## Table of Contents

**Preface** ........................................................................................................... 5

**Chapter 1: Introduction to FTP Connector....................................................... 6**
- FTP Connector Overview. .................................................................................. 6
- FTP Connector Capabilities. ............................................................................. 6
- FTP Connector Implementation. ....................................................................... 7
- Repository Files. ............................................................................................... 7

**Chapter 2: FTP Connections............................................................................. 9**
- Basic Connection Properties. .......................................................................... 9
- FTP Connection Properties. ............................................................................ 10

**Chapter 3: FTP Event Sources...................................................................... 14**
- FTP Connector Event Sources. ....................................................................... 14
- Basic Event Source Properties. ....................................................................... 14
- File Location Properties. ................................................................................ 15
- File Operations and Polling Properties. ............................................................ 16
- File Read Lock Properties. .............................................................................. 17
- File Parsing and Content Type Properties. ...................................................... 17
- Fixed Width Files. ........................................................................................... 21
- Event Source Endpoints. .................................................................................. 21

**Chapter 4: FTP Event Targets..................................................................... 22**
- FTP Connector Event Targets. ....................................................................... 22
- Basic Event Target Properties. ....................................................................... 22
- Event Target File Properties. ......................................................................... 23
- Delimited Content Writer Properties. ............................................................... 24

**Chapter 5: Process Objects with FTP Connector......................................... 25**
- Event Process Objects Overview. ................................................................. 25
- FTP Connector Process Objects. ................................................................. 26
- FTP Monitor Process Objects. ....................................................................... 26
- Built-In Process Object Output. ................................................................. 27
- Custom Object Fields Output. ....................................................................... 28
- File Information Process Objects. ................................................................. 29
- Event Target Process Objects. ................................................................. 29
- Delimited Content Objects. .......................................................................... 30

**Chapter 6: Process Design with FTP Connector......................................... 33**
- Process Design Considerations. ................................................................. 33
The FTP Connector Guide for Application Integration contains information about how to set up and use FTP connections. It assumes you have an understanding of the FTP requirements for your environment, and an understanding of how to create connections using Application Integration.
CHAPTER 1

Introduction to FTP Connector

This chapter includes the following topics:

- FTP Connector Overview, 6
- FTP Connector Implementation, 7
- Repository Files, 7

FTP Connector Overview

The FTP Connector provides connectivity between Application Integration and remote FTP servers. With the connector, you can monitor an FTP server for new files, move files, read and write content from files, and select options to handle processed files.

FTP is a common protocol for file transfer that enables access to a remote FTP server. SFTP, or Secure File Transfer Protocol, encrypts the data transferred to the FTP server. With SFTP, you can access the FTP server's file system on a secure channel.

You specify connection parameters for the FTP or SFTP protocol and other options to receive or stream content through the connection. You might use the FTP 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 FTP 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 FTP Connector runs on an Informatica Secure Agent.

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

FTP Connector Capabilities

The FTP Connector includes these capabilities:

- Monitor a remote FTP server for new files and process files when they are added.
- Perform file operations in a remote folder like move or delete.
- Parse the contents of delimited files to create process objects so the contents are available for use in Process Designer.
- Create event targets that are published as event services you can call from Process Designer.
- Serialize a set of process objects to a delimited format or to XML/JSON format and save to a file.
• Write and read text content to a file.
• Write and read binary content to and from files found in the remote folder using a binary or attachment format.
• Parse JSON and XML files found in the remote folder and convert them to a set of process objects.
• Read files with fixed width columns and no delimiter characters.

FTP Connector Implementation

The FTP Connector includes a single set of connection properties you need to interact with FTP/SFTP servers. This means that you can specify the connection settings once and define multiple Event Sources or Event Targets to perform different FTP monitoring or file writing operations. For example, you can monitor one directory for new files while you also read and write file content in other directories. The event sources and event targets generate process objects you can consume or invoke in Process Designer.

The following authentication types are available when you define an FTP connection:

• Anonymous access.
• Password-based authentication.
• Key-based authentication (for SFTP only).

The following Event Source types are available:

• FTP Monitor. Monitors a remote FTP server location for new files.
• FTP Fixed Width Parser. Monitors a remote FTP location for new files and parses the fixed-width file contents of each file.
• FTP Delimited Content Parser. Monitors a remote FTP location for new files and parses the delimited file contents of each file.

