

EQUIFAX[®]



FTPConnect User Guide

August 2020



Contents

FTPConnect Overview	3
Overview.....	3
How IT Works.....	4
Getting Started	5
Install FTPConnect	5
Configure FTPConnect	5
Configure Staging/Relay Service URL	5
Configuring Connections	6
Setting Up and Testing FTPConnect without Encryption	7
White Listing FTPConnect.....	12
Integrate With FTPConnect	12
FTPConnect API.....	13
FTPConnect Job Examples.....	19
Importing data Using FTPConnect Jobs.....	19
Exporting Data With FTPConnect Jobs	24
Scheduled FTPConnect Jobs.....	24
Encrypt/Decrypt with FTPConnect.....	24
Manage FTPConnect Sessions	33
Session History, License Terms and Usage	34
Managing Sandbox License.....	34
Troubleshooting and Debugging	36



FTPConnect Overview



Overview

FTPConnect is a powerful tool to help you move data between your Salesforce.com instance and one or more Secure FTP (SFTP) servers. Whether you need to import data from an SFTP site into Salesforce.com or export data from Salesforce.com to an SFTP site, FTPConnect gives you the tools to do this in a secure, simple and straightforward manner. FTPConnect gives you the ability now to securely and easily integrate your Salesforce.com instance with SFTP. With it, you can easily:

- Load delimited (comma delimited, pipe delimited, tab delimited) or fixed format files from an SFTP site into one or more database objects in Salesforce.com
- Export data from your Salesforce.com instance onto an SFTP site
- Get files from an SFTP site to store as Attachments or as Documents on your Salesforce.com instance
- Store Attachments or Documents from your Salesforce.com instance onto an SFTP site

In addition to handling the transfer of data between an SFTP site and your Salesforce.com instance, FTPConnect also provides you with the ability to encrypt data you transfer to the SFTP site and to decrypt data that you pull into Salesforce.com from an SFTP site. FTPConnect supports the following cryptography algorithms:

- PGP
- AES
- DES
- Blowfish

In addition, you can use FTPConnect to generate keys for AES, DES and Blowfish

Quick Start

All data transfers can now be initiated from within your Salesforce.com instance. There is no need to install expensive and complicated ETL software on your servers. Within minutes of installing the FTPConnect managed package onto your Salesforce.com instance, you can start moving data between Salesforce.com and your SFTP site.

Steps to start Data Transfer:

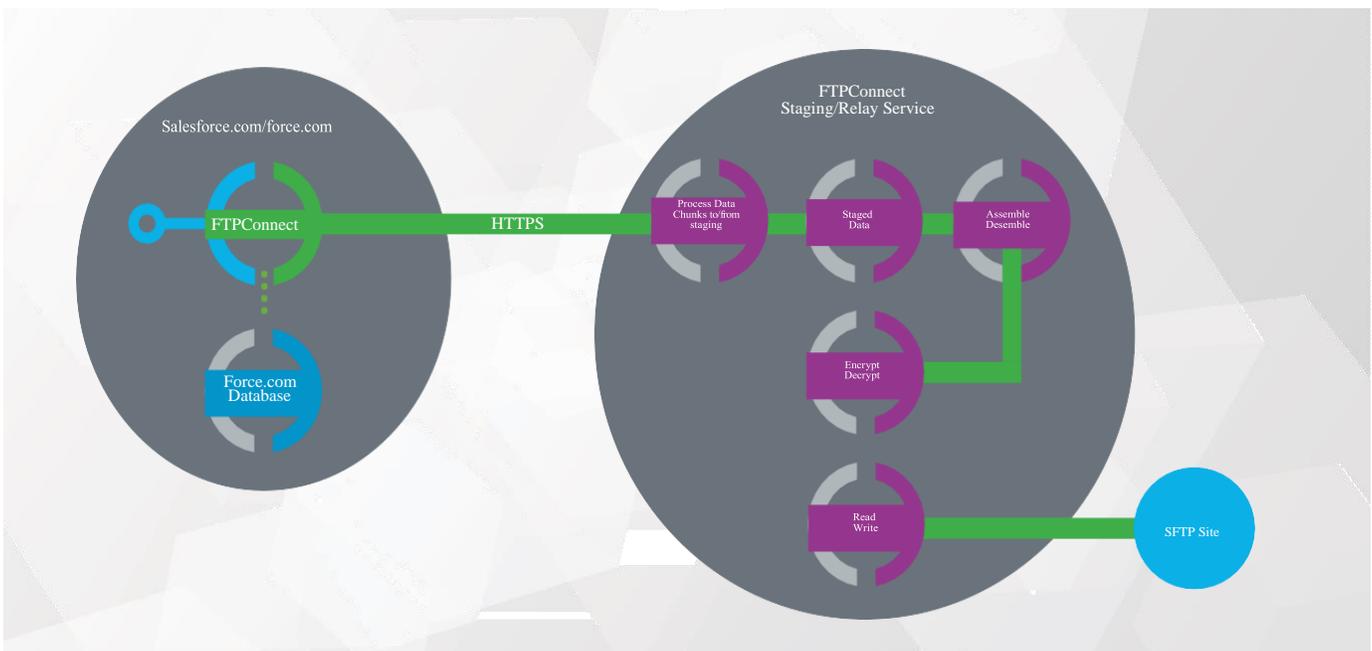
- 1** Install the FTPConnect managed package
- 2** Configure FTPConnect
- 3** With the very easy to use FTPConnect API and with the Apex programming language, quickly implement processes to exchange data between your Salesforce.com instance and one or more Secure FTP sites

How IT Works

Before now, if you needed to integrate your Salesforce.com environment with an SFTP site, you may have started by searching Salesforce user and developer forums for a way to do this, only to find out that the Force.com platform does not provide an API for SFTP integration. This was a problem we came across when our own customers asked us to help them integrate their Salesforce.com instances with SFTP sites. FTPConnect leverages the communication protocol that Salesforce.com already supports very well – HTTPS. **FTPConnect handles for you the difficult processing issues related to Force.com limitations:**

- SFTP Communication.
- Governor limit on total size of an HTTPS request.
- Governor limit on database queries and manipulation.
- Governor limit on heap memory.
- Encryption and decryption.

FTPConnect solves this problem by utilizing a “Staging/Relay” service. The FTPConnect managed package securely communicates with the Staging/Relay service using the HTTPS. The Staging service lives on the cloud between Salesforce.com and your SFTP site. For large file transfers, the FTPConnect managed package provides a batch process that sends/receives data from the Staging service in small chunks. All of the hard parts then are handled by the FTPConnect staging service.



When **sending data from Salesforce.com to an SFTP site**, the relay server securely stores each chunk as it is sent one at a time through the FTPConnect managed package on Salesforce.com. Once all data for a transmission is staged, the FTPConnect Staging service assembles the data, encrypts it and writes it to your file on the SFTP site. When getting a file from an SFTP site, the FTPConnect relay service disassembles the file into small manageable chunks and then hands them to your application on Salesforce.com (via the FTPConnect managed package) one manageable chunk at a time.

When **receiving data from an SFTP site into salesforce.com**, the Staging/Relay service reads the file from the SFTP site, breaks it down into small manageable chunks and temporarily stages those chunks for retrieval by the FTPConnect managed package running on Salesforce.com.

Getting Started

Install FTPConnect

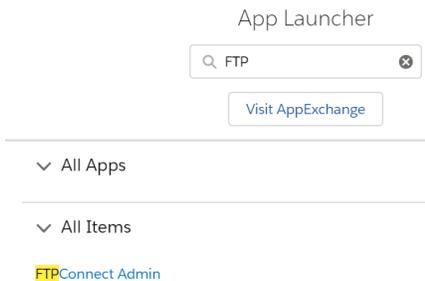
To get started, install the FTPConnect managed package onto your Salesforce.com instance. This starts a 30 day free trial. After 30 days, just purchase FTPConnect application licenses to continue using FTPConnect.

Configure FTPConnect

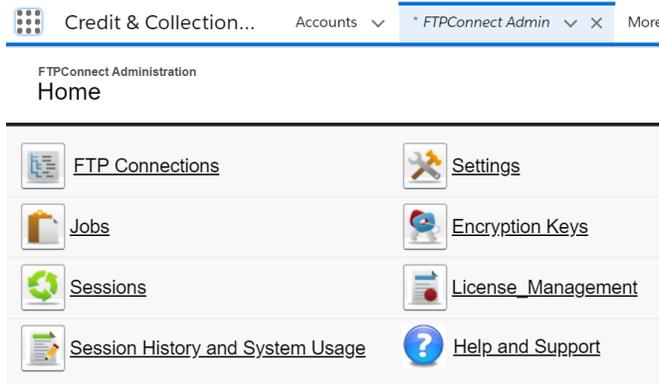
Once you install FTPConnect, you can begin configuring it.

Open FTPConnect Admin App:

- Click on 
- Search for FTPConnect in the App Lightning App Launcher Search and Select FTPConnect Admin



FTPConnect Admin Tab gives you access to all configuration and management features provided in the product.



Configure Staging/Relay Service URL

The Relay_Service_URL setting should only be set under the direction of the Equifax BusinessConnect support team. It is used to override the default URL used by the application to connect to the production Staging/Relay service. Do not configure this setting without consulting with the Equifax BusinessConnect support team.

