



Informatica® Application Integration  
April 2024

# File Connector Guide

Informatica Application Integration File Connector Guide  
April 2024

© Copyright Informatica LLC 1993, 2024

Publication Date: 2024-10-15

# Table of Contents

<b>Preface</b> .....	<b>4</b>
<b>Chapter 1: Introduction to File Connector</b> .....	<b>5</b>
File Connector Overview. ....	5
File Connector Capabilities. ....	5
File Connector Implementation. ....	6
Repository Files. ....	6
<b>Chapter 2: File Connections</b> .....	<b>8</b>
Basic Connection Properties. ....	8
<b>Chapter 3: File Event Sources</b> .....	<b>9</b>
Basic Event Source Properties. ....	9
File Location Properties. ....	10
File Operations and Polling Properties. ....	10
File Read Lock Properties. ....	12
File Parsing and Content Type Properties. ....	12
Fixed Width Files. ....	16
<b>Chapter 4: File Event Targets</b> .....	<b>17</b>
File Connector Event Targets. ....	17
Basic Event Target Properties. ....	17
Event Target File Properties. ....	18
Delimited Content Writer Properties. ....	19
<b>Chapter 5: Process Objects with File Connector</b> .....	<b>21</b>
Event Process Objects Overview. ....	21
File Connector Process Objects Overview. ....	22
Built-In Process Object Output. ....	22
Custom Object Fields Output. ....	23
File Information Process Objects. ....	24
Event Target Process Objects. ....	24
Delimited Content. ....	25
<b>Chapter 6: Process Definitions with File Connector</b> .....	<b>28</b>
Process Design Considerations. ....	28
Example Process with File Connector. ....	28
<b>Index</b> .....	<b>30</b>

# Preface

Read the *File Connector Guide* to learn how to set up and use file connections.

This guide assumes that you have an understanding of the file systems in your environment and how to create connections and processes in Application Integration.

# CHAPTER 1

## Introduction to File Connector

This chapter includes the following topics:

- [File Connector Overview, 5](#)
- [File Connector Implementation, 6](#)
- [Repository Files, 6](#)

### File Connector Overview

The File Connector provides connectivity between Application Integration and file systems so you can monitor file systems for new files, move files, read and write content from files, and select options to handle processed files. You might use the File Connector, for example, to write log files needed as part of a larger integration, or to monitor a file system for new .csv files, read the delimited content from the files, and use the generated XML in a set of process objects.

When you create event sources and event targets with the File Connector, the Process Designer also generates a set of process objects you can access to obtain file information, such as the number of records or file size.

The File Connector runs on an Informatica Secure Agent.

This guide assumes you are familiar with Process Designer, process objects, and the options available for any file systems you want to work with.

### File Connector Capabilities

The File Connector allow you to read and write file content, including text, delimited file content, XML and JSON. Using one of the configurable event sources or event targets, you can:

- Monitor a file folder and process files when they are added to a folder.
- Perform file operations like move or delete.
- Parse the contents of delimited files to create process objects and make the contents available for use in Process Designer.
- Create event services (event targets) that can be called from Process Designer.
- Serialize a set of process objects in a delimited format and save to a file.
- Serialize process objects to XML/JSON format.
- Write and read text content to a file.

- Write and read binary content to and from files using a binary or attachment format.
- Parse JSON and XML files and convert them to a set of process objects.

For more information about setting up a delimited file reader and file writer in Application Integration, see the following community article:

[https://knowledge.informatica.com/s/article/HOW-TO-Setup-a-Delimited-File-Reader-Writer-in-Cloud-Application-Integration?language=en\\_US](https://knowledge.informatica.com/s/article/HOW-TO-Setup-a-Delimited-File-Reader-Writer-in-Cloud-Application-Integration?language=en_US)

## File Connector Implementation

With the File Connector, you can define a single File connection and include one or more Event Sources or Event Targets. These event sources and event targets generate process objects you can then consume or invoke in Process Designer.

The following Event Source types are available:

- File Monitor. Monitors a directory for new files.
- File Parser. Monitors a directory for new files and parses the file contents of each processed file.
- Delimited Content Parser. Monitors a directory for new files and parses the delimited file contents of each processed file.

The following Event Target types are available:

- File Writer. Writes files to a target directory.
- Delimited Content Writer. Writes delimited files to a target directory.

Because each Event Source and Event Target can have unique properties, one File Connector can monitor several different folders for new files, read delimited content from several other folders, and write to another folder.

You can combine a set of related tasks in one connection or split your work into several separate connections.

## Repository Files

The File Connector uses a file-based cache to hold a list of processed entries.

When needed, it also stores idempotent repository files in a special folder called ".aedata". These "idempotent" files are safe files that record data about the files previously processed. The connector creates these files only when:

1. You enable the No File Operation option for the connection.
2. You do not delete processed files or move them to another location.

**Note:** If you move files after processing, be sure they are not in the monitored file path as they are processed. You can also move them to a folder name that starts with a period (for example, ".done" or ".error"). This will ensure that the connector skips the moved files during subsequent folder scans.