The following Event Target types are available:

• FTP Writer. Writes files to a remote FTP server location.
• FTP Delimited Content Writer. Writes delimited files to a target directory on a remote FTP server.
• FTP File Transferer. Transfers files to a remote FTP server.

Repository Files

The FTP Connector can also use a file-based cache to hold a list of processed entries, in "idempotent" repository files. The idempotent files are safe files that record data about the files previously processed. Based on the list of entries in the repositories, the connection can skip files during subsequent monitoring events.

Note: The connector can also move files after they are processed to a specific directory (the default is ".done"). In that case, you cannot use idempotent repository files.
When you select this option, the event source clears all entries from the repository file during startup, to reduce repository size and remove entries for files that no longer exist. To use these files:

- In the connection properties, enable **Idempotent Repository**. Provide the path to an existing directory where the connection should store the idempotent files. The repository file name has the following format. It stores the relative path, file size and modification timestamp to detect changes to each file:
  
  tenant_connectionName_eventSourceName.Aerepo

- In the event source properties, enable **No File Operation** (disabled by default). When you work in this mode, the options to move and delete files are automatically disabled.

You can also manually clear the idempotent repository.

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.
FTP Connections

This chapter includes the following topics:

- Basic Connection Properties, 9
- FTP Connection Properties, 10

Basic Connection Properties

The following table describes the basic properties available on the Properties tab of the Connection page:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Required. Unique name for this connection that identifies it in the Process Designer.</td>
</tr>
<tr>
<td>Location</td>
<td>The location of the project or folder you want the connection to reside in. To select a location for the connection, browse to the appropriate project or folder, or use the default location.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional. Description of the connection.</td>
</tr>
<tr>
<td>Type</td>
<td>Required. The connector or service connector to use for this connection. Select the type you want to configure.</td>
</tr>
<tr>
<td>Run On</td>
<td>Required. The Cloud Server or the Secure Agent where this connection should run.</td>
</tr>
<tr>
<td>Connection Test</td>
<td>If supported for the connection type, displays the results of the last connection test.</td>
</tr>
<tr>
<td>OData-Enabled</td>
<td>If supported for the connection type, select Yes to enable OData feeds and specify the allowed users and groups.</td>
</tr>
<tr>
<td>Allowed Roles for OData</td>
<td>Optional. The roles that have access to the connection at design time. You can enter a custom role or a system-defined role. You can enter more than one role in this field.</td>
</tr>
</tbody>
</table>

Along with these basic properties, depending on the connector, you also define:

- Properties applicable to the connection type.
- The Event Sources and Event Targets, if available.
- Metadata displays the process objects generated when you publish the connection.
## FTP Connection Properties

The following tables describe the connection properties available for the FTP connector:

### FTP Protocol Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP Protocol</td>
<td>Required. Determines which FTP protocol to use. Select SFTP if you want to enable a secure channel with data encryption. Default: FTP</td>
</tr>
</tbody>
</table>

### FTP Host settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP Host</td>
<td>Required. The FTP host name. For example, enter an IP address as &quot;127.0.0.1.&quot; The question mark (?) is a prohibited character in this field.</td>
</tr>
<tr>
<td>FTP Port</td>
<td>Optional. The FTP server port number. If not specified here, the default port number is 21 (FTP) or 22 (SFTP).</td>
</tr>
<tr>
<td>Known Hosts File (SFTP only)</td>
<td>Optional. The path to a known_hosts file that contains a list of host names and public keys to be verified when the connection occurs. If you supply a file path here and disable Strict Host Key Checking, the FTP Connector adds new hosts to the file.</td>
</tr>
</tbody>
</table>

