Informatica

Informatica® Application Integration
April 2024

Amazon S3 Connector Guide

Informatica Application Integration Amazon S3 Connector Guide
April 2024

Publication Date: 2024-04-04

Table of Contents

Preface

Read the *Amazon S3 Connector Guide* to learn how to set up and use connections with the Amazon Simple Storage Service (S3).

This guide assumes that you have an understanding of your Amazon S3 environment and how to create connections and processes in Application Integration.

CHAPTER 1

Introduction to Amazon S3 Connector

This chapter includes the following topics:

Amazon S3 Connector

Application Integration supports several types of listener-based connectors that can be used for event-based processing of messages and files. These connectors expose a wide range of connection attributes you can use to configure a listener-based connection.

You can configure the Amazon Simple Storage Service (S3) Connector as a file monitor that makes data objects, service calls, and events available to the Process Designer.

Using the Amazon S3 Connector, you can work with files that reside in the S3 storage system similar to how you handle local files using the File Connector. For example, you can:

1. Monitor an S3 bucket for new objects and generate events for processes. You can access S3 object metadata or S3 object content as plain text, binary data, attachment or a set of process objects that are built from parsed XML, JSON, or delimited content.
2. Write S3 objects to an S3 bucket as delimited content, arbitrary string data, binary data, attachments or list of process objects (represented as XML or JSON).

Amazon S3 Object Store

Amazon S3 is a key-based object store. The unique object key assigned by the user can later be used to retrieve the data. Each object is stored and retrieved using a unique key.

Each account contains one or more buckets to store data and to organize the Amazon S3 namespace. An object is uniquely identified within a bucket by a key (unique identifier) and a version ID.

In this example:

```
http://mybucket.s3.amazonaws.com/myfiles/photo1.jpg
```

the bucket is "mybucket" and the key is "myFiles/photo1.jpg"

Each object in S3 has a data and metadata component. The metadata describes the object using name-value pairs and include, for example, the date last modified and other standard HTTP metadata, such as Content-Type. A bucket can also specify the geographical region where Amazon S3 will store it.

For more information on S3, refer to the Amazon product information, such as http://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html.

S3 Connector Implementation

The S3 Connector provides configurable start events (Event Sources) and event services (Event Targets) that can be defined in a Connection object using Process Designer.

The Event Sources (start events) and Event Targets (event service definitions) share common attributes, so a single S3 Connection can monitor several different buckets for new objects, read delimited content from other buckets, and write to another bucket, provided that all the buckets are related to a single access/secret key pair (the S3 account).

With this flexible implementation, you can combine a set of related tasks in one connection or split your work into several separate connections.

Processing S3 Objects with Event Sources

After the S3 Connector processes the S3 objects in a bucket, those objects are automatically deleted to ensure they are not processed again. As a result, S3 buckets used with the connector must be defined either as temporary storage or as a data exchange buffer between an isolated application and a Secure Agent, from which the objects can be processed and moved to another location.

The connector handles each object as follows:

- The start event (Event Source) you define reads the S3 object from the bucket and sends it to the processes handling the content.
- After the content has been processed, the S3 object is removed from the bucket and the event source reads the next object.
- If a problem is encountered during object processing, the processing terminates and an error message is printed to the log file. An error might be occur if:
  - An S3 object causes an error during processing and cannot be deleted because its content was not processed.
  - An S3 object cannot be moved to the local folder and deleted it from the bucket because it might cause an unexpected result. For example, if there is an error in the configuration of the connection where the process is running.
  - The Event Source settings have an incorrect encryption algorithm or password.
  - The connection is attempting to read encrypted S3 objects without first decrypting them. This is not permitted because the encryption keys would be removed from the bucket along with the original objects.

Encryption Method

Amazon S3 Connector supports client-side encryption using a master key that you store within your application.

When you upload an object to Amazon S3, you must provide a client-side master key. Amazon S3 Connector uses the master key to encrypt the encryption data key. Then, Amazon S3 Connector uses the encryption data key to encrypt the object.

When you download an object from Amazon S3, Amazon S3 Connector downloads the encrypted object from Amazon S3. Amazon S3 Connector uses the master key to decrypt the encryption data key. Then, Amazon S3 Connector uses the encryption data key to decrypt the object.

For more information about using a client-side master key, see the Amazon S3 user documentation.

Note: If you try to consume encrypted objects from Amazon S3 without enabling data encryption in the Amazon S3 connection, the connection fails.

Encryption Algorithms

The implementation is based on the S3 client-side encryption. The S3 Connector supports several different encryption algorithms. The encryption settings are used for both encryption of Event Sources and decryption of Event Targets. Before you can read an encrypted S3 object, you *must* decrypt the object.

