FAXCOM *Anywhere* Web Services

July 2011 Revised Edition
# Introduction to FAXCOM Anywhere Web Services

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>4</td>
</tr>
<tr>
<td>Audience</td>
<td>4</td>
</tr>
<tr>
<td>Using Parameters with SOAP</td>
<td>4</td>
</tr>
<tr>
<td>Sample SOAP Request</td>
<td>5</td>
</tr>
<tr>
<td>SOAP Response</td>
<td>5</td>
</tr>
</tbody>
</table>

## Developer's Guide

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>6</td>
</tr>
<tr>
<td>Java Walk Through</td>
<td>6</td>
</tr>
<tr>
<td>C# Walk Through</td>
<td>12</td>
</tr>
<tr>
<td>Python Walk Through</td>
<td>17</td>
</tr>
</tbody>
</table>

## API Reference

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>22</td>
</tr>
<tr>
<td>Response</td>
<td>23</td>
</tr>
<tr>
<td>Login</td>
<td>27</td>
</tr>
<tr>
<td>ReleaseSession</td>
<td>27</td>
</tr>
<tr>
<td>NewFaxMessage, NewFaxMessageEx</td>
<td>28</td>
</tr>
<tr>
<td>AddRecipient, AddRecipientEx</td>
<td>28</td>
</tr>
<tr>
<td>AddAttachment</td>
<td>29</td>
</tr>
<tr>
<td>SendFax, SendFaxEx</td>
<td>29</td>
</tr>
<tr>
<td>GetPendingMessageByUniqueID</td>
<td>30</td>
</tr>
<tr>
<td>GetPendingMessages, GetPendingMessagesEx</td>
<td>30</td>
</tr>
<tr>
<td>GetPendingMessagesCount</td>
<td>31</td>
</tr>
<tr>
<td>GetMessageStatusByUniqueID, GetMessageStatusByUniqueIDEx</td>
<td>31</td>
</tr>
<tr>
<td>GetMessageStatuses, GetMessageStatusEx</td>
<td>32</td>
</tr>
<tr>
<td>GetMessageStatusesCount</td>
<td>32</td>
</tr>
<tr>
<td>RemoveMessageStatusByUniqueID</td>
<td>33</td>
</tr>
<tr>
<td>GetActivityRecords</td>
<td>33</td>
</tr>
<tr>
<td>GetActivityRecordsCount</td>
<td>34</td>
</tr>
<tr>
<td>GetTxStatusForJob</td>
<td>34</td>
</tr>
</tbody>
</table>
Introduction to FAXCOM Anywhere Web Services

FAXCOM Anywhere Web Services is a simple web services interface that gives the capability of Office Account users the ability to submit faxes securely using any programming language that can consume web services. All web service requests support the SOAP message protocol over an HTTP connection. The easiest way to use the SOAP requests is to use the convenience classes that we give you for making those calls. The convenience classes are written in C#, Java and Python. Users in a FAXCOM Anywhere Office account, after authenticating using their username and password, can create a fax message with a cover page, a memo and add attachments and send it to multiple recipients. FAXCOM Anywhere Web Services comes with a WSDL (Web Services Description Language) file located at http://webfax.faxcomanywhere.com/fawebservice/fawebservice.asmx?WSDL. Users can now leverage the full power and scalability of FAXCOM Anywhere servers for faxing by using the web services interface for their applications.

Security
To access the web services in a secure manner, use https://webfax.faxcomanywhere.com/fawebservice/fawebservice.asmx. If you use the SDK, you will automatically access the web service through SSL.

Audience
This guide is intended for developers who are building web-based applications which send faxes, and getting the statuses of those faxes. This guide assumes that you are familiar with either Java or C#, but this is not required to use the interface. It also assumes that you have a valid FAXCOM Anywhere Office account that is web enabled. If you do not have an account, please speak to one our FAXCOM Anywhere sales representatives and an account can be created for you.

Using Parameters with SOAP
The API Reference in this guide describes the parameters and their values for each operation of the web interface. You can view the WSDL file directly to see how the request parameters appear in the XML that your toolkit generates. You can also see how your toolkit makes the operations available to your application code. The convenience classes make all this available to you in a class library. No knowledge of SOAP is required.
Sample SOAP Request
The following sample is the XML for a SOAP message that calls the NewFaxMessage operation. The convenience classes contain the nuts and bolts to make such a request with the proper headers.

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <NewFaxMessage xmlns="http://www.faxcompanywhere.com/">
      <priority>1</priority>
      <sendTime>0.0</sendTime>
      <resolution>1</resolution>
      <subject></subject>
      <memo/></memo>
      <senderName></senderName>
      <senderFax></senderFax>
      <recipientName>John Doe</recipientName>
      <recipientCompany>Global Travel</recipientCompany>
      <recipientFax>5085551212</recipientFax>
      <recipientVoice></recipientVoice>
      <recipientAccount></recipientAccount>
    </NewFaxMessage>
  </soap:Body>
</soap:Envelope>
```

SOAP Response
Following is the sample response.

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <NewFaxMessageResponse xmlns="http://www.faxcompanywhere.com/">
      <NewFaxMessageResult>
        <Result>true</Result>
        <Detail>Fax message submitted for delivery. UniqueJobID: WEBFAX-MAMB-01_WS_0901072025200001</Detail>
        <Type>INFO</Type>
        <Data>WEBFAX-MAMB-01_WS_0901072025200001</Data>
      </NewFaxMessageResult>
    </NewFaxMessageResponse>
  </soap:Body>
</soap:Envelope>
```
Developer’s Guide

Introduction
The quickest way to start programming with FAXCOM Anywhere Web Services is to start with the Java and C# convenience classes. These classes wrap the stubs generated by the WSDL, making it easier for the developer to start creating, sending, and checking for faxes. All connections are done securely using SSL. Use the “Walk throughs” to get started with your language preference.

Java Walk Through
If you don’t already have a FAXCOM Anywhere account call a FAXCOM Anywhere sales rep at (800) 477-2472. Make sure that the account is web enabled and create a user with the username and password you will be using for web services. After receiving the Web Services package from the download the fa-webservice.jar and put it in your class path. The class libraries for the Java integration come from stubs generated by Axis. You will therefore need to have the Axis libraries in your class path as well. You can find the Axis releases here: http://ws.apache.org/axis/java/releases.html. You also need the JavaBeans Activation Framework (http://java.sun.com/javase/technologies/desktop/javabean/jaf/downloads/index.html) and the JavaMail API (http://java.sun.com/products/javamail/) in your class path. The convenience classes and the stub classes generated by Axis from the WSDL are in the fa-webservice.jar. Add the following imports to your Java program.

```java
import java.io.*;
import java.util.*;
import java.text.*;
import com.faxcomanywhere.www.SimpleClient.*;
```

All calls to the Web Service start with the Client class. The login method takes the username and password of a user in your Office account as parameters.

```java
// See if we can log in
Client client = new Client();
ResultMessage output = client.login("john", "letmein");
if (!output.isResult()) {
    System.out.println("Incorrect login credentials");
    return;
} else {
    System.out.println("Logged in with correct credentials (Version " + client.getVersion() + ")");
}
```
// OK. We are in. Start making a fax.
Fax fax = client.getFaxObject();

If the login is successful, you can obtain a Fax object from the Client class factory. The Fax object handles creating the fax message, the recipients, the attachment to be sent, and it keeps track of the status of the fax as it gets posted to FAXCOM Anywhere until its final delivery. Once we have the Fax object we can specify the recipients with the Recipient class and add them.

// Create the recipients
Recipient john = new Recipient();
john.setName("John Smith");
john.setFax("5559012307");
john.setCompany("Global Solutions");
Recipient michael = new Recipient();
michael.setName("Michael Doe");
michael.setFax("5559012307");
michael.setCompany("Island Hopper");
// Add recipients
fax.addRecipient(john);
fax.addRecipient(michael);

You can then use the Fax object to specify additional information such as the subject, including a memo, specifying the TSI, the sender’s name, the sender’s fax number, and which cover page to use.

// Specify a header, subject, a memo, the TSI, the sender name // and sender's fax number. Specify the cover page.
fax.setSubject("");  
fax.setHeader("");  
fax.setMemo("");  
fax.setTsi("Global Fax");  
fax.setSenderName("");  
fax.setSenderFax("");  
fax.setCoverPage(Fax.NO_COVER_PAGE);

There are two ways you can add an attachment to the fax message. You can create an AttachmentFile object by specifying a path to a file that you want to add as an attachment. Then add it as an attachment by calling the Fax.addAttachment() method. Check for any errors by examining the ResultMessage.

