



Cognitier HomeWeb Service Getting Started Guide

Version 1.1

November 1, 2011

Copyright © 2011 Cognitier, Inc. All rights reserved.

Cognitier and the Cognitier logo are trademarks of Cognitier, Inc. All other company and product names referred to may be trademarks of their respective owners. This documentation is copyrighted material and is intended exclusively for use by licensed users of Cognitier software. The information in this document is subject to change without notice.

Table of Contents

1	Introduction	4
2	How the Service Works	4
3	Service User Responsibilities.....	4
4	Getting Started	5
5	Message Peer For PHP Installation	5
5.1	Prerequisites	5
5.2	Installation Process	5
5.3	Post-Installation Configuration	6
5.3.1	Identifying Your “web” Location	6
5.3.2	Configuring a “web” on the Windows Server	6
5.3.3	Creating the HomeWeb Service User.....	7
5.3.4	Starting the Local Service	8
6	Writing Routines on Your Computer.....	9
6.1	Exercise 1 – Listing Files on Your Computer.....	9
6.2	Exercise 2 – Retrieve a File From Your Computer	19
6.3	Exercise 3 – Email a File to Yourself	28
6.4	Exercise 4 – Call a Routine From Your Website	30
6.5	Sample Exercise Conclusions.....	35
7	Getting Additional Support	36

1 Introduction

The HomeWeb service is intended to allow a Windows computer behind a firewall (such as a home computer) to respond to requests initiated from the public internet. You (the user of the service), write scripts to define the commands (routines) to which your computer will respond and how the commands will be carried out. Then you create the means by which the remote commands will be issued. This usually means you have a web page with a PHP or ASP script hosted by a hosting provider. The PHP or ASP script issues the command for your computer to carry out.

There are two primary uses for the HomeWeb service:

1. Consumers may use this service for remote access to their home computers. Instead of using a service which allows you to log into your home computer from a remote location, the HomeWeb service allows you to issue discrete commands for your home computer to carry out. It is a much quicker process to issue the discrete command, but the commands have to be pre-defined with scripts ahead of time.
2. Developers hosting web sites can route certain operations to be completed on a computer that is behind a firewall. Routing a request via the HomeWeb service is analogous to making a web service call against the firewalled computer, although the mechanics are different.

2 How the Service Works

You will need to fill out the signup form and download and install two software applications from Cognitier. The Messaging Peer For PHP application is a service that runs on your computer that enables your computer to respond to requests as long as your computer is running and connected to the internet. SimpleIPC (or SimpleIPC Express) is an application that allows you to prepare routines that can be run in response to these external requests. When you sign up for the HomeWeb service, Cognitier will host a “web” for you. That “web” is a location on the public internet. To send a request to your home computer, you actually send the request to the hosted “web”, and then the request gets routed to your home computer.

Note: Your computer must be running and connected to the internet in order for the service to operate. If your computer loses internet connectivity, then it cannot respond to requests. Likewise, if your computer is configured to suspend itself after a period of inactivity, then it cannot respond to requests.

3 Service User Responsibilities

The Terms Of Service are posted on the Cognitier website; however, a few items are highlighted here:

1. Enabling your computer to respond to commands originating from the public internet is inherently risky. You should assume that any routine you define could be executed by an intruder. If, for example, you define a routine that has full read and write access to your entire hard drive, you are taking a risk that personal data might be stolen, important files could be deleted or maliciously altered, etc.

2. Due, in part, to these inherent risks, you are not allowed to use the service if you are under the age of 18.
3. Please be aware that the HomeWeb service does use some of your computer's processing capacity and network bandwidth – even when it is not in the process of executing commands. In particular, you should assume the service will use at least 12MB of network bandwidth per day. This is important if your internet plan does not include unlimited bandwidth.
4. We ask that you be conscientious and maintain high standards in the material you transact by means of this service. We ask that you not use this service to distribute material related to hate speech, gambling, pornography, spam, infringement of intellectual property rights, etc. This is not an all-inclusive list.

4 Getting Started

Find the link for the HomeWeb service signup on the Cognitier website. Download and install SimpleIPC or SimpleIPC Express (SimpleIPC Express is the free version of the product, and it is adequate for most users). The User's Guide for SimpleIPC is available for download. Next, download and install the Messaging Peer For PHP. This Getting Started Guide is a substitute for the User's Guide that is available for Messaging Peer For PHP. The Messaging Peer For PHP User's Guide assumes that you will set up and maintain your own "web". With the HomeWeb service, Cognitier is setting up and maintaining that "web" for you. This Getting Started Guide is a sort of "crash course" for using the HomeWeb service. The User's Guide for the Messaging Peer For PHP provides more in-depth information on the inner workings of the service and its components, and may be of interest to "power users".

