SAP Conversion Agent by Itemfield

Conversion Agent Engine Developer's Guide

Version 4

Conversion Agent Engine Developer's Guide

SAP AG
http://www.sap.com

Publication Information:

Version: 4
Date: September 2006

Contents

1. Overview 1

Getting Started 1

Prerequisite: Deploy a Conversion Agent Service 1

2. Command-Line Interface 3

Platform Support 3

Accessing the Interface 3

Command-Line Syntax 3

- Options 3
- Examples 5
- Usage Message 6
- Error Messages 6

3. API Programming 7

API Approach 7

Supported Input/Output Locations 7

.NET API Programming 8

COM API Programming 8

Java API Programming 8

C and C++ API Programming 8

4. CGI Interface 11

Installing the CGI Interface Component 11
- Platform Support 11
- Installation Procedure 11

Using the CGI Interface 12
- Parameters 12
- Example 13
- Testing 13

5. Event Logs 14

Log Generation 14

Log Configuration 15

Engine Initialization Event Log 16

Remote Support Interface 16

Other Troubleshooting Tools 16

6. Conversion Agent Server 17

Platform Support 17

Configuring Conversion Agent to Use the Server 18

Invoking Conversion Agent Server 18

Troubleshooting 18

7. External Components 20

Example 20

Property Support 21

Developing an External Component in Java 21

Developing an External Component in C or C++ 23

Configuration Instructions 25

Other Ways to Run Custom Code 28

Index 30

1 Overview

After you develop a Conversion Agent data transformation, you need to run the transformation in Conversion Agent Engine. There are several methods for doing this:

- By using the Conversion Agent *command-line interface*
- By programming an application that calls one of the *Conversion Agent APIs*.
- By using the HTTP protocol to access the Conversion Agent *CGI interface*

Chapters 2 to 4 of this book explain these approaches. In the case of the command-line and CGI interfaces, the book provides full details. Regarding the APIs, the book introduces the approach and provides references to the other Conversion Agent documentation.

The succeeding chapters of the book cover the following subjects:

- The chapter on *Event Logs* describes troubleshooting procedures for Conversion Agent Engine.
- The chapter on *Conversion Agent Server* explains how to run Conversion Agent Engine on 64-bit operating systems or out-of-process on 32-bit systems.
- The chapter on *External Components* explains how to program custom components, which operate within a Conversion Agent service.

Getting Started

Before you read this book, you may wish to review the last chapter of *Getting Started with Conversion Agent*, which contains a tutorial exercise called *Running Conversion Agent Engine*. The exercise illustrates the use of Conversion Agent Engine and presents the source code of an API application.

Prerequisite: Deploy a Conversion Agent Service

A prerequisite for all the approaches described in this book is that you configure a Conversion Agent project, and deploy it as a Conversion Agent service. To learn how to do that, see the *Conversion Agent Studio User's Guide* (especially the last chapter, *Deploying a Conversion Agent Service*).

A service can run a parser, a serializer, a mapper, or a transformer. These components, and the subcomponents from which they are constructed, are fully explained in the *User's Guide*.

In this book, we assume that you have already deployed a Conversion Agent service, and we explain how to run the service in Conversion Agent Engine.

2 Command-Line Interface

The simplest way to run Conversion Agent Engine is by using the Conversion Agent command-line interface. You can use the command-line interface for testing, or you can automate it in a batch file or a shell script.

Platform Support

The command-line interface runs on all operating systems where Conversion Agent Engine can be installed.

Accessing the Interface

To access the command-line interface, open a command prompt and run the `CM_console` command. The command executes `CM_console.exe`, which is located in the Conversion Agent installation folder.

Command-Line Syntax

The syntax of the `CM_console` command is as follows:

```
CM_console service_name options
```

Here, *`service_name`* is the name of the Conversion Agent service to run. The *`options`* are switches preceded by a hyphen. Immediately following the switch, you can enter parameters.

You can enter the parameters with or without surrounding quotes. If a parameter contains spaces, you must surround it with quotes.

Options

The following table lists the supported options of the command-line interface.

You should specify at most one of the input options: `-u -f -t`. If the service runs a serializer, mapper, or transformer, an input option is required. If the service runs a parser, the behavior is as follows:

- If you specify an input option, 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 options, 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 `-l` and `-p` 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 properties.

`-u`

URL of the input.

`-f`

Path and filename of the input.

`-t`

Text string, which is the input of the service.

