation Engine Developer Guide* to learn how to run Data Transformation services from the command line, applications that use an API, and web applications.

You can also use the guide to lean how to program custom components that run within Data Transformation.

The guide is written for software developers who test and activate transformations in a production environment. It assumes that you are familiar with programming languages such as C++ and Java.

# Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

## Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit https://network.informatica.com.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

## Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

## Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

## Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

## Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find Informatica Velocity resources at http://velocity.informatica.com. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

## Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at https://marketplace.informatica.com.

## Informatica Global Customer Support

You can contact a Global Support Center by telephone or through the Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

To find online support resources on the Informatica Network, visit https://network.informatica.com and select the eSupport option.

CHAPTER 1

# Command-Line Interface

This chapter includes the following topics:



## 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:

| Option | Argument | Description |
|---|---|---|
| - | ServiceName | Required. Specifies the name of the service. |
| -f | InputDocument | Optional. Specifies a path and file name on the local file system. By default, the service uses the document defined in the **example_source** property of the startup component. |
| -t | InputDocument | Optional. Specifies a string surrounded by double quotes. |
| -u | InputDocument | Optional. Specifies a URL. |
| -a | ServiceParameter=InitialValue | 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. |
| -o | FileName<br>[Path]FileName | 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. |
| -r | curr | Optional. Specifies the directory from which you ran the CM_console command. |
| -r | res | Optional. Specifies the `results` subdirectory under the directory that holds the service in the filesystem repository. |
| -r | spec=OutputDirectory | Optional. Specifies a directory on the local file system. |
| -r | guid | 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. |
| -l | UserName | Required when you use HTTP authentication. Specifies the user name for HTTP authentication.<br>**Note:** This option is a lower-case L. |
| -p | Password | Required when you use HTTP authentication. Specifies the password for HTTP authentication. |
| -v | - | 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. |
| -S | - | Required if the startup component of the service is a streamer. You must also use the -f option to define the input file. |
| -xf | InputPortName=InputDocument | 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. |

| Option | Argument | Description |
|---|---|---|
| -xt | InputPortName=InputDocument | Optional. InputPortName specifies the name of an **AdditionalInputPort** defined in the service. InputDocument specifies a string surrounded by double quotes. You can enter multiple input ports, separated by spaces. |
| -xu | InputPortName=InputDocument | Optional. InputPortName specifies the name of an **AdditionalInputPort** defined in the service. InputDocument specifies a URL. You can enter multiple input ports, separated by spaces. |
| -xo | OutputPortName=OutputDocument | Optional. OutputPortName specifies the name of an **AdditionalOutputPort** defined in the service. OutputDocument specifies a path and file name on the local file system. You can enter multiple output ports, separated by spaces. |
| -e | - | Optional. By default, the CM_console command terminates with an exit code of 1 for success and greater than 1 for error. When you include the -e option, the CM_console command terminates with an exit code of 0 for success and greater than 1 for error. |

For example:

```
CM_console XYZparser -fInputFile.txt -aMaxLines=1000 -oResults.xml -rcurr
```

This example calls the XYZparser service, using `InputFile.txt` as the main input document. It gives the value of 1000 to the *MaxLines* parameter, and writes the output to the `Results.xml` file in the directory from which you ran the CM_console command.

# 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 Developer. You can also define a list of documents to read in the production environment and in the Developer.

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 Developer.

Use the **sources_to_extract** property of the main parser to define a list of input documents for both the Developer 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 Developer.

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

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.

CHAPTER 2

# API Interfaces

This chapter includes the following topics:



## 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

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 User Guide*.

By combining these approaches, you can use a 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 Server.

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

# Guidelines for .NET API

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

Data Transformation requires .NET 4 Framework or later.

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

To build a .NET application, see "Building a .NET Application" on page 13.

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

## Building a .NET Application

To build a .NET application, first compile the project with the current DLL version, and register or copy the DLL.

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.

# 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 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 `CM_Engine.lib` file 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).