During the signup, you will be asked to provide a username and a password. The username will become the last portion of your "web" name (discussed later). You will receive an email notification when your "web" has been set up and is ready for use. Your signup information for the HomeWeb service will undergo manual review, and as such your "web" will not be immediately available.

5 Message Peer For PHP Installation

5.1 Prerequisites

Please be sure to install SimpleIPC or SimpleIPC Express before installing the Messaging Peer For PHP. If your home computer is capable of running SimpleIPC, then it will be capable of running the Messaging Peer For PHP.

5.2 Installation Process

There are very few selectable options during the Messaging Peer installation process. The default installation directory is a sub-directory of \Program Files (or \Program Files (x86) on 64-bit Windows). It is important to note that the Messaging Peer infrastructure does create log files and other server state-maintenance files, and for that reason you may choose to install the product to an alternate location. The installation process creates a "Service" on the Windows server. The service requires post-installation configuration before it can be used.

At the time of this writing, a separate installer version for 64-bit Windows operating systems is not available. If your computer is running a 64-bit version of Windows, then after installing SimpleIPC and the Messaging Peer (and before launching the console application for either application), please download and run the registry patch for 64-bit Windows operating systems. The link for this registry patch is located on the same page as the download link for SimpleIPC.

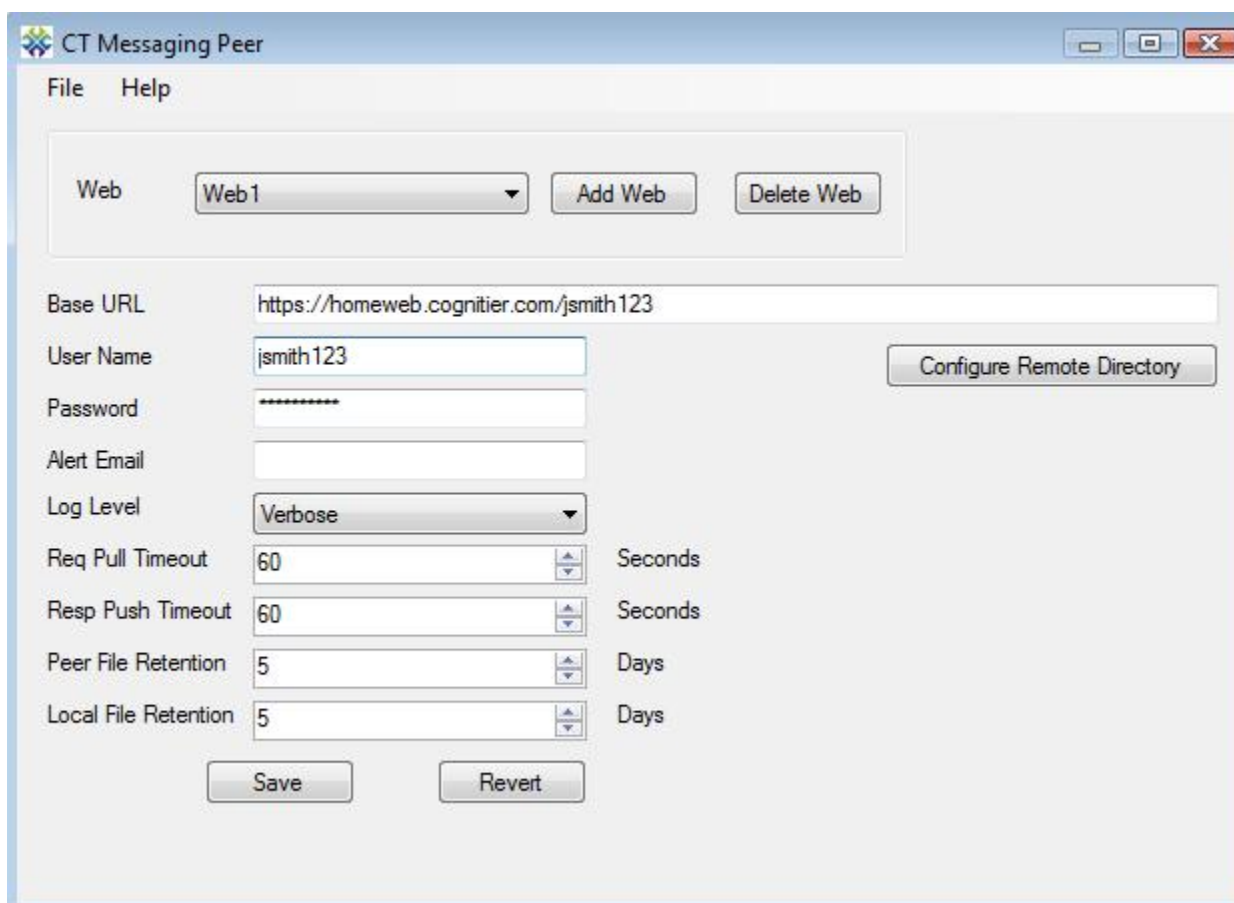
5.3 Post-Installation Configuration

5.3.1 Identifying Your “web” Location

You will receive an email notification when your “web” has been set up by Cognitier and is available for use.