// Add an attachment using a file.
Attachment attachment1 = new AttachmentFile("C:\temp\attachment1.pdf");
output = fax.addAttachment(attachment1);
if (!output.isResult()) {
    System.out.println("Could not add attachment " + 
                      output.getDetail());
    client.release();
    return;
}
You can also specify an InputStream to use for the source of the contents of the attachment and use an AttachmentStream object. To use a stream as a source for the attachment just pass the stream to the constructor as a parameter. The stream can be any subclass of InputStream. Then add it the same way an AttachmentFile object is added, but this time give the attachment a name so we can determine the file type. Any name with a valid file extension will do.

```
// Try adding an attachment using a file stream.
FileInputStream in = null;
try {
in = new FileInputStream("C:\temp\attachment2.pdf");
} catch (Exception e) {
    System.out.println("Could not add attachment " + output.getDetail());
    client.release();
    return;
}
Attachment attachment2 = new AttachmentStream(in);
// Need to give it a name with extension to set the file type.
attachment2.setName("attachment2.pdf");
output = fax.addAttachment(attachment2);
if (!output.isResult()) {
    System.out.println("Could not add attachment " + output.getDetail());
    client.release();
    return;
}
```

The Fax object allows you to set the priority, the type of delay, and scheduled time for the fax as specified in the Web Service API. In the sample code, normal priority and immediate faxing are specified. To set the schedule a time for the fax to go out, create a java.util.Date object that has the date and time you want the fax to go out. Then call the sendFax() method to send the fax.

```
// Set the time and priority of the fax.
fax.setPriority(Fax.PRIORITY_NORMAL);
/* To set send immediately */
fax.setDelay(Fax.SCHEDULE_IMMEDIATE);
/* To send fax at an off peak time (after 12:00 AM midnight) */
// fax.setDelay(Fax.SCHEDULE_OFF_PEAK);
/* To send fax at a specific time */
// fax.setDelay(Fax.SCHEDULE_DELAY);
// Calendar cal = new GregorianCalendar();
// cal.set(2009, Calendar.JANUARY, 14, 11, 15, 0);
// fax.setSendTime(cal.getTime());
output = fax.sendFax();
```

Check the ResultMessage to see if the fax got posted. If it is true then the message got posted on our servers and faxes to all the recipients will be sent. The following sample code loops through checking statuses until all possible faxes have been sent.

```
SimpleDateFormat df = new SimpleDateFormat("yyyy'/'MM'/'dd hh':'mm':'ss aa");
while (!fax.isAllSent()) {
```
```
fax.lookForStatusesEx();
ArrayList<Recipient> recipients = fax.getRecipients();
for (Recipient recipient : recipients) {
    GenericStatus status = recipient.getStatusRecord();
    String start = status.getStartTime() == null ? "" : df.format(status.getStartTime());
    String current = status.getTime() == null ? "" : df.format(status.getTime());
    System.out.println("Status UniqueJobID: "+ status.getUniqueJobID());
    System.out.println("Fax: " + recipient.getFax());
    System.out.println("Start: " + start);
    System.out.println("Time: " + current);
    System.out.println("Action Taken: "+ status.getActionCode());
    System.out.println("StatusText: "+ status.getStatusText());
    System.out.println("Text: "+ status.getText());
    System.out.println("Name: "+ status.getName());
    System.out.println("ErrorCode: "+ status.getErrorCode());
    System.out.println("Attempts: "+ status.getAttempts());
    System.out.println("Pages: "+ status.getPagesTransmitted());
    System.out.println("IsInTransit: "+ status.isInTransit());
    System.out.println("ID Tag: "+ status.getIdTag());
    System.out.println("Should Resend: "+ status.shouldResend());
    System.out.println();
}
try {
    Thread.sleep(2000);
}
catch (InterruptedException e) {} 
```

Each recipient in the fax contains the status of the fax going to him. Each `GenericStatus` can have the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>NOT_POSTED. The Job did not go to our servers.</td>
</tr>
<tr>
<td>-1</td>
<td>FAILED. The transmission failed to reach the recipient.</td>
</tr>
<tr>
<td>0</td>
<td>PREPARING. Preparation for submission to our servers has begun.</td>
</tr>
<tr>
<td>1</td>
<td>POSTED. Our servers received the job.</td>
</tr>
<tr>
<td>2</td>
<td>QUEUED. Our servers put the job in queue for transmission.</td>
</tr>
<tr>
<td>3</td>
<td>SENT. The job was sent to the recipient.</td>
</tr>
<tr>
<td>4</td>
<td>SERVER. The job is at the server.</td>
</tr>
</tbody>
</table>
Please see the appendix for a description of all possible error codes and their descriptions. Once the session is done release the session. If `GenericStatus.shouldResend()` is true then the message can be sent again with the possibility of not encountering the same error.

```java
// We are all done.
client.release();
```

The following example shows how we can access faxes sent to a user.

```java
String myMailBoxPath = "C:\\Users\\John Doe\\Documents\\FAXCOM Anywhere\\MailBox";
// See if we can login
Client client = new Client();
ResultMessage output = client.login("john", "letmein", myMailBoxPath);
if (!output.isResult())
{
    System.out.println("Incorrect login credentials");
    return;
}
else
{
    System.out.println("Logged in with correct credentials (Version " + client.getVersion() + ")");
}
```

Start by logging in with a third parameter specifying the location of your mailbox.

```java
MailBox myMailBox = client.getMailBoxObject();
System.out.println("Getting inbox count...");
int count = myMailBox.getCount();
System.out.println("Number of faxes stored online: " + count);
System.out.println("Down loading faxes...");
```

Then get a `MailBox` object. You can get the number of faxes in your mail box from the `MailBox` object.

```java
output = myMailBox.downloadReceivedFaxes(false);
if (!output.isResult())
{
    System.out.println("Could not download faxes " + output.getDetail());
}
else
{
    System.out.println("Faxes are in the store.");
}
for (ReceivedFax rx : myMailBox.getReceivedFaxesInStore())
{
    System.out.println(rx.getPathID() + " " + rx.getActualCallerID() + " " + rx.getImagePath() + " " + rx.getReceiveTime() + " " + rx.getStatusName());
}
```
You can now download the faxes to your local mail box. The Boolean value in the `downloadReceivedFaxes` method determines whether the faxes in the online mail box should be deleted after the download. Otherwise the faxes will remain online for a specified amount of time. Afterwards, you can look at the meta data associated with the faxes by going through the `ReceivedFaxesInStore` object. Check the sample code for how to access your mail box using time periods.

If you need to look up the status of fax at any time, and check its history you can do the following.

```java
Client client = new Client();
ResultMessage output = client.login("johndoe", "letmein");
if (!output.isResult()) {
    System.out.println("Incorrect login credentials");
    return;
} else {
    System.out.println("Logged in with correct credentials (Version " + client.getVersion() + ")");
}
String uniqueJobID = "ARLEF-DMSWE-01_WX_0906092106000234";
int recipientIndex = 0;
GenericStatus status = client.getTransmitObject(uniqueJobID, recipientIndex);
System.out.println("Status UniqueJobID: " + status.getUniqueJobID());
System.out.println("Time: " + status.getTime());
System.out.println("StatusText: " + status.getStatusText());
System.out.println("Text: " + status.getText());
System.out.println("Name: " + status.getName());
System.out.println("ErrorCode: " + status.getErrorCode());
System.out.println("Attempts: " + status.getAttempts());
System.out.println("IsInTransit: " + status.isInTransit());
List<Action> actions = status.getTransmitActions();
if (actions != null && actions.size() > 0) {
    for (Action action : actions) {
        System.out.println("Date " + action.getDate() + ", Attempt " + action.getAttemptNo() + ", Type " + action.getType() + ", Description " + action.getDescription() + ", Status " + action.getStatusName() + ", Pages " + action.getPages() + ", Connect Time " + action.getConnectTime() + ", CSI " + action.getCsi() + ", Number " + action.getDialedNumber());
    }
}
client.release();
```
You can specify the `uniqueJobID` and get a status from the `getTransmitObject` class factory. The status also contains a history of the actions that took place to transmit the fax or a history of actions taken they led up to the failure to transmit the fax.

**C# Walk Through**

If you don’t already have a FAXCOM Anywhere account call a FAXCOM Anywhere sales rep at (800) 477-2472. Make sure that the account is web enabled and create a user with the username and password you will be using for web services. After receiving the Web Services package from the sales rep, run the `Setup.msi` file. This will install the C# libraries to access the web services. In your C# project add a reference and use the browse tab to the browse to the `C:\Program Files\Biscom\FAXCOM Anywhere WS Library\FAXCOMAnywhereWS.dll` or the path you selected in the install. Add the following code to your project.