### Client Authentication Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Name</td>
<td>Optional. The user name to use when the connection logs in to the remote file system. If not provided, the connection attempts an anonymous login. The question mark (?) is a prohibited character in this field.</td>
</tr>
<tr>
<td>Password</td>
<td>Optional. The pass word to use to log in to the remote file system. Leave blank to configure an anonymous login. Default: No</td>
</tr>
<tr>
<td>Private Key File (SFTP only)</td>
<td>Optional. The private key file to use for private key verification. The connector assumes that the corresponding public key is in the file with the same name and the suffix, &quot;.pub&quot;.</td>
</tr>
<tr>
<td>Private Key Passphrase (SFTP only)</td>
<td>Optional. The passphrase for the private key file.</td>
</tr>
</tbody>
</table>
### Proxy Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use HTTP Proxy</td>
<td>Enable this setting if you use an HTTP proxy server. An HTTP proxy server is a dedicated computer or software system that acts as an intermediary between an endpoint device and a server from which a user or client requests a service. Use an HTTP proxy server to create a connection between an FTP client and an FTP server. <strong>Important:</strong> If you enable this setting for FTP, you must set the Use Passive Mode property under the FTP Connection Settings to Yes. Default: No</td>
</tr>
<tr>
<td>Proxy Host</td>
<td>Required. The IP address or host name of the machine that hosts the HTTP proxy server. For example, “192.176.3.761” or “name.mycompany.tech.”</td>
</tr>
<tr>
<td>Proxy Port</td>
<td>Optional. The port number of the HTTP proxy server. For example, “8060”. Default: 80.</td>
</tr>
<tr>
<td>User Name</td>
<td>Optional. The user name to use when the connection logs in to the HTTP proxy server. If you leave this field empty, you configure an anonymous login.</td>
</tr>
<tr>
<td>Password</td>
<td>Optional. The password to use when the connection logs in to the HTTP proxy server. If you leave this field empty, you configure an anonymous login.</td>
</tr>
</tbody>
</table>

### FTP Connection Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Transfer Format</td>
<td>Required. The format for the file transfer, either Binary or ASCII. In most cases, you can use Binary, the most universal transfer format. Default: <strong>Binary</strong> Select ASCII only if your FTP connection has specific limitations that require this option.</td>
</tr>
<tr>
<td>Use Passive Mode (FTP only)</td>
<td>Determines whether the connection uses passive mode for data connections. <strong>Important:</strong> If you set the Use HTTP Proxy setting under Proxy Settings to Yes, you must set the Use Passive Mode property to Yes. Default: No</td>
</tr>
<tr>
<td>Stepwise</td>
<td>Determines whether the connection should traverse directories one directory at a time. Default: Yes</td>
</tr>
<tr>
<td>Fast Exists Check</td>
<td>Determines whether the connection should refer to a list file to check if the file exists, if supported by the FTP server. This option may improve performance if the FTP server has a large number of files. Default: No</td>
</tr>
<tr>
<td>Server Alive Interval (SFTP only)</td>
<td>Required for SFTP. Seconds to wait before sending a server alive message to the FTP server. If set to 0, no messages are sent to the FTP server. Default: 0</td>
</tr>
<tr>
<td>Server Alive Count Max (SFTP only)</td>
<td>Required for SFTP. The maximum number of server alive messages that can be sent. Default: 1</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Connection Timeout</td>
<td>Required. Seconds to wait before connection times out. Default: <strong>10</strong> seconds</td>
</tr>
<tr>
<td>Data Channel Timeout (FTP only)</td>
<td>Required for SFTP. Seconds to wait before the data channel times out. Default: <strong>30</strong>.</td>
</tr>
<tr>
<td>Socket Timeout</td>
<td>Required. Seconds to wait before the socket times out. Default: <strong>300</strong>.</td>
</tr>
</tbody>
</table>

**Reconnection Settings**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Reconnection Attempts</td>
<td>Required. Determines the maximum number of reconnection attempts to make when the connection tries to reach the remote FTP server. Enter 0 to disable. Default: <strong>3</strong></td>
</tr>
<tr>
<td>Reconnection Delay</td>
<td>Required. Seconds to wait between reconnection attempts. Default: <strong>1</strong></td>
</tr>
<tr>
<td>Fail On Connection Error</td>
<td>Determines whether the connection attempt should fail after the maximum number of reconnection attempts. If enabled, the connection throws an exception, the connection closes, and the related event source is stopped. You can then address connection issues and republish the connection before proceeding. Otherwise, the connection tries to establish a new connection with the FTP server until it is successful.</td>
</tr>
</tbody>
</table>

**SFTP Data Encryption Settings**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ciphers (SFTP only)</td>
<td>Optional. A comma-separated list of ciphers, in order of preference, to use for encryption during data transfer. For example, the list might include aes128-3tr, aes128-cbc, 3des-cbc, or blowfish-cbc. The FTP server may not support all available cipher names. If a list is not specified, the connection applies the default list from JSch (Java Secure Channel). JSch is a Java implementation of SSH2. Default: <strong>No</strong></td>
</tr>
</tbody>
</table>