5.3.2 Configuring a “web” on the Windows Server

After installing the Messaging Peer on the Windows server, launch the CTMsgWorkerConsole (configuration console) via Start>>Programs>>Cognitier>>Msg Peer For PHP>> CTMsgWorkerConsole. Use the console to configure a “web”. Within the console, a “web” is a collection of settings which dictates how the Windows server will participate in the HomeWeb service.



A “web” can be configured with the following settings:

Base URL: This is the “web” location that you receive in email after signing up for the service.

User Name: This is the username you selected while signing up for the service.

Password: This is the password you selected while signing up for the service.

Alert Email: This setting is not relevant for users of the HomeWeb service. It is only relevant for users hosting their own “web”.

Log Level: Choose the desired level of logging. This will define the volume of log information generated on your computer. More information is useful for debugging, but it consumes space on your hard drive.

Req Pull Timeout: This timeout setting represents the number of seconds your computer will wait for a request to complete when your computer is pulling data from the internet as part of the service. A higher setting will result in less bandwidth consumption over time. However, the setting must be at least 20 seconds less than the network timeout. A typical network timeout is 90 seconds, so the default setting of 60 seconds is probably appropriate.

Resp Push Timeout: This timeout setting represents the number of seconds your computer will wait for a request to complete when your computer is pushing data to the internet as part of the service. Unless you define routines to push very large quantities of data, the default setting of 60 seconds is probably appropriate.

Peer File Retention: This setting is not relevant for users of the HomeWeb service. It is only relevant for users hosting their own “web”.

Local File Retention: This setting represents the number of days after which files in all of the Messaging Peer sub-directories will be purged on your computer. The files in these directories are useful for troubleshooting, but many files are created, so there is a disk space consideration.

After entering the appropriate settings, click the Save button to save your changes.

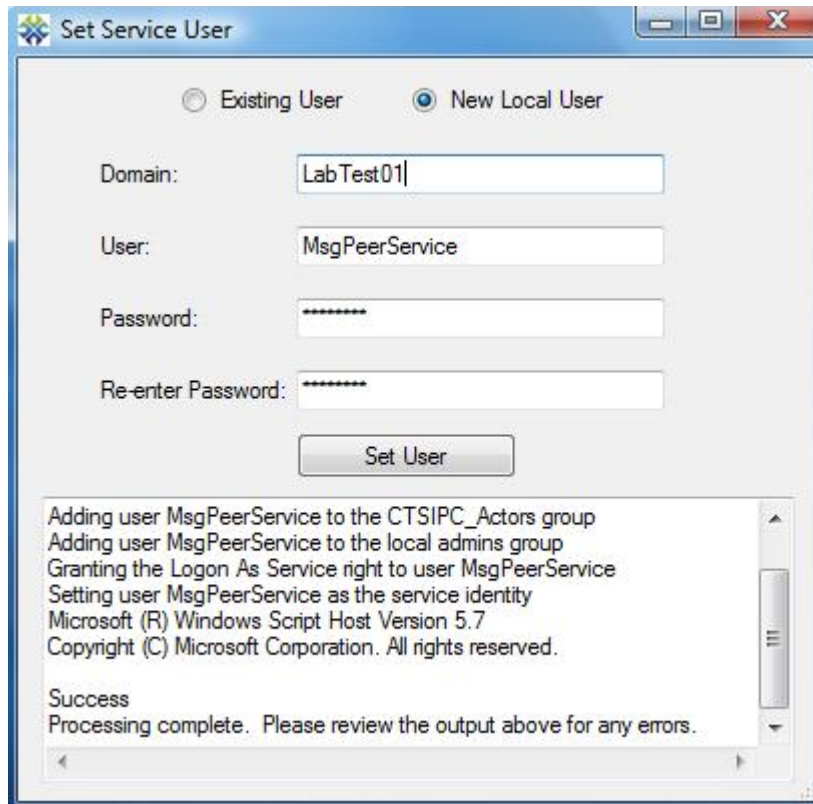
The “Configure Remote Directory” button is not relevant for users of the HomeWeb service. It is only relevant for users hosting their own “web”.

5.3.3 Creating the HomeWeb Service User

You have to create a new user account on your computer for the HomeWeb service. This way, you can change the permissions for that user later without affecting other software on your computer. To create this user account, do the following:

1. Launch the CTMsgWorkerConsole (if it is not already running) via Start>>Programs>>Cognitier>>Msg Peer For PHP>> CTMsgWorkerConsole.
2. From the menu, select File>>Set Service Account.
3. Select the “New Local User” radio button.
4. Leave the “Domain” field as is (it will be your computer name).