For keys generated by the connection and based on the user password string, the encryption key is based on the PBKDF2WithHmacSHA1 key derivation algorithm with 4096 iterations, using the fixed salt, "Informatica".

For a user-provided encryption key, any key is acceptable provided the key length is correct.

Also see the Client-Side Encryption Settings below.

Java Cryptography Extension (JCE)

To use the encryption algorithms, you might need to install the Java Cryptography Extension (JCE). You can download the JCE from the Oracle website. To install it:

1. Extract the files from the JCE zip file you downloaded.
2. Copy the following jar files to *$JAVA_HOME/jre/lib/security:*
   local_policy.jar

   US_export_policy.jar

   Note: If these jar files already reside in the specified directory, make a backup copy and then*overwrite* them.
3. Restart your Secure Agent to begin using the new jar files.

After you perform these steps, you should be able to use all the encryption algorithms supported by the S3 Connector.

Amazon S3 Event Types

After you create an S3 connection, you can define one or more Event Sources and Event Targets that are associated with the connection.
The following event source types are available for Amazon S3:

- Monitor
- Delimited Content Parser
- Fixed Width Content Parser

The following event target types are available for Amazon S3:

- Writer
- Delimited Content Writer

CHAPTER 2

Amazon S3 Connections

This chapter includes the following topics:

Basic connection properties

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

Property	Description
Name	Required. Unique name for this connection that identifies it in the Process Designer.
Location	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.
Description	Optional. Description of the connection.
Type	Required. The connector or service connector to use for this connection. Select the type you want to configure.
Run On	Required. The Cloud Server or the Secure Agent where this connection should run.
Connection Test	If supported for the connection type, displays the results of the last connection test.
OData-Enabled	If supported for the connection type, select Yes to enable OData feeds and specify the allowed users and groups.
Allowed Roles for OData	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.

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

When you publish the connection, the **Metadata** tab displays the generated process objects.

Amazon S3 Connection Properties

IAM authentication provides secured access to Amazon S3 resources. You can use the AWS IAM system to map policies to IAM roles or external resources and to determine the list of permissions that can be assigned to the IAM roles.

Note: The IAM role that you configure in the Amazon S3 connection properties will have access to all the queues that you include in the Amazon S3 event source and event target properties. You must create a separate app connection to configure an IAM role to access a different set of Amazon S3 queues.

When you select **Amazon S3** as the connection type, you can configure Amazon S3 specific connection properties. The following table defines the Amazon S3 connection properties that you must configure in the **Connection Properties** section:

AWS Signature Settings	Description
Access Key ID	AWS access key ID of the requester. Alternatively, you can pass the following access key in the AWS SDK in the Java system properties: `aws.accessKeyId` You can pass the key using the `AWS_ACCESS_KEY_ID` environment variable.
Secret Access Key	AWS secret key of the requester. Alternatively, you can pass the following secret key in the AWS SDK in the Java system properties: `aws.secretKey` You can pass the secret key using the `AWS_SECRET_ACCESS_KEY` environment variable.
Use EC2 Role to Assume Role	Enables the EC2 role to assume another IAM role specified in the IAM Role ARN option.
IAM Role ARN	The Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role assumed by the user to use the dynamically generated temporary security credentials.
External Id	Provides a more secure access to the Amazon S3 bucket when the Amazon S3 bucket is in a different AWS account. You can use external ID to grant access to your AWS resources to a third party.

The following table describes the Amazon S3 policy and region settings, which apply to all event sources and event targets in the connection:

Amazon S3 Settings	Description
Policy	Optional. The bucket policy (using the JSON-based access policy language required by Amazon) to be applied to any new bucket created by the process.
Amazon S3 Endpoint	Optional. The region-specific website endpoint where the S3 bucket must reside if the process creates a new bucket. For example: `s3-website-us-east-1.amazonaws.com`
Region	Optional. The region where a new bucket must be located. For example: `us-east1`

Some regions support Amazon S3 buckets that support both Signature Version 2 and Signature Version 4 authentications. By default, the connection uses the Signature Version 2 authentication.
You must specify the target region explicitly in the **Amazon S3 Endpoint** field to run a process to read or write data to an Amazon S3 bucket that requires Signature Version 4 authentication. Else, the connection uses the Signature Version 2 authentication and the mapping fails. For more information about the Amazon S3 bucket regions support, see Amazon S3 documentation.