**File Operations Settings**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idempotent Repository Folder</td>
<td>Optional. The path to a local directory where the connection can store idempotent repository files. Applicable only when you also select the Event Source property, No File Operations. For more information, see &quot;Repository Files&quot; on page 7.</td>
</tr>
<tr>
<td>Clear Idempotent Repository</td>
<td>Determines whether to clear the idempotent repository file when an event source starts up. Default: <strong>No</strong></td>
</tr>
</tbody>
</table>
Other Guidelines for FTP Connections

When you select the FTP connection properties, note that:

- If you supply values that are inapplicable to the protocol (FTP or SFTP), they are generally ignored. However, to configure anonymous access, leave the user name, password, and private key fields empty.

- You can use a known_hosts file with or without strict host key checking, to prevent connections to unverified hosts. With these options:
  - The list of valid hosts should include the host address and public key.
  - You can specify a known_hosts file without strict host key checking, the connection appends the host names it accesses to the known_hosts file.

- Some settings, including Passive Mode, Stepwise, and Fast Exists Check are server-specific. Verify that your FTP server supports these options before you use them.

- You can use the Fail On Connection Error, Max Reconnection Attempts, and Reconnection Delay to handle connection failures and prevent continuous connection attempts. If a connection is lost and the event source stops attempting to connect based on these settings, an error message is logged. See the property descriptions above for more information.
This chapter includes the following topics:

- FTP Connector Event Sources, 14
- Basic Event Source Properties, 14
- File Location Properties, 15
- File Operations and Polling Properties, 16
- File Read Lock Properties, 17
- File Parsing and Content Type Properties, 17
- Fixed Width Files, 21
- Event Source Endpoints, 21

### FTP Connector Event Sources

The FTP Connector includes these event source types:

- FTP Monitor. Monitors a remote folder to get information about newly added files. You can configure this event source to receive only file metadata or to process the plain text, binary, attachment, XML, or JSON file contents.
- Delimited Content Parser. Monitors a specified directory for new files and parses delimited file contents.
- Fixed Width Content Parser. Monitors a specified directory for new files and parses file contents that contain fixed width columns and no delimiters.

### Basic Event Source Properties

For each connection you configure, you can add one or more event sources. 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. 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 the connection, from the Event Sources tab, click Add Event Source. Choose the event source type from the available list.
The following table describes the Event Source properties available for all event source types:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Required. The name must be unique for this connection.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Select <strong>Yes</strong> to make this event source available immediately after it is published. Select <strong>No</strong> to disable the event source until you are ready to use it.</td>
</tr>
</tbody>
</table>

**File Location Properties**

The following table describes the file location properties.

**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/](https://regex101.com/) to ensure that the expression syntax is correct and to learn more about Java-style regular expressions.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disconnect (FTP and SFTP connections only)</td>
<td>Determines whether to disconnect from the remote FTP server each time an event source polls the directory. If the event source checks the remote directory infrequently, select <strong>Yes</strong> to disconnect and establish a new connection for each polling event. Default: <strong>No</strong></td>
</tr>
<tr>
<td>Directory</td>
<td>Required. Specify the path of the directory to monitor. For example: Windows: C:\AppName\DataFiles or \DataFiles Linux: /datafiles The question mark (?) is a prohibited character in this field. <strong>Note:</strong> With an FTP or SFTP connection, always specify a relative path.</td>
</tr>
<tr>
<td>Recursive</td>
<td>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. <strong>Note:</strong> The connector ignores any directory name that begins with a period (&quot;.&quot; character (for example, &quot;.\done&quot;). 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.</td>
</tr>
<tr>
<td>Include Files</td>
<td>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 .txt file extension, the regular expression would be similar to: ..txt It uses a back slash instead of a forward slash.</td>
</tr>
<tr>
<td>Exclude Files</td>
<td>Optional. Enter a regular expression you can enter to select files that should be excluded. Use a Java-style regular expression.</td>
</tr>
</tbody>
</table>
File Operations and Polling Properties