`-o`

The output filename.

Optionally, the name may contain an absolute or relative path. A relative path is resolved relative to the folder that you specify in the `-r` option. An absolute path overrides the `-r` option.

If you omit the `-o` option, the command writes to the standard output (typically the screen display).

`-r`

The output folder. Enter one of the following values:

- `curr`: The current folder, from which you executed the `CM_console` command.
- `res`: In the Conversion Agent repository, in the `Results` subfolder of the Conversion Agent service.
- `spec=<path>`: A specified absolute path, for example, `-rspec=c:\temp`.
- `guid`: (Default) The output is stored in a system-generated folder, `CMReports\Tmp\<Unique_Name>`, where `CMReports` is the Conversion Agent log location (defined in the Configuration Editor), and `<Unique_Name>` is a unique identifier constructed from the server name and a GUID. The output of each `CM_console` execution is stored in a separate `<Unique_Name>` folder.

`-l`

A user name that the service should use for HTTP authentication, if needed.

`-p`

A password that the service should use for HTTP authentication, if needed.

`-a`

The initial value of a variable (known as a *service parameter*, because you can use this feature to pass parameters to a Conversion Agent service). Enter the value as follows:

```
-avariable_name=value
```

To pass multiple service parameters, you can enter the `-a` switch multiple times. If a value is a string containing spaces, you must enclose it in quotes.

For example, suppose that a data transformation has a string variable called `Name`, an integer variable called `Age`, and a boolean variable called `IsDeveloper`. You can pass the initial values in this way:

```
-aName="Ron Lehrer" -aAge=27 -aIsDeveloper=true
```

`-v`

Displays the Conversion Agent version number, the Conversion Agent Engine syntax version number, the setup package identifier, license validity information, and other information.

Examples

The following command runs a Conversion Agent parsing service called `Tutorial_2` (which is described in the book *Getting Started with Conversion Agent*).

```
CM_console Tutorial_2 -fc:\temp\hl7-obs.txt -omsg.xml -rspec=c:\temp
```

The input of the service is the file `c:\temp\hl7-obs.txt`, whose content is an HL7 message:

```
MSH|^~\&|LAB||CDB||||ORU^R01|K172|P
PID|||PATID1234^5^M11||Jones^William||19610613|M
OBR||||80004^Electrolytes
OBX|1|ST|84295^Na||150|mmol/l|136-148|Above high normal|||Final results
OBX|2|ST|84132^K+||4.5|mmol/l|3.5-5|Normal|||Final results
OBX|3|ST|82435^Cl||102|mmol/l|94-105|Normal|||Final results
OBX|4|ST|82374^CO2||27|mmol/l|24-31|Normal|||Final results
```

After a moment, the output file `c:\temp\msg.xml` is generated. You can view the XML file in Internet Explorer:

```
<?xml version="1.0" encoding="windows-1252"?>
<Message type="ORU" id="K172">
  <Patient id="PATID1234^5^M11" gender="M">
```

```
    <f_name>William</f_name>
    <l_name>Jones</l_name>
    <birth_date>19610613</birth_date>
  </Patient>
  <Test_Type test_id="80004">Electrolytes</Test_Type>
  <Result num="1">
    <type>Na</type>
    <value>150</value>
    <range>136-148</range>
    <comment>Above high normal</comment>
    <status>Final results</status>
  </Result>
  ...
```

The following example runs a service on the string input, `"hello world"` and stores the output in the file `c:\temp\Hello.xml`. Notice that the string `"hello world"` must be enclosed in quotes because it contains a space character. The quotes around `"c:\temp\Hello.xml"` are optional.

```
CM_console MyService –t"hello world" -o"c:\temp\Hello.xml"
```

The following example runs a service on a URL input. The output is a file called `out.xml`, stored in the project `Results` folder.

```
CM_console MyService –uhttp://www.example.com -oout.xml -rres
```

Usage Message

If you enter the `CM_console` command without any service name or options, it displays a usage message, which explains how to use the command.

```
Usage:
CM_console Service name [Additional options]
...
```

Error Messages

In the event of an error, such as an incorrect parameter, the `CM_console` command displays an XML string containing an error message.

If the service encounters an error or a failure, it generates an event log in the project `Results` folder. To view the log, open the `*.cme` file in Conversion Agent Studio (see Chapter 5, *Event Logs*).