It is possible to set up your instance to use an alternative Staging/Relay service. The first task is to set up your connection to the FTPConnect Staging/Relay service:

Links to install the latest version of the FTPConnect managed package are found on the FTPConnect support site:

[FTPConnect Support](#)



The screenshot shows the 'Add a Setting' form in the FTPConnect Administration interface. The 'Setting Name' dropdown is set to 'RELAY_SERVER_URL' and the 'Setting Value' text field contains the URL 'http://ftpconnectuatprivate.herokuapp.com'. An 'Add Setting' button is visible below the form. Below the form is a table with the following structure:

Action	Setting Name	Setting Value
--------	--------------	---------------

- Go to the FTPConnect Admin tab.
- Click on the Settings link.
- Select RELAY_SERVER_URL in the Setting Name dropdown list.
- In the Setting Value field, type or paste this URL setting: <http://ftpconnectuatprivate.herokuapp.com>
- Click the Add Setting button.

Configuring Connections

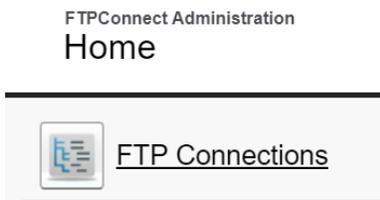
To get started, you need to create and test connections with an SFTP site. To create an FTP connection, you need the following information:

- The domain name of an SFTP site you will be getting files from or putting files to
- The username and password for your SFTP site
- If the SFTP site requires Public Key Authentication, then you will need username, private key and optionally the private key passphrase (if the private key is encrypted).
 - **For public key authentication:** you need a valid SSH key pair (with public and private keys). Work with your security team to obtain your key pair. The administrator of the SFTP site you are connecting to will need to install the public key on the site and set it up for public key authentication. You will install the private key and optionally a passphrase for that key in the FTP Connections setup pages.
 - The easiest way to generate a key pair that works with SFTP is to use the ssh-keygen command which is available in Linux and other versions of Unix.
- **If files on your SFTP site are to be encrypted at rest, then you also need the following:**
 - The name of the encryption algorithm you plan to use (i.e. AES, PGP, DES, Blowfish are supported)
 - The encryption and decryption keys in PEM format
 - Paths and file names on the SFTP site where files will be read and written.

Setting Up and Testing FTPConnect without Encryption

The following bullets demonstrate setting up and testing connections in FTPConnect without encryption. See the section titled **Encrypt/ Decrypt with FTPConnect** for details on how cryptography keys are stored and used with an FTPConnect connection.

- Click **FTP Connections** to view all configured connections



- Click **New** to configure a new connection



- For **Simple Connections to SFTP sites** that do not use public key authentication fill these fields out:

- **Connection Name** – Any name you want works.
- **Service Type** – Just select SFTP (SFTP is the only supported protocol).
- **URL** – Provide the domain name of your SFTP server.
- **Username** – Provide the user name you will use to connect to your SFTP server.
- **Password** – Provide the password you will use to connect to your SFTP Server.

FTPConnect Administration
Edit Connection

[Save](#) [Cancel](#)

Name & Description

Connection Name

Description

Connection Parameters

Service Type

URL

Username

Use Public Key Authentication

Password

Encryption Settings

Encryption Algorithm

Encryption Key

Decryption Key

[Save](#) [Cancel](#)

■ For **Public Key Authentication** fill in these fields:

- **Connection Name** – Any name you want works.
- **Service Type** – Just select SFTP (SFTP is the only supported protocol).
- **URL** – Provide the domain name of your SFTP server.
- **Username** – Provide the user name you will use to connect to your SFTP server.
- **SSL Private Key** – Private key from the key pair.
- **SSL Key Pass Phrase** (Optional) – Provide this if the private key is encrypted with a pass phrase.

FTPConnect Administration
Edit Connection

Save Cancel

Name & Description

Connection Name

Description

Connection Parameters

Service Type

URL

Username

Use Public Key Authentication

SSL Private Key (for pub key auth)

SSL Key Pass Phrase

Encryption Settings

Encryption Algorithm

Encryption Key

Decryption Key

Save Cancel

■ **Encryption Settings:**

- **Encryption Algorithm:** Select Encryption type:
 - AES
 - Blowfish
 - DES
 - DESede
 - PGP: Special Stipulations for PGP Setup:
 - When you edit or create a new connect PGP Connection you will see a checkbox called 'Signature Required' which you will need to check
- **Encryption Key:** Lookup your Encryption Key
- **Decryption Key:** Lookup your Decryption Key

■ Press **Save** to take you back to Connections Page

Testing FTPConnect Connections

- Once back on Connections page press the **Test** Action

FTPConnect Administration
Connections

Success:
Connection edits were canceled.

New Admin Home

Action	Connection Name	Service Type	URL	Username	Encryption Algorithm	Decryption Key	Encryption Key
Edit Del Test	test		test	1234565			

- This will bring you to the Test Connections Page

FTPConnect Administration
Test Connections

Admin Home

▼ Connection Details

FTP Connection: test

File Path:

Get Data Put Data Delete

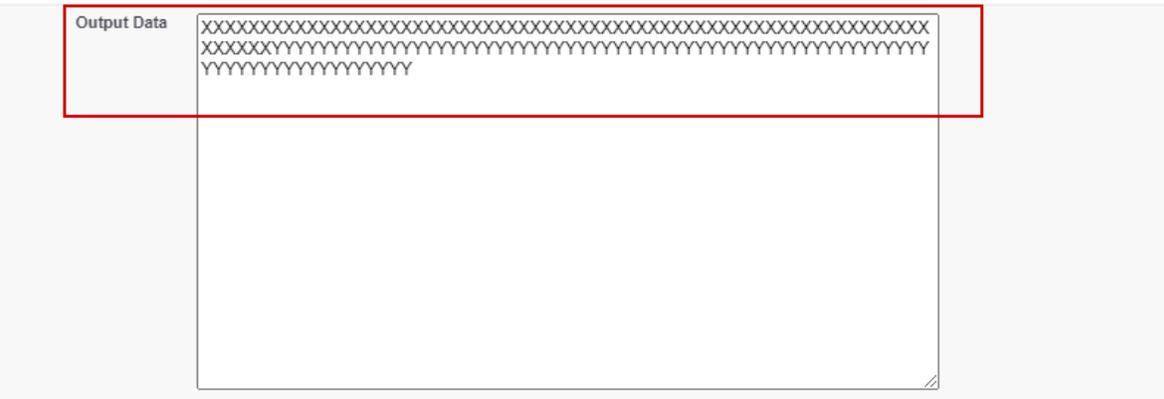
▼ Step-by-Step Test Options

FTP "Get" File Steps: 1) Get Start 2) Number of Records: Get Records

FTP "Put" File Steps: 1) Put Start 2) Put Write 3) Put Done 4) Check File Write Status

▼ Test Data

- Two terms you will see on the FTPConnect Test Page are “**Put...**” and “**Get...**”. Both **Put** and **Get** are the technical terms for the basic FTP operations.
 - Put** – is an operation where a file is written to an FTP site.
 - Get** – in an operation where a file is read from an FTP site.
- First, test writing a file to the SFTP site. On the test page, you provide the name and path of the file that you want to write to on the SFTP site and provide some data that you want written to that file.
 - File Path** – This is the Path and Name of the file.
 - Input Data** – Provide some data to write to that file. In the example shown in the screen shot, the user just types a bunch of X’s and Y’s.
 - Click the **Put Data** button to place the data into a file on the SFTP site
 - You should see a success message at the top of the page.



- Note that you might get errors and this is why you test. The error descriptions should be descriptive enough for you to solve any problems. A likely error for example is:
 - Incorrect domain name for the SFTP site
 - Incorrect username or password for the SFTP site
 - An incorrect file path in the File Path field
 - Lack of permissions on your SFTP site to read or write on one of the directories in the file path

White Listing FTPConnect

If the SFTP site for a connection requires white listing of SFTP clients that connect to it, then have the SFTP server administrator add these IP addresses to the SFTP site white-list:

- 34.237.214.24
- 34.204.23.183
- 52.203.56.194
- 34.204.30.215

Integrate With FTPConnect

FTPConnect provides three different API's for transmitting data between a Salesforce.com instance and SFTP site. The API you use will depend on how you want your application to work and on the size of the file to be written to or read from the FTP site.

For files smaller than 3MB:

- The FTPConnect API provides a single API call that can read/write the entire content of a file with one method call.

For files larger than 3MB, FTPConnect provides two batch options:

- **FTPConnect Jobs** – You configure a job and write a very simple APEX class to handle:
 - Parsing/formatting of data
 - Insertion/Updates of data or Selection of data to/from the Salesforce.com database. Your APEX class simply extends a global FTPConnect class and then you write code to override a couple of methods. Then FTPConnect job then handles all of the details related to the batch process.
- **Batch Class** – You write your own APEX batch class that calls a lower level FTPConnect API.

FTPConnect API



The FTPConnect API provides two types of access. First, through the **ftpconnect.FtpService** class, you can write APEX code that has access to the low level features of FTPConnect. Second, for importing or exporting large amounts of data, FTPConnect provides an easy API that allows you to create classes that can be used with **FTPConnect Jobs**.