The following table describes the client-side encryption settings, which apply to all event sources and event targets in the connection:

Client-Side Encryption Settings	Description
Use Data Encryption	If you select **Yes**, the content of the S3 objects will be decrypted and encrypted by event sources and targets. Default is **No**.
Encryption Algorithm	Optional. Encryption algorithm that must be used to decrypt and encrypt the S3 objects.
Encryption Key	Optional. Base64-encoded secret key that must be used for encryption. The length of the key must match the required key length of the selected encryption algorithm (AES - 128, 192 or 256 bits; DES - 64 bits; 3DES - 192 bits).
Encryption Password	Optional. Encryption password that should be used to generate an encryption key. You can either enter the encryption key or generate the key using this parameter. If both parameters are provided, the encryption key is used.

CHAPTER 3

Amazon S3 Event Sources

This chapter includes the following topics:

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 **Yes** to make the event source available immediately after it is published. Select **No** to disable the event source until you are ready to use it. Default is **Yes**.

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*.

Amazon S3 Event Source Properties

After you define an Amazon S3 connection and deploy it in a process, you can monitor a bucket for new objects. When the S3 Connector detects a new object, it reads the S3 object and sends it to the process that subscribed to the Event Source, which acts as a start event.

For each connection, you can add multiple event sources.

A connection begins processing objects in the bucket immediately after it is published, even if you have not yet defined any processes to consume or subscribe to the events. You can use the initial delay setting to postpone processing the first bucket until you have published a process to handle events from the Event Source.

The following table describes the file location settings applicable to all S3 event source types:

Property	Description
S3 Bucket Name	Required. Amazon S3 bucket name that contains the objects to be processed by the event source. The question mark (?) character is prohibited in this field.
S3 Prefix	The S3 prefix that determines which objects are processed. Event source can use this attribute to consume only subset of S3 objects that share common prefix. If no prefix is specified, the event source reads all objects in the bucket.

The following table describes the **Object Operations Settings** applicable to all S3 event source types:

Property	Description
Delete S3 Objects	Required. Use this property to set whether you want to delete none, some, or all processed S3 objects from the source bucket. Choose one of the following options: - No: Do not delete any processed S3 object. - Consumed: Delete all successfully processed S3 objects. - All: Delete all, faulted or successfully processed, S3 objects. **Note:** If you select **No** or **Consumed**, the connector uses the move settings to prevent an object from being consumed twice. If you have not set a **Move to Bucket** name, the connector uses the default bucket. For example, if you select **No**, the connector uses the **Move To Bucket** and **Move Prefix** properties to move processed S3 objects. Default: No.
Move To Bucket	Optional. Specify the bucket to which you want to move successfully processed S3 objects. If you do not specify a **Move Prefix**, it is a good practice to specify a move to bucket different from the source bucket. Default: Source bucket.
Move Prefix	Optional. Specify the prefix you want to add to the keys of successfully processed S3 objects. If you want to move processed objects to the source bucket, it is a good practice to specify a move prefix. Default: `.ae-done/`
Move Failed to Bucket	Optional. Specify the bucket to which you want to move failed S3 objects. If you do not specify a **Move Failed Prefix**, it is a good practice to specify a move failed to bucket that is different from the source bucket and the move to bucket. Default: Source bucket.

Property	Description
Move Failed Prefix	Optional. Specify the prefix you want to add to the keys of failed S3 objects. If you want to move failed objects to the source bucket, it is a good practice to specify a move prefix. Default: `.ae-error/`
Backup In Folder	Optional. Specify a local directory in which you want to store a copy of successfully processed S3 objects.
Backup Failed In Folder	Optional. Specify a local directory in which you want to store a copy of failed S3 objects.

The following table describes the **Failure Handling Settings** applicable to all S3 event source types:

Property	Description
Max Retry Interval	In case of a fault, specify the maximum time (in seconds) that you want an event source to wait before making a recovery attempt. The delay interval between retries increases exponentially until the connector reaches the maximum retry interval. Default maximum: 3600 seconds or 1 hour. Minimum allowed: 1 second Maximum allowed: 86400 seconds or 24 hours.

The following table describes the polling and other settings applicable to all S3 event source types:

Property	Description
Initial Delay	Seconds to wait before polling the S3 bucket. Default: 1 second. Set the initial delay that you need, for example, to publish the related process or take other required steps before polling can begin.
Delay	Seconds to wait before the next poll to check whether new files have arrived. Default is 1 second. Set a longer delay between polls to reduce the number of S3 API calls.
Max Messages Per Poll	The maximum number of objects to retrieve in each poll. Default: 10. Set a value that enables you to segment the load and manage performance if you anticipate that the monitor will find a large number of objects to process during each poll.

File Content Properties for Monitor Event Sources

You can use the Monitor event type to monitor an S3 bucket for new objects and be notified when a new object is found.