5. Enter the name to assign to the new user.
6. Enter a complex password (6 or more characters, upper and lower case letters, at least one number).
7. Click the “Set User” button.
8. Examine the output in the text area at the bottom of the form and ensure that it indicates that the user was created successfully.



The Messaging Peer service will start SimpleIPC servers for the processing of requests sent by the Linux server, so those servers will run under the same account as the Messaging Peer service.

5.3.4 Starting the Local Service

Go to Start>>Control Panel>>Administrative Tools>>Services on your computer and look through the list of services to find the “CT Messaging Peer For PHP” service. Right-click on this service and select “Start”. If the starting of the service appears to timeout on the first attempt, then make one more attempt to start the service before taking any other troubleshooting steps.

6 Writing Routines on Your Computer

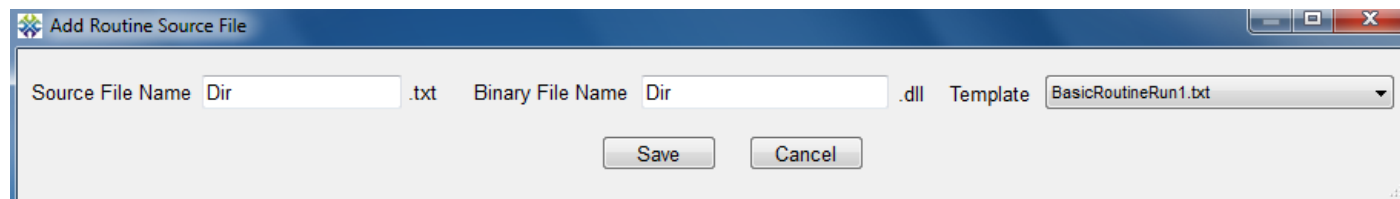
At this point, your computer is ready to execute remotely-issued commands, but no commands have been defined yet. In the HomeWeb service, the command to be executed takes the form of a “routine”. A routine is a piece of code written so that it can be “plugged into” SimpleIPC (SimpleIPC is the first product you installed to get started with the HomeWeb service). You can actually obtain pre-written routines from someone else and plug them into your copy of SimpleIPC. This document assumes you will write new routines yourself. The SimpleIPC User’s Guide covers the programming of routines in depth. This document will provide working samples without explaining each line of code. The routines provided will be scripted using SimpleIPC’s built-in editor for JScript.NET. You have the option of writing routines in C# or VB.NET using Microsoft Visual Studio .NET. Please refer to the SimpleIPC User Guide for a complete discussion of authoring and registering routines with SimpleIPC.

6.1 Exercise 1 – Listing Files on Your Computer

The following procedure will establish a command that can be run remotely to list the files and directories starting from a given root directory on your computer – similar to the “Dir” command available from a command prompt.

Step 1: Open Task Manager and view all of the processes running on your computer. Use the “Show processes from all users” option. If there are any occurrences of CTServerInst1.exe running, then terminate them. These are the SimpleIPC processes which run the routines you define.

Step2: Open the SimpleIPC console by selecting Start>>Programs>>Cognitier>>SimpleIPC>>CTConsole. Go to the Author Routines tab. This is where the code for routines is entered. On this screen, click the “New Source File” button. Give the new routine a name of “Dir” and choose the BasicRoutineRun1 template.



Step3: Copy and paste the following code into the source code text area:

```
import System;
import System.Collections;
import System.Reflection;

[assembly:AssemblyVersion("1.0.0.0")]
public class Dir implements CTSHost.IBasicRoutineRun1
{
    public function RunRoutine(oServerContextAccessor: CTSHost.ServerContextAccessor,
oSessionContextAccessor: CTSHost.SessionContextAccessor, oCallParamsAccessor:
CTSHost.CallParamsAccessor) : System.Boolean
    {
        var oLogUtil: CTCommon.LogUtil;
        var bRet: System.Boolean;
```

```

    oLogUtil = CTCCommon.LogUtil.Instance;
    bRet = false;
    try
    {
        //We expect one input argument - the directory name
        if (oCallParamsAccessor.InputArgs.Count == 0)
        {
            oCallParamsAccessor.OutputErrors.Add("No directory name provided");
        }
        else
        {
            var sInput: System.String;
            sInput = oCallParamsAccessor.InputArgs[0];

            var sDirName: System.String;
            var sDirNameList: System.String[] =
System.IO.Directory.GetDirectories(sInput);
            var oDirInfo: System.IO.DirectoryInfo;
            var sFileName: System.String;
            var sFileNameList: System.String[] = System.IO.Directory.GetFiles(sInput);
            var oFileInfo: System.IO.FileInfo;
            var iCount: System.Int32;
            var oStringBuilder: System.Text.StringBuilder = new
System.Text.StringBuilder();
            iCount = 0;
            while (iCount < sDirNameList.Length)
            {
                oStringBuilder.Remove(0, oStringBuilder.ToString().Length);
                oDirInfo = new System.IO.DirectoryInfo(sDirNameList[iCount]);
                oStringBuilder.Append(oDirInfo.CreationTime.ToString("g"));
                oStringBuilder.Append("\t\t");
                oStringBuilder.Append("DIR");
                oStringBuilder.Append("\t\t");
                oStringBuilder.Append(oDirInfo.Name);
                oCallParamsAccessor.OutputArgs.Add(oStringBuilder.ToString());
                iCount++;
            }
            iCount = 0;
            while (iCount < sFileNameList.Length)
            {
                oStringBuilder.Remove(0, oStringBuilder.ToString().Length);
                oFileInfo = new System.IO.FileInfo(sFileNameList[iCount]);
                oStringBuilder.Append(oFileInfo.CreationTime.ToString("g"));
                oStringBuilder.Append("\t\t\t");
                oStringBuilder.Append(oFileInfo.Length);
                oStringBuilder.Append("\t");
                oStringBuilder.Append(oFileInfo.Name);
                oCallParamsAccessor.OutputArgs.Add(oStringBuilder.ToString());
                iCount++;
            }
            bRet = true;
        }
    }
    catch (e)
    {
        //Set errors into errors collection - must be strings