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No File Operations</td>
<td>Determines whether to disallow any move or delete file operations, to handle read-only data.</td>
</tr>
<tr>
<td>Delete Processed Files</td>
<td>Determines whether to delete each file after it is processed successfully. <strong>Note:</strong> Do not use this option with Move To.</td>
</tr>
<tr>
<td>Before Move</td>
<td>Optional. Enter a regular expression to dynamically set the new file name and location before processing it.</td>
</tr>
<tr>
<td>Move To</td>
<td>Optional. Enter a regular expression to dynamically set the file name and location where you want to move files after they are successfully processed. If you enable the Recursive property, use this option with care to avoid repeatedly processing moved files. <strong>Default:</strong> .done <strong>Note:</strong> Do not use this option with Delete Processed Files.</td>
</tr>
<tr>
<td>Move if Failure</td>
<td>Optional. Enter a regular expression used to dynamically set a different target directory if the Move To operation fails. <strong>Default:</strong> .error <strong>Note:</strong> Do not use this option with Delete Processed Files.</td>
</tr>
<tr>
<td>Initial Delay</td>
<td>Required. Seconds before polling begins.</td>
</tr>
<tr>
<td>Poll Interval</td>
<td>Required. Seconds to wait before polling again.</td>
</tr>
<tr>
<td>Max Messages Per Poll</td>
<td>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. <strong>Default:</strong> 0</td>
</tr>
<tr>
<td>Other Attributes</td>
<td>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. <strong>Note:</strong> If you add any of the specific attributes that are already exposed in the connector, such as Poll Interval, Process Designer ignores them.</td>
</tr>
</tbody>
</table>
### File Read Lock Properties

The following table describes the file read lock properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Lock</td>
<td>Required. 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.</td>
</tr>
<tr>
<td></td>
<td>The event source waits until the file lock is granted. Options:</td>
</tr>
<tr>
<td></td>
<td>- <strong>changed</strong>: 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.</td>
</tr>
<tr>
<td></td>
<td>- <strong>rename</strong>: Tries to rename the file as a test to decide whether an exclusive read-lock is available for the file. Default.</td>
</tr>
<tr>
<td></td>
<td>- <strong>none</strong>: Do not use read lock.</td>
</tr>
<tr>
<td></td>
<td>For some connection types (like the File Connector), these read lock options are also available:</td>
</tr>
<tr>
<td></td>
<td>- <strong>markerFile</strong>: Creates a marker file and holds a lock on it.</td>
</tr>
<tr>
<td></td>
<td>- <strong>fileLock</strong>: Acquires a read lock on the file. Not recommended when accessing a remote file system with a mount/share unless the file system supports distributed file locks.</td>
</tr>
<tr>
<td>Read Lock Timeout</td>
<td>Enter the number of seconds to wait before skipping the current file, if a read lock cannot be acquired. 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. Default: 10 seconds</td>
</tr>
<tr>
<td>Read Lock Check Interval</td>
<td>Enter the number of seconds to wait between attempts to acquire a read lock on a file. Default: 1 second Note: 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.</td>
</tr>
<tr>
<td>Read Lock Min Length</td>
<td>Enter the minimum file size, in bytes, that should be processed by the event source. Note: Use this property only when Read Lock is set to changed. Default: 1 byte</td>
</tr>
<tr>
<td>Read Lock Logging Level</td>
<td>Required. Select the logging level to use when a read lock cannot be acquired: OFF, INFO, WARN, or ERROR. Default: WARN</td>
</tr>
</tbody>
</table>