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.IO;
using FAXCOMAnywhereWS;
using FAXCOMAnywhereWS.com.faxcomanywhere.webfax;

These are the references to the convenience classes that use the Web Services and the FAXCOM Anywhere Web Services themselves. All calls to the Web Service start with the `Client` class. The `login` method takes the `username` and `password` of a user in your Office account as parameters.

```csharp
// See if we can login
Client client = new Client();
ResultMessage output = client.LogIn("john", "letmein");
if (!output.Result)
{
    Console.WriteLine("Incorrect login credentials");
    return;
} else
{
    Console.WriteLine("Logged in with correct credentials");
}
// OK. We are in. Start making a fax.
Fax fax = client.FaxObject;
```

If the login is successful, you can obtain a `Fax` object from the `Client` class factory. The `Fax` object handles creating the fax message, the recipients, the attachment to be sent, and it keeps track of the status of the fax as it gets posted to FAXCOM Anywhere until its final delivery. Once we have the `Fax` object we can specify the recipients with the `Recipient` class and add them.
// Create the recipients
Recipient john = new Recipient();
john.Name = "John Smith";
john.Fax = "5559012307";
john.Company = "Global Solutions";
Recipient michael = new Recipient();
michael.Name = "Michael Doe";
michael.Fax = "5559012307";
michael.Company = "Island Hopper";
// Add recipients
fax.AddRecipient(john);
fax.AddRecipient(michael);

You can then use the Fax object to specify additional information such as the subject, including a memo, specifying the TSI, the sender’s name, the sender’s fax number, and which cover page to use.

// Specify a header, subject, a memo, the TSI, the sender name // and sender's fax number. Specify the cover page.
fax.Header = "";
fax.Subject = "";
fax.Memo = "";
fax.Tsi = "Global Fax";
fax.SenderName = "";
fax.SenderFax = "";
fax.CoverPage = Fax.NO_COVER_PAGE;

There are two ways you can add an attachment to the fax message. You can create an AttachmentFile object by specifying a path to a file that you want to add as an attachment. Then add it as an attachment by calling the Fax.AddAttachment() method. Check for any errors by examining the ResultMessage.

// Add the attachment.
IAttachment attachment1 = new AttachmentFile(@"C:\temp\tst.pdf");
output = fax.AddAttachment(attachment1);
if (!output.Result) {
    Console.WriteLine("Failed to attach: " + output.Detail);
    client.Release();
    return;
}

You can also specify an Stream to use for the source of the contents of the attachment and use an AttachmentStream object. To use a stream as a source for the attachment just pass the stream to the constructor as a parameter. The stream can be any subclass of Stream. Then add it the same way an AttachmentFile object is added, but this time give the attachment a name so we can determine the file type. Any name with a valid file extension will do.

// Try adding an attachment using a file stream.
using (FileStream fs = File.OpenRead(@"C:\temp\att2.pdf")) {
    IAttachment attachment2 = new AttachmentStream(fs);
    // Need to give it a name with extension to set the file type.
    attachment2.Name = "att2.pdf";
The `Fax` object allows you to set the priority, the type of delay, and scheduled time for the fax as specified in the Web Service API. In the sample code, normal priority and immediate faxing are specified. To set the schedule a time for the fax to go out, create a `System.DateTime` object that has the date and time you want the fax to go out. Then call the `SendFax()` method to send the fax.

```csharp
// Set the time and priority of the fax.
fax.Priority = Fax.PRIORITY_NORMAL;
/* To set send immediately */
fax.Delay = Fax.SCHEDULE_IMMEDIATE;
/* To send fax at an off peak time (after 12:00 AM midnight) */
// fax.Delay = Fax.SCHEDULE_OFF_PEAK;
/* To send at a specific time */
// fax.Delay = Fax.SCHEDULE_DELAY;
// fax.SendTime = new DateTime(2009, 8, 23);
output = fax.SendFax();
```

Check the `ResultMessage` to see if the fax got posted. If it is true then the message got posted on our servers and faxes to all the recipients will be sent. The following sample code loops through checking statuses until all possible faxes have been sent.

```csharp
while (!fax.IsAllSent()) {
    fax.LookForStatusesEx();
    List<Recipient> recipients = fax.Recipients;
    foreach (Recipient recipient in recipients) {
        GenericStatus status = recipient.StatusRecord;
        Console.WriteLine("Status UniqueJobID: \0\nFax: \1\nStart: \2\nTime: \3\nAction Taken: \4\nStatusText: \5\nText: \6\nName: \7\nErrorCode: \8\nAttempts: \9\nPages: \10\nIsInTransit: \11\nID Tag: \12\nShould Resend: \13\n\n", status.UniqueJobID, recipient.Fax, start, current, status.ActionCode, status.StatusText, status.Text, status.Name, status.ErrorCode, status.Attempts, status.PagesTransmitted, status.IsInTransit, status.IDTag, status.ShouldResend);
    }
    Thread.Sleep(2000);
}
```
Each recipient in the fax contains the status of the fax going to him. Each `GenericStatus` can have the following values:

-2 NOT_POSTED. The Job did not go to our servers.
-1 FAILED. The transmission failed to reach the recipient.
0 PREPARING. Preparation for submission to our servers has begun.
1 POSTED. Our servers have received the job.
2 QUEUED. Our servers have put the job in queue for transmission.
3 SENT. The job was sent to the recipient.
4 SERVER. The job is at the server.

To look up statuses for any particular job use the following.

```csharp
string uniqueJobID = "XXXXXXXX-XXXX-XXXX_0905201837040002";
int recipientIndex = 0;
GenericStatus status = client.TransmitObject(uniqueJobID, recipientIndex);
List<Action> actions = status.TransmitActions;
if (actions != null && actions.Count > 0)
{
    foreach (Action action in actions)
    {
        Console.WriteLine("Date {0}, Attempt {1}, Type {2}, Description {3}, Status {4}, Pages {5}, Connect time {6}, CSI {7}, Number {8}",
                       action.Date, action.AttemptNo, action.Type, action.Description, action.StatusName, action.Pages, action.ConnectTime, action.Csi, action.DialedNumber);
    }
}
```

This will give you a complete history of the transmit process.

Please see the appendix for a description of all possible error codes and their descriptions. Once the session is done release the session. If `GenericStatus.ShouldResend` is true then the message can be sent again with the possibility of not encountering the same error.

```csharp
// We are done.
client.Release();
```

The following example shows how we can access faxes sent to a user.

```csharp
string myMailBoxPath = @"C:\Users\John Doe\Documents\FAXCOM Anywhere\MailBox";
```
// See if we can login
Client client = new Client();
ResultMessage output = client.LogIn("john", "letmein", myMailBoxPath);
if (!output.Result)
{
    Console.WriteLine("Incorrect login credentials");
    return;
} else
{
    Console.WriteLine("Logged in with correct credentials (Version {0})", client.Version);
}

Start by logging in with a third parameter specifying the location of your mailbox.

MailBox myMailBox = client.MailBoxObject;
Console.WriteLine("Getting inbox count...");
int count = myMailBox.GetCount();
Console.WriteLine("Number of faxes stored online: " + count);

Then get a MailBox object. You can get the number of faxes in your mail box from the MailBox object.

Console.WriteLine("Down loading faxes...");
output = myMailBox.DownloadReceivedFaxes(false);
if (!output.Result)
{
    Console.WriteLine("Could not download faxes " + output.Detail);
} else
{
    Console.WriteLine("Faxes are in the store.");
}
foreach (ReceivedFax rx in myMailBox.ReceivedFaxesInStore)
{
    Console.WriteLine(rx.PathID + " " + rx.ActualCallerID + " " + rx.ImagePath + " " + rx.ReceiveTime + " " + rx.StatusName);
}
client.Release();

You can now download the faxes to your local mail box. The Boolean value in the DownloadReceivedFaxes method determines whether the faxes in the online mail box should be deleted after the download. Otherwise the faxes will remain online for a specified amount of time. Afterwards, you can look at the meta data associated with the faxes by going through the ReceivedFaxesInStore object. Check the sample code for how to access your mail box using time periods.
**Python Walk Through**

If you don’t already have a FAXCOM Anywhere account call a FAXCOM Anywhere sales rep at (800) 477-2472. Make sure that the account is web enabled and create a user with the username and password you will be using for web services. After receiving the Web Services package from the sales rep, download the Python.zip file and extract the files into your Python directory. The Python WSDL client stub modules for FAXCOM Anywhere were generated using ZSI 2.1 (http://sourceforge.net/projects/pywebsvcs/files/). You will need to download and ZSI to use them.

After you download and install ZSI make sure you run `python setup.py install`.

Add the following code to your project.

```python
import sys
from datetime import datetime
from Client import *
from AttachmentFile import *
from AttachmentStream import *
from Recipient import *
from Fax import *
from GenericStatus import *

These are the references to the convenience classes that use the Web Services and the FAXCOM Anywhere Web Services themselves. All calls to the Web Service start with the `Client` class. The `login` method takes the `username` and `password` of a user in your Office account as parameters.