3 API Programming

Conversion Agent offers several APIs for application development:

- The .NET and COM APIs, which run on Microsoft Windows platforms
- The Java, C, and C++ APIs, which run on both Windows and non-Windows platforms

Using any of these APIs, you can run services in Conversion Agent Engine.

API Approach

The APIs share a similar programming approach:

1. Program an application that listens for incoming source documents, and instructs Conversion Agent to process them.
2. The application constructs a request and submits it to Conversion Agent Engine. The request includes information such as the name of the Conversion Agent service to run, and the input/output locations.

   Optionally, the application can pass the initial values of variables (known as *service parameters*) to the service. Currently, the Java, C, C++, and .NET APIs support this feature.
3. Conversion Agent executes the request and stores the output in the specified location.
4. As appropriate, the application sends the output to its final destination, such as an information or messaging system.

Supported Input/Output Locations

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

You can use components such as actions and custom document processors to obtain input from additional sources, such as databases and message queues (see the chapters on *Document Processors* and *Actions* in the *Conversion Agent Studio User's Guide*). Likewise, you can use actions to write output to any location.

By combining these approaches, you can use a limitless number of input/output locations in your applications.

.NET API Programming

You can call the .NET API from any language that supports the Microsoft .NET technology, such as C#, J#, or Visual Basic .NET. For information about the .NET object model, see the *Conversion Agent .NET API Reference*. The *API Reference* contains syntax examples in C#. You can use the Object Browser of Visual Studio .NET to view the syntax in other .NET languages.

Your projects should reference the following assembly, which is located in the Conversion Agent installation directory:

`api\lib\Itemfield.ContentMaster.DotNetAPI.dll`

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

COM API Programming

COM API applications should reference the file `CM_COM3.dll`, which is located in the Conversion Agent `bin` directory. For information about the COM object model, see the *Conversion Agent COM API Reference*.

Java API Programming

Java API programs should use the library `CM_JavaAPI.jar`, which is located in the Conversion Agent `api/lib` directory. For information about the Java object model, see the *Java API Reference*.

C and C++ API Programming

The C API contains a small number of functions that access Conversion Agent Engine directly. The C++ API is an object-oriented wrapper for the C API. For information about the syntax and the object model, see the *C and C++ API Reference*.

C applications should include the file `CApi.h`, which is located in the Conversion Agent `api/include` directory. C++ applications should include `Api.h`.

The following paragraphs provide compilation and linking instructions on the supported platforms. The dynamic link libraries (shared objects) are located in the Conversion Agent `api/bin` directory.

AIX

On an IBM AIX platform, you should link to `libCM_Engine.a`. You should compile and link as follows:

```
xlC_r -qthreaded -DIFUNIX -I${IFCONTENTMASTER_HOME}/include -c source.cc

xlC_r -qthreaded -brtl -bM:UR -o <app name> source.o
    -L${IFCONTENTMASTER_HOME}/bin -lCM_Engine
```

If the `-bm:UR` flag was omitted and the service runs a Java document processor, you must set the following environment variable. AIX 5.3 prompts you to set the variable.

```
setenv LDR_CNTRL USERREGS
```

HP-UX

On an HP-UX PA-RISC platform, you should link to `libCM_Engine.sl`. You should compile and link as follows:

```
g++ -pthread -DIFUNIX -I${IFCONTENTMASTER_HOME}/include -c source.cc

g++ -pthread -o <app name> source.o -L${IFCONTENTMASTER_HOME}/bin
    -lCM_Engine
```

On an HP-UX ia64 platform, you should link to `libCM_Engine.so`. You should compile and link as follows:

```
aCC -mt -AA -DIFUNIX -I${IFCONTENTMASTER_HOME}/include -c source.cc

aCC -mt -lpthread -lstd_v2 -lCsup -lunwind –lc -o <app name> source.o
    -L${IFCONTENTMASTER_HOME}/bin -lCM_Engine
```

Linux

On a Linux platform, you should link to `libCM_Engine.so`. You should compile and link as follows:

```
g++ -pthread -DIFUNIX -I${IFCONTENTMASTER_HOME}/include -c source.cc

g++ -pthread -o <app name> source.o -L${IFCONTENTMASTER_HOME}/bin
    -lCM_Engine
```

Solaris

On a Sun Solaris platform, you should link to `libCM_Engine.so`. You should compile and link as follows:

```
CC -mt -DIFUNIX -xarch=v8plus -I${IFCONTENTMASTER_HOME}/include
    -c source.cc

CC -mt -xarch=v8plus -o <app name> source.o -L${IFCONTENTMASTER_HOME}/bin
    -lCM_Engine
```

Windows

On a Microsoft Windows platform, you should link to `CM_Engine.lib`. In Visual Studio .NET 2003, you should set the project properties as follows:

1. In the left pane, expand the tree and select Configuration Properties > C/C++ > Code Generation.
2. In the right pane, set the Runtime Library option to Multi-threaded DLL (/MD).