### 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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Applies To</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character Set</td>
<td>All Event Source Types</td>
<td>Required. Determines the encoding of the file. If none specified, the connector uses UTF-8.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>Delimited Content Parser</td>
<td>Required. The delimiter character for delimited files. To specify a space or a tab character as a delimiter, use the escaped character, &quot;\s&quot; or &quot;\t&quot;.</td>
</tr>
<tr>
<td>Text Qualifier</td>
<td>Delimited Content Parser</td>
<td>Required. The text qualifier for delimited files. To specify a space or a tab character as a qualifier, use the escaped character, &quot;\s&quot; or &quot;\t&quot;.</td>
</tr>
<tr>
<td>Ignore First Record</td>
<td>Delimited Content Parser</td>
<td>Determines whether the connector processes the first row of a delimited file as a data row. You can select from the following options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Yes: The connector does not process the first row of a delimited file as a data row.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- No: The connector processes the first row of a delimited file as a data row. If you select No, you must provide custom headers by using the Columns Descrip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tor attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is Yes.</td>
</tr>
<tr>
<td>Split Rows</td>
<td>Delimited Content Parser</td>
<td>Determines whether to process each row separately, convert each row to a process object, and generate a separate event for each of them.</td>
</tr>
<tr>
<td></td>
<td>Fixed Width Content Parser</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Applies To</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Columns Descriptor</td>
<td>Delimited Content Parser</td>
<td>Defines the fixed width column headers to be processed. Specify the values in a comma-separated list of names and sizes (in parentheses).</td>
</tr>
</tbody>
</table>
|                                | Fixed Width Content Parser      | Delimited Content Parser example:  
User Name, Password, Email  
Fixed Width Content Parser example:  
User Name(40), Password(8), Email(14)  
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). |
| Ignore Column Inconsistencies  | Delimited Content Parser        | For Delimited Content Parser, determines:  
- Whether to ignore extra columns in data rows.  
- Whether to add columns with empty string values if columns are missing. If not enabled, the event source throws an exception when it encounters extra or missing columns.  
For Fixed Width Content Parser, determines:  
- Whether to ignore extra characters in rows.  
- Whether to add characters with empty spaces if defined columns have missing values.  
Default: No. |
|                                | Fixed Width Content Parser      |                                                                                                                                              |

The following tables describe all the content type properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Applies To</th>
<th>Description</th>
</tr>
</thead>
</table>
| Content Format             | File Parser        | Required. Format of the content to be processed:  
- **Ignore.** Do not process the content.  
- **Plain Text.** Content is a string.  
- **Binary.** Content should be converted to a Base64-encoded string.  
- **XML and JSON.** Content should be parsed and converted to an object or a list of process objects.  
- **Attachment.** Content is an attachment. |
<p>|                            | File Monitor       |                                                                                                                                            |
|                            | FTP Monitor        |                                                                                                                                            |
| Simplify Content           | File Parser        | 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. Default: No |
|                            | File Monitor       |                                                                                                                                            |
|                            | FTP Monitor        |                                                                                                                                            |</p>
<table>
<thead>
<tr>
<th>Property</th>
<th>Applies To</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Object Mode</td>
<td>File Parser, File Monitor, FTP Monitor</td>
<td>If enabled, XML content is converted to a single object. Default: No</td>
</tr>
<tr>
<td>Use Built-in Process Objects</td>
<td>Delimited Content Parser, Fixed Width Content Parser</td>
<td>Select <strong>Yes</strong>, 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 <strong>No</strong> 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 &quot;Event Process Objects Overview&quot; on page 25</td>
</tr>
<tr>
<td>Custom Object</td>
<td>Delimited Content Parser, Fixed Width Content Parser</td>
<td>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: Name,Street,City,State,Postal Code,Country,Phone <strong>Note:</strong> You cannot use &quot;index&quot; 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 <strong>NCName</strong> format, to remove any prohibited characters from the header names and ensure they are valid process object field names.</td>
</tr>
</tbody>
</table>

**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.

**Event Source Endpoints**

Each event source you define must have an endpoint that is unique both within the connection and among all published connections in the organization. You can view the endpoints for a connection in the Details area of the Design Home page. When you define connections, the Process Designer validates that these endpoints are unique, to avoid a URI conflict. A URI conflict might otherwise occur, even for different event source types. A delimited content parser and file monitor, for example, could both use the same URI to monitor files (file:* or to write content (ftp:*)).

If a conflict is detected when you attempt to publish an event source, you can change the value for a single attribute to make the URIs unique and resolve the conflict. For example, the different poll interval (delay) shown in these URIs is sufficient:

file://D:\%5Ccamel_test%5Cfile%5Csource?
copyAndDeleteOnRenameFail=false&delay=1000&delete=false&initialDelay=1000&maxMessagesPerPoll=0&move=.done&moveFailed

file://D:\%5Ccamel_test%5Cfile%5Csource?
copyAndDeleteOnRenameFail=false&delay=2000&delete=false&initialDelay=1000&maxMessagesPerPoll=0&move=.done&moveFailed
CHAPTER 4

FTP Event Targets

This chapter includes the following topics:

- FTP Connector Event Targets, 22
- Basic Event Target Properties, 22
- Event Target File Properties, 23
- Delimited Content Writer Properties, 24

FTP Connector Event Targets

The FTP Connector includes the following event target types:

- FTP Writer. Write data in plain text, binary, attachment, JSON, or XML format to a remote file via FTP or SFTP.
- FTP Delimited Content Writer. Serialize process objects to delimited content format and write the result to a remote file via FTP or SFTP.
- FTP File Transfer. Copy local files to a remote location with FTP or SFTP. This event target streams the content of a file directly to FTP, a more efficient file transfer if you do not need to handle any of the file content in your process.