```python
# # Attempt to login
# client = Client()

# # Show the version of the SDK we are using
# print 'Using version ' + client.getVersion() + ' of the SDK'

output = client.logIn('your_username', 'your_password')

if output.isResult():
    print 'Login successful'
else:
    print 'Login unsuccessful: ' + output.getMessage()
    sys.exit()

# OK. We are in. Start making a fax.
fax = client.getFaxObject()
```
If the login is successful, you can obtain a Fax object from the Client class factory. The Fax object handles creating the fax message, the recipients, the attachment to be sent, and it keeps track of the status of the fax as it gets posted to FAXCOM Anywhere until its final delivery. Once we have the Fax object we can specify the recipients with the Recipient class and add them.

```python
# Create the recipients
john = Recipient()
john.setName('John Smith')
john.setFax('5559012307')
john.setCompany('Global Solutions')
michael = Recipient()
michael.setName('Michael Doe')
michael.setFax('5559012308')
michael.setCompany('Island Hopper')
# Add recipients
fax.addRecipient(john)
fax.addRecipient(michael)
```

You can then use the Fax object to specify additional information such as the subject, including a memo, specifying the TSI, the sender’s name, the sender’s fax number, and which cover page to use.

```python
# Specify a header, subject, a memo, the TSI, the sender name
# and sender's fax number. Specify the cover page.
fax.setHeader('This Is My Header')
fax.setMemo('The memo')
# fax.setSubject('Q1 Results')
fax.setTsi('Global Fax')
fax.setSenderName('')
fax.setSenderFax('')
# Use this if you don't want a cover page.
# fax.setCoverPage(Fax.NO_COVER_PAGE)
# Use this is you want FAXCOM Anywhere's default cover page.
# fax.setCoverPage(Fax.DEFAULT_COVER_PAGE)
fax.setCoverPage('your_coverpage.cpg')
```

There are two ways you can add an attachment to the fax message. You can create an AttachmentFile object by specifying a path to a file that you want to add as an attachment. Then add it as an attachment by calling the Fax.addAttachment() method. Check for any errors by examining the Output.

```python
# Add an attachment using a file.
attachment1 = AttachmentFile('C:\temp\attachment1.pdf')
output = fax.addAttachment(attachment1)
if not output.isResult():
    print 'Could not add attachment ' + output.getMessage()
    client.release()
sys.exit()
```

You can also specify an existing memory buffer to use for the source of the contents of the attachment and use an AttachmentStream object. To use a memory buffer as a source for the attachment just pass the buffer to the constructor as a parameter. Then add it the same way an AttachmentFile object is added, but this time give the attachment a name so we can determine the file type. Any name with a valid file extension will do.
# Try adding an attachment using a buffer.
f = None
try:
f = open('C:\temp\attachment2.pdf', 'rb')
buffer = f.read()
except IOError as (errno, strerror):
    print failurePoint + ' I/O error({0}): {1}'.format(errno, strerror)
client.release()
sys.exit()
finally:
    if not f is None:
        f.close()
attachment2 = AttachmentStream(buffer)
# Need to give it a name with extension to set the file type.
attachment2.setName('attachment2.pdf')
output = fax.addAttachment(attachment2)
if not output.isResult():
    print 'Could not add attachment ' + output.getMessage()
    client.release()
sys.exit()

The Fax object allows you to set the priority, the type of delay, and scheduled time for the fax as specified in the Web Service API. In the sample code, normal priority and immediate faxing are specified. To set the schedule a time for the fax to go out, create a datetime object that has the date and time you want the fax to go out. Then call the sendFax() method to send the fax.

# Set the time and priority of the fax.
fax.setPriority(Fax.PRIORITY_NORMAL)
# To set send immediately
fax.setDelay(Fax.SCHEDULE_IMMEDIATE)
# To send fax at an off peak time (after 12:00 AM midnight)
# fax.setDelay(Fax.SCHEDULE_OFF_PEAK)
# To send at a specific time
# fax.setDelay(Fax.SCHEDULE_DELAY)
# mytime = datetime(2010,3,31,10,30,0)
# fax.setSendTime(mytime)
output = fax.sendFax()

Check the Output.getResult() to see if the fax got posted. If it is True then the message got posted on our servers and faxes to all the recipients will be sent. The following sample code loops through checking statuses until all possible faxes have been sent.

    while not fax.isAllSent():
        fax.lookForStatusesEx()
        recipients = fax.getRecipients()
        for recipient in recipients:
            # This is a GenericStatus
            status = recipient.getStatusRecord()
            if status.getStartTime() is None:
                start = ''
            else:
                start = status.getStartTime().strftime('%m/%d/%Y %I:%M:%S %p')
            if status.getTime() is None:
                current = ''
else:
    current = status.getTime().strftime('%m/%d/%Y %I:%M:%S %p')
    print 'Status UniqueJobID: ' + status.getUniqueJobID()
    print 'Fax: ' + recipient.getFax()
    print 'Start: ' + start
    print 'Time: ' + current
    print 'Action Taken: ' + status.getActionCode()
    print 'StatusText: ' + status.getStatusText()
    print 'Text: ' + status.getText()
    print 'Name: ' + status.getName()
    print 'ErrorCode: %d' % (status.getErrorCode(),)
    print 'Attempts: %d' % (status.getAttempts(),)
    print 'Pages: %d' % (status.getPagesTransmitted(),)
    print 'IsInTransit: ' + str(status.isInTransit())
    print 'ID Tag: ' + status.getIdTag()
    print 'Should Resend: ' + str(status.shouldResend())
    print

    time.sleep(2)

Each recipient in the fax contains the status of the fax going to him. Each GenericStatus can have the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>NOT_POSTED. The Job did not go to our servers.</td>
</tr>
<tr>
<td>-1</td>
<td>FAILED. The transmission failed to reach the recipient.</td>
</tr>
<tr>
<td>0</td>
<td>PREPARING. Preparation for submission to our servers has begun.</td>
</tr>
<tr>
<td>1</td>
<td>POSTED. Our servers have received the job.</td>
</tr>
<tr>
<td>2</td>
<td>QUEUED. Our servers have put the job in queue for transmission.</td>
</tr>
<tr>
<td>3</td>
<td>SENT. The job was sent to the recipient.</td>
</tr>
<tr>
<td>4</td>
<td>SERVER. The job is at the server.</td>
</tr>
</tbody>
</table>

To look up statuses for any particular job use the following.

```python
gleak = 'ZZZZYX-333Y-XX_YY_1004021254170083'
recipientIndex = 0
# Get a GenericStatus object
status = client.getTransmitObject(uniqueJobID, recipientIndex)
print 'Status UniqueJobID: ' + status.getUniqueJobID()
print 'Time: ' + status.getTime().strftime('%m/%d/%Y %I:%M:%S %p')
print 'StatusText: ' + status.getStatusText()
print 'Text: ' + status.getText()
print 'Name: ' + status.getName()
print 'ErrorCode: %d' % (status.getErrorCode(),)
print 'Attempts: %d' % (status.getAttempts(),)
print 'IsInTransit: ' + str(status.isInTransit())
print```
Associated actions taken for job:

```python
actions = status.getTransmitActions()
if not actions is None and len(actions) > 0:
    for action in actions:
        print '===='
        print 'Date ' + action.getDate().strftime('%m/%d/%Y %I:%M:%S %p') + ', Attempt %d' % (action.getAttemptNo(),) + ', Type ' + action.getType()
        print 'Description ' + action.getDescription() + ', Status ' + action.getStatusName() + ', Pages %d' % (action.getPages(),)
        print 'Connect Time %d' % (action.getConnectTime(),) + ', CSI ' + action.getCsi() + ', Number ' + action.getDialedNumber()
        print '===='
```

This will give you a complete history of the transmit process.