```

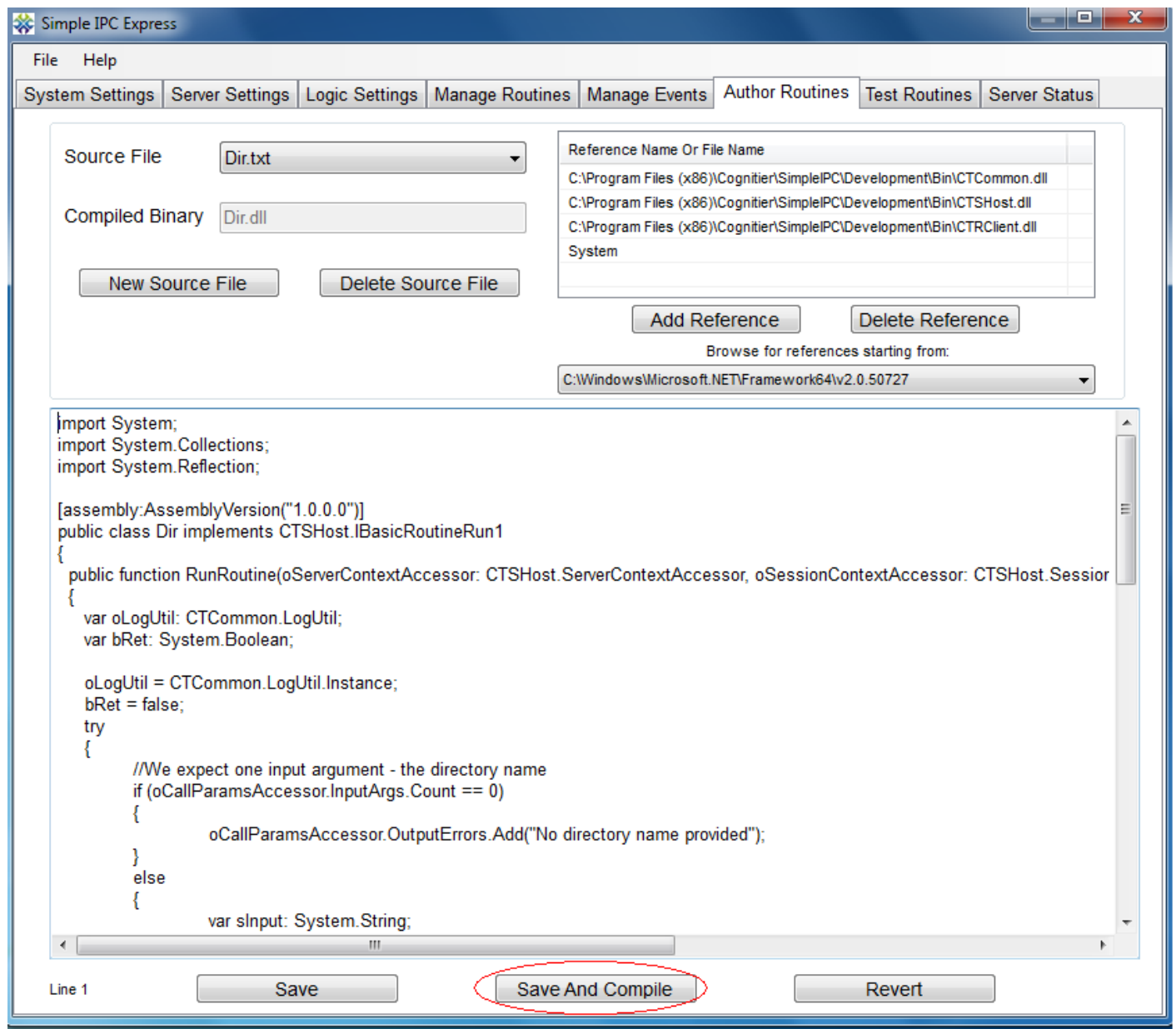
```

oCallParamsAccessor.OutputErrors.Add(e.Message);

//Also write errors to the server log file
//Log levels are Errors (1), Warnings (2), Info (3), Verbose (4)
oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, e);
}