Basic Event Target Properties

For each connection 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. Then choose the event target type from the available list.
### Event Target File Properties

The following table describes the file location properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory</td>
<td>Required. Specify the path of the directory where you want to store files. For example: Windows: C:\AppName\DataFiles or DataFiles Linux: /datafiles The question mark (?) is a prohibited character in this field. With an FTP or SFTP connection, always specify a relative path. <strong>Note:</strong> Processes can create only nested folders.</td>
</tr>
</tbody>
</table>
| File Exists    | Required. Determines what to do if a file already exists with the same name:  
  - **Override** (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.  
  - **Append** to add content to the existing file. **Do not** specify a Temp Prefix or Temp File Name.  
  - **Fail** to skip the conflicting file and throw an exception to indicate that the file name already exists.  
  - **Ignore** to skip the conflicting file and silently ignore the problem (no exception thrown).  
  - **Move** 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.  
  - **Try Rename** 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. |
| Move Existing  | 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:  
backup\${file:name}_\${file:modified}  
This expression creates a new file in the backup folder every time existing files are moved and appends the timestamp to the file name. For more information on using the Apache Camel File Expression language, see: [http://camel.apache.org/file-language.html](http://camel.apache.org/file-language.html) |
| Eager Delete   | Select **Yes** 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). Select **No** 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. |
The following table describes the file writing properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temp Prefix</td>
<td>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. <strong>Note:</strong> Ignored if you select Append as the File Exists option.</td>
</tr>
<tr>
<td>Temp File Name</td>
<td>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. <strong>Note:</strong> Ignored if you select Append as the File Exists option.</td>
</tr>
<tr>
<td>Force Writes</td>
<td>Select <strong>Yes</strong> 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. Select <strong>No</strong> 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.</td>
</tr>
<tr>
<td>Write Buffer Size</td>
<td>Determines the size of the write buffer (in bytes). Default: 128kb.</td>
</tr>
<tr>
<td>Character Set</td>
<td>Required. Determines the encoding of the file. If not specified, the Event Target uses UTF-8 file encoding.</td>
</tr>
</tbody>
</table>

### 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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delimiter</td>
<td>Required. The delimiter character to use in files generated by this Event Target. <strong>Note:</strong> To use a space or a tab, enter it as an escaped character (&quot;\s&quot; or &quot;\t&quot;).</td>
</tr>
<tr>
<td>Text Qualifier</td>
<td>Required. The text qualifier to use in files generated by this Event Target. <strong>Note:</strong> To use a space or a tab, enter it as an escaped character (&quot;\s&quot; or &quot;\t&quot;).</td>
</tr>
<tr>
<td>Skip Headers</td>
<td>Required. Determines whether the generated content has header names. Select <strong>Yes</strong> to store the delimited content without header names.</td>
</tr>
<tr>
<td>Line Ending Style</td>
<td>Required. Determines the line ending style in the generated content (Windows style is \r\n. Unix style is \n.)</td>
</tr>
</tbody>
</table>
Chapter 5

Process Objects with FTP Connector

This chapter includes the following topics:

- Event Process Objects Overview, 25
- FTP Connector Process Objects, 26
- FTP Monitor Process Objects, 26
- Built-In Process Object Output, 27
- Custom Object Fields Output, 28
- File Information Process Objects, 29
- Event Target Process Objects, 29

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, its 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.
FTP Connector Process Objects

The following process objects are used with the FTP Connector:

- **FileContent.** Generated for each new file added to the target directory when you configure an FTP Monitor event source to include the file contents.
- **FileInformation.** File metadata generated for each file read by an event source. If you configure an FTP Monitor event to exclude the file contents, this is the only information returned.
- **<EventSource>Content.** Generates a custom delimited content object. If applicable, Field, Header, and Record process objects are also created.
- **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.