Please see the appendix for a description of all possible error codes and their descriptions. Once the session is done release the session. If `GenericStatus.shouldResend` is true then the message can be sent again with the possibility of not encountering the same error.

```python
# Always release the session when you are done.
#
output = client.release()
```

The following example shows how we can access faxes sent to a user.

```python
import sys
from Client import *
from MailBox import *
from Output import *

myMailBoxPath = 'C:\\Users\\John Smith\\Documents\\FAXCOM Anywhere\\MailBox'

# See if we can login
client = Client()
output = client.logInWithMailBox('my_username', 'my_password', myMailBoxPath)
if not output.isResult():
    print 'Incorrect login credentials: ' + output.getMessage()
    sys.exit()
else:
    print 'Logged in with correct credentials (Version ' + client.getVersion() + ')
```
Start by logging in with a third parameter specifying the location of your mailbox.

```python
myMailBox = client.getMailBoxObject()
print 'Getting inbox count...' 
count = myMailBox.getCount()
print 'Number of faxes stored online: %d' % (count,)
print 'Downloading faxes...' 
```

Then get a MailBox object. You can get the number of faxes in your mail box from the MailBox object.

```python
output = myMailBox.downloadReceivedFaxes(False)
if not output.isResult():
    print 'Could not download faxes ' + output.getMessage()
else:
    print 'Faxes are in the store.'

for rx in myMailBox.getReceivedFaxesInStore():
    callerID = rx.getActualCallerID()
    if callerID is None:
        callerID = ''
    print rx.getPathID() + ' ' + callerID + ' ' + rx.getImagePath() + ' ' +
    rx.getReceiveTime().strftime('%m/%d/%Y %I:%M:%S %p') + ' ' +
    rx.getStatusName()
```

You can now download the faxes to your local mail box. The Boolean value in the 
downloadReceivedFaxes method determines whether the faxes in the online mail box should be 
deleted after the download. Otherwise the faxes will remain online for a specified amount of time. 
 Afterwards, you can look at the meta data associated with the faxes by going through the 
ReceivedFaxesInStore object. Check the sample code for how to access your mail box using time 
periods. The first two samples deal with sending faxes and the last three samples deal with receiving 
 faxes.

**API Reference**

**Parameters**

This section describes the parameters used by the API.

**Priority**

There are four levels of priority. Each level is an integer value. Low is 0. Normal is 1. High is 2. Urgent is 3. Generally, there shouldn’t be any need to go above normal priority. (Please be a good NET citizen and use Low or Normal priority.)
Scheduling
Faxes can be sent immediately, off peak, or delayed. To send a fax immediately set the scheduling time to a string equal to “0.0.” To send a fax off peak, set the scheduling time to a string equal to “1.0.” (Off peak faxes are sent after midnight.) To schedule a fax for particular time set the scheduling time using GMT in this format: MM/dd/yyyy hh:mm:ss tt where MM is the month, dd is the day of the month, yyyy is the year, hh is the hour in a twelve hour clock, mm is the minutes, ss is the seconds and tt is either AM or PM.

For example, in order to send a fax at 5:30 PM EST on December 22, 2008, set the schedule time to "12/22/2008 10:30:00 PM." (EST is 5 hours behind GMT.)

You can specify the correct time in Java using the SimpleDateFormat class with “MM’/’dd’/’yyyy hh’:'mm' : 'ss aa” as the string in the constructor. To set the string value int the GMT time zone use the setTimeZone method as follows
SimpleDateFormat.setTimeZone(TimeZone.getTimeZone("GMT")) and then format the string.

In C# you can set a DateTime value to GMT in the correct format using
DateTime.ToUniversalTime().ToString("MM’/’dd’/’yyyy hh’:'mm' : 'ss tt").

Resolution
The fax resolution can either be high (equal to 1) or low (equal to 0). The default is 1. High resolution faxes have twice as many scan lines in the vertical direction. Unless there is some special reason to use the low resolution, we recommend that you just stick to using the high resolution setting. Our fax servers are capable of using the highest available baud rate for fax transmission.

Cover Page
In order to fax using a cover page, specify its file name. These cover pages were uploaded by the account the account manager. They have file extensions of “.CPG.” Specify “” for the default cover page, and “(none)” for no cover page.

TSI
You can set the Transmitting Subscriber Identification to let the receiving fax machine know where the fax is coming from.

Response

ResultMessage
This is the return value of most of the API function calls. The ResultMessage returns whether the operation succeeded, a description of any errors if they occurred, and any data associated with the operation upon successful completion.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td>True for success; false for failure.</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td><strong>Detail</strong></td>
<td>A description of the failure.</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>Data associated with the result.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>The type of message: INFO, WARNING or ERROR.</td>
</tr>
</tbody>
</table>

### RxStatus

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActualCallerID</td>
<td>The received message Caller ID.</td>
</tr>
<tr>
<td>BaudRate</td>
<td>The baud rate of the transmission.</td>
</tr>
<tr>
<td>CallerID</td>
<td>The received fax caller identification string. This is the same as the TSi if the caller used a TSi.</td>
</tr>
<tr>
<td>CompletionTime</td>
<td>The completion time the fax was received.</td>
</tr>
<tr>
<td>ConnectTime</td>
<td>The telephone connect time in seconds.</td>
</tr>
<tr>
<td>DID</td>
<td>The caller's DID string.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Returns the size of the faxed image in Kbytes.</td>
</tr>
<tr>
<td>ID</td>
<td>The received fax ID.</td>
</tr>
<tr>
<td>NumberOfPages</td>
<td>The number of pages in the received fax.</td>
</tr>
<tr>
<td>ReceiveTime</td>
<td>The date and time when the fax was received.</td>
</tr>
<tr>
<td>Status</td>
<td>The error status of the received fax.</td>
</tr>
<tr>
<td>StatusName</td>
<td>A four letter alphanumeric code of the status.</td>
</tr>
<tr>
<td>StatusText</td>
<td>A description of the error status.</td>
</tr>
<tr>
<td>TSi</td>
<td>The transmitting station ID.</td>
</tr>
<tr>
<td>Image</td>
<td>A Base64 encoding of the image of the fax. This will be null if the loadImage flag is set to false.</td>
</tr>
<tr>
<td>ImageFileName</td>
<td>The file name of the image file.</td>
</tr>
</tbody>
</table>

### TxStatus

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueJobID</td>
<td>The unique ID obtained from the SendFax or SendFaxEx operation.</td>
</tr>
<tr>
<td>RecipientIndex</td>
<td>The recipient index of the transmission. The recipient index is 0 based. The recipient index represents the order in which the recipient was added to the message before transmission.</td>
</tr>
<tr>
<td>Attempt</td>
<td>The number of the attempt in which this action occurred.</td>
</tr>
<tr>
<td>ActionType</td>
<td>A descriptive code for the type of action taken in the process of sending the fax.</td>
</tr>
<tr>
<td>ActionDate</td>
<td>The date and time the action occurred in UTC.</td>
</tr>
<tr>
<td>ActionDescription</td>
<td>A description of the action taken.</td>
</tr>
<tr>
<td>StatusValue</td>
<td>A four letter name of the status code. This is the status associated with a particular action.</td>
</tr>
<tr>
<td>StatusDescription</td>
<td>A description of the transmission status code.</td>
</tr>
<tr>
<td>Csi</td>
<td>The caller subscriber ID. Not blank if the action taken results in a successful fax and the CSI is</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ConnectTime</td>
<td>The connection time for the fax transmission in seconds. Greater than 0, if the action taken results in a successful fax.</td>
</tr>
<tr>
<td>Pages</td>
<td>The total number of pages in the fax image. Greater than 0, if the action taken results in a successful fax.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>The fax telephone number. Not blank if the action taken results in a successful fax.</td>
</tr>
</tbody>
</table>

**PendingMessage**

This is the response for any operation that involves getting a pending message. A pending message identifies whether a fax is in queue to be sent or retried if an attempt to send it has been made.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>The message job ID.</td>
</tr>
<tr>
<td>AssignedID</td>
<td>Optional ID assigned by the creator of the fax message.</td>
</tr>
<tr>
<td>DeliveryTime</td>
<td>The delivery time of the fax if the fax was scheduled.</td>
</tr>
<tr>
<td>Status</td>
<td>This is a flag with the following bit values: if bit 0 is set it is on hold, if bit 1 is set it is active and if bit 5 is set it is deleted.</td>
</tr>
<tr>
<td>HoldValue</td>
<td>If 0 it is on hold, if 1 it is active, and if 5 it is deleted.</td>
</tr>
<tr>
<td>Priority</td>
<td>The priority of the fax: low, normal, high or urgent.</td>
</tr>
<tr>
<td>Attempts</td>
<td>The number of attempts already made to fax the recipient.</td>
</tr>
<tr>
<td>FaxNumber</td>
<td>The fax number of the recipient.</td>
</tr>
<tr>
<td>SenderName</td>
<td>The name of the sender.</td>
</tr>
<tr>
<td>IsSelected</td>
<td>N/A</td>
</tr>
<tr>
<td>PreviewAllowed</td>
<td>N/A</td>
</tr>
<tr>
<td>AuthorizeAllowed</td>
<td>N/A</td>
</tr>
<tr>
<td>PreviewPath</td>
<td>N/A</td>
</tr>
<tr>
<td>IsSender</td>
<td>N/A</td>
</tr>
<tr>
<td>IsAuthorizor</td>
<td>N/A</td>
</tr>
<tr>
<td>IDTag</td>
<td>The IDTag set in the fax message when the message was created with NewFaxMessage or NewFaxMessageEx.</td>
</tr>
<tr>
<td>UniqueID</td>
<td>The unique ID returned by SendFax or SendFaxEx.</td>
</tr>
<tr>
<td>RecipientIndex</td>
<td>The recipient index.</td>
</tr>
</tbody>
</table>
**MessageStatus**