If you restart your server or republish the connection, the in-memory copy of the repository list is no longer available and the connector loads the list from the file.

The repository file name has the following format:

```
.aedata/tenant_connectionName_eventSourceName.Aerepo
```

This file stores the relative path, file size and modification timestamp to detect changes to the file.

Each time the connector loads a repository file, it cleans the repository to reduce repository size and removes entries for files that no longer exist. You can copy previously removed files to a monitored folder if you want to process them again.

## CHAPTER 2

# File Connections

This chapter includes the following topic:

- [Basic Connection Properties, 8](#)

## Basic Connection Properties

The following table describes the basic properties that you can configure on the **Properties** tab of the connection creation page:

Properties	Description
Name	Required. Unique name for the File connection that identifies it in the Process Designer. The name must start with an alphabet and can contain only alphabets, numbers, or hyphens (-).
Location	Optional. The location of the project or folder where you want to save the connection. Click <b>Browse</b> to select a location. If the <b>Explore</b> page is currently active and a project or folder is selected, the default location for the connection is the selected project or folder. Otherwise, the default location is the location of the most recently saved asset.
Description	Optional. Description of the connection.
Type	Required. The type of connection that you want to use for the connector or service connector. Select <b>File</b> .
Run on	Required. The name of the Secure Agent group or the Secure Agent machine where the connection must run.
Connection Test	Not supported for File Connector.
OData-Enabled	Not supported for File Connector.

After you configure the basic properties, you must also define the following properties:

- The properties applicable to the File connection type
- The event source properties and event target properties for the File connection

The **Metadata** tab displays the process objects generated when you publish the File connection.



# CHAPTER 3

## File Event Sources

This chapter includes the following topics:

- [Basic Event Source Properties, 9](#)
- [File Location Properties, 10](#)
- [File Operations and Polling Properties, 10](#)
- [File Read Lock Properties, 12](#)
- [File Parsing and Content Type Properties, 12](#)
- [Fixed Width Files, 16](#)

### Basic Event Source Properties

You can add one or more event sources for a connection. An event source serves as a start event that listens or monitors a specified location for new files or messages. After you define an event source for a connection, you can publish the connection only on a Secure Agent and not on the Cloud Server. You can then access the event source in a process and deploy the process only on a Secure Agent to consume the process objects generated by the event source downstream.

To create event sources for a connection, from the **Event Sources** tab, click **Add Event Source**, and select the event source type from the available list.

The following table describes the event source properties that are available for all event source types:

Property	Description
Name	Required. Unique name for the event source.
Description	Optional. Description for the event source.
Enabled	Select <b>Yes</b> to make the event source available immediately after it is published. Select <b>No</b> to disable the event source until you are ready to use it. Default is <b>Yes</b> .

You can view the status of each event source in the published connection. If the status of the event source is stopped, you can republish the connection and restart the event source. When you republish the connection, all the event sources in the connection start by default.

For more information about starting and stopping event sources in a listener-based connection, see *Connectors for Cloud Application Integration and Monitor*.

# File Location Properties

The following table describes file location properties available for each event source.

**Note:** When you use regular expressions to define filters to include and exclude files, you can use a tool like the one available at <https://regex101.com/> to ensure that the expression syntax is correct and to learn more about Java-style regular expressions.

Property	Description
Directory	Required. Specify the relative path of the directory to monitor. For example: <code>data/docs</code> The question mark (?) is a prohibited character in this field.
Recursive	Determines whether the event source looks for files in all the subdirectories of the specified Directory. Use care with this option. For example, if you move processed files into a subdirectory, those files would be monitored again. This could cause unintended results, including a loop that creates a series of nested directories. <b>Note:</b> The connector ignores any directory name that begins with a period (".") character (for example, ".done"). You can use this as a prefix when you want to skip all files in a particular directory. The best practice is to use this method to store processed files in a subdirectory and avoid issues with recursion.
Include Files	Optional. Enter a regular expression to select files that should be processed. Use a Java-style regular expression. For example, to select only files with the <code>.txt</code> file extension, the regular expression would be similar to: <code>.*\.txt</code> It uses a back slash instead of a forward slash.
Exclude Files	Optional. Enter a regular expression you can enter to select files that should be excluded. Use a Java-style regular expression.

# File Operations and Polling Properties

The following table describes the file operation and polling properties available for each Event Source:

Property	Description
No File Operations	Determines whether to disallow any move or delete file operations, to handle read-only data. Default is <b>No</b> .
Delete Processed Files	Determines whether to delete each file after it is processed successfully. <b>Note:</b> Do not use this option with Move To. Default is <b>No</b> .
Before Move	Optional. Enter a regular expression to dynamically set the new file name and location before processing it.