The Monitor event source includes hat allow you to configure event sources and process different content types based on the S3 object data and the data format you need to receive:

The following table describes the content type settings applicable to all S3 monitor event source types:

Property	Description
Content Format	Required. Select the format of the content to be processed: - **Ignore**: Use this option when you need only metadata that you can obtain directly from the S3 object and not the content. - **Plain Text**: Use this option to get the S3 content as plain text. - **Binary**: Use this option to convert the S3 content to a Base64- encoded string for both text and other file types. - **XML** and **JSON**: Use this option if you need to parse and convert the S3 object content to an object or a list of process objects of any type, and send them together with an event to your process. - **Attachment**: Use this option if you do not need to obtain the S3 object content as part of an event, which might be more efficient. The content of an S3 object will be included as an attachment in the generated event.
Simplify Content	Required. Select **Yes** if the XML or JSON content to be processed is not in a valid process object structure and you want the event source to modify the structure of the content so it uses the process object's format. For example, if your XML object contains XML attributes that are normally skipped and you enable this option, the attributes are converted to nested XML elements and processed as valid object fields.
Single Object Mode	Required. Select **Yes** if you want to convert all XML content to a single process object. Use this option only if you need to work with XML content.

Delimited and Fixed Width Content

The Amazon S3 Connector allows you to parse new S3 objects with a delimited content or fixed width event source. You can provide the results to your process as an event.

The following properties are available for delimited and fixed width content:

Property	Description
Delimiter (Delimited Content Parser only)	Required. Specify the character delimiter for delimited content. If you need to use a space or a tab character as a delimiter, use the escaped character, \s or \t.
Text Qualifier (Delimited Content Parser only)	Specify the text qualifier for delimited content. If you need to use a space or a tab character as a qualifier, use the escaped character, \s or \t.
Ignore First Record (Delimited Parser Content only)	Determines whether the connector processes the first row of a delimited file as a data row. You can select from the following options: - **Yes**: The connector does not process the first row of a delimited file as a data row. - **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 Descriptor attribute. The default value is **Yes**.

Property	Description
Split Rows	Select **Yes** to process each row of delimited content separately, convert each row to a process object and generate a separate event for each of them. Select **No** to process the contents of the delimited content all at once and generate a single event.

Columns Descriptor	Defines the fixed width column headers to be processed. Specify the values in a comma-separated list of names and sizes (in parentheses).
	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	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.
Use Built-in Process Objects	Select **Yes** to use built-in process objects to map S3 object records to process objects. The event source then represents the S3 object content as a set of process objects. This is useful when you work with S3 objects that have different contents and use a different set of fields. Select **No** to simplify representation of delimited content and provide a list of field names in Custom Object Fields. This is suitable if you can reliably anticipate a set of fields (headers) in each the delimited content. The event source will generate a simple process object for each record of the delimited content.
Custom Object	A comma-delimited list of process object field names (headers) to use to represent the file contents. The names should match the headers of the delimited files. Specify values here when Use Built-In Process Objects is set to No. Do not use "index" as a field name here. It is reserved for row index information. Enter a comma-delimited list of process object field names (headers) that will be used to represent the file contents. The names should match the headers of the delimited files. When the file is parsed, if a header you enter here does not exist, the related field of the generated objects will be empty. Required if you select No in Use Built-in Process Objects. **Note:** You cannot use "index" as a field name here. It is reserved for row index information. Generated custom objects will use the provided set of field names but names will be in *NCName* format, to remove any prohibited characters from the delimited content header names and ensure they are valid process object field names.

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

Amazon S3 Event Targets

This chapter includes the following topics:

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.

Amazon S3 Event Target Properties

The event target properties allow you to control object creation and create an event service that can be called from Process Designer.

For each event target, you select the type (Writer or Delimited Content Writer) and specify a unique name. All event targets also require the following properties:

Amazon S3 Settings	Description
S3 Bucket Name	Required. Amazon S3 bucket name. If the provided bucket name entered here does not exist, the event target automatically creates it using the Amazon S3 settings specified in the connection properties. The question mark (?) character is prohibited in this field.
Storage Class	Required. The S3 storage class to set in the PUT object request. Select STANDARD or REDUCED_REDUNDANCY. See the Amazon S3 documentation for more details. Default is **STANDARD**.

Writer Properties

If you define an event target using the S3 Writer, you can write content as an S3 object to the S3 storage bucket. No additional properties are required.

Delimited Content Writer Properties

If you define an event target using the S3 Delimited Content Writer, you can serialize custom process objects or an *S3DelimitedContent* object to a delimited file format and store the result as a S3 object in a bucket.