return bRet;
} //end function
} //end class

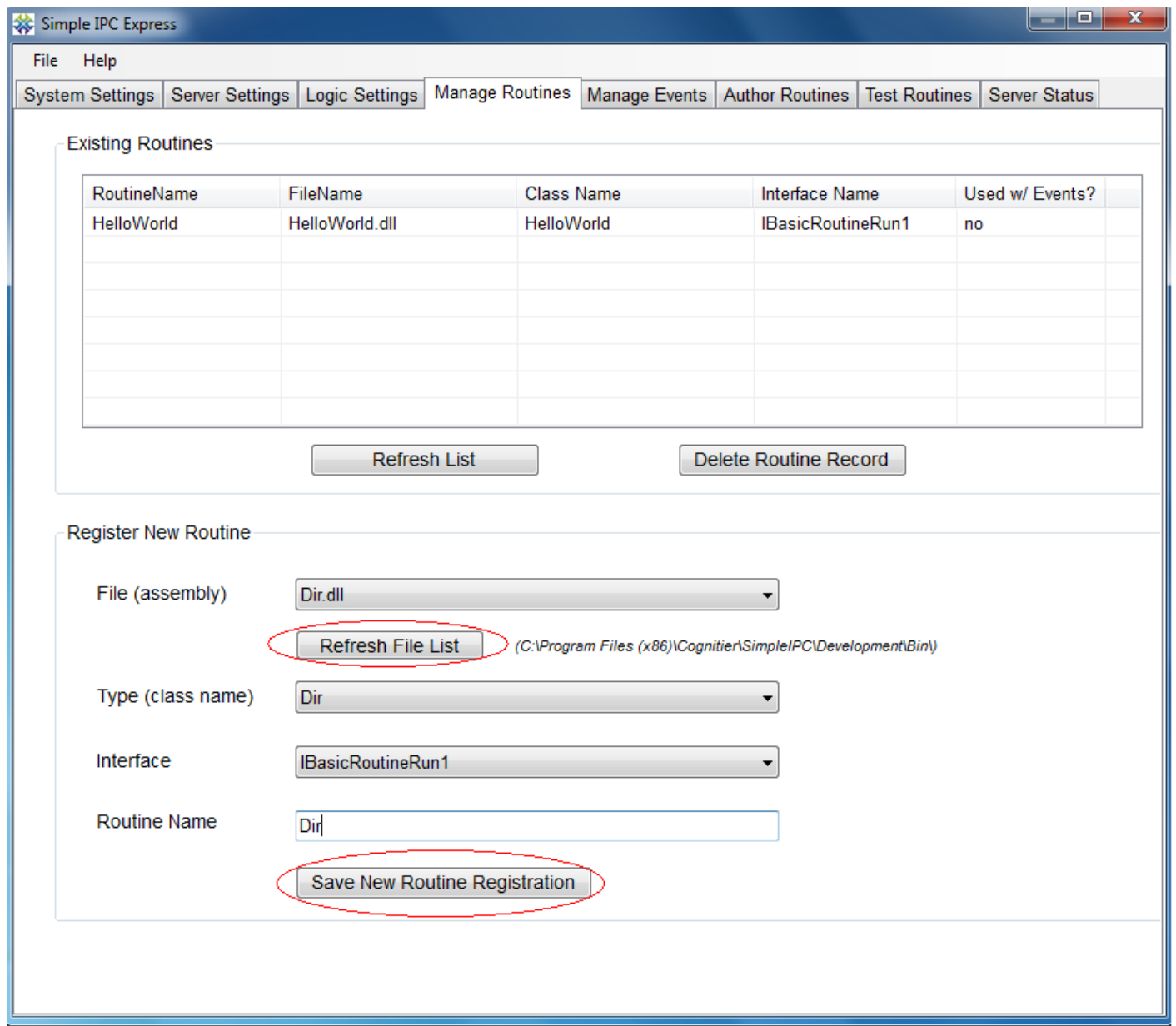
```

Step 4: After entering this code, click the “Save And Compile” button. Accept the resulting prompts – including the prompt to copy the compiled assembly to the alternative directory.