Property	Description
Move To	<p>Optional. Enter a regular expression to dynamically set the file name and location where you want to move files after they are successfully processed.</p> <p>If you enable the Recursive property, use this option with care to avoid repeatedly processing moved files.</p> <p>Default is .done.</p> <p><b>Note:</b> Do not use this option with Delete Processed Files.</p>
Move if Failure	<p>Optional. Enter a regular expression used to dynamically set a different target directory if the Move To operation fails.</p> <p>Default is .error.</p> <p><b>Note:</b> Do not use this option with Delete Processed Files.</p>
Initial Delay	<p>Seconds before polling begins.</p> <p>Default is 1 second.</p>
Poll Interval	<p>Seconds to wait before the next poll to check whether new files have arrived.</p> <p>Default is 1 second.</p>
Max Messages Per Poll	<p>The maximum number of objects to retrieve each time the location is polled. If you do not want to set an upper limit, enter 0.</p> <p>Default is 0.</p>
Other Attributes	<p>Optional. You can supply a list of other parameters that might be available with this event source type. There is no need to enter the attributes in URI-encoded format because the connector encodes them for you.</p> <p><b>Note:</b> If you add any of the specific attributes that are already exposed in the connector, such as Poll Interval, Process Designer ignores them.</p>

# File Read Lock Properties

The following table describes the file read lock properties:

Property	Description
Read Lock	<p>Required.</p> <p>Select a file read lock mode to avoid processing files that are in use at the same time the event source is accessing the files. The event source waits until the file lock is granted. Options:</p> <ul style="list-style-type: none"><li>- <b>changed</b>: Uses file length and modification timestamp to detect whether the file is currently being copied or not. This option slows processing due to the delay required to detect changes.</li><li>- <b>rename</b>: Tries to rename the file as a test to decide whether an exclusive read-lock is available for the file. Default.</li><li>- <b>none</b>: Do not use read lock.</li></ul> <p>For some connection types (like the File Connector), these read lock options are also available:</p> <ul style="list-style-type: none"><li>- <b>markerFile</b>: Creates a marker file and holds a lock on it.</li><li>- <b>fileLock</b>: Acquires a read lock on the file.</li></ul> <p><i>Not recommended</i> when accessing a remote file system with a mount/share unless the file system supports distributed file locks.</p>
Read Lock Timeout	<p>Enter the number of seconds to wait before skipping the current file, if a read lock cannot be acquired.</p> <p>During the next poll, the event source makes another attempt to process the skipped file. If set to 0, the event source waits as long as required.</p> <p>Default: 10 seconds</p>
Read Lock Check Interval	<p>Enter the number of seconds to wait between attempts to acquire a read lock on a file.</p> <p>Default: 1 second</p> <p><b>Note:</b> Set Read Lock Timeout to at least 3 times the value of this property to ensure that the event source can complete a file lock attempt on each file.</p>
Read Lock Min Length	<p>Enter the minimum file size, in bytes, that should be processed by the event source.</p> <p><b>Note:</b> Use this property only when <b>Read Lock</b> is set to <b>changed</b>.</p> <p>Default: 1 byte</p>
Read Lock Logging Level	<p>Required.</p> <p>Select the logging level to use when a read lock cannot be acquired: OFF, INFO, WARN, or ERROR.</p> <p>Default: WARN</p>

# File Parsing and Content Type Properties

You can configure an event source based on the specific event source types available for your connection type.

In the Event Source properties, you determine how the content is handled as it is parsed. After the event source is published, the connection waits for a new file, parses the contents on arrival, and represents the contents according to the properties for:

- File Parsing
- Content Type

The following table describes the file parsing properties:

Property	Applies To	Description
Character Set	All Event Source Types	Required. Determines the encoding of the file. If none specified, the connector uses UTF-8.
Delimiter	Delimited Content Parser	Required. The delimiter character for delimited files. To specify a space or a tab character as a delimiter, use the escaped character, "\s" or "\t".
Text Qualifier	Delimited Content Parser	Required. The text qualifier for delimited files. To specify a space or a tab character as a qualifier, use the escaped character, "\s" or "\t".
Ignore First Record	Delimited Content Parser	Determines whether the connector processes the first row of a delimited file as a data row. You can select from the following options: - <b>Yes:</b> The connector does not process the first row of a delimited file as a data row. - <b>No:</b> The connector processes the first row of a delimited file as a data row. If you select <b>No</b> , you must provide custom headers by using the Columns Descriptor attribute. The default value is <b>Yes</b> .
Split Rows	Delimited Content Parser Fixed Width Content Parser	Determines whether to process each row separately, convert each row to a process object, and generate a separate event for each of them.