These properties are available for the generated output:

Delimited Content Generation Settings	Description
Delimiter	Required. The character delimiter for delimited files. To specify a space or tab character as a delimiter, use an escaped character (\s or \t).
Text Qualifier	Required. The text qualifier for delimited files. To specify a space or tab character as a text qualifier, use an escaped character (\s or \t).
Skip Headers	Select **Yes** to store the delimited content without header names.
Line Ending Style	Choose the line ending style (Windows=\r\n, Unix=\n).

CHAPTER 5

Amazon S3 Process Objects

This chapter includes the following topics:

Monitor Event Objects

When you use an S3 Monitor event source, each time a new object is added to the bucket, the connection generates an *S3MonitorEvent* event object similar to the example below.

Note: Remember that S3 objects are deleted from the bucket and, unless you specify a local folder in the event source properties, the local file information is not available.

The event includes all the available S3 object metadata. If metadata is not available, the field is empty:

```
<S3MonitorEvent>
  <!-- local file information, may be empty if local backup copying is not used -->
  <localFileInfo>
    <lastModified> 2015-06-23T13:04:46.281Z </lastModified>
    <dir> D:/s3_test/s3 monitor </dir>
    <name> test1 </name>
    <path> D:/s3test/s3 monitor/test1.txt </path>
    <fullName> test1.txt </fullName>
    <ext> txt </ext>
    <size> 9 </size>
  </localFileInfo>
  <!--S3 object information -->
  <s3ObjectInfo>
    <lastModified> 2015-06-22T14:27:00Z </lastModified>
    <contentControl/>
    <s3VersionId/>
    <s3Key> test1.txt </s3Key>
    <contentType> text/plain </contentType>
    <contentEncoding> UTF-8 </contentEncoding>
    <contentDisposition/>
    <contentLength> 9 </contentLength>
    <bucketName> dvilaverde </bucketName>
    <s3ETag> b3d5cf638ed2f6a94d6b3c628f946196 </s3ETag>
  </s3ObjectInfo>
  <!-- if Content Format is equal Ignore content of the object will be skipped -->
  <!-- or if Content Format is equal PlainText content will be provided as a string
field -->
    <content>S3 object content string</content>
  <!-- or if Content Format is equal Binary content will be provided as a Base64 encoded
string field -->
    <content>dGVzdCBzdHJpbmc=</content>
```

```
  <!-- or if Content Format is equal XML or JSON content will be provided as a list of
one or more process objects -->
    <content>
    <lastName>Smith</lastName>
    <phone>111122-222</phone>
    <firstName>Bob</firstName>
    </content>
    <content>
    <lastName>Smith2</lastName>
    <phone>111122-222</phone>
    <firstName>Bob2</firstName>
    </content>
  <!-- or if Content Format is equal Attachment content will be provided as an
attachment field -->
    <attachment>cid:29cde59c-e85b-41e0-834c-22bbed5a5b9a</attachment>
</S3MonitorEvent>
```

Using Single Object Mode

When processing an XML document, you can choose whether to create a list of user objects or as a single process object.

For example, note that the following XML contains a list of user objects where the users element is a root element for the list:

```
<users>
  <user>
    <firstName>Bob</firstName>
    <lastName>Smith</lastName>
    <phone>111122-222</phone>
  </user>
  <user>
  <user>
    <firstName>Bob2</firstName>
    <lastName>Smith2</lastName>
    <phone>111122-222</phone>
  </user>
</users>
```

In this case, if you want to convert the whole XML document to a single *users* process object, where *user* is an object list field, *enable* **Single Object Mode**. If you want to convert the XML to a list of *user* objects, *disable* **Single Object Mode**.

Delimited Content Parser Process Objects

After the event source is configured and published, it begins to monitor the specified S3 bucket for new objects. When a new object is found, the contents are parsed and an event is generated that contains S3 object information, local copy information (if any) and a set of process objects that represent the parsed content. The generated event is sent to the processes listening for these events.

As described above, you can handle generated objects as built-in process objects or custom process objects.

Use these factors as a guide to choose your approach:

- Use Built-in Process Objects to process S3 objects with different structures and you do not know the content headers in advance.
- Use Custom Object Fields to process S3 objects with the same structure when you know the headers in advance. This method enables you to extract only the required data. Generated objects are simpler and they require less code to handle them in your processes.

Whatever method you choose, be aware of the differences in the generated process objects.

Built-in Process Object Output

When each S3 Delimited Content object is found and processed, it generates an event (*S3DelimitedContentParserEvent*) with a single parameter (*delimitedContent)* to indicate whether you are using built-in process objects (*S3DelimitedContent*) or custom process objects (*CustomS3DelimitedContent*).