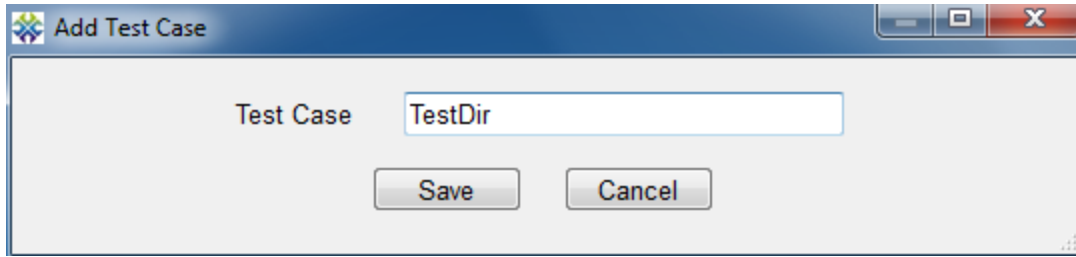


Note: If at any time you receive a message saying the code cannot be compiled or copied because the file is in-use, then terminate all instances of CTServerInst1.exe via Task Manager and close and re-launch the SimpleIPC console (CTConsole).

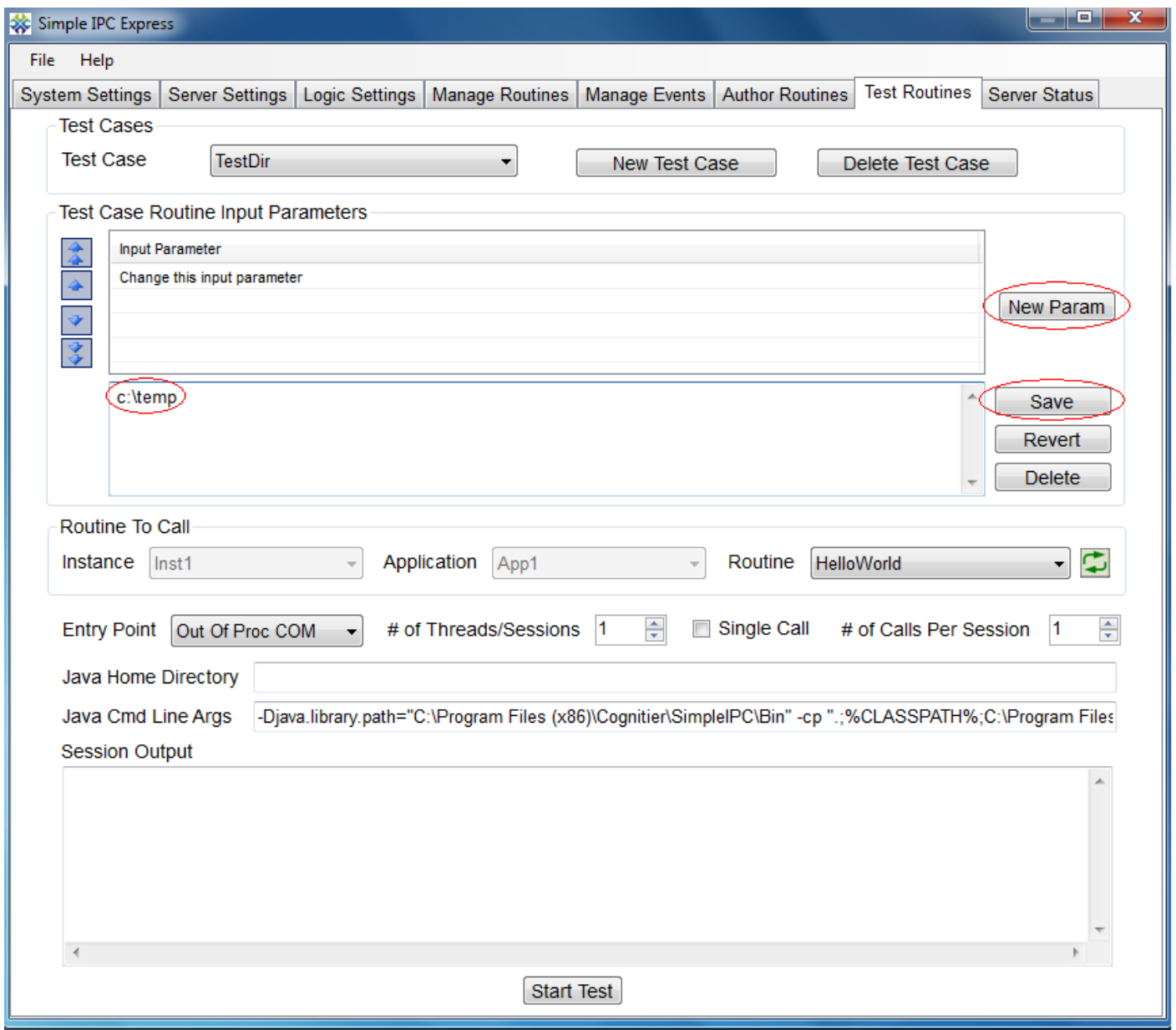
Step 5: After the routine has been coded and compiled, it must be registered within SimpleIPC to make it available for execution. Go to the “Manage Routines” tab. Click the “Refresh Files List” button to make the new assembly available in the drop-down above the button. Select “Dir.dll” from the “File (assembly)” drop-down. At the bottom of the form, enter a Routine Name of “Dir” and click the “Save New Routine Registration” button.