Property	Applies To	Description
Columns Descriptor	Delimited Content Parser Fixed Width Content Parser	<p>Defines the fixed width column headers to be processed. Specify the values in a comma-separated list of names and sizes (in parentheses).</p> <p>Delimited Content Parser example: User Name, Password, Email</p> <p>Fixed Width Content Parser example: User Name(40), Password(8), Email(14)</p> <p>These column headers also determine the process object header names when you use Custom Objects, unless you specify a different set of headers (see below).</p>
Ignore Column Inconsistencies	Delimited Content Parser Fixed Width Content Parser	<p>For Delimited Content Parser, determines:</p> <ul style="list-style-type: none"> <li>- Whether to ignore extra columns in data rows.</li> <li>- Whether to add columns with empty string values if columns are missing.</li> </ul> <p>If not enabled, the event source throws an exception when it encounters extra or missing columns.</p> <p>For Fixed Width Content Parser, determines:</p> <ul style="list-style-type: none"> <li>- Whether to ignore extra characters in rows.</li> <li>- Whether to add characters with empty spaces if defined columns have missing values.</li> </ul> <p>Default: No.</p>

The following tables describe all the content type properties:

Property	Applies To	Description
Content Format	File Parser File Monitor FTP Monitor	<p>Required. Format of the content to be processed:</p> <ul style="list-style-type: none"> <li>- <b>Ignore</b>. Do not process the content.</li> <li>- <b>Plain Text</b>. Content is a string.</li> <li>- <b>Binary</b>. Content should be converted to a Base64-encoded string.</li> <li>- <b>XML and JSON</b>. Content should be parsed and converted to an object or a list of process objects.</li> <li>- <b>Attachment</b>. Content is an attachment.</li> </ul>
Simplify Content	File Parser File Monitor FTP Monitor	<p>If enabled, the event source parses XML/JSON content that does not match a valid process object structure and attempts to modify the content structure to match the format of the process objects.</p> <p>Default: No</p>

Property	Applies To	Description
Single Object Mode	File Parser File Monitor FTP Monitor	If enabled, XML content is converted to a single object. Default: No
Use Built-in Process Objects	Delimited Content Parser Fixed Width Content Parser	Select <b>Yes</b> , for example, if you work with files that use a different set of fields or you do not know the file headers in advance. The File Connector represents the records as a set of process objects based on the file contents.  Select <b>No</b> if you provide a list of field names in Custom Object. This is suitable if you work with similar files and you can reliably anticipate a set of fields (headers) in each one. The event source generates a simple process object for each record of the delimited content file. Default is No.  For details about working with process objects, see <a href="#">"Event Process Objects Overview" on page 21</a>
Custom Object	Delimited Content Parser Fixed Width Content Parser	If you do not select Use Built-in Process Objects, enter a comma-delimited list of process object header names that represent the file contents. This method enables you to extract only the required data. Generated objects are simpler to work with and require less code to handle in a process.  The names must be an exact match with the header names in the file or the headers defined with the Columns Descriptor. For example:  <code>Name, Street, City, State, Postal Code, Country, Phone</code>  <b>Note:</b> You cannot use "index" as a field name. It is reserved for row index information.  When the file is parsed, if a header you enter here does not exist, the related field is empty in the generated objects.  The custom objects are represented in <i>NCName</i> format, to remove any prohibited characters from the header names and ensure they are valid process object field names.

### Escape Characters in Custom Objects

If you need to specify a comma character in a column header, add a backslash (\) as an escape character so the file is parsed correctly. For example, if the file contains:

```
First, Name, Last Name, Address, local, Phone
```

In this case, the backslash ensures that the comma is not used as a delimiter character when the content is parsed:

```
First\, Name, Last Name, Address\, local, Phone
```

## Fixed Width Files

The Fixed Width Parser allows you to handle fixed width files that have no column headers or delimiter characters. In the event source properties, you define column descriptors that enable you to generate process objects that represent the fixed width content.

For example, you might define the structure of your data files in the Columns Descriptor property, you might enter a list of column names and column lengths (in parentheses):

```
ID(3), User Name(100), Password(20), Nick Name(25), Email(20)
```

This enables the Fixed Width Parser to read the fixed width file and generate a process object that structures the records based on these values. This is similar to the Delimited Content Parser, with the addition of column widths. These values also define the default set of field names if you use custom process objects.

Both the built-in and custom process objects are available for this event source type.

Because the connector verifies data consistency, you might encounter an exception error if some rows in the fixed width file are longer or shorter than the specified column descriptors. To ignore extra characters and add trailing spaces to rows with fewer characters than expected, you can enable the Ignore Column Inconsistencies property.



# CHAPTER 4

## File Event Targets

This chapter includes the following topics:

- [File Connector Event Targets, 17](#)
- [Basic Event Target Properties, 17](#)
- [Event Target File Properties, 18](#)
- [Delimited Content Writer Properties, 19](#)

### File Connector Event Targets

The File Connector includes the following event target types:

- File Writer. Write data in plain text, binary, attachment, JSON, or XML format to a file in the specified directory.
- Delimited Content Writer. Serialize process objects to delimited content format and write the result to a file in the specified directory.

### Basic Event Target Properties

For each connection that you define, you can include one or more event targets that specify operations for writing files or messages, or when the event target is called from a process. For example, you might define an event target that reads from a process object and writes to comma-delimited files.