For example, if you process a simple delimited content object with the following data, you might read a file similar to this:

```
Country Capital Area Region
USA Washington 11111 "North America"
Ukraine Kiev 22222 Europe
Japan Tokyo 33333 "Asia Pacific"
```

S3 object metadata field is always added to the generated events. Based on the metadata is available in the local file, the generated process object is similar to the following:

```
<S3DelimitedContent>
  <!-- S3 object information -->
  <s3ObjectInfo>
    <lastModified>2015-06-23T14:20:22Z</lastModified>
    <contentControl/>
    <s3VersionId/>
    <s3Key>test.csv</s3Key>
    <contentType>text/plain</contentType>
    <contentEncoding>UTF-8</contentEncoding>
    <contentDisposition/>
    <contentLength>130</contentLength>
    <bucketName>myBucket</bucketName>
    <s3ETag>ddfadc6de31dd30a6588bee8c01e157e</s3ETag>
  </s3ObjectInfo>

        <!-- local copy file information, will be empty if local copying is not used -->
  <localFileInfo>
    <lastModified>2015-06-23T14:20:22.086Z</lastModified>
    <dir>D:/camel_test/s3 monitor</dir>
    <name>test</name>
    <path>D:/camel_test/s3 monitor/test.csv </path>
    <fullName>test.csv</fullName>
    <ext> csv </ext>
    <size>130</size>
  </localFileInfo>

        <!-- total number of rows in the result -->
        <totalRowsCount>3</totalRowsCount>

  <!-- list of file headers with names and indexes -->
  <header>
    <name>Country</name>
    <fieldIndex>1</fieldIndex>
  </header>
  <header>
    <name>Capital</name>
    <fieldIndex>2</fieldIndex>
  </header>
  <header>
    <name>Area</name>
    <fieldIndex>3</fieldIndex>
  </header>
  <header>
    <name>Region</name>
    <fieldIndex>4</fieldIndex>
  </header>

  <!-- delimited content records -->
  <record>
  <field>
    <value>USA</value>
  </field>
```

```
    <field>
      <value>Washington</value>
    </field>
    <field>
      <value>1111</value>
    </field>
    <field>
      <value>North America</value>
    </field>
      ...
  </S3DelimitedContent>
```

The generated event contains an *S3DelimitedContent* process object. Local copy information might be empty if the event source does not store local copies of processed S3 objects.

You can see that the generated result contains information about the source object, list of header objects, list of records and total row count. The S3 object content is represented with the header name and value for each field.

If you work in split rows mode for delimited content, the S3 object content is divided into separate rows and for each row, the S3 connection produces a delimited content object with headers and one record.

Custom Object Fields Output

If you process the same S3 object, using, instead, the Custom Object Fields, you might provide the header names, "Country, Capital, Area" (but not "Region").

In that case, the results for the same object look similar to this:

```
<AwsS3DelimitedContentParserContent>
  <!-- S3 object information -->
        <s3ObjectInfo>
    <lastModified>2015-06-23T14:20:22Z</lastModified>
    <contentControl/>
    <s3VersionId/>
    <s3Key>test2.csv</s3Key>
    <contentType>text/plain</contentType>
    <contentEncoding>UTF-8</contentEncoding>
    <contentDisposition/>
    <contentLength>130</contentLength>
    <bucketName>myBucket</bucketName>
    <s3ETag>ddfadc6de31dd30a6588bee8c01e157e</s3ETag>
  </s3ObjectInfo>

        <!-- local copy file information, will be empty if local copying is not used -->
  <localFileInfo>
    <lastModified>2015-06-23T14:20:22.086Z</lastModified>
    <dir>D:/camel_test/s3 monitor</dir>
    <name>test</name>
    <path>D:/camel_test/s3 monitor/test.csv </path>
    <fullName>test2.csv</fullName>
    <ext>csv</ext>
    <size>130</size>
  </localFileInfo>

        <!-- total number of rows in the result -->
        <totalRowsCount>3</totalRowsCount>

  <!-- custom delimited content records with specified by user fields -->
  <record>
    <index>1</index>
    <Area>11111</Area>
    <Capital>Washington</Capital>
    <Country>USA</Country>
  </record>
  <record>
    <index>2</index>
    <Area>22222</Area>
    <Capital>Kiev</Capital>
```

```
    <Country>Ukraine</Country>
  </record>
  <record>
    <index>3</index>
    <Area>33333</Area>
    <Capital>Tokyo</Capital>
    <Country>Japan</Country>
  </record>
</AwsS3DelimitedContentParserContent>
```

Here, the process object contains the S3 object information, local copy information, and total number of records. "Region" data was excluded because it was not specified as a custom object field.

Notice that when you use custom object fields:

- The generated output still includes S3 object information.
- The process object name takes the format *<sourceName>Content*.