CHAPTER 3

# Custom Script Components

This chapter includes the following topics:



## 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:

| Property | Description |
|---|---|
| KeepLineBreaks | A Boolean property. When true, the processor preserves the line-break characters in the binary data. |
| MaxLineLength | An integer property. Specifies the maximum length of the text lines to output. |
| Ignore | A string property. Tells the processor to ignore data fields beginning with the specified string. |

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:

| Component type | Description |
|---|---|
| Document processor | Language: Java<br>Maximum properties: 4 |
| Document processor | Language: C or C++<br>Maximum properties: 5 |
| Transformer | Language: Java<br>Maximum properties: 10 |
| Transformer | Language: C or C++<br>Maximum properties: 10 |

# Developing a Custom Component in Java

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

| Component Type | Description |
|---|---|
| Document processor | Input: File<br>Interface: CMXFileProcessor |
| Document processor | Input: Buffer<br>Interface: CMXByteArrayProcessor |
| Transformer | Input: String<br>Interface: CMXStringTransformer |
| Transformer | Input: Buffer<br>Interface: CMXByteArrayTransform |

   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 20.

   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:

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

The following table describes the `CMXFileProcessor` parameters:

| Parameter | Description |
|---|---|
| context | 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. |
| in | In: The full path of the file upon which the component should operate. |
| additionalFilesDir | 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. |
| reporter | In: An object providing the report method, which the component can use to write events to the event log. |
| reporter | Out: The full path of a file that contains the output of the component. |

## 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:

| Sample | Description |
|---|---|
| FilePP.java | A document processor accepting file input. |
| ByteArrayPP.java | A document processor accepting buffer input. |
| StringTT.java | A transformer accepting string input. |
| ByteArrayTT.java | A transformer accepting buffer input. |

# Developing a Custom Component in C or C++

1. Create a C or C++ project.
2. Add the following files to the project:

   ```
   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:

| Component Type | Description |
|---|---|
| Document processor | Input Type: File<br>Interface: CMXProcessFile |
| Document processor | Input type: Multiple files<br>Interface: CMXProcessMultipleFiles |
| Document processor | Input type: Buffer<br>Interface: CMXProcessBuffer |
| Document processor | Input type: C++ stream<br>Interface: CMXProcessStream |
| Transformer | Input type: Null-terminated string<br>Interface: CMXTransformBuffer |
| Transformer | Input type: Buffer input that is not null-terminated<br>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 20.

   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:

```
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:

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

## 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:

| Sample | Description |
|---|---|
| Processor.c | A document processor accepting either file or buffer input. |
| Transformer.c | A transformer accepting null-terminated string input. |

# 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:

| For | ComponentType |
|---|---|
| A Java document processor with 0 to 4 properties | `ExternalJavaProcessorNoParamsT`<br>`ExternalJavaProcessor1ParamsT`<br>`ExternalJavaProcessor2ParamsT`<br>`...` |
| A C or C++ document processor with 0 to 5 properties | `ExternalProcessorNoParamsT`<br>`ExternalProcessor1ParamsT`<br>`ExternalProcessor2ParamsT`<br>`...` |

| For | ComponentType |
|---|---|
| A Java transformer with 0 to 10 properties | `ExternalJavaTransformerNoParamsT`<br>`ExternalJavaTransformer1ParamsT`<br>`ExternalJavaTransformer2ParamsT`<br>`...` |
| A C or C++ transformer with 0 to 10 properties | `ExternalTransformerNoParamsT`<br>`ExternalTransformer1ParamsT`<br>`ExternalTransformer2ParamsT`<br>`...` |

`<ClassName>` is the fully qualified name of the Java class. On Windows, `<DllName>` is the name of the DLL, without the `*.dll` extension. On Linux or UNIX, it is the name of the shared object, without the `lib` prefix or the `*.so`, extension.

`<CustomPropertyName1>` and `<CustomPropertyName2>` are the names of the properties that you configured in step 2.

4. Save the `*.tgp` file.
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.

| Type | Component | Description |
|---|---|---|
| Transformer | XSLTTransformer | Applies an XSLT transformation to XML input text. |
| Action | CalculateValue | Performs a computation defined in a JavaScript expression. |
| Action | JavaScriptFunction | Runs a JavaScript function. |

| Type | Component | Description |
|---|---|---|
| Action | WriteValue | Writes data to an external location. Among other options, you can use custom code to write the data. |
| Action | XSLTMap | Runs an XSLT transformation on a branch of an XML document. |

# INDEX

# T