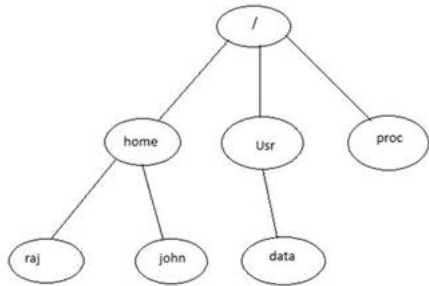
To set event target properties for the connection, from the **Event Targets** tab, click **Add Event Target**, and select the event target type from the available list.

The following table describes the basic properties:

Property	Description
Name	Required. Unique name for the event target.
Description	Optional. Description of the event target.

# Event Target File Properties

The following table describes the file location properties:

Property	Description
Directory	<p>Required. Specify the relative path of the directory where you want to store files. For example: <code>data/docs</code></p> <p>The question mark (?) is a prohibited character in this field.</p> <p>If you use an FTP or SFTP connection, always specify a path that is relative to your FTP or SFTP home directory on the FTP or SFTP server. You can find out the home directory by accessing the FTP or SFTP endpoint through a GUI client. The home directory is the first directory that you are connected to when you access an FTP or SFTP endpoint.</p> <p>For example, consider the following directory structure where <code>raj</code> is the home directory:</p>  <pre> graph TD     root("/") --- home("home")     root --- usr("usr")     root --- proc("proc")     home --- raj("raj")     home --- john("john")     usr --- data("data")     </pre> <p>If you want to point to the <code>data</code> directory, you must specify the relative path in the <b>Directory</b> field as follows: <code>../../usr/data</code></p> <p>FTP Connector navigates two directories back from the <code>raj</code> directory to the root directory and then navigates to the <code>usr</code> directory followed by the <code>data</code> directory.</p> <p><b>Note:</b> Processes can create only nested folders.</p>
File Exists	<p>Required.</p> <p>Determines what to do if a file already exists with the same name:</p> <ul style="list-style-type: none"> <li>- <b>Override</b> (the default) to replace the existing file. Also select an Eager Delete option to determine how to handle existing files when the targets are generated.</li> <li>- <b>Append</b> to add content to the existing file. <i>Do not</i> specify a Temp Prefix or Temp File Name.</li> <li>- <b>Fail</b> to skip the conflicting file and throw an exception to indicate that the file name already exists.</li> <li>- <b>Ignore</b> to skip the conflicting file and silently ignore the problem (no exception thrown).</li> <li>- <b>Move</b> to move any existing files before writing the target file. Use with Move Existing to specify the folder where you want to move existing files.</li> <li>- <b>Try Rename</b> to rename the file from the temporary name to the actual name without checking to see if the file name exists. Use only if you also specify Temp File Name. This might be faster on some file systems or FTP servers.</li> </ul>

Property	Description
Move Existing	<p>Use with the File Exists "Move" option to specify a file location for existing files when writing the target. Simply enter "backup" in this field to move existing files to the backup folder. To rename the backup files as they are moved, so subsequent operations do not replace backup files with a newer version, you can enter an expression to determine the file name. For example:</p> <pre>backup\\${file:name}_\${file:modified}</pre> <p>This expression creates a new file in the backup folder every time existing files are moved and appends the timestamp to the file name.</p> <p>For more information on using the Apache Camel File Expression language, see: <a href="http://camel.apache.org/file-language.html">http://camel.apache.org/file-language.html</a></p>
Eager Delete	<p>Select <b>Yes</b> to delete the target file before the temp file is written. In that case, you must also select Override as the File Exists option and specify a Temp File Name (see below).</p> <p>Select <b>No</b> to wait and delete the target file only when the temp file is ready to be written and renamed to the output file name. You might want to use this if you want to ensure that the existing file is available during the time interval it takes for the write operation to complete.</p>

The following table describes the file writing properties:

Property	Description
Temp Prefix	<p>Enter a prefix for the file name if you want to write the file to a temporary name and, after the write operation completes, rename it to the original name.</p> <p><b>Note:</b> Ignored if you select Append as the File Exists option.</p>
Temp File Name	<p>Enter an expression to determine the file name for temporary files, instead of a prefix. You can take advantage of the Apache Camel File Expression language.</p> <p><b>Note:</b> Ignored if you select Append as the File Exists option.</p>
Force Writes	<p>Select <b>Yes</b> to force the file system to write all the data to a target file, to make sure that in the event of a system failure, all data is retained.</p> <p>Select <b>No</b> if you work, for example, with log data and are not concerned with loss of a data fragment in the event of a file system failure. This might yield a small performance improvement.</p>
Write Buffer Size	<p>Determines the size of the write buffer (in bytes).</p> <p>Default: 128kb.</p>
Character Set	<p>Required. Determines the encoding of the file</p> <p>If not specified, the Event Target uses UTF-8 file encoding.</p>

## Delimited Content Writer Properties

If you use the Delimited Content Writer, you can serialize existing process objects to delimited file format. You can also serialize process objects created by a delimited event source.