The **MessageStatus** is a record that keeps information about the outgoing fax. Use the unique job ID and the recipient index to retrieve information about a particular outgoing fax when the transmission has ended.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ID of the message.</td>
</tr>
<tr>
<td>TransmitTime</td>
<td>The date and time the fax was transmitted.</td>
</tr>
<tr>
<td>FaxNumber</td>
<td>The fax telephone number.</td>
</tr>
<tr>
<td>NumberOfAttempts</td>
<td>The number of transmission attempts.</td>
</tr>
<tr>
<td>RecipientName</td>
<td>The name of the recipient.</td>
</tr>
<tr>
<td>Subject</td>
<td>The subject of the fax.</td>
</tr>
<tr>
<td>PagesTransmitted</td>
<td>The number of pages in the fax.</td>
</tr>
<tr>
<td>TransferRate</td>
<td>The transmission speed of the last page in baud.</td>
</tr>
<tr>
<td>FallBack</td>
<td>If true, the transmission speed fell back to a lower rate.</td>
</tr>
<tr>
<td>TransmissionStatus</td>
<td>The status code of the fax transmission. 0 indicates success.</td>
</tr>
<tr>
<td>StatusText</td>
<td>A description of the transmission status code.</td>
</tr>
<tr>
<td>StatusName</td>
<td>A four letter name of the status code.</td>
</tr>
<tr>
<td>ConnectTime</td>
<td>The connection time for the fax transmission in seconds.</td>
</tr>
<tr>
<td>CSI</td>
<td>The caller subscriber ID.</td>
</tr>
<tr>
<td>PortNumber</td>
<td>N/A</td>
</tr>
<tr>
<td>TotalPages</td>
<td>The total number of pages in the fax image.</td>
</tr>
<tr>
<td>AttachmentCount</td>
<td>The number of attachments included in the fax.</td>
</tr>
<tr>
<td>EnvelopeName</td>
<td>N/A</td>
</tr>
<tr>
<td>Index</td>
<td>The recipient index of the transmission. The recipient index is 0 based.</td>
</tr>
<tr>
<td>IDTag</td>
<td>A user supplied ID for the fax transmission.</td>
</tr>
<tr>
<td>AssignedID</td>
<td>N/A</td>
</tr>
<tr>
<td>UniqueJobID</td>
<td>The unique ID obtained from the SendFax or SendFaxEx operation.</td>
</tr>
<tr>
<td>IsSelected</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Activity**

The **Activity** record contains information about an outgoing fax. Unlike the MessageStatus record, these records are transient. Only a certain number of them can be retained and records are eliminated, starting with the oldest.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastError</td>
<td>N/A</td>
</tr>
<tr>
<td>ID</td>
<td>The fax job identification.</td>
</tr>
</tbody>
</table>
### DateTime
The date and time the transmission was completed.

### Type
The type of fax, either Tx for transmission or Rx for receive.

### Retries
The number of retries to send the fax.

### FaxNumber
The recipient’s fax number.

### Sender
The name of the sender.

### TransmissionStatus
Status code.

### StatusText
A description of the status code.

### ConnectTime
The connection time to transmit the fax in seconds.

### TotalPages
The total pages in the fax image.

### TransferRate
The baud rate for the transmission.

### FinalRetry
True if the retry attempt is the last attempt to send the fax.

### AuthorizeGroup
N/A

### Authorizer
N/A

## Login

### Description
Before you can start faxing you need to start a valid session. Simply call this procedure and supply the username and password of a user in your Office account. Make sure you maintain session support for all other calls. When you are done with everything, you should call `ReleaseSession`.

### Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>the username of the user in the Office account</td>
</tr>
<tr>
<td>Password</td>
<td>the password of the user in the Office account</td>
</tr>
</tbody>
</table>

### Response Element
See `ResultMessage`.

## ReleaseSession

### Description
After logging in and doing whatever fax operations that are needed, call this method to end the session.

### Response Element
See `ResultMessage`. 
**NewFaxMessage, NewFaxMessageEx**

**Description**

NewFaxMessage and NewFaxMessageEx create a new message for faxing. More recipients and attachments can be added later, but this is the starting point for creating a fax. A successful return means that the fax will go out to all the recipients. A status will be returned for each recipient, which determines if the fax has reached its destination.

**Request Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>The priority of the fax message (see above).</td>
</tr>
<tr>
<td>Schedule</td>
<td>When to send the fax message (see above).</td>
</tr>
<tr>
<td>Resolution</td>
<td>Set the resolution of the fax (see above).</td>
</tr>
<tr>
<td>Subject</td>
<td>The subject of the fax (appears on cover page).</td>
</tr>
<tr>
<td>Coverpage</td>
<td>The .CPG name of the cover page.</td>
</tr>
<tr>
<td>Memo</td>
<td>The memo of the fax (appears on cover page).</td>
</tr>
<tr>
<td>SenderName</td>
<td>The name of the sender (appears on cover page).</td>
</tr>
<tr>
<td>SenderFax</td>
<td>The fax number of the sender (appears on cover page).</td>
</tr>
<tr>
<td>RecipientName</td>
<td>The name of the recipient.</td>
</tr>
<tr>
<td>RecipientCompany</td>
<td>The name of the company of the recipient.</td>
</tr>
<tr>
<td>RecipientFax</td>
<td>The fax number to send this message.</td>
</tr>
<tr>
<td>RecipientVoice</td>
<td>The telephone number of the recipient.</td>
</tr>
<tr>
<td>RecipientAccount</td>
<td>The account number of the recipient (information purposes only).</td>
</tr>
<tr>
<td>Tsi</td>
<td>The TSI of the fax (only in the NewFaxMessageEx call).</td>
</tr>
</tbody>
</table>

**Response Element**

See ResultMessage.

**AddRecipient, AddRecipientEx**

**Description**

After creating a fax message with NewFaxMessage or NewFaxMessageEx more recipients can be added to the message with this operation.

**Request Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the recipient.</td>
</tr>
<tr>
<td>Company</td>
<td>Company of the recipient.</td>
</tr>
<tr>
<td>FaxNumber</td>
<td>Fax number of the recipient.</td>
</tr>
<tr>
<td>VoiceNumber</td>
<td>Voice number of the recipient.</td>
</tr>
<tr>
<td>Address1</td>
<td>Address of the recipient.</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Address2</td>
<td>Additional address information of the recipient.</td>
</tr>
<tr>
<td>Address3</td>
<td>Additional address information of the recipient.</td>
</tr>
<tr>
<td>Account</td>
<td>For information purposes only.</td>
</tr>
<tr>
<td>Coverpage</td>
<td>Specify a cover to use for this recipient.</td>
</tr>
</tbody>
</table>

**Response Element**

See ResultMessage.

**AddAttachment**

**Description**

This operation adds a file as an attachment to the fax. Your toolkit should allow you to pass the file as a byte array. The convenience class only requires a valid file system path to the file.

**Request Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttName</td>
<td>Name of the attachment.</td>
</tr>
<tr>
<td>Attachment</td>
<td>Base64 encoded contents of the file.</td>
</tr>
</tbody>
</table>

**Response Element**

See ResultMessage.

**SendFax, SendFaxEx**

**Description**

After specifying everything that is needed for a fax message, this operation sends the fax specified in the NewFaxMessage operation.

**Request Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IdTag</td>
<td>A user specified identification string for the fax. (SendFaxEx only)</td>
</tr>
</tbody>
</table>

**Response Element**

The data field in the ResultMessage contains the unique ID for the fax message.

**Sample Response**

```xml
<?xml version="1.0" encoding="utf-8"?>
```
GetPendingErrorMessageByUniqueID

Description
This operation retrieves a pending status from a job specified by a unique ID string from SendFax or SendFaxEx. A pending status means that the fax job is in queue to be sent to its recipient. Because a single SendFax/SendFaxEx can have multiple recipients, the status retrieved is uniquely identified by the combination of the unique ID obtained in the sending operation and the recipient index, which is a one-based order of the recipients added to the NewFaxMessage/NewFaxMessageEx operation. In other words, the recipient index represents the order in which the recipients were added to create the fax message, so that the first recipient would have an index of 1 and the second recipient would have and index of 2 and so on in the exact order in which they were added.

Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique id of the fax job obtained from SendFax or SendFaxEx operation.</td>
</tr>
<tr>
<td>recipient Index</td>
<td>The one-based order of the recipient of the recipients in the fax job.</td>
</tr>
</tbody>
</table>

Response Element
See PendingMessage.