This document provides two detailed examples of setting up and coding **FTPConnect Jobs** which is the easy and recommended approach for using FTPConnect for sending or receiving large files to/from SFTP. The other approach is to write your own batch job that uses the low level FTPConnect api.

- For examples that demonstrate the lower level API (`ftpconnect.FtpService`), please see the FtpConnect sample library found on the **FTPConnect Support** site.

FtpConnect Job API

As mentioned before, the preferred approach to exchanging large files between Salesforce.com and an SFTP server with FTPConnect is to configure an FTPConnect job. With this approach, you configure the job in the FTPConnect admin tab. In the job configuration, you must indicate if the job is a **Get** or **Put** job and you provide the name of a custom class which implements one of these two packaged Apex interfaces:

- `ftpconnect.PutFileProcessor`
- `ftpconnect.GetFileProcessor`

With this approach, you focus solely on the high level code that manages the format of the data in your SFTP files and manages the access to the database.

- For **Get** jobs, you are required to provide one method that parses one chunk of data from your SFTP and writes that chunk to some object(s) in your database.
 - Note that for getting CSV delimited files from SFTP, the package also provides the **ftpconnect.CsvGetFileProcessor** class.
 - Your custom class can extend this class rather than directly implement **ftpconnect.GetFileProcessor** to leverage features in that class that take care of record and field parsing.
- For **Put** jobs, you are required to provide two methods:
 - Define a SOQL query to select the data you want to send to SFTP
 - Define a method that formats records from one batch (meaning Salesforce.com Apex batch).
 - Note that with version 1.70 and later, the package now provides **ftpconnect.PutFileProcessorBase** class which allows you to define headers and footers that can be written to your SFTP file.
 - We recommend extending **PutFileProcessorBase** rather than directly implementing **PutFileProcessor** interface

As mentioned above, with **FTPConnect Jobs**, you focus solely on writing code to access your database objects and to handle the data format of the files sent or received to/from your SFTP server. Then FTPConnect takes care of the rest of the tedious details.

Below are detailed descriptions of the interfaces and/or classes you might choose to implement or extend.

ftpconnect.PutFileProcessorBase Class:

- While you can instead choose to have your class directly implement **ftpconnect.PutFileProcessor** interface, we instead recommend that you declare your custom class to extend **ftpconnect.PutFileprocessorBase**. Extend this class if your FTPConnect job configuration has “Put File” as the selected action.

- **global override Blob getData(Integer batchCount, List<Object>scope)** – Formats database records provided in scope argument and returns formatted data as a blob.
 - **Returns** – Formatted data.
 - **Parameters:**
 - **Integer batchCount** – Provides the count of the current batch being processed by the FTPConnect job.
 - **List<Object> scope** – A list of database records to be processed by the current batch.

- **global override String getQuery()** – Returns a valid SOQL query which specifies which records are to be processed and exported by the job.
 - **Returns** – A string containing a valid SOQL query.

- **Global override blob getFooter()** – returns a string that the job will write at the very end of the file. If you don't want to write a footer to your file, then do not include this method in your extension class.

- **Global override blob getHeader()** – returns a string that the job will write at the very start of the file. If you do not want to write a header to your file, then do not include this method in your extension class.

ftpconnect.GetFileProcessor:

When defining an FTPConnect job, you must provide a class that implements **ftpconnect.GetFileProcessor** if the action selected for the job is “**Get File**”. You can choose to declare your class to either directly implement **ftpconnect.GetFileProcessor** or if your file format is csv (or delimited with other special characters) then you can instead extend **ftpconnect.CsvGetFileProcessor**.

However, if your file has any other format, then your class should implement **ftpconnect.GetFileProcessor**.

The methods you need to implement for this interface are described below:

- **void processData(Integer batchCount)** – This method is called by the FTPConnect job every time it receives data from your file from the Staging/Relay service. You implement your business logic to processes a chunk of data received from the file on the SFTP site. This method must handle all parsing of data and handle writing the data to the Salesforce.com database.
 - Parameters:
 - **Integer batchCount** – The count of batches that have been processed.

- **void setData(String rawData)** – This method is called by the FTPConnect job when it receives data from the Staging/Relay service. This method prepares the data for processing by storing it in an instance variable that can later be accessed by **processData(Integer batchCount)**.
 - Parameters:
 - **String rawData** – Raw data that FTPConnect job obtained from the staging service.

Ftpconnect.CsvGetFileProcessor

Methods you override:

- **global override void processData(Integer batchSize)** – you must override this method to process a chunk of data sent back from the relay server. Use the `getRecord(...)` and `getField(...)` methods along with other helper methods to get records and fields; and to process the data as defined by your requirements. See description of specific helper methods in the next section below.
- **global virtual void init()** – optional. Use this to initialize settings using

CSV Helper Methods:

- **String getRecord()** – Reads the next record in the data chunk into a record buffer. It also returns the full record as a string.
- **String getField()** – Reads the next field in the record buffer.
- **String getField(String fieldName)** – Reads a field in the record buffer using the related field name.
- **Void setTestData()** – Initializes the data buffer with test data. This is convenient to use in your unit test methods. Use this in unit tests methods to setup test cases with data that reflects all of your use cases.
- **Void setDelimiter(String delimiter)** – Overrides the default delimiter (comma) with some other character. Can be called from your `init()` override method to specify the character used as delimiter in your input files.
- **Void setNameBasedFieldAccess(Boolean value)** – Turns on/off name based field mapping. Can be called from your `init()` override method to specify if your file provides a header for each of the fields in your records.
- **Void ignoreStartLines(Integer numberOfLinesToIgnore)** Tells processor to ignore some number of lines at the beginning of the import file.
- **Void writeErrorRecord(String errorMessage)** – Writes current record being processed to an error file. Appends error message to the record (separated using the record delimiter).
- **Void writeErrorRecord(Integer index, String errorMessage)** – Writes record from your batch that corresponds with the passed index value to an error file. Appends error message to the record (separated using the record delimiter).

FTPConnect Low Level API

FtpConnectErrorException Class

The FTPConnect API will throw **FtpConnectErrorException** for fatal exceptions related to communication with the FTPConnect Staging/Relay service and with FTPConnect configuration.

Methods:

- Global String **getFTPConnectErrorMessage()** – Gets a description of the error.

FtpConnect.FtpService Class

As described earlier, the `FtpService` class exposes the lowest level services that are available in `FtpConnect`. This API can be accessed from any Apex class such as page controllers or a custom batch class.

Constructors:

- **FtpService(String fsName)** – This is the preferred constructor for the **FtpService** class. Pass the name of a connection that was defined in the Connection detail page in FTPConnect Admin tab.
 - String **fsName** – Name of an FTPConnect connection that was defined using the FTPConnect admin tab.

- **FtpService(Id FileTransferServiceId)**
 - Id **FileTransferServiceId** – Salesforce record id of an FTPConnect connection that was defined using the FTPConnect admin tab.

- **FtpService(String svctype, String relayServerUrl, String ftpUrl, String ftpUserName, String ftpPassword, String filePathName, String encryptionKey, String decryptionKey, String encryptionAlgorithm)**

- This constructor can be used without doing any configuration. However, we highly recommend using the **FTPService(String fsName)** constructor rather than this one to make trouble-shooting and managing changes to your connection information easier.
 - String **svctype** – This should always be 'SFTP'.
 - String **relayServerUrl** – The production relay server url (https://ftpconnectprod.herokuapp.com/).
 - String **ftpUrl** – The domain name for your SFTP site (i.e. 'bytebuckets.com').
 - String **ftpUserName** – The user name to connect to your SFTP site.
 - String **ftpPassword** – The password to connect to your SFTP site.
 - String **filePathName** – The path of the file to access on the SFTP site.
 - String **encryptionKey** – The encryption key in PEM format.
 - String **decryptionKey** – The decryption key in PEM format.
 - String **encryptionAlgorithm** – The encryption algorithm. Valid options are "PGP", "AES", "Blowfish" and "DES".

Global Methods

- void **setFilePathName(String pathAndName)** – Sets the full file path and name of file to be accessed on the SFTP site.
 - Parameters
 - String **pathAndName** – Full path and name of a file.

- void **setSessionName(String sessionName)** – Sets the session name to shown in the session monitor.
 - **Parameters:**
 - String **sessionName** – The session name to appear in the session monitor.
 - String **getall()** – Submits a request to get the entire content of a file in a single request. This is not recommended if the file size exceeds 3MB. After calling **getall()**, use **getGetDataBlob()** or **getGetDataString()** to get the file content.
 - **Returns** – Unparsed XML string which will contains status values, error values and content of the requested file.
 - **String getrecordsstart()** – Submits a request to the Staging/Relay service to initialize a session to get a large file from an SFTP site. Subsequent **getRead(...)** method calls will retrieve groups of records from the file.
 - **Returns** – Unparsed XML string which will contains status values and error values.
 - **String getbytesstart()** – Submits a request to the Staging/Relay service to initialize a session to get a large file from an SFTP site. Subsequent **getRead(...)** method calls will retrieve chunks of data in terms of bytes.
 - **Returns** – Unparsed XML string which will contains status values and error values.
 - **String getread(integer numberOfUnits)** – Called after either **getrecordstart()** or **getbytesstart()** to get a chunk of content from the file.
 - **Returns** – Unparsed XML string which will contains status values, error values and content of the requested file.
 - Parameters:
 - **Integer numberOfUnits** – Specifies either the number of records or the number of bytes of content that should be retrieved from the staging service.