The following table describes properties for the Delimited Content Writer:

Property	Description
Delimiter	Required. The delimiter character to use in files generated by this Event Target. <b>Note:</b> To use a space or a tab, enter it as an escaped character ("\" or "\t"). Default is ",".
Text Qualifier	Required. A Text Qualifier is added to the Delimited Content only when a cell in a CSV file contains special characters, such as the delimiter char, a quote char, or spans multiple lines. Default is "".
Skip Headers	Required. Determines whether the generated content has header names. Select <b>Yes</b> to store the delimited content without header names. Default is <b>No</b> .
Line Ending Style	Required. Determines the line ending style in the generated content (Windows style is \r\n. Unix style is \n.) Default is <b>Windows</b> .

## CHAPTER 5

# Process Objects with File Connector

This chapter includes the following topics:

- [Event Process Objects Overview, 21](#)
- [File Connector Process Objects Overview, 22](#)
- [Built-In Process Object Output, 22](#)
- [Custom Object Fields Output, 23](#)
- [File Information Process Objects, 24](#)
- [Event Target Process Objects, 24](#)

## Event Process Objects Overview

A process object is a set of structured data that enables you to handle data sent to or returned from a service.

Process Designer generates process objects based on the event sources and event targets defined for each connection. These process objects are available in processes that access the connection.

The Event Source process objects act as start events that monitor a queue or file system and trigger file operations. After you publish an event source, the connector begins to monitor the specified directory. When the connector detects a new file, it parses the contents and converts them to a process object, then sends the process object to the process(es) listening for these events.

**Note:** By default, a connection begins to generate events as soon as you publish the connection, even if no published processes have started to monitor the directory. You can use the Initial Delay setting to postpone the start time for monitoring.

The Event Target process objects act as event services you can use to generate content in a specific format. After you publish an event target, you can invoke it from a process in Process Designer. An event target is sometimes called an event service.

# File Connector Process Objects Overview

The following process objects are used with the File Connector:

- DelimitedContent. Generated from a parsed file when you use built-in process objects. In this case, Field, Header, and Record process objects are also created.
- <EventSource>Content (a collection of records and related information) and <EventSource>Record (a single row). Generated from a parsed file when you use custom objects.
- FileInformation. File metadata generated for each file read by an event source.
- FileWriteTask. Represents a request to a File Writer (Event Target) service.
- SerializeToDelimitedContentTask. Represents a request to a Delimited Content Writer (Event Target) service.
- SerializeToDelimitedContentResult. Returned as a response from a Delimited Content Writer service.

When you define a process that uses the File connection, you can work with these process objects.

In addition, these process objects represent the content of files handled by the File Connector:

- PlainFileContent. Content of a plain text or binary file.
- ParsedFileContent. Content of a file represented as XML or JSON in a list of process objects (or a single object).
- AttachmentFileContent. Content of a file provided as an attachment.

## Built-In Process Object Output

**Note:** Applicable only to Delimited Content Parser and Fixed Width Content Parser

When you choose the built-in process objects with delimited content and fixed width files, the output is represented in a DelimitedContent process object. For example, if you process a simple .csv file with several records, you might read a file similar to this:

```
User Name,Age,Born
Bob Smith,22,1987
Bill White,45,1999
```

Based on this .csv file, the file contents are represented with the header name and value for each field. The process object includes file information about the source file, list of header objects, list of records and total row count.

**Note:** If you work in spilt rows mode, the file is divided into separate rows and for each row Process Designer produces a DelimitedContent process objects with headers and one record.

```
<DelimitedContent>
  <fileInfo>
    <!-- for FTP/SFTP only -->
    <host>127.0.0.1</host>

    <lastModified>2015-04-29T14:37:35.448Z</lastModified>
    <dir>DataFiles</dir>
    <name>users</name>
    <path>/DataFiles/users.csv</path>
    <fullName>users.csv</fullName>
    <ext>csv</ext>
    <size>78</size>
  </fileInfo>
  <header>
```

```

        <name>User Name</name>
        <fieldIndex>1</fieldIndex>
    </header>
    <header>
        <name>Age</name>
        <fieldIndex>2</fieldIndex>
    </header>
    <header>
        <name>Born</name>
        <fieldIndex>3</fieldIndex>
    </header>
    <record>
        <field><value>Bob Smith</value></field>
        <field><value>22</value></field>
        <field><value>1987</value></field>
        <index>1</index>
    </record>
    <record>
        <field><value>Bill White</value></field>
        <field><value>45</value></field>
        <field><value>1999</value></field>
        <index>2</index>
    </record>
    <totalRowsCount>3</totalRowsCount>
</DelimitedContent>

```

## Custom Object Fields Output

**Note:** Applicable only to Delimited Content Parser and Fixed Width Content Parser

If you specify custom object fields:

- The output includes an empty element for any headers not found in the parsed file. To skip some fields, simply *omit* them from the list of Custom Object Fields in the Event Source properties.
- The output still includes file information.
- The process object name is derived as <sourceName>Content, to represent a collection of record objects and other details.
- When you have several delimited content or fixed width event sources, each of them uses its own custom content and record objects. The record object uses the format <sourceName>Record.
- Field names are converted to NCName format, to remove any prohibited characters from the header names and ensure they are valid process object field names.