GetPendingMessages, GetPendingMessagesEx

Description
Retrieve an array of PendingMessage records.

Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortColumn</td>
<td>The sorting order for the PendingMessage collection. 1 = JobID, 2 = Date and time, 3 = Priority, 4 = Sender, 5 = Fax number, 6 = Attempts, 7 =</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>ascending</td>
<td>True if sort column in ascending order.</td>
</tr>
<tr>
<td>start</td>
<td>The PendingMessage number to start retrieving.</td>
</tr>
<tr>
<td>count</td>
<td>The number of pending messages available.</td>
</tr>
</tbody>
</table>

**Response Element**

An array of **PendingMessage** records. See **PendingMessage**.

**GetPendingMessagesCount**

**Description**

Retrieve the size of the **PendingMessage** array. A return value of -1 indicates an error. This could mean that the account was not web-enabled or that there was an invalid session.

**Response Element**

Returns the **GetPendingMessagesCountResponse**, which contains **size of the PendingMessage array**.

**Sample Response**

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetPendingMessagesCountResponse xmlns="http://www.faxcompany.com/">
      <GetPendingMessagesCountResult>112</GetPendingMessagesCountResult>
    </GetPendingMessagesCountResponse>
  </soap:Body>
</soap:Envelope>
```

**GetMessageStatusByUniqueID, GetMessageStatusByUniqueIDEx**

**Description**

After a fax has been sent, successfully or unsuccessfully, a message status is generated. This message status can be retrieved by the unique ID returned by **SendFax** or **SendFaxEx**. If multiple recipients received the fax, the message status can be retrieved by the unique ID and the recipient index.

**Request Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniqueID</td>
<td>The unique ID of the fax message returned by <strong>SendFax</strong> or <strong>SendFaxEx</strong>.</td>
</tr>
<tr>
<td>recipientIndex</td>
<td>0 based order of the recipient in the recipients list added to the fax message.</td>
</tr>
</tbody>
</table>
Response Element
See MessageStatus.

GetMessageStatuses, GetMessageStatusEx

Description
Retrieve an array of MessageStatus records.

Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortColumn</td>
<td>The item in the MessageStatus to sort by. 1 = ID, 2 = Date and Time, 3 =</td>
</tr>
<tr>
<td></td>
<td>Recipient Name, 4 = Recipient fax number, 5 = Number of transmission</td>
</tr>
<tr>
<td></td>
<td>attempts, 6 = Status name, 7 = Status code, 8 = Status description, 9 =</td>
</tr>
<tr>
<td></td>
<td>Transfer rate, 10 = Pages sent, 11 = Attachment count, 12 = Connect time,</td>
</tr>
<tr>
<td></td>
<td>13 = subject, 14 = total pages, 15 = tag ID, 16 = CSI, 19 = Unique job ID.</td>
</tr>
<tr>
<td>ascending</td>
<td>If true, sort in ascending order.</td>
</tr>
<tr>
<td>start</td>
<td>Retrieve record starting at this index.</td>
</tr>
<tr>
<td>count</td>
<td>The number of records to retrieve.</td>
</tr>
</tbody>
</table>

(GetMessageStatusEx only)

Response Element
An array of MessageStatus records. See MessageStatus.

GetMessageStatusesCount

Description
Retrieve the size of the MessageStatus array. A return value of -1 indicates an error. This could mean that the account was not web-enabled or that there was an invalid session.

Response Element
Returns the GetMessageStatusesCountResponse, which contains size of the MessageStatus array.

Sample Response
<?xml version="1.0" encoding="utf-8"?>
  <soap:Body>
    <GetMessageStatusesCountResponse xmlns="http://www.faxcompanywhere.com/">
      <GetMessageStatusesCountResult>112</GetMessageStatusesCountResult>
    </GetMessageStatusesCountResponse>
  </soap:Body>
</soap:Envelope>

RemoveMessageStatusbyUniqueID

Description
Removes a given message status record from the MessageStatus array based on its unique ID. It is good practice to remove the MessageStatus after it has been accessed. This will free up resources on the server.

Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniqueID</td>
<td>The unique ID of the message status record. This is the same ID returned by the SendFax and SendFaxEx operation.</td>
</tr>
<tr>
<td>recipientIndex</td>
<td>This index is the 0 based add order of the recipient when the fax message was created.</td>
</tr>
</tbody>
</table>

Response Element
See ResultMessage.

GetActivityRecords
Retrieve an array of Activity records.

Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortColumn</td>
<td>The item in the Activity record to sort by. 1 = ID, 2 = Attempt time, 3 = Type, 4 = Retries, 5 = Fax number, 6 = Sender name, 7 = Pages transmitted, 8 = Error code, 9 = Error description, 10 = Connect time, 11 = Baud rate, 12 = Priority, 13 = Unique ID, 14 = Caller ID, 19 = ID tag.</td>
</tr>
<tr>
<td>ascending</td>
<td>If true, sort in ascending order.</td>
</tr>
<tr>
<td>start</td>
<td>Retrieve record starting at this index. (GetActivityRecordsEx only)</td>
</tr>
<tr>
<td>count</td>
<td>The number of records to retrieve. (GetActivityRecordsEx only)</td>
</tr>
</tbody>
</table>
Response Element
An array of Activity records. See Activity.

GetActivityRecordsCount

Description
Retrieve the size of the Activity records array. A return value of -1 indicates an error. This could mean that the account was not web-enabled or that there was an invalid session.

Response Element
Returns the GetActivityRecordsCountResponse, which contains the size of the Activity records array.

Sample Response
<?xml version="1.0" encoding="utf-8"?>
xmns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetActivityRecordsCountResponse xmlns="http://www.faxcompanywhere.com/"
      GetActivityRecordsCountResult>112</GetActivityRecordsCountResult>
  </GetActivityRecordsCountResponse>
</soap:Body>
</soap:Envelope>

GetTxStatusForJob

Description
Retrieve an array of TxStatus records. The array is sorted in ascending order by ActionDate. The array lists all the actions taken to submit a fax job to its destination. An ActionType of JobDone means that there are no more attempts or actions remaining. Actions ending in TxFailure result in a failed fax transmission. Actions ending in JobCompletion result in a successful fax transmission.

Key Action Types
The action types to look for are
- JobDone
- JobCompletion
- TxFailure
- EnqueueJob

JobDone signals the completion of a job. That means there are no more attempts being made for this job and results for these attempts have come in. Whether it is a failure or not is decided by the result
Action Types. The result action types are JobCompletion and TxFailure. EnqueueJob signals the arrival of the job in the pending queue en route to the fax server.

Determining the fate of a job based on Result Action Types
If the latest result action type is TxFailure then the job failed and the reason for the failure can be found in the status text, status name and status value codes. If there is an action type of JobCompletion then the job succeeded with the number of pages sent, the connect time, and CSI if present. The presence of the TxFailure action type does not mean the job failed. The job fails only if the last attempt has a TxFailure followed by the JobCompletion action type.

Response Element
Returns an array of TxStatus records. See TxStatus.

Sample Response
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetTxStatusForJobResponse xmlns="http://www.faxcomanywhere.com/">
      <GetTxStatusForJobResult>
        <TxStatus>
          <UniqueJobID>XXXXX_XXXXXXX_0905182124080006</UniqueJobID>
          <RecipientIndex>0</RecipientIndex>
          <Attempt>1</Attempt>
          <ActionType>EnqueueJob</ActionType>
          <ActionDate>2009-05-18T21:24:08</ActionDate>
          <ActionDescription>FAXCOM Queue service added the job to the pending queue.</ActionDescription>
          <StatusValue>0000</StatusValue>
          <StatusDescription>Success</StatusDescription>
          <StatusName>ok</StatusName>
          <Csi />
          <ConnectTime>-1</ConnectTime>
          <Pages>-1</Pages>
          <DialedNumber />
        </TxStatus>
        <TxStatus>
          <UniqueJobID>XXXXX_XXXXXXX_0905182124080006</UniqueJobID>
          <RecipientIndex>0</RecipientIndex>
          <Attempt>1</Attempt>
          <ActionType>SubmitToServer</ActionType>
          <ActionDate>2009-05-18T21:24:08</ActionDate>
          <ActionDescription>Job submitted to the FAXCOM Server.</ActionDescription>
          <StatusValue>0000</StatusValue>
          <StatusDescription>Success</StatusDescription>
          <StatusName>ok</StatusName>
          <Csi />
          <ConnectTime>-1</ConnectTime>
          <Pages>-1</Pages>
          <DialedNumber />
        </TxStatus>
      </GetTxStatusForJobResult>
    </GetTxStatusForJobResponse>
  </soap:Body>
</soap:Envelope>
<ArrayOfTxStatus>
<TxStatus>
<UniqueJobID>XXXXX_XXXXXXX_0905182124080006</UniqueJobID>
<RecipientIndex>0</RecipientIndex>
<Attempt>1</Attempt>
<ActionType>QueueToConvert</ ActionType>
<ActionDate>2009-05-18T21:24:08</ActionDate>
<ActionDescription>Job waiting for conversion at FAXCOM Server.</ActionDescription>
>StatusValue>0000</StatusValue>
>StatusDescription>Success</StatusDescription>
>StatusName>ok</StatusName>
<Csi />
<TxStatus>
<UniqueJobID>XXXXX_XXXXXXX_0905182124080006</UniqueJobID>
<RecipientIndex>0</RecipientIndex>
<Attempt>1</Attempt>
<ActionType>JobCompletion</ ActionType>