- **String putall(String content)** – Writes a string to a file on the SFTP site.
 - **Returns** – Unparsed XML string which will contains status values and error values.
 - Parameters:
 - **String content** – The data to write to the file on the SFTP site.
- **String putall(Blob contentBlob)** – Writes binary content in passed blob argument to the file on the SFTP site.
 - **Returns** – Unparsed XML string which will contains status values and error values.
 - Parameters:
 - **Blob contentBlob** – The data to write to the file on the SFTP site.
- **String putstart()** – Submits a request to the Staging/ Relay service to initialize a session to write a large file to an SFTP site. Subsequent **putwrite()** calls will send chunks of data to write to the file.
 - **Returns** – Unparsed XML string which contains status values and error values.
- **String putwrite(String content)** – Submits a request to the Staging/Relay service to stage a chunk of data to be eventually written to the file on the SFTP site.
 - **Returns** – Unparsed XML string which contains status values and error values.
 - Parameters:
 - **String content** – A chunk of data to stage for writing to the file on the SFTP site.
- **String putwrite(Blob content)** – Submits a request to the Staging/Relay service to stage a binary chunk of data to be eventually written to the file on the SFTP site.
 - **Returns** – Unparsed XML string which contains status values and error values.
 - **Parameters:**
 - **Blob content** – A chunk of data to stage for writing to the file on the SFTP site.
- **String putdone()** – Submits a request to the Staging/Relay service to indicate that there is no more data to be sent for the current session. This prompts the Staging/Relay service to assemble, encrypt and write all staged chunks of data to the file on the SFTP site.
 - **Returns** – Unparsed XML string which contains status values and error values.
- **String getStatusCode()** – Returns the status of a previous request to the Staging/Relay service.
 - **Returns** – status code
 - **-1** – indicates failure
 - **0** – indicates success
- **String getStatusDescription()** – Returns the status description of a previous request to the Staging/Relay service.
 - **Returns** – Description of the status.
- **String requestfiletransmitstatus()** –Submits a request to the Staging/Relay service to obtain the status of transmission of the resulting file to the SFTP server related to the current session. This method should be called after the **putdone()** request. Note that **putdone()** prompts the Relay/Staging service to write the file to the SFTP server. The writing of the file is executed asynchronously which implies **putdone()** returns without knowing if the file transmission is successful or not.
 - **Returns** –An xml response from the relay server that includes the status and a lot of other information.
- **String getfiletransmitstatus()** –Should be called immediately **after requestfiletransmitstatus()**. Returns the status of the file transmission. If pending is returned, then subsequent calls can be made until Success or Failure is returned.
 - **Returns** –‘Pending’, ‘Success’ or ‘Failure’
- **String getfiletransmiteerror()** –If the status is ‘Failure’, then this returns the error description for the related failure.
 - **Returns** – Description of file transmission failure.

- **String getFileStatus()** – Returns status of the file after a **getRead()** request.
 - **Returns** – Status of file after a **getRead()** request
 - **'EOF'** – Indicates you are done with reading the file. There is no more staged data remaining to read.
 - **'NOT_READY'** – The staging service has not yet staged the data from the file. (The staging of data after **getrecordsstart()** or **getbytesstart()** is performed asynchronously.) Ordinarily, you will repeat **getread()** until the **'NOT_READY'** status goes away.
 - **'MOREDATA'** – Indicates that there is more data from the file to get from the Staging/Relay service.
 - **'ERROR'** – An error was encountered reading or processing the file.

- **String getErrorCode()** – If the status indicates an error, then an error code is returned.
 - **Returns** – If **getStatusCode()** returns '-1', then **getErrorCode()** will return one of the following codes:
 - FtpService. ERROR_CODE_GENERAL_ERROR
 - FtpService. ERROR_CODE_TOO_MANY_SESSIONS

- **String getErrorDescription()** – Returns a description of the error.
 - **Returns** – Description of error from Staging/ Relay service.

- **String getGetDataString()** – Returns the content from the previous **getread()** call as a string.
 - **Returns** – String content from previous **getread()** call.

- **Blob getGetDataBlob()** – Returns the content from the previous **getread()** call as a string.
 - **Returns** – Binary content from previous **getread()** call.

FTPConnect Job Examples

FTPConnect Jobs is the quickest way to build an import or export process using an SFTP site. There are two steps:

1. Write a simple APEX class that implements either ftpconnect. GetFileProcessor or extends ftpconnect.PutFileProcessorBase.
2. Configure the job in the FTPConnect Admin tab.

Importing data Using FTPConnect Jobs

If the file to be imported from your SFTP site is a delimited file (with commas, tabs, pipes,...), then have your custom GetProcessor class extend **ftpconnect.CsvGetFileProceesor** rather than implement **ftpconnect.GetFileProcessor** and then leverage the rich library of methods that make it easy to access records and fields in your csv formatted data.

Some points about the example below that you should understand include:

- By default, the ftpconnect.CsvGetFileProceesor class assumes comma's are the delimiter and assumes the file does not have a header record with column names. You can override the init() method, as shown in this example, to use a different delimiter and to use the column names in the header record to map fields in your records.
- In this example, mapping of fields using column names is accomplished with the statement **setNameBasedFieldAccess(true)**. Because the file we want to read from the SFTP site uses the pipe symbol (“|”) for the delimiter, the statement **setDelimiter('|')** is included in the **init()** method.
- The **processData()** method, which you need to implement, focuses on the task of extracting records and fields from data mapping the values of these records and to sObject records to be inserted or updated in Salesforce.com database. It also takes care of any database access that is necessary to save the data.
- In this example, the data is getting loaded into the standard Account object. So there are statements for creating account records, assigning data from fields in the file to fields on the account record. Finally, there is a statement to insert all account records in a batch.
- In the processData() method shown in the example, the code calls getField(string) passing a string which contains the name for the field given in the header record. We can use the field name in this way because the file has a header record in it and because we included the setNameBasedFieldAccess(true) statement in the init() method.

- At runtime, an FTPConnect job calls out to the processData() method each time if receives a chunk of data from the file on the SFTP site. When you configure the job you will define how many records FTPConnect should pass to this method at one time. You should code this method in a manner that allows it to process each batch of records without reaching governor limits. Note that in this example, the code is bulkified. Rather than perform insert on each record, it adds each record to a list of SObjects and then executes insert on the list, thereby minimizing the number of inserts executed.

```
global class MyGetProcessor extends
ftpconnect.CsvGetFileProcessor {
    global override void init() {
        setDelimiter('|'); // override default delimiter (default is a
comma)
        //setNameBasedFieldAccess(false); //default is true
        //ignoreStartLines(3); //causes job to ignore first few
lines of file
    }
    global override void processData(Integer batchCount){
        String record = getRecord();
        List<Account> acctInsertList; = new List<Account>();
        while (record != null) {

            try {
                Account acct = new Account();

                acct.name = getField('Company Name');
                acct.BillingStreet = getField('Billing Street');
                acct.BillingCity = getField('Billing City');
                acct.BillingState = getField('Billing State');
                acct.BillingPostalCode = getField('Postal Code');
                acctInsertList.add(acct);
                record = getRecord();
            }
            catch (Exception e) {
                // will write last read record to error log with error
message
                //appended
                writeErrorMessage(e.getMessage());
            }
            recordIndex++;
        }

        Insert acctInsertList; // note that if apex throws
exception here,
// the job log will write all records in this
// batch to the error attachment on the job
log.
// the dml error will be appended to each
record.
    }
}
```



Create a Job in FTPConnect

Once you create a “Get Processor” class, you can create a job in FTPConnect from the FTPConnect Admin tab:

- Select **Jobs** from the **FTPConnect Admin** tab.

BusinessConnect f... Home * FTPConnect Admin X More

FTPConnect Administration
Home

- FTP Connections
- Settings
- Jobs**
- Encryption Keys
- Sessions
- License Management
- Session History and System
- Help and Support

Usage

- Click **New** to define a new job.

FTPConnect Administration
Jobs

New Jobs Monitor Admin Home

Action	File Transfer Job Name	Connection	File Path and Name	Number of Batches	Records Per Batch	Delete File

New Jobs Monitor Admin Home

On the Edit Job page, provide data for the following fields and then save:

FTPConnect Administration
Edit Job

Job Name	My Get Accounts Job
File Path and Name	/gptest.csv
Connection	TestConnection
Action	get file
Delete File Upon Completion	<input type="checkbox"/>
Number of Batches	100
Records Per Batch	100
Processor Class Name	MyGetProcessor
Next Job	
After Job Process Class	

- **Job Name** – Provide a descriptive name for the import job.
- **File Path and Name** – Provide the full path and name on the file. The path should begin with “/”. Note you can also use special character sequences to look for today’s date (date at the runtime of the job) embedded in the file.
 - {YYYY} – Embed the full year of the current date into the file path/name string.
 - {YY} – Embeds the last two digits of the year of the current date.
 - {MM} – Embeds the two digit numeric representations of month (Jan -> 01).
 - {DD} – Embeds the day of the month (will include leading 0 for days 1-9).
 - {HH} – Embeds the hour of the day (will include leading 0 for hours 0-23).
 - {MIN} – Embeds the minute of the hour (will include leading 0 for hours 0-59).
 - {SS} – Embeds the second of the minute (will include leading 0 for hours 0-59)NOTE: The wildcard character * is not supported.
- **Connection** – Select the connection (Which you created in the Connections area of the FTPConnect admin tab).
- **Action** – Choose “get file”.
- **Number of Batches** – You must estimate the number of batches in the file you are going to read. The number of batches is determined by the size of the file and the number of records in each batch. (see Records Per Batch field below). This will set up the batch size in the salesforce.com batch process. Take the number of records in your file, divide by your batch size and then add a safety buffer. It is ok to put in a number that is too big. You want the batch job to process all records in your file.
- **Records Per Batch** – The size you choose here should be based on the usage of governed resources in the processData() method that you created in your GetProcessor. If you execute one or more SOQL (select...) or DML (insert, update, delete) statements for each record, then you will need to choose a fairly small batch size. However, if you took care to Batchify your processData() method, then you can choose a relatively large number (hundreds). Also, you must take into consideration the amount of heap space each record uses. The total heap usage cannot exceed 3mb. Multiply the size of each record by the batch size to estimate the amount of heap storage that is needed for a given batch size.

- **Process Class Name** – Put in the exact name of your Get Processor class. In our example, it is “MyGetProcessor”.
- **Delete File Upon Completion** – Applies to data import jobs only. Check the box to make the job delete the file upon a successful completion.

Once saved, you can submit the job manually or you can schedule it to run at specific times daily, weekly, monthly and at a variety of other intervals. To simply run the job right now, click on the **Submit** link in the job list.

FTPConnect Administration
Jobs

Action	File Transfer Job Name	Connection	File Path and Name	Number of Batches	Records Per Batch	Delete File Upon Completion	Processor Class Name
Detail Del Submit	My Get Accounts Job	TestConnection	/gptest.csv	100	100	false	MyGetProcessor
Detail Del Submit	PutAllAccounts	NewTestService	/gptest.csv		100	false	MyPutProcessor
Detail Del Submit	Putjob1	NewTestService	/gptest1.txt		100	false	MyPutProcessor
Detail Del Submit	putjob2	NewTestService	/gptest2.csv		300	false	MyPutProcessor
Detail Del Submit	Test123	NewTestService	/gptest.csv	12	20	false	MyGetProcessor

A success message will appear if the job was successfully submitted. **THIS DOES NOT MEAN THAT THE JOB COMPLETED SUCCESSFULLY.** After it is submitted, click on the **Jobs Monitor** button to go to the **Apex Jobs** monitor and view the status of the job.

Apex Jobs

[Help for this Page](#)

Monitor the status of all Apex jobs, and optionally, abort jobs that are in progress.

View: All [Create New View](#)

Action	Submitted Date	Job Type	Status	Status Detail	Total Batches	Batches Processed	Failures	Submitted By	Completion Date	Apex Class	Apex Method	Apex Job ID
Abort	8/24/2020, 7:51 PM	Batch Apex	Processing		100	0	0	Kharg_Saona		GetBatch		7078g000020YoT1

By refreshing the Apex Jobs page, you can view progress (total batches, batches processed, status). You will also only see fatal errors on this page under the Status Detail column.

After the job is complete, you can view results in the **File Transfer Results** tab. By clicking on the detail record for a job, you can view processing details such as warnings and non-fatal errors.

File Transfer Job Re...	File Transfer Job	Header	Had Errors	Had Warni...	Session ID
<input type="checkbox"/> jobres-00001	job-00001	FTP Batch Status: Completed	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	a4c8a155-7eb3-4052-9632-a9991c31d9b2
<input type="checkbox"/> jobres-00002	job-00000	FTP Batch Status: Completed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ecac8cae-803f-40cc-baf8-ecae43de815c
<input type="checkbox"/> jobres-00003	job-00000	FTP Batch Status: Completed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	abedbdad-040e-4b0a-bba9-773f6d7e46d5
<input type="checkbox"/> jobres-00004	job-00001	FTP Batch Status: Completed	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	4045c85c-ad99-417b-8ee7-4c6d8c0947fa
<input type="checkbox"/> jobres-00005	job-00000	FTP Batch Status: Completed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	9fae55b0-fc4-4583-8d60-2c13c0d9f205
<input type="checkbox"/> jobres-00006	job-00001	FTP Batch Status: Completed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0de98e5a-1cc6-498c-831d-4f36f4c8e60c
<input type="checkbox"/> jobres-00007	job-00000	FTP Batch Status: Completed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	a2dc528b-4208-4d0e-818f-9f4978616123

In the list shown above and in the detail record below, the **Had Errors** checkbox indicates that a job had errors and the **Had Warnings** checkbox indicates that a job had warnings.

File Transfer Job Result
jobres-00004

Related **Details**

File Transfer Job Results Name
jobres-00004

File Transfer Job
[job-00001](#)

Header ⓘ
FTP Batch Status: Completed 

Session ID ⓘ
4045c85c-ad99-417b-8ee7-4c6d8c0947fa 

Had Errors ⓘ
 

Had Warnings ⓘ
 

File Transmission Status ⓘ 

Details about errors and warnings are stored in an attachment (**FileTransferResults...**). Additional details about specific records that cause errors are also stored in an attachment (**RecordsWithErrors...**).

In the **Related** tab, click on the attachment link to download it and view with WordPad.

File Transfer Job Result
jobres-00004

Related Details

 **Notes & Attachments (2)** Upload Files

 [FileTransferResults2020_5_21_9_39_49](#)
May 21, 2020 · Attachment

 [RecordsWithErrors__2020_5_21_9_39_49](#)
May 21, 2020 · Attachment

[View All](#)

Exporting Data with FTPConnect Jobs

FTPConnect Jobs can also be built to export data from Salesforce.com to an FTP site. The example below shows the simple code required to export a large amount of data to a file on an SFTP site using an FTPConnect job. The class extends `ftpconnect.PutFileProcessorBase`.

Minimally, you must override two simple methods:

- `getQuery()`.
- `getData()`.

First, override the `getQuery()` method. In `getQuery()`, your apex needs to return a string that defines a SOQL query that selects the database records you want to export.

Second, override the `getData()` method. In `getData()`, you write Apex code that formats the records you are exporting into the format needed for the output file. In the example that follows, the `getData()` method we define formats the records into output records that are delimited with a pipe (“|”) symbol. The `getData()` method also takes care of creating a header record.

Note how it checks the batch count before it adds a header record because we only want the header record to appear once at the top of the file rather than repeated with each batch.

Global Sharing Class:

```
global with sharing class MyPutProcessor extends ftpconnect.putFileProcessorBase {  
  
    public override String getQuery() {  
        return 'select id, name, billingStreet, billingCity, '  
            + 'billingPostalCode, billingState '  
            + 'from Account!';  
    }  
  
    public override Blob getData(Integer batchCount, List<SObject> scope) {  
        String dataBuffer = "";  
        if (batchCount == 0)  
            dataBuffer += 'ID|CompanyName|BillingStreet|BillingCity|BillingState|BillingPostalCode';  
  
        for (SObject obj : scope) {  
            Account acct = (Account) obj;  
  
            dataBuffer += '\n'  
                + acct.id + '|' +  
                + acct.name + '|' +  
                + acct.billingStreet + '|' +  
                + acct.billingCity + '|' +  
                + acct.billingState + '|' +  
                + acct.billingPostalCode;  
        }  
  
        return Blob.valueOf(dataBuffer);  
    }  
}
```

Public with Sharing class:

```
public with sharing class MyPutProcessor implements ftpconnect.PutFileProcessor {  
  
    public Blob getData(Integer batchCount, List<SObject> scope) {  
  
        String dataBuffer = "";  
  
        if (batchCount == 0)  
            dataBuffer += 'ID|CompanyName|BillingStreet|BillingCity|BillingState|BillingPostalCode';  
  
        for (SObject obj : scope) {  
            Account acct = (Account) obj;  
            dataBuffer += '\n'  
                + acct.id + '|' +  
                + acct.name + '|' +  
                + acct.billingStreet + '|' +  
                + acct.billingCity + '|' +  
                + acct.billingState + '|' +  
                + acct.billingPostalCode;  
        }  
  
        return Blob.valueOf(dataBuffer);  
    }  
  
    public String getQuery() {  
        return 'select id, name, billingStreet, billingCity, '  
            + '    billingPostalCode, billingState '  
            + 'from Account';  
    }  
}
```

You configure this export job in FTPConnect similarly to how the import job was configured. The required fields in the job definition are slightly different from the import job that we looked at earlier.

FTPConnect Administration

Edit Job

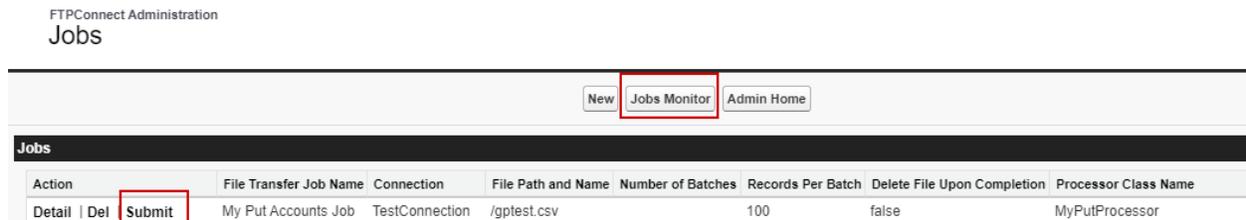
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	
Job Name	<input type="text" value="My Put Accounts Job"/>
File Path and Name	<input type="text" value="/gptest.csv"/>
Connection	<input type="text" value="TestConnection"/> 
Action	<input type="text" value="put file"/> 
Records Per Batch	<input type="text" value="100"/>
Processor Class Name	<input type="text" value="MyPutProcessor"/>
File Termination String	<input type="text" value="any string you want"/>
Do Not Create File When No Data	<input type="checkbox"/>
Next Job	<input type="text" value=""/> 
After Job Process Class	<input type="text" value=""/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

- **Job Name** – Provide a descriptive name for the export job.
 - **File and Path Name** – Provide the full path and name of the file. The path should begin with “/”. Note you can also use special character sequences to look for today’s date (date at the runtime of the job) embedded in the file.
 - {YYYY} – Embed the full year of the current date into the file path/name string.
 - {YY} – Embeds the last two digits of the year of the current date.
 - {MM} – Embeds the two digit numeric representations of month (Jan -> 01).
 - {DD} – Embeds the day of the month (will include leading 0 for days 1-9).
 - {HH} – Embeds the hour of the day (will include leading 0 for hours 0-23).
 - {MIN} – Embeds the minute of the hour (will include leading 0 for hours 0-59).
 - {SS} – Embeds the second of the minute (will include leading 0 for hours 0-59).
- NOTE: The wildcard character * is not supported.

- **Connection** – Select the connection (Which you created in the Connections area of the FTPConnect admin tab).
- **Action** – Choose “put file”
- **Records Per Batch** – Choose a size that allows your getData() to operate on the full batch without exhausting your governed Apex resources (memory, SOQL statements, DML statements).
- **Processor Class Name** – Put in the exact name of your Put Processor class.
- **File Termination String** – A fixed string that the job will write to the end of the export file.

Once your PutProcessor class is deployed and you have configured the Put Job that uses the new class, you can submit the job by clicking **Submit** from the **FTPConnect Jobs** list page or you can schedule the job.

After the page indicates that the job was successfully submitted with a message at the top, you can click the **Jobs Monitor** button to view go to Salesforce’s **Apex Jobs** page to view the progress.

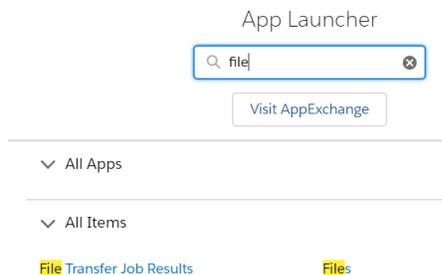


On the **Apex Jobs** page, refresh until you see that the **Status** field indicates the job **Completed**.



Review File Transfer Results after the job is completed:

- Go to the **File Transfer Job Results** tab to see if the job was successful.



- First, in the list you can immediately see if the job encountered errors by viewing the **Had Errors** checkbox.

File Transfer Job Results						
All						
15 Items · Sorted by Created Date · Filtered by all file transfer job results · Updated a few seconds ago						
	File Transfer Job Res...	File Transfer Job	Header	Had Errors	Had Warni...	Session ID
1	<input type="checkbox"/> jobres-00016	job-00005	FTP Batch Status: Completed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	15b0ee39-fa21-447e-b1d5-bd0d2852dd77

- To see more details, open up the detail record. For a put job, it is also very important to check the **File Transmission Status** field as shown below.

File Transfer Job Result
jobres-00016

Related **Details**

File Transfer Job Results Name
jobres-00016

File Transfer Job
[job-00005](#)

Header ⓘ
FTP Batch Status: Completed

Session ID ⓘ
15b0ee39-fa21-447e-b1d5-bd0d2852dd77

Had Errors ⓘ

Had Warnings ⓘ

File Transmission Status ⓘ
Success

- Upon completion of the FTPConnect PutBatch job, all data processed by your job is staged on the Relay Server, but final transmission of the data is queued on the relay server to be executed asynchronously.
- The Apex job before final completion queries the relay server for status of the transmission which may be either **'Pending'**, **'Success'** or **'Failure'**. If it is Pending, it re-queries a limited number of times.

Often, with smaller transmissions, the job knows the transmission status before it completes and the **File Transmission Status** field will indicate either Success or Failure. However, with bigger jobs that transmit a lot of data, the job may finish with **File Transmission Status** still pending. If you open **File Transmission Result** detail page and see that **File Transmission Status** is pending, you can click on the **Get File Transmit Status** button to invoke a request to the relay server for an updated status.

Get File Transmit Status

✓ **Success:**
File Transmission Status successfully obtained. Return to the Job Results detail page to view the status.

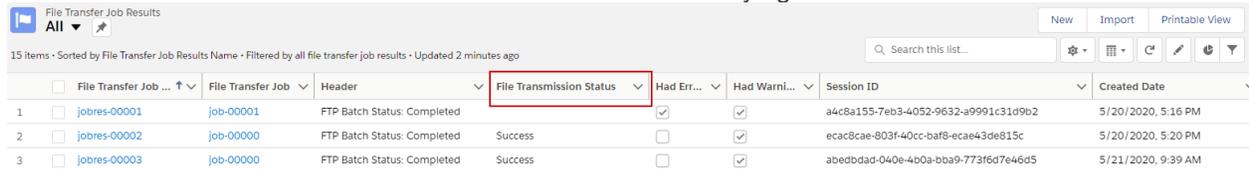
[Return To File Transfer Job Results](#) [Get File Transmit Status](#)

Scheduled FTPConnect Jobs

Further, if you have FTPConnect jobs that are scheduled to run on a regular basis, you can schedule `ftpconnect`.

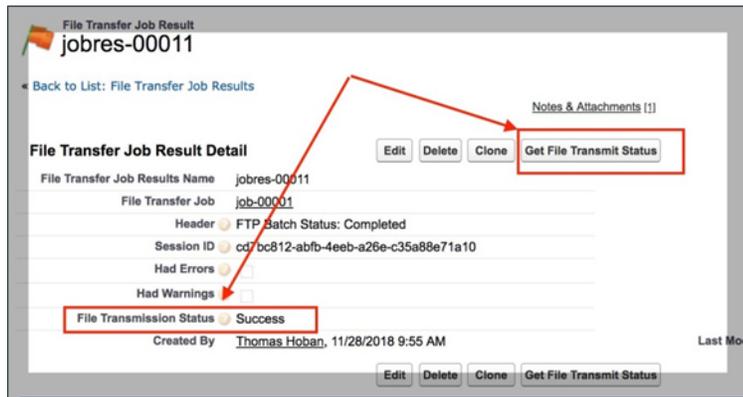
- **PutJobFileTransmissionChecker** - This is a schedulable Apex class that updates the status of all result records with pending file transmission status. Schedule it at intervals that make sense for your application.
- For automatic notifications of jobs that fail or have status other than Success, we recommend you setup standard Salesforce.com workflow on the **File Transfer Job Results** object to notify any user who needs to be aware of failures or problems.

Note that the **File Transfer Job Results** list view can be modified to include **File Transmission Status** as shown here. See Salesforce.com documentation for instructions on modifying tab views.



File Transfer Job ...	File Transfer Job	Header	File Transmission Status	Had Err...	Had Warni...	Session ID	Created Date	
1	jobres-00001	job-00001	FTP Batch Status: Completed	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	a4c8a155-7eb3-4052-9632-a9991c31d9b2	5/20/2020, 5:16 PM	
2	jobres-00002	job-00000	FTP Batch Status: Completed	Success	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ecac8cae-803f-40cc-baf8-ecae43de815c	5/20/2020, 5:20 PM
3	jobres-00003	job-00000	FTP Batch Status: Completed	Success	<input type="checkbox"/>	<input checked="" type="checkbox"/>	abedbdad-040e-4b0a-bba9-773f6d7e46d5	5/21/2020, 9:39 AM

Also note that you may need to edit the page layout of the **File Transfer Job Results** detail page to include the **Get File Transmit Status** button and the **File Transmission Status** field as show below. See Salesforce.com documentation for instructions on modifying page layouts.



Encrypt/Decrypt with FTPConnect

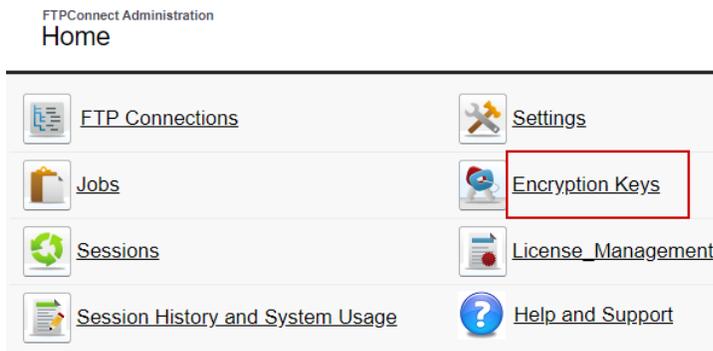
Incorporating encryption in your FTPConnect implementation is easy. FTPConnect supports several cryptography algorithms and even provides tools for generating encryption keys. The steps are easy:

- Install your encryption keys into FTPConnect or use FTPConnect to generate your keys.
- Assign an encryption algorithm and related encryption keys to your FTPConnect connection.

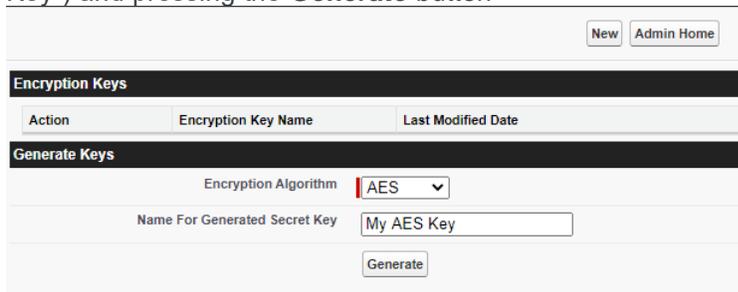
That's all! Once your connection is defined with an encryption algorithm and with valid encryption keys, then FTPConnect will encrypt outbound data (from salesforce.com to your SFTP site) and will decrypt inbound data (from your SFTP site to salesforce.com) on any file transfer using that connection.

For example, let's say the owner of the SFTP site where you need to place your data, agrees the data will be encrypted with AES using a key that you will provide him, then you can use FTPConnect to generate an encryption key as follows:

- Go to the FTPConnect Admin tab and select **Encryption Keys**



- Generate the key by selecting **AES** for the **Encryption Algorithm**, giving a name (for instance “My AES Key”) and pressing the **Generate** button



- If the key is successfully created, you will see success messages at the top of the page and you will see the key listed in your list of encryption keys. Click on the **Edit** link for the newly listed key to see the key string that was generated.

Success:
Key was generated.

New Admin Home

Encryption Keys

Action	Encryption Key Name	Last Modified Date
Edit Del	My AES Key	8/24/2020 10:32 PM

Generate Keys

Encryption Algorithm: AES

Name For Generated Secret Key: My AES Key

Generate

- You can cut and paste this key to share with the SFTP site owner but take care to get it to the owner in a safe and secure manner.

Save Cancel

Encryption Key Name: My AES Key

Key: SnVzw3IBcTLxt7Z+v4MpRXcrJM3hFU4BGcToQ5K4w8kkENCyqqnoQCixhUGJqlec

Save Cancel

- You may prefer to generate your key using a tool outside of FTPConnect or to use a key that the SFTP site owner shared with you. If this is the case, then just add the key to FTPConnect. Start by navigating to the main encryption key page.

FTPConnect Administration
Home

- FTP Connections
- Jobs
- Sessions
- Session History and System Usage
- Settings
- Encryption Keys**
- License Management
- Help and Support

- On the **Encryption Keys** Page click the **New** button to create a new key.

New Admin Home

Encryption Keys

Action	Encryption Key Name	Last Modified Date
Edit Del	My AES Key	8/24/2020 10:38 PM

Generate Keys

Encryption Algorithm: []

Generate

- Give the key a name, paste the key into the Key field and click **Save**. Notice the key itself is in a format called PEM. This format is required. The PEM format usually starts and ends with comments “-----BEGIN ...” and “-----END...” as shown here.

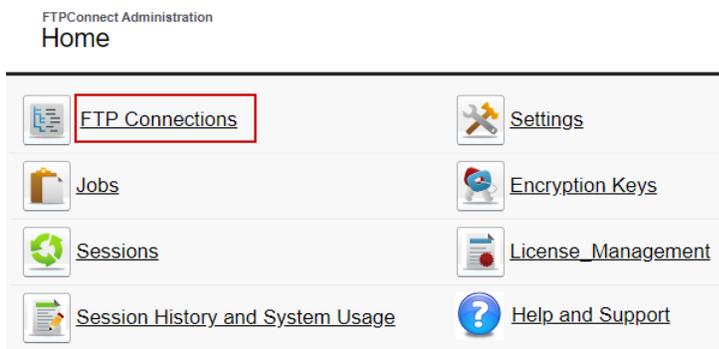
Save Cancel

Encryption Key Name

Key

```
-----BEGIN AES KEY-----
asGS3serv4ftaFTlhfsdDa57sgdstaASFas=
-----END AES KEY-----
```

- Once you have your keys, then you change your connections to use encryption. First navigate to your list of FTPConnect connections.



- Click **Edit** on the connection that you need to modify.

New Admin Home

Action	Connection Name	Service Type	URL
Edit Del Test	demo.wftpserver.com:2222	SFTP	demo.wftpserver.com:2
Edit Del Test	MyTestService	SFTP	uat.mft2.equifax.com
Edit Del Test	EncryptedSSHKey	SFTP	uat.mft2.equifax.com
Edit Del Test	NewTestService	SFTP	uat.mft2.equifax.com
Edit Del Test	EncryptedSSHKey	SFTP	uat.mft2.equifax.com
Edit Del Test	TestConnection	SFTP	uat.mft2.equifax.com

- On the detail page, select an encryption algorithm, select a key for encryption and select a key for decryption.

Encryption Settings

Encryption Algorithm  AES 

Encryption Key  

Decryption Key  

- Click **Save**

Manage FTPConnect Sessions

When conducting large data transfers with FTPConnect, the FTPConnect Staging/Relay service maintains a session. With every FTPConnect session that you start, a measured amount of system resources are allocated in the Staging/Relay service. In order to keep the usage of resources under control, a specific Salesforce.com instance is allowed a limited number of simultaneous sessions. When the limit is reached, you are no longer allowed to begin new sessions until your existing sessions have completed or have been cleared by you. When transferring a large file to an SFTP site, your salesforce.com application, using FTPConnect, initializes a session with Staging/Relay service by calling the **FtpService.putstart()** method. After the session is initialized, then the application sends data for that session to the Staging/Relay service by calling the **FtpService.putwrite()** method. The session does not end until your application calls the **FtpService.putdone()** method.

Similarly, when transferring a large file from an SFTP site, your salesforce.com application, using FTPConnect, initializes a session with the Staging/Relay service by calling either **FtpService.getrecordsstart()** or **FtpService.getbytesstart()**. It then pulls chunks of data from the Staging/Relay service using **FtpService.getread(...)**. The relay service then stages the file by reading it, decrypting it, breaking it up into manageable chunks and storing those chunks in a temporary holding area. Your ftpconnect application then calls the **getread()** method repeatedly until there are no more chunks of data to get from the Staging/Relay service. When the EOF status (see **FtpService.getFileStatus()**) is reached, the Staging/Relay service then clears the session.

If either type of session terminates prematurely, then unfinished sessions sit in limbo on the Staging/Relay service until someone clears them.

FTPConnect provides an interface you can use to view current sessions and to clear them if necessary. To view your existing sessions, navigate to the Sessions page.

FTPConnect Administration
Home

- FTP Connections
- Settings
- Jobs
- Encryption Keys
- Sessions**
- License Management
- Session History and System Usage
- Help and Support

On the Sessions page, you can view current sessions and clear old sessions that failed to complete.

Admin Home

Action	Session ID	Session Name	User Id	FTP URL	File Path
Clear Session	f252c172-5b7d-4934-b9dc-9cd9466b33d4		0056g000004HB76AAG	uat.mft2.equifax.com	/SFGUATtest.txt

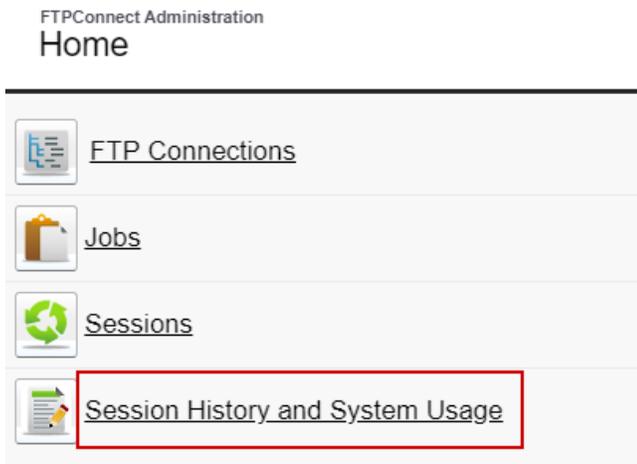
Admin Home



In order to keep the usage of resources under control, a specific Salesforce.com instance is allowed a limited number of simultaneous sessions.

Session History, License Terms and Usage

Your FTPConnect installation is subject limits on usage which vary with the type of license assigned to it. The limits apply to the number of sessions allowed per month, the number of bytes transferred per month and the amount of data transferred in a single session. To view your session history, the limits associated with the terms of your license, and the usage of resources against your limits, go to the FTPConnect Sessions History Page.



Usage and limits are displayed at the top of the page. The most recent history of your sessions are shown below the limits.

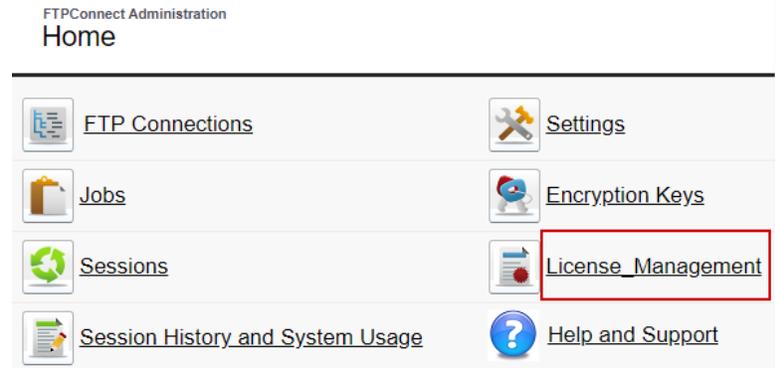
Usage for the Current Month						
Sessions Used	11	Bytes Used	286541			
Sessions Per Month Limit	200	Bytes Per Month Limit	100000000			

Processed Sessions (Most recent 20 sessions displayed)								
Session ID	Session Name	User Id	FTP URL	File Path	FTP Get Unit Type	Algorithm	File Size	Chunks M
15b0ee39-fa21-447e-b1d5-b09d2852d077	MyPutAccountsJob	0056g000004HB76AAG	uat.mft2.equifax.com	/gpctest.csv			109297	16
2b07900c-49f7-4a07-bbd1-		0056g000004HB76AAG	uat.mft2.equifax.com	/test.out			142	0

Managing Sandbox License

You can install the FTPConnect managed package on any Salesforce.com instance and try it out for 30 days free of charge. After 30 days, your instance must have a production licensed to continue using FTPConnect.

From the **FTPConnect Admin** tab, you can navigate to the License page to view the FTPConnect licensing status.

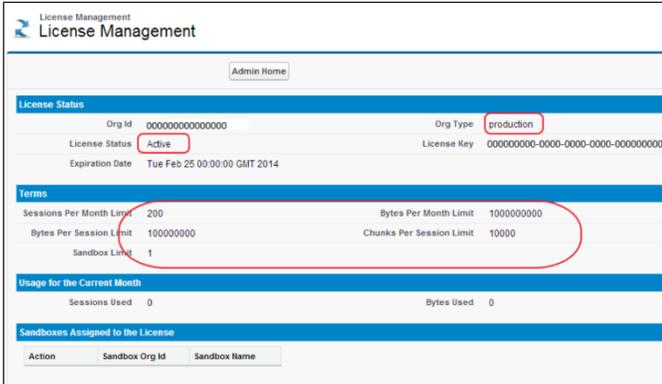


When first installed on a production/sandbox instance, the instance's license status is 'Trial' as shown below.

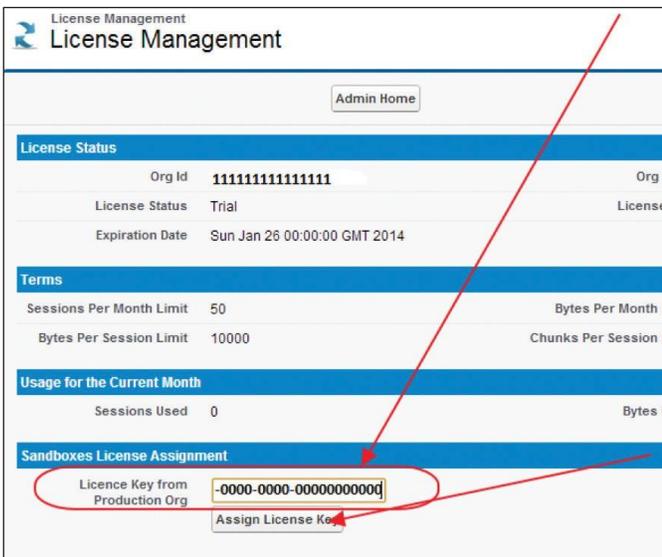
License Management	
Admin Home	
License Status	
Org Id	Org Type production
License Status Trial	License Key
Expiration Date	
Terms	
Sessions Per Month Limit	100
Bytes Per Month Limit	10000000
Bytes Per Session Limit	1000000
Chunks Per Session Limit	5000
Simultaneous Sessions	3
Sandbox Limit	0
Usage for the Current Month	
Sessions Used	0
Bytes Used	0
Sandboxes Assigned to the License	
Action	Sandbox Org Id
	Sandbox Name

Managing Sandbox License (continued)

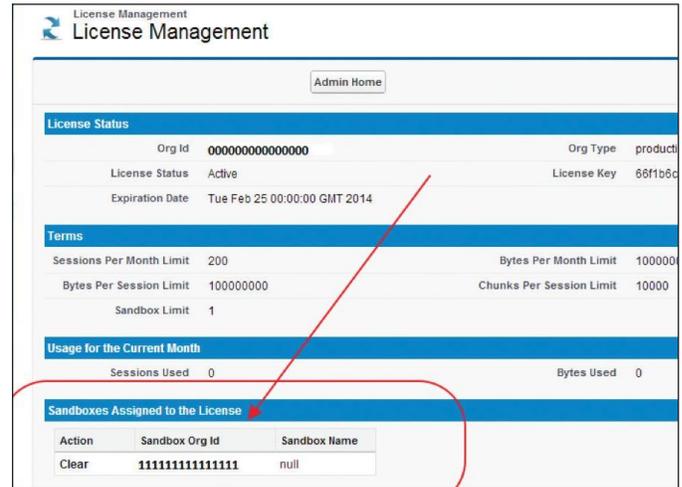
After purchasing a license, on your Salesforce.com production instance, the license page will indicate that the license status is active. Also, notice that the limits are increased.



On the production instance page shown above, notice that the sandbox limit is currently 1, but in the **Sandboxes Assigned to the License** list at the bottom of the page, we can see that there are no sandboxes using the license. To assign the license to a sandbox, take the license key, shown in the page above, and enter it into the License page on the sandbox as shown below.



Now, back on the production instance, you can see that a sandbox is using one of your licenses.



Notice that on the page above, that the production administrator has the ability to de-assign the license by clicking **Clear**.





Troubleshooting and Debugging

Common sources of problems and errors include:

- Incorrect configuration of FTPConnect Staging/Relay Service URL.
- Incorrect configuration of an FTPConnect connection.
- Incorrect configuration of an FTPConnect job.
- Encryption/Decryption issues:
 - Wrong encryption algorithm
 - Wrong or invalid key
 - Decrypting unencrypted data.
 - Neglecting to decrypt encrypted data.
- Hanging sessions.
- Apex programming errors.

Common Troubleshooting steps:

- **Test your connections:**
 - For almost any problem, the first item to check is the connection. If your job or program cannot connect to your SFTP site, then nothing works. Use the connect test page (discussed earlier in this document) to confirm that you can send and receive data to and from your SFTP site.
 - To get additional debug information on the test page, you can turn on debug mode. Go to the FTPConnect Admin tab. Select the Settings link. Select “**Debug**” in the Setting Name drop down list. Type “**True**” in the Value field. Click the Add Setting button.
- **Test encryption and decryption:**
 - After configuring your connection with an encryption algorithm and keys, transport a simple one line file to the SFTP site using the connection test page. Confirm with the party consuming your data that they are able to decrypt your file using their key. On the reverse, have the SFTP site owner place a small encrypted file on the SFTP site. Then use the test page to get that file from the site. Confirm that the data read from the site is readable.
- **For FtpConnect jobs, verify the following:**
 - Check status of job in the Apex Jobs Monitor (Setup->Monitoring->Apex Jobs). Sometimes errors will be shown under the “**Status**” column.
 - If the Status indicates the job completed, check the **Sessions** Page. If the job completed with no errors, then there should be no pending sessions. If the job failed and left a pending session showing on this page, then the page might also show an error status. After viewing the error status, clear the session.
 - Check the **Sessions History and Usage** page. If the Staging/Relay Service encountered errors, then they will be shown here.
 - Check the **File Transfer Job Results** tab. This is where the FTPConnect Job batch class writes all error and log information that it traps.
- **Test and debug your Apex code:**
 - First, use the tools that the force.com platform gives you.

- Write unit tests to assure you code performs as expected.
- Use the Apex Debug Logs (Setup->Monitoring->Debug Logs).
- If you have a class that extends **ftpconnect**.
- **CsvGetFileProcessor**, include try/catch blocks to catch and report exceptions that might be thrown by ftpconnect layers.
- If using the **FtpService api**, catch and report FtpErrorException.
- If using the FtpService api, check the results and status of all service requests. See **getStatusCode()**, **getStatusDescription()**, **getFileStatus()**, **getErrorCode()** and **getErrorDescription()**. Also see use of these methods in sample classes.

By default, the FTPConnect managed package connects to our production relay server using this endpoint:

<https://ftp.ftpconnectrelay.com/>

However, in some cases working in troubleshooting issues, our support team might ask you to override the default endpoint with another endpoint. This is done by creating an entry for the **RELAY_SERVER_URL** setting on the settings page:

- Go to the FTPConnect Admin tab.
- Click on the Settings link.
- Select **RELAY_SERVER_URL** in the Setting Name dropdown list.
- In the Setting Value field, type or paste this URL setting:
<https://ftp.ftpconnectrelay.com/>
- Click the **Add Setting** button.