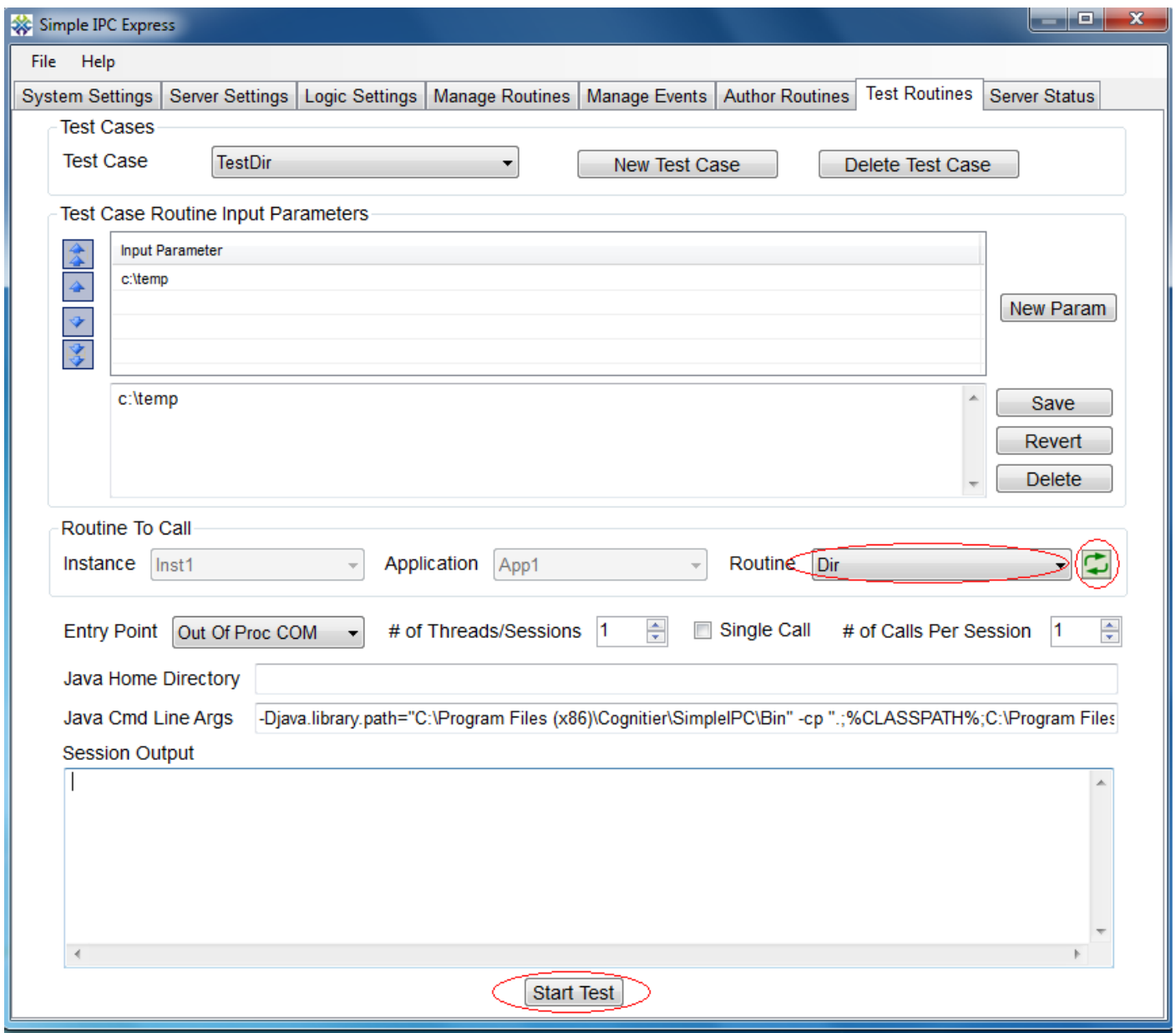
Step 6: At this point, the new routine is ready for execution. Test the routine locally by going to the “Test Routines” tab. From this tab, click the “New Test Case” button at the top of the form. Give the new test case a name of “TestDir”.