<ActionDate>2009-05-18T21:24:08</ActionDate>
<ActionDescription>The job delivery completed successfully.</ActionDescription>
>StatusValue>0000</StatusValue>
>StatusDescription>Success</StatusDescription>
>StatusName>ok</StatusName>
<Csi />
<TxStatus>
<UniqueJobID>XXXXX_XXXXXXX_0905182124080006</UniqueJobID>
<RecipientIndex>0</RecipientIndex>
<Attempt>1</Attempt>
<ActionType>JobDone</ ActionType>
<ActionDate>2009-05-18T21:24:11</ActionDate>
<ActionDescription>The final attempt for a job has ended.</ActionDescription>
>StatusValue>0000</StatusValue>
>StatusDescription>Success</StatusDescription>
>StatusName>ok</StatusName>
<Csi />
<TxStatus>
</ArrayOfTxStatus>
</GetTxStatusForJobResult>
</GetTxStatusForJobResponse>
</soap:Body>
</soap:Envelope>
GetReceivedMessages, GetReceivedMessagesEx

**Description**
Retrieve an array of RxStatus records.

**Request Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortColumn</td>
<td>The item in the collection to sort by. 2 = ID, 3 = Date and Time, 4 = File Name, 5 = TSI, 6 = Caller ID, 7 = Number of Pages, 8 = File Size, 9 = Status Name, 10 = Status Code, 11 = Status Description, 12 = Connect Time, 13 = DID number, 17 = Completion Time.</td>
</tr>
<tr>
<td>ascending</td>
<td>If true, sort in ascending order.</td>
</tr>
<tr>
<td>start</td>
<td>Retrieve record starting at this index.</td>
</tr>
<tr>
<td>count</td>
<td>The number of records to retrieve.</td>
</tr>
<tr>
<td>loadImage</td>
<td>If true, load the image data into the record.</td>
</tr>
</tbody>
</table>

**Response Element**
An array of RxStatus records. See RxStatus.

GetReceivedMessagesCount

**Description**
Retrieve the size of the RxStatus array. A return value of -1 indicates and error. This could mean that the account was not web-enabled or that there was an invalid session.

**Response Element**
Returns the GetReceivedMessagesCountResponse, which contains size of the RxStatus array.

**Sample Response**
```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetReceivedMessagesCountResponse xmlns="http://www.faxcomanywhere.com/">
      <GetReceivedMessagesCountResult>int</GetReceivedMessagesCountResult>
    </GetReceivedMessagesCountResponse>
  </soap:Body>
</soap:Envelope>
```
RemoveReceivedMessagebyName

Description
This operation removes a received message from the received message collection based on the image’s path name (see RxStatus. ImageFileName).

Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pathName</td>
<td>A string value of the path to the image.</td>
</tr>
</tbody>
</table>

Response Element
See ResultMessage.

Sample Response
<?xml version="1.0" encoding="utf-8"?>
  <soap:Body>
    <RemoveReceivedMessagebyPathNameResponse xmlns="http://www.faxcompanywhere.com/">
      <RemoveReceivedMessagebyPathNameResult>
        <Result>true</Result>
        <Detail>Received Message object: 3495A, succesfully deleted.</Detail>
        <Type>INFO</Type>
        <Data></Data>
      </RemoveReceivedMessagebyPathNameResult>
    </RemoveReceivedMessagebyPathNameResponse>
  </soap:Body>
</soap:Envelope>

SetSenderInformation

Description
Set the default sender information.

Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the sender.</td>
</tr>
<tr>
<td>faxNumber</td>
<td>Fax number of the sender.</td>
</tr>
<tr>
<td>voiceNumber</td>
<td>Voice number of the sender.</td>
</tr>
<tr>
<td>email</td>
<td>Email address of the sender.</td>
</tr>
<tr>
<td>company</td>
<td>The company of the sender.</td>
</tr>
<tr>
<td>address1</td>
<td>The address of the sender.</td>
</tr>
<tr>
<td>address2</td>
<td>Additional address information of the sender.</td>
</tr>
<tr>
<td>address3</td>
<td>Additional address information of the sender.</td>
</tr>
</tbody>
</table>
Appendix A: Status Codes

The following table lists, with explanations, the status codes returned by `GenericStatus` and `MessageStatus`. In the event of an error, trying a resend of the fax may result in a successful transmission of the fax if the Resend column is ‘Yes’.

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Explanation</th>
<th>Resend</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ok</td>
<td>All pages were successfully transmitted.</td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>hup</td>
<td>Lost communication with called fax machine</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>nak</td>
<td>Received page quality deemed unacceptable by fax machine.</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>nois</td>
<td>FAXCOM Anywhere unable to transmit due to noisy phone connection.</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>voic</td>
<td>FAXCOM Anywhere detected voice answer.</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>dis1</td>
<td>Early disconnect. Remote fax machine requested a disconnect before image was transmitted.</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>dis2</td>
<td>Late disconnect; remote fax machine requested a disconnect after image was transmitted.</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>tone</td>
<td>No dial tone detected.</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>rset</td>
<td>FAXCOM Anywhere reset during transmission.</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>busy</td>
<td>Called fax machine busy.</td>
<td>Yes</td>
</tr>
<tr>
<td>14</td>
<td>nbsy</td>
<td>Phone network busy.</td>
<td>Yes</td>
</tr>
<tr>
<td>15</td>
<td>noan</td>
<td>Called phone number ringing but no answer.</td>
<td>Yes</td>
</tr>
<tr>
<td>16</td>
<td>grpl</td>
<td>Incompatible Group I fax machine.</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>grp2</td>
<td>Incompatible Group II fax machine.</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>?mdm</td>
<td>Unknown modem. A modem answer tone was detected, but it is not a fax modem.</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>noph</td>
<td>No destination phone number specified.</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>Error reading font during translation.</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>Specified form was not detected.</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>Incompatible G III fax receiver detected.</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>Cannot dial due to incoming call.</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>dead</td>
<td>After dialing, no telephone network response.</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>rvoi</td>
<td>FAXCOM Anywhere answered call, but no fax at other end.</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>rdis</td>
<td>FAXCOM Anywhere receive mode; disconnect requested.</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>All pages blank; nothing transmitted.</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>nops</td>
<td>FAXCOM Anywhere not configured with PostScript option.</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>Postscript option not loaded in this version.</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>npcl</td>
<td>HP PCL Interpreter did not create any pages.</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>tout</td>
<td>Job being collected timed-out (no activity detected as per FCL timeout setting).</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>neof</td>
<td>Internal FAXCOM queue error; resubmit job.</td>
<td></td>
</tr>
<tr>
<td>671</td>
<td></td>
<td>Error converting attachment to fax format.</td>
<td></td>
</tr>
<tr>
<td>699</td>
<td></td>
<td>Image translation error; no output file created by translator; unable to</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Result</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>716</td>
<td>Fax board was reset.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>727</td>
<td>Fax board temporarily out of service; firmware is being reloaded.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>770</td>
<td>Dial tone was detected after dialing the number.</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>771</td>
<td>Internal error; resend the job; if error occurs repeatedly, contact FAXCOM Anywhere.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>772</td>
<td>Attempt to attach a fax port was unsuccessful (possibly due to a configuration error or hardware problem).</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>773</td>
<td>Attempt to reset a fax port was unsuccessful (possibly due to a configuration error or hardware problem).</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>774 – 777, 779 - 781</td>
<td>Internal error; resend the job; if error occurs repeatedly, contact FAXCOM Anywhere.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>783</td>
<td>Miscellaneous hangup.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>Internal error; resend the job; if error occurs repeatedly, contact FAXCOM Anywhere.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>9012</td>
<td>Non-compatible attachment.</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>9013</td>
<td>Error creating fax message.</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>9014</td>
<td>Cannot mix PostScript, PCL, and Epson files.</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>9017</td>
<td>env Too many addresses.</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>9018</td>
<td>addr No phone number in address field.</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>