**Note:** An automated step, aetgt:automatedStepRequest, handles the FTP file transfer. It returns information about the file transfer, similar to the FileInformation process object.

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

In addition, these process objects represent the content of files handled by the FTP 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.

FTP Monitor Process Objects

When you use the FTP Monitor event source to access files on a remote FTP server, you generate one of two process objects:

1. **FileContent.** Includes the file’s contents in addition to the file metadata. If you choose the option to ignore content format, only the FileInformation event object is generated.
2. **FileInformation.** Similar to the FileInformation object generated by other event sources, this contains only the file metadata.

For example, the FileContent object contains a `<fileInfo>` element followed by the content. The content is represented as shown, based on the content type:

```xml
<FileContent>
  <!-- remote file information -->
  <fileInfo>
    <host>127.0.0.1</host>
    <lastModified>2015-09-21T10:05:20Z</lastModified>
    <dir>documents/json</dir>
    <name>users</name>
    <path>documents/json/users.json</path>
    <fullName>users.json</fullName>
    <ext>json</ext>
    <size>380</size>
  </fileInfo>
</FileContent>
```
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 split 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>
    <!-- for FTP/SFTP only -->
    <size>78</size>
  </fileInfo>
  <header>
    <name>User Name</name>
    <fieldIndex>1</fieldIndex>
  </header>
</DelimitedContent>
<header>
  <name>Age</name>
</header>
<record>
  <field><value>Bob Smith</value></field>
  <field><value>22</value></field>
  <field><value>1987</value></field>
</record>
<record>
  <field><value>Bill White</value></field>
  <field><value>45</value></field>
  <field><value>1999</value></field>
</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>2015-04-29T14:37:35.4482</lastModified>
    <dir>DataFiles</dir>
    <name>users</name>
    <path>/DataFiles/users.csv</path>
    <fullName>users.csv</fullName>
    <ext>csv</ext>
    <size>78</size>
  </fileInfo>
</MyUserFileContent>
```
File Information Process Objects

The FileInfo 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:

```xml
<FileInfo>
  <!-- for FTP/SFTP only -->
  <host>127.0.0.1</host>
  <!-- dateTime : file modified time -->
  <lastModified>2015-04-29T14:37:35.4482</lastModified>
  <!-- string: full path and name of resource. File separator is normalized to forward slash. -->
  <path>path/documents/users.csv</path>
  <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 -->
  <!-- double: file size in bytes -->
  <size>78</size>
</FileInfo>
```

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 -->
  <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 29.

### 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> 222222 </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>
</SerializeToDelimitedContentTask>
```
Whatever approach you take, after each file is processed, the results display:

```
<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>
```

Event Target Process Objects 31
<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 Design with FTP Connector

This chapter includes the following topics:

- Process Design Considerations, 33
- Example Process with FTP Connector, 33

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 FTP Connector

The following outline illustrates one way you can use the FTP Connector to read and parse content from a delimited file that resides on a remote FTP server. The content is added to a process object so you can use the data to make a service call and iterate through a list of records.

First, configure the Connection:

1. Define the basic connection properties and select FTP as the Connection Type. Be sure to select an agent where the connection will run.
2. In the FTP Connection Properties, enter or select the necessary details for the FTP connection. The required properties depend on the remote FTP server but these options are the minimum required:
   - FTP Protocol
   - FTP Host
• FTP Port

3. On the Event Sources tab, select **Add Event Source > FTP Delimited Content Parser**.

4. Enter a name for this event source and enable it so 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 now but must enable it before you can run the process successfully.

5. Specify the directory that should be monitored on the FTP server, where delimited files can be accessed. This field is required. You can specify other optional properties to exclude or include specific files or the subdirectories.

6. Enter other file parsing options as needed.

7. 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

8. Select any optional **File Read Lock Settings** you want to use.

9. 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:

   \[ \text{list:head}($\text{temp.TempIterator}) \]

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. For example, you might make a service call to an external service like RequestBin and pass in each record read from the file, so the service can generate a list of data.

8. Add an Assignment step to remove the processed record from the list of process objects. Specify Formula as the source of TempCurrent, and use the following:

   \[ \text{list:remove}($\text{temp.TempIterator,1}) \]

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 Is unset branch, you can make another service call or take some other action. In this example, the Is unset branch calls reads the FileInfo from the process object and calls a service to publish the record count to the client.

11. End the process.