For example, to process this .csv file, you might specify custom object fields with the header names "Age, Born, User Name, Salary":

```

Age,Born,User Name
22,1987,Bob Smith
45,1999,Bill White

```

In this case, the results appear similar to the following example. The <Salary> element is empty because it was specified as a custom object field but the source file did not contain the "Salary" header:

```

<MyUserFileContent>
  <fileInfo>
    <!-- for FTP/SFTP only -->
    <host>127.0.0.1</host>

    <lastModified> 2015-04-29T14:37:35.448Z </lastModified>
    <dir>DataFiles</dir>
    <name> users </name>
    <path>/DataFiles/users.csv </path>

```

```

        <fullName> users.csv </fullName>
        <ext> csv </ext>
        <size> 78 </size>
    </fileInfo>
    <record>
        <Age> 22 </Age>
        <Born> 1987 </Born>
        <Salary/>
        <User_Name> Bob Smith </User_Name>
    </record>
    <record>
        <Age> 45 </Age>
        <Born> 1999 </Born>
        <Salary/>
        <User_Name> Bill White </User_Name>
    </record>
    <totalRowCount>3</totalRowCount>
</MyUserFileContent>

```

## File Information Process Objects

The FileInformation process object for each event source contains information about the file but not the contents.

**Note:** The file path strings always use a forward slash as the path delimiter. The timestamp is always in UTC format.

For example:

```

<FileInformation>
  <!-- for FTP/SFTP only -->
  <host>127.0.0.1</host>

  <!-- dateTime : file modified time -->
  <lastModified> 2015-04-29T14:37:35.448Z</lastModified>
  <!-- string: full path and name of resource. File separator is normalized to
  forward slash. -->
  <path>path/documents/users.csv</path>
  <!-- string: path to parent directory -->
  <dir>path/documents/</dir>
  <!-- string: file name with extension -->
  <fullName> users.csv</fullName>
  <!-- string: separate name only and extension only -->
  <name> users</name>
  <ext> csv</ext>
  <!-- double: file size in bytes -->
  <size> 78</size>
</FileInformation>

```

## Event Target Process Objects

Event Target definitions enable you to work with process objects to write files and serialize delimited content.

### File Writer

When you publish a connection that includes a file writer definition as an Event Target, Process Designer also creates a service.



The FileWriteTask process object contains the name of the target file, the relative file path (based on the event target's base directory) and the string content to write to the target.

```
<FileWriteTask>
  <!-- file path, if required, which must be relative for FTP/SFTP -->
  <filePath> documents </filePath>
  <!-- target file name -->
  <fileName> test.txt</fileName>

  <!-- content format, which determines the applicability of other fields -->
  <format>PlainText|Binary|Attachment|JSON|XML</format>

  <!-- applies only if format is PlainText or Binary - if Binary, content is Base64-
  encoded string -->
  <content>Test file content</content>

  <!-- applies if content is XML or JSON, in which case, use object or objects -->
  <object>process object</object>
  <objects>a list of process objects</objects>

  <!-- optional, provide the ObjectName and listName -->
  <objectName>order</objectName>
  <listName>orders</listName>
</FileWriteTask>
```

**Note:** If the file or any folders in the path do not exist, they are automatically created.

For details on the FileInformation process object, see ["File Information Process Objects" on page 24](#).

## Delimited Content

When you publish a connection that contains a delimited content event target, Process Designer creates a service that you can use to serialize your process objects into a delimited file.

You can serialize both delimited content process objects, which represent the general model of a delimited content file, and custom process objects.

### Serialize Custom Process Objects

The process objects created to handle delimited files in an Event Target are:

1. Use `SerializeToDelimitedContentTask` to access the request object.
2. Use `SerializeToDelimitedContentResult` to access the results of the serialization.

For example:

```
<SerializeToDelimitedContentTask>
  <!-- for FTP/SFTP only -->
  <host>127.0.0.1</host>

  <fileName> users2.csv </fileName>
  <filePath> CustomModelProcess </filePath>
  <delimiter> ; </delimiter>
  <skipHeaders> true </skipHeaders>
  <customObjects>
    <Email> bob@test.com </Email>
    <Password> 22222 </Password>
    <Phone_number> 333-3333-5554 </Phone_number>
    <User_name> Bob </User_name>
  </customObjects>
  <customObjects>
    <Email> bill@test.com </Email>
    <Password> 3333 </Password>
    <Phone_number> 444-222-111 </Phone_number>
```

```

        <User_name> Bill </User_name> </customObjects>
    <header>
        <name> User_name </name>
        <fieldIndex> 1 </fieldIndex>
    </header>
    <header>
        <name> Phone_number </name>
        <fieldIndex> 2 </fieldIndex>
    </header>
    <header>
        <name> Password </name>
        <fieldIndex> 3 </fieldIndex>
    </header>
</SerializeToDelimitedContentTask>

```

The request should include the target file name, relative file path, and a list of custom objects.