  Note: The custom record object uses the format *<sourceName>Record*. If you have several delimited content event sources, each of them uses its own custom content and record objects.
- Field names are converted to *NCName* format to remove any prohibited characters from the delimited content header names and ensure they are valid process object field names.
- If the source S3 object does not contain the required header, this field is empty in the generated output. In the above example, if "Region" had been specified as a custom object field and the source S3 object did not contain the "Region" header, the field would appear in the process object but be empty.
- As shown above, to skip some fields, simply omit them from the list of Custom Object Fields.

Writing Content with Event Targets

When you publish a connection that includes an event target definition for S3, you create a service that you can call from a process to generate S3 objects.

Writer Event Targets

`StoreS3ObjectRequest` is the process object that contains the target object parameters and contents, as shown here. This is the input to the event target:

```
<StoreS3ObjectRequest>
        <!-- create S3 object metadata -->

  <s3ObjectParameters>
    <!-- object key, is required -->
    <awsS3Key>test.txt</awsS3Key>
    <!-- content type, by default text/plain type is used -->
    <contentType>text/plain</contentType>
    <!-- content control -->
    <contentControl>…</contentControl>
    <!-- content disposition which is a default file name (similar to Content-
Disposition HTTP header) -->
    <contentDisposition>…</contentDisposition>
    <!-- content encoding, by default this property is empty -->
    <contentEncoding>…</contentEncoding>
    <!-- last modified timestamp in milliseconds -->
    <lastModified>…</lastModified>
    <!-- storage class (STANDARD or REDUCED_REDUNDANCY), can be used to overwrite the
same attribute
         of the event target -->
    <storageClass>…</storageClass>
    <!-- canned acl (Private, PublicRead, PublicReadWrite, AuthenticatedRead,
LogDeliveryWrite,
```

```
        BucketOwnerRead, BucketOwnerFullControl). -->
    <cannedAcl>…</cannedAcl>
    <!-- a well-constructed Amazon S3 Access Control List xml string. -->
    <s3Acl>…</s3Acl>
  </s3ObjectParameters>
  <!-- content format (PlainText|Binary|Attachment|XML|JSON) depending on what format
you use the content
       should be provided differently -->
    <format>PlainText|Binary|Attachment|XML|JSON</format>

  <!-- this field can be used only if you set format to PlainText or Binary.
       if you use PlainText format this field should contain some textual information
that will be written as
       is to the bucket; if you use Binary format here you should provide a Base64
encoded string that will be
       decoded and written to the target bucket -->
    <content>Hello World</content>

  <!-- if you chose the XML or JSON format you should use either the object or objects
field to provide one or
       several process objects that should be converted to XML/JSON content -->
    <object>some process object</object>
    <objects>a list of process objects</objects>

  <!-- optionally you can provide objectName and listName attribute values, details see
below this example -->
    <objectName>user</objectName>
    <listName>users</listName>

</StoreS3ObjectRequest >
```

For details on the Access Control List (ACL) parameters, see the Amazon documentation on Access Control List Overview and CannedAccessControlList.

The only required parameters are:

- AwsS3Key
- format

You can omit all other parameters. Depending on the format you select, the parameters vary.

Note:

StoreS3ObjectResponse is the process object returned by the Writer event target similar to the following:

```
<StoreS3ObjectResponse>

  <!-- S3 object version (if available) -->
    <awsS3VersionId/>
  <!-- S3 object ETag -->
    <awsS3ETag>b10a8db164e0754105b7a99be72e3fe5</awsS3ETag>

</StoreS3ObjectResponse>
```

XML/JSON Format

When you use XML/JSON format, the S3 connection expects an *object* (a single process object) or *objects* (a list of process objects). If you pass in a single process object using the objects field, the result is a list with a single item. If you pass in the same object using object fields, the result is a single XML/JSON object (not a list).

You can use the optional parameters, *objectName* and *listName*, to change the generated XML and JSON content as shown below.

XML: When you serialize a list of *user* process objects to XML format, the output is similar to:

```
<objects>
  <object>
    <firstName>Bob</firstName>
```

```
    <lastName>Smith</lastName>
    <phone>111122-222</phone>
  </object>
  <object>
    <firstName>Bob2</firstName>
    <lastName>Smith2</lastName>
    <phone>111122-222</phone>
  </object>
</objects>
```

The root elements have generic names (object and objects) by default because Process Designer does not have information about any object list or individual object names.

If you need to change the root element names, you can use the *objectName* and *listName* parameters to specify element names.

JSON: To serialize the same objects list to JSON format, you can use the *objectName* and *listName* parameters to add a wrapper JSON object to the generated result. Without these parameters, the output is a valid JSON array:

```
[{
  "firstName": "Bob",
  "lastName": "Smith",
  "phone": "1111-222-333"
},
{
  "firstName": "Bob2",
  "lastName": "Smith",
  "phone": "1111-222-333"
}]
```

You can also serialize the same process objects list as a JSON object as a single array by adding the "users" wrapper in the *listName* parameter:

```
{
  "users": [{
                "firstName": "Bob",
                "lastName": "Smith",
                "phone": "1111-222-333"
            },
            {
                "firstName": "Bob2",
                "lastName": "Smith",
                "phone": "1111-222-333"
            }]
}
```

As shown above, the JSON object now contains a single JSON object wrapper with a single field that contains the list *user* objects.

To wrap a single object in the same way, use the *objectName* parameter instead.

Attachment Format

If you use the Attachment format, the approach is different. Instead of providing content as part of the *StoreS3ObjectRequest* object,, you can use a separate *S3Writer* service call parameter, *attachment*.

Delimited Content Writer Event Targets

You can serialize both built-in delimited content process objects and a list of custom process objects.

When you publish a connection that contains a delimited content event target, Process Designer creates the following process objects:

- StoreS3DelimitedContentRequest
- StoreS3DelimitedContentResponse

Whether you serialize the *S3DelimitedContent* process object or a list of custom process objects, the request and response are nearly identical.

For example:

You can serialize both built-in delimited content process objects and a list of custom process objects.

```
<StoreS3DelimitedContentRequest>
  <!-- create S3 object metadata -->
  <s3ObjectParameters>
  <!-- object key, is required -->
    <awsS3Key>test.csv</awsS3Key>
  <!-- other optional parameters (the same as S3 Writer request parameters) -->
  </s3ObjectParameters>

  <!—delimiter char; overwrites the same setting of the event target -->
    <delimiter>;</delimiter>
  <!-- text qualifier char; overwrites the same setting of the event target -->
    <textQualifier>”</textQualifier>
  <!-- whether to skip headers; overwrites the same setting of the event target -->
    <skipHeaders>false</skipHeaders>

  <!-- built-in S3DelimitedContent process object model, in this case not used -->
    <delimitedContentObject/>

  <!-- list of custom headers -->
    <header>
  <!-- header name (custom object field name) -->
    <name>name</name>
  <!-- header index (column position index) -->
    <fieldIndex>1</fieldIndex>
    </header>
    <header>
    <name>address</name>
    <fieldIndex>2</fieldIndex>
    </header>
    <header>
    <name>phone</name>
    <fieldIndex>3</fieldIndex>
    </header>

  <!-- list of custom process objects -->
  <customObjects>
    <id>1</id>
    <name>Bill</name>
    <phone>123-11-22</phone>
    <address>Lviv</address>
  </customObjects>
  <customObjects>
    <id>2</id>
    <name>Bob</name>
    <phone>222-333-222</phone>
    <address>London</address>
  </customObjects>

</StoreS3DelimitedContentRequest>
```

Note: The object metadata parameters are the same as those used in the S3 Writer. If needed, you can overwrite some of the event target attributes (delimiter, text qualifier, and skip headers).

Handling Headers with Custom Process Objects

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. In this case, you should leave the *delimitedContentObject* field

(see above) empty. During serialization, Process Designer generates a list of headers using the first process object's simple field names. Sometimes this method is useful but it has several disadvantages:

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

You can eliminate these disadvantages if you provide 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.

If you work with the built-in process objects, when you define the request object, you can include the *S3DelimitedContent* process object in the request. The generated delimited content then matches the headers and records described by this model object.

Note: The *s3ObjectInfo* and *localFileInfo* fields in the *S3DelimitedContent* object are ignored during serialization so you can leave them empty if you create a new object yourself or retain them if you received the delimited content process object from somewhere and these fields are set

Serialization Results

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

- Status of the operation.
- Optional message string.
- Two counters with the number of processed records and number of records successfully written to S3 object content.
- Object metadata for the new S3 objects.

StoreS3DelimitedContentResponse

The response process object contains:

```
<StoreS3DelimitedContentResponse>
  <!-- true if success, false in case of error -->
    <success>true</success>
  <!-- textual message that may contain some warnings or errors -->
    <message>…</message>
  <!-- number of processed records (number of records that are provided in request) -->
    <processedRecordsCount>3</processedRecordsCount>

  <!-- number of successfully serialized and written to S3 object content records -->
    <writtenRecordsCount>3</writtenRecordsCount>

  <!-- created S3 object information -->
  <s3ObjectInfo>
  <!-- S3 object version (if available) -->
    <awsS3VersionId/>
  <!-- S3 object ETag -->
    <awsS3ETag>ae784bc7446c154e2f802c8440a530a0</awsS3ETag>
  </s3ObjectInfo>
</StoreS3DelimitedContentResponse>
```