4 CGI Interface

The Conversion Agent CGI interface lets web applications run Conversion Agent 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 Conversion Agent service. Typically, it returns the output of the service as the HTTP response.

Installing the CGI Interface Component

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

The component is called `CM_CGI.exe`, and it is located in the Conversion Agent installation folder.

Platform Support

The CGI component runs on any Windows or Unix platform where Conversion Agent Engine is installed. It can run under any standard web server, such as IIS or Apache.

Installation Procedure

To install the CGI component on the web server, follow this procedure:

1. Copy `CM_CGI.exe` from the Conversion Agent folder to the web server's CGI folder. Usually, the name of this folder is `cgi-bin`. Make sure that the access rights allow the execution of this module.
2. Confirm that the Conversion Agent installation folder is in the system path. This is required for the CGI component to activate Conversion Agent Engine.
3. Confirm that the CGI user has the following permissions:
   - Read and execute permission for the Conversion Agent installation folder
   - Read permission for the Conversion Agent repository (by default, `ConversionAgent\ServiceDB`)
   - Read and write permission for the log location and its subfolders (`CMReports`, see Chapter 5, *Event Logs*)

Using the CGI Interface

To use the CGI interface, a web application should 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. You should insert your host name and path.

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

`parameter=value&parameter=value&...`

The query string should use the standard URL encoding. For example, space characters should be encoded as `%20`.

Parameters

The following table lists the supported parameters of the CGI interface.

You should 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.

`service`
: The Conversion Agent service to run.

`file`
: Path and filename of the input.

`URL`
: URL of the input.

`text`
: Text string, which is the input of the service.

`output`
: An output filename.

Optionally, the name may contain an absolute or relative path. A relative path is resolved relative to the folder that you specify in the `outputlocation` option.

If you omit this parameter, the CGI interface returns the output as the HTTP response.

`outputlocation`
The folder for file output. Assign one of the following values:

- `curr`: The current folder.
- `res`: The project `Results` folder.
- `spec`: A specified path (use `outputlocationpath` to define the path)
- `guid`: (Default) The output is stored in a system-generated folder, `CMReports\Tmp\<Unique_Name>`, where `CMReports` is the Conversion Agent log location (defined in the Configuration Editor), and `<Unique_Name>` is a unique identifier constructed from the server name and a GUID. The output of each execution is stored in a separate `<Unique_Name>` folder.

`outputlocationpath`
If `outputlocation` = `spec` or `guid`, the output path.

`user`
A user name that the service should use for HTTP authentication, if needed (overrides the user name in the project properties).

`password`
A password that the service should use for HTTP authentication, if needed (overrides the password in the project properties).

Example

The following example runs a Conversion Agent 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

You can 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
```

5 Event Logs

The main troubleshooting tool of Conversion Agent Engine is the event log. The event log contains a wealth of information about the operations that Conversion Agent performs while processing a document.

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

For information about interpreting the event log, see the chapter on *Testing and Debugging* in the *Conversion Agent Studio User's Guide*.

Log Generation

By default, Conversion Agent Engine does not generate event logs. If it encounters an error or failure while running a service, it re-runs the request and generates an event log.

The logs are stored under the `CMReports` folder, whose location can be defined in the Configuration Editor.

- On Microsoft Windows platforms, the default location is:

  `c:\Documents and Settings\<USER>\Application Data\SAP\ConversionAgent\4.0\CMReports\Logs`

  where `<USER>` is the user name under which the Conversion Agent application runs.

- On Unix platforms, the default location is:

  `<INSTALL_DIR>/CMReports/Logs`

  where `<INSTALL_DIR>` is the Conversion Agent installation directory.

Within this folder, the logs are organized by date and service name, for example,

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

where `A115656584-14611-19812-12403` is a GUID identifier.

To view a log, drag the `*.cme` file to the Events view of Conversion Agent Studio.

In addition to the event log, Conversion Agent Engine saves a copy of the source document in a subfolder of the log folder. If you double-click an event in the log,

Conversion Agent Studio displays the text that caused the event in the IntelliScript editor.

✓ *If you use the `CM_console` command to run Conversion Agent Engine, or if you test a project in Conversion Agent Studio, the logs are stored in the project `Results` folder, and not in the above location.*

Log Configuration

You can configure the Engine event logs in the Conversion Agent Configuration Editor. For instructions on using the Configuration Editor, see the *Conversion Agent Administrator's Guide.*

Log Location

To change the log location, edit the following settings in the Configuration Editor.

`CM Configuration/General/CM Reports directory`
: The path of the reports folder.

  If the Configuration Editor does not display this setting, you can add it by right-clicking the `CM Configuration/General` node, and click Add > CM Reports Directory. You may then enter the setting value.

`CM Configuration/General/CM log files directory`
: Subfolder of the reports folder, where Conversion Agent stores the event logs (by default, `Logs`).

Days to Keep History

Conversion Agent purges old Engine event logs and old copies of source documents periodically. The following parameter configures the behavior:

`CM Configuration/CM Engine/Days to keep history`
: The number of days that Conversion Agent should save the event information before purging (by default, 4 days).

Multiple Users

If Conversion Agent 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. To set up the multiple configurations, see *Configuration Editor* in the *Conversion Agent Administrator's Guide*.

Engine Initialization Event Log

In addition to the logs of service events, there is an Engine initialization event log, which reports problems that occur when Conversion Agent Engine starts, without reference to any service or input data. You can view this log to diagnose installation problems such as missing environment variables.

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

Remote Support Interface

On SAP XI systems, you can use the *remote support interface* to view event logs in a web browser over an HTTP connection. For information, see the manual *Deploying and Using Conversion Agent*.

Other Troubleshooting Tools

In edition to the event logs, you can use the visual tools provided by Conversion Agent Studio to troubleshoot a Conversion Agent project. For information, see the *Conversion Agent Studio User's Guide*.

6 Conversion Agent Server

Conversion Agent Server is a module that allows an application to run Conversion Agent Engine out-of-process. This has the following advantages:

- It allows 64-bit applications to activate 32-bit versions of Conversion Agent Engine.
- Because Conversion Agent Engine runs out-of-process, an Engine failure is less likely to disrupt the calling application.

Platform Support

Conversion Agent Server is available for use on any platform where Conversion Agent Engine is installed. It does not require a separate installation.

32-Bit Support

On 32-bit operating systems, the use of Conversion Agent Server is optional. You can do this in order to run Conversion Agent Engine out-of-process.

64-Bit Support

On certain 64-bit operating systems, Conversion Agent Engine runs as a native 64-bit application. On these platforms, a 64-bit application can activate Conversion Agent Engine in-process. You do not need to use Conversion Agent Server.

On other 64-bit operating systems, Conversion Agent Engine runs as a 32-bit application. On these platforms, a 64-bit application cannot run Conversion Agent Engine in-process. It must use Conversion Agent Server to activate Conversion Agent Engine out-of-process.

For a detailed list of the operating systems where Conversion Agent runs as a 64-bit or 32-bit application, see the *Conversion Agent Administrator's Guide*.

Limitations

Some of the Conversion Agent APIs may be unavailable when running out-of-process (see *Invoking Conversion Agent Server* below). Running in-process may give faster performance than out-of-process.

Configuring Conversion Agent to Use the Server

For instructions on configuring Conversion Agent to use the Server, see the section entitled *Running In-Process or Out-of-Process* in the *Conversion Agent Administrator's Guide.*

Invoking Conversion Agent Server

After you complete the Conversion Agent Server configuration, the Server works transparently. All requests that you submit to Conversion Agent Engine are processed via the Server. There are no special API calls or commands to use it.

Supported Invocation Methods

The Conversion Agent Server supports requests received from:

- The Conversion Agent Java API
- The Conversion Agent process module for SAP XI
- Please contact SAP regarding the Server support for requests submitted by other methods, such as the command-line interface, other APIs, and the CGI interface.

Instructions for Java API

There are no special instructions for invoking the Server from the Java API. The same Java executables can use either in-process invocation or the Server.

Terminating the Server Threads

When an API application invokes the Conversion Agent 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 should terminate the threads by calling `System.exit`.

Troubleshooting

Firewall Settings

Conversion Agent Server communicates with Java client applications by TCP/IP. If an application fails to activate the Server, the problem may be a firewall that is installed on the computer, which blocks the communication.

You should configure the firewall to allow both Conversion Agent Server and the client application to access the local network.

The firewall must allow Conversion Agent Server to receive and initiate connections. On Windows, the firewall must be open to `cm_server.exe`. On Unix, it must be open to `cm_server` and `cm_server.sh`. The files are located in the Conversion Agent `bin` directory.

7 External Components

When you design and configure a Conversion Agent service, you can incorporate a large number of out-of-the-box components, which are described in the *Conversion Agent Studio User's Guide*. In addition to these built-in components, you can program custom *external components*, such as document processors and transformers.

The external components operate in exactly the same way as the built-in components. In Conversion Agent Studio, you can configure data transformations that use the external components. You can then deploy the data transformations as Conversion Agent services, which run the components.

You can use external components in any combination with the built-in components. For example, you can construct a parser that activates an external transformer, in addition to built-in transformers. By assigning the properties of the external transformer, the parser can pass data or option-settings to the transformer.

You can implement the external components in Java, C, or C++. This chapter provides programming and configuration guidelines. For details of the interfaces that you must implement, see the *External-Component Java Interface Reference* or the *External-Component C and C++ Interface Reference*.

In addition to the approach described in this chapter, there are other components and interfaces that let you execute custom code within data transformations. For a summary, see *Other Ways to Run Custom Code* at the end of the chapter.

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, which is easier to parse.

To do this, you can program an external document processor, which you might call `MyBinaryToText`. The processor might have properties such as the following:

`KeepLineBreaks`
: A boolean property. If true, the processor preserves the line-break characters that exist in the binary data.

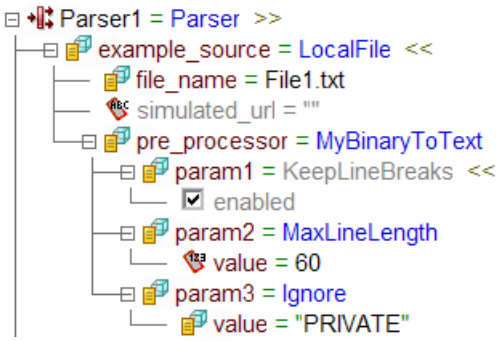
`MaxLineLength`

An integer property, specifying the maximum length of the text lines that the processor should output.

`Ignore`

A string property, which tells the processor to ignore data fields beginning with the specified string.

After you develop the processor, you can install it in Conversion Agent and use it in data transformation. For example, you might configure it as follows in the IntelliScript.



Property Support

External components that have 0 to 4 properties are supported. The properties can have integer, boolean, string, or list-of-string data types. The integer and string types support dynamic assignment. That is, in the IntelliScript, the user can assign either a constant property value or the name of a data holder that contains the value.

It is not compulsory to display all the properties in the IntelliScript. For example, a component may support four properties. In its TGP configuration file, you can configure it to display only the first two properties in the IntelliScript (see the *Configuration Instructions* below). Conversion Agent passes only the displayed properties to the component. The component can assign its own default values to the undisplayed properties.

Developing an External Component in Java

To develop an external component in Java:

1. Create a class that implements one or more of the following interfaces:

Component type	Type of input	Interface
Document processor	File	`CMXFileProcessor`
	Buffer	`CMXByteArrayProcessor`
Transformer	String	`CMXStringTransformer`
	Buffer	`CMXByteArrayTransform`

For detailed 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 Conversion Agent `externLibs\user` directory. You must do this on the Conversion Agent Studio computer and on any other computer where you plan to use the component in a Conversion Agent service.
4. Create a TGP script file that defines the display name of the component and its properties. Store the file in the Conversion Agent `autoInclude\user` directory. For more information, see *Configuration Instructions* below.

You can then use the external component in data 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 meaning of the parameters is as follows:

`context`

(In) An object containing the properties that Conversion Agent 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, Conversion Agent 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.

(Out) The full path of a file, which contains the output of the component.

Online Samples

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

`samples\SDK\ExternalParameters\Java_SDK\Java`

The directory contains the following samples:

`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 an External Component in C or C++

To develop an external 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 Conversion Agent installation directory, under:

   `samples/SDK/ExternalParameters/Cpp_SDK/Cpp`
3. Include all `*.h` files located in the following Conversion Agent directories:

   ```
   samples/SDK/ExternalParameters/Cpp_SDK/Cpp/include
   api/include
   ```
4. Set the linker option to add the following Conversion Agent subdirectory:

   `api/lib`
5. Create a module that implements the appropriate functions:

Component type	Type of input	Interface

Component type	Type of input	Interface
Document processor	File	CMXProcessFile
	Multiple files	CMXProcessMultipleFiles
	Buffer	CMXProcessBuffer
	C++ stream	CMXProcessStream
Transformer	Null-terminated string	CMXTransformBuffer
	Buffer input that is not null-terminated	CMXTransformBinaryBuffer

For detailed information about these interfaces, see the *External-Component C and C++ Interface Reference*.

Note: There are some restrictions on whether a single module can implement more than one of the above functions. For details, see the *Reference*.

6. For use on Windows platforms, compile the project to a DLL. For use on Unix-type platforms, compile to a shared object.
7. Store the DLL or the shared object in the Conversion Agent `externLibs\user` directory. You must do this on the Conversion Agent Studio computer and on any other computer where you plan to use the component in a Conversion Agent service.
8. Create a TGP script file that defines the display name of the component and its properties. Store the file in the Conversion Agent `autoInclude\user` directory. For more information, see *Configuration Instructions* below.

You can then use the external component in data transformations.

Limitation

The property values that Conversion Agent can pass to a C or C++ external component can have lengths of up to 4000 characters. For information about how to increase the maximum size, please contact SAP support.

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 meaning of the parameters is as follows:

`sessionToken`
: (In) A pointer to the current session.

`params`
: (In) A structure containing the properties that Conversion Agent 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, which 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 Conversion Agent installation directory:

`samples\SDK\ExternalParameters\Cpp_SDK\Cpp`

The directory contains the following samples:

`Processor.c`
: A document processor accepting either file or buffer input.

`Transformer.c`
: A transformer accepting null-terminated string input.

Configuration Instructions

After you develop an external component, you must prepare a TGP script file that defines the component in Conversion Agent. You cannot prepare the TGP file in the IntelliScript editor of Conversion Agent Studio. Instead, you should prepare it in a text editor such as Notepad.

After you install the component and the TGP file, the external component becomes configurable in the IntelliScript.

The procedure is as follows:

1. In Notepad, create a file and save it with a `*.tgp` filename, such as `MyExternalComponents.tgp`. It is permitted to 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 the property, which you want to display in the IntelliScript. *DataType* is the data type of the property, which must be `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, which you want to display in the IntelliScript. *ComponentType* is one of the following values:

For:	ComponentType
A Java document processor, with 0 to 4 properties respectively	ExternalJavaProcessorNoParamsT ExternalJavaProcessor1ParamsT ExternalJavaProcessor2ParamsT ExternalJavaProcessor3ParamsT ExternalJavaProcessor4ParamsT
A C or C++ document processor, with 0 to 4 properties respectively	ExternalProcessorNoParamsT ExternalProcessor1ParamsT ExternalProcessor2ParamsT ExternalProcessor3ParamsT ExternalProcessor4ParamsT
A Java transformer, with 0 to 4 properties respectively	ExternalJavaTransformerNoParamsT ExternalJavaTransformer1ParamsT ExternalJavaTransformer2ParamsT ExternalJavaTransformer3ParamsT ExternalJavaTransformer4ParamsT
A C or C++ transformer, with 0 to 4 properties respectively	ExternalTransformerNoParamsT ExternalTransformer1ParamsT ExternalTransformer2ParamsT ExternalTransformer3ParamsT ExternalTransformer4ParamsT

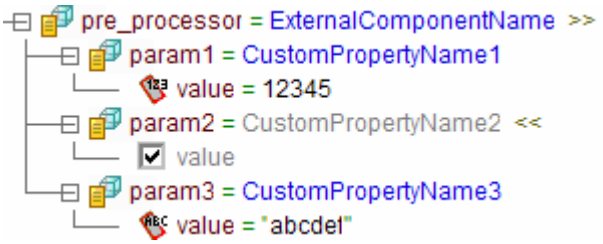
ClassName is the fully qualified name of the Java class. On Windows, *DllName* is the name of the DLL(without the `*.dll` extension). On Unix, it is the name of the shared object (without the `lib` prefix or the `*.so`, etc., extension).

CustomPropertyName1, *CustomPropertyName2*, etc., are the names of the properties, which you configured in step 2.

4. Save the `*.tgp` file.
5. Store the file in the `autoInclude\user` subdirectory of your Conversion Agent installation directory. You must do this on the Conversion Agent Studio computer and on every computer where you want Conversion Agent Engine to use the component.
6. If Conversion Agent Studio is currently open, close and re-open it.

   If an `autoInclude` error is displayed, review the `*.tgp` file for syntax errors or naming inconsistencies, and try to open Conversion Agent Studio again.

7. In Conversion Agent Studio, open a project, and try to insert the external component in the IntelliScript. The external component name, which you assigned in step 3 above, should be displayed in the IntelliScript drop-down list. The IntelliScript should display its properties.



IntelliScript illustrating an external component with three properties. The properties have integer, boolean, and string data types, respectively.

Online Samples

For online samples of the TGP files, see the following subdirectories of the Conversion Agent installation directory.

`samples\SDK\ExternalParameters\Java_SDK\autoInclude`

`samples\SDK\ExternalParameters\Cpp_SDK\autoInclude`

Other Ways to Run Custom Code

The external components described above in this chapter are not the only way to run custom code within a Conversion Agent data transformation. You can use components such as the following to run custom code of various types. For detailed information, see the *Conversion Agent Studio User's Guide*.

✓ *The word* deprecated*, in the description of a component, means that the component is supported for backwards compatibility with existing applications. For new applications, we recommend that you use the external-component approach that is described above. Among other advantages, the latter approach lets Conversion Agent pass properties to the components.*

Document Processors

`ExternalCOMPreProcessor`

Runs a custom document processor, implemented as a COM DLL (for use on Microsoft Windows platforms only).

`ExternalJavaPreProcessor`

(Deprecated) Runs a custom document processor, implemented in Java.

`ExternalPreProcessor`
: (Deprecated) Run a custom document processor, implemented as a C++ DLL.

Transformers

`ExternalTransformer`
: (Deprecated) Runs a custom transformer that is implemented as a C++ DLL.

`JavaTransformer`
: (Deprecated) Runs a custom transformer that is implemented in Java.

`XSLTTransformer`
: Applies an XSLT transformation to XML input text.

Actions

`CalculateValue`
: Performs a computation defined in a JavaScript expression.

`ExternalCOMAction`
: Runs a custom action that is implemented as a COM DLL (for use on Microsoft Windows platforms only).

`JavaScriptFunction`
: Runs a JavaScript function.

`WriteValue`
: Writes data to an external location. Among other options, you can use custom code to write the data.

`XSLTMap`
: Runs an XSLT transformation on a branch of an XML document.

Index

.

.NET API
  programming instructions, 8

6

64-bit applications
  activating Conversion Agent in, 17

A

APIs
  C and C++ overview, 7
  COM overview, 7
  Java overview, 7
autoInclude
  external components, 25

B

batch files
  running Conversion Agent in, 3

C

C and C++ API
  programming instructions, 8
C/C++
  external components, 23
CGI interface
  Conversion Agent Engine, 11
COM API
  programming instructions, 8
command-line interface
  Conversion Agent Engine, 3
compilation
  C and C++ API, 8
Conversion Agent Server, 17
Conversion Agent Server
  invoking, 18
custom code
  running in Conversion Agent, 28
custom components
  external, 20

D

days to keep history, 15
document processor
  in command-line interface, 4
document processors
  custom, 20

E

error messages
  command-line interface, 6
event log
  days to keep, 15
  Engine initialization, 16
  for multiple users, 15
event logs
  configuration, 15
  Conversion Agent Engine, 14
  when generated, 14
external components, 20
  C/C++, 23
  example, 20
  Java, 21
  properties of, 21

F

firewall
  Conversion Agent Server connections, 18

H

header files
  C and C++ API, 8
HTTP interface
  Conversion Agent Engine, 11

I

include files
  C and C++ API, 8
initialization
  event log, 16
in-process
  running Conversion Agent Engine, 17

J

Java
external components, 21
Java API
programming instructions, 8
using with Conversion Agent Server, 18

L

linking
C and C++ API, 8
logs
event, 14

N

NET
.NET API programming instructions, 8

O

out-of-process
running Conversion Agent Engine, 17

P

processors
custom, 20
properties
of external components, 21

Q

query string
CGI interface, 12

R

remote support interface, 16

S

Server
Conversion Agent, 17
service parameters
passing via API, 7
shell files
running Conversion Agent in, 3

T

TGP file
for external components, 25
threads
Conversion Agent Server, 18
transformers
custom, 20

V

variables
initializing in API, 7

W

web interface
Conversion Agent Engine, 11