**Note:** You can also provide delimited and text qualifier characters if you need to overwrite default values that are set in the event target's properties. For example, you might want to overwrite the skipHeaders attribute if you generate process objects one by one. In that case, you can write headers in the first record and then skip headers as you append all other records in the file.

## Handling Headers

Process Designer can automatically serialize simple objects but skips complex fields (such as references and object lists) when using the custom process objects. The generated file includes a list of headers using the first process object's simple field names. Sometimes this method is useful but it has several disadvantages:

- The order of fields in the resulting file is not defined because the process object's fields do not rely on any specific order.
- If the first object does not contain optional fields, these fields are ignored even if they are provided in all other objects.
- You cannot skip unnecessary fields.

You can eliminate these disadvantages if you include a set of custom headers in a request object. Process Designer then uses the headers to generate delimited records with only the specified fields in the specified order.

**Note:** Use custom headers only if you serialize custom process objects. If you serialize a built-in delimited content process object, it already contains information about headers so there is no need to use custom headers.

## Serialization Results

Whatever approach you take, after each file is processed, the results display:

- Two counters with the number of processed records and number of records that have been successfully written to the target file.
- The modification date, path, filename, extension, and file size of the generated file.
- Status of the operation.
- An optional message string.

For example:

```

<SerializeToDelimitedContentResult>
    <message/>
    <processedRecordsCount> 10 </processedRecordsCount>
    <writtenRecordsCount> 8 </writtenRecordsCount>
    <success> true </success>
    <fileInfo> <lastModified> 2015-05-15T14:07:11.475Z </lastModified>
        <dir> D:/MyDirectory/DelimitedFiles/CustomModelProcess </dir>

```

```
<name> users2 </ns7:name>
<path> D:/MyDirectory/DelimitedFiles/CustomModelProcess/users2.csv </path>
<fullName> users2.csv </fullName>
<ext> csv </ext>
<size> 116 </size>
</fileInfo>
</SerializeToDelimitedContentResult>
```

## CHAPTER 6

# Process Definitions with File Connector

This chapter includes the following topics:

- [Process Design Considerations, 28](#)
- [Example Process with File Connector, 28](#)

## Process Design Considerations

When you design a process with this connector, note that:

- The input field that holds the file content is defined in the connection properties.
- The process must run on a Secure Agent.
- After you save and publish the process, you also need to enable each event source or event target you plan to use in the connection properties so it is available for the process to consume.

Other considerations depend on the type of application integration you are handling. Refer to the example process outlined below to get started.

## Example Process with File Connector

The following example illustrates one way you can use the File Connector to read delimited content, convert it to a list of process objects, call a service to make a list of phone numbers and publish the record count to the client.

**First**, configure the Connection:

1. Define the basic connection properties and select File as the Connection Type. Be sure to select an agent where the connection will run.
2. On the Event Sources tab, add a Delimited Content Writer event.
3. Enter a name and be sure to enable the event source if you want the connection to be available as soon as you publish it. If you do not want the connection to be immediately available, you can disable it.
4. Specify the directory that should be monitored. This field is required. You can specify other optional properties to exclude or include specific files or the subdirectories.

5. Enter other file parsing options as needed.
6. Select No for Use Built-In Process Objects and, for Custom Object, enter a comma-delimited list of header names that are exactly the same as the names of the headers in the delimited file header you are reading. For example:

```
Name,Street,City,State,PostalCode,Country,Phone
```

7. Select any optional File Read Lock Settings you want to use. The Other Attributes field allows you to specify other parameters supported by the Apache Camel File Listener but not exposed in the File Connector.
8. Save, test, and publish the process.

**Second**, create the Process:

1. Create a process and be sure it runs on the same Secure Agent defined in the connection.
2. On the Start tab, select the connection, DelimitedContent, to automatically make the input field available that will hold the file content when the connection detects a new file in the monitored directory.
3. Create two temporary variables in the process properties so you can iterate through the delimited content in the file:
  - a. One to hold the full set of records. For example, TempIterator.
  - b. One to hold each object as you iterate through the file. For example, TempCurrent.
4. Create an Assignment step to assign the file records (process objects list) to the TempIterator field.
5. Create an Assignment step to populate the current record from the list into a temporary variable, TempCurrent. Specify Formula as the source of TempCurrent, and use the following to get the first record:
 

```
list:head($temp.TempIterator)
```
6. Add a Decision step to validate that the record is set.
7. In the Is set branch, perform the functions you need to process the data. This example makes a service call to RequestBin, passes in each record read from the file, and the service generates a list of phone numbers.
8. Add an Assignment step to remove the processed record from the list of process objects. Specify Formula as the source of TempIterator, and use the following:
 

```
list:remove($temp.TempIterator,1)
```
9. Add a Jump step to loop back to the Assignment step that populates TempCurrent and get the next record from the process object.
10. In the Unset branch, this example calls a service to publish the record count to the client, reading the FileInfo from the process object.
11. End the process.

# INDEX

## B

Basic connection properties

File [8](#)

## F

File

Basic Connection properties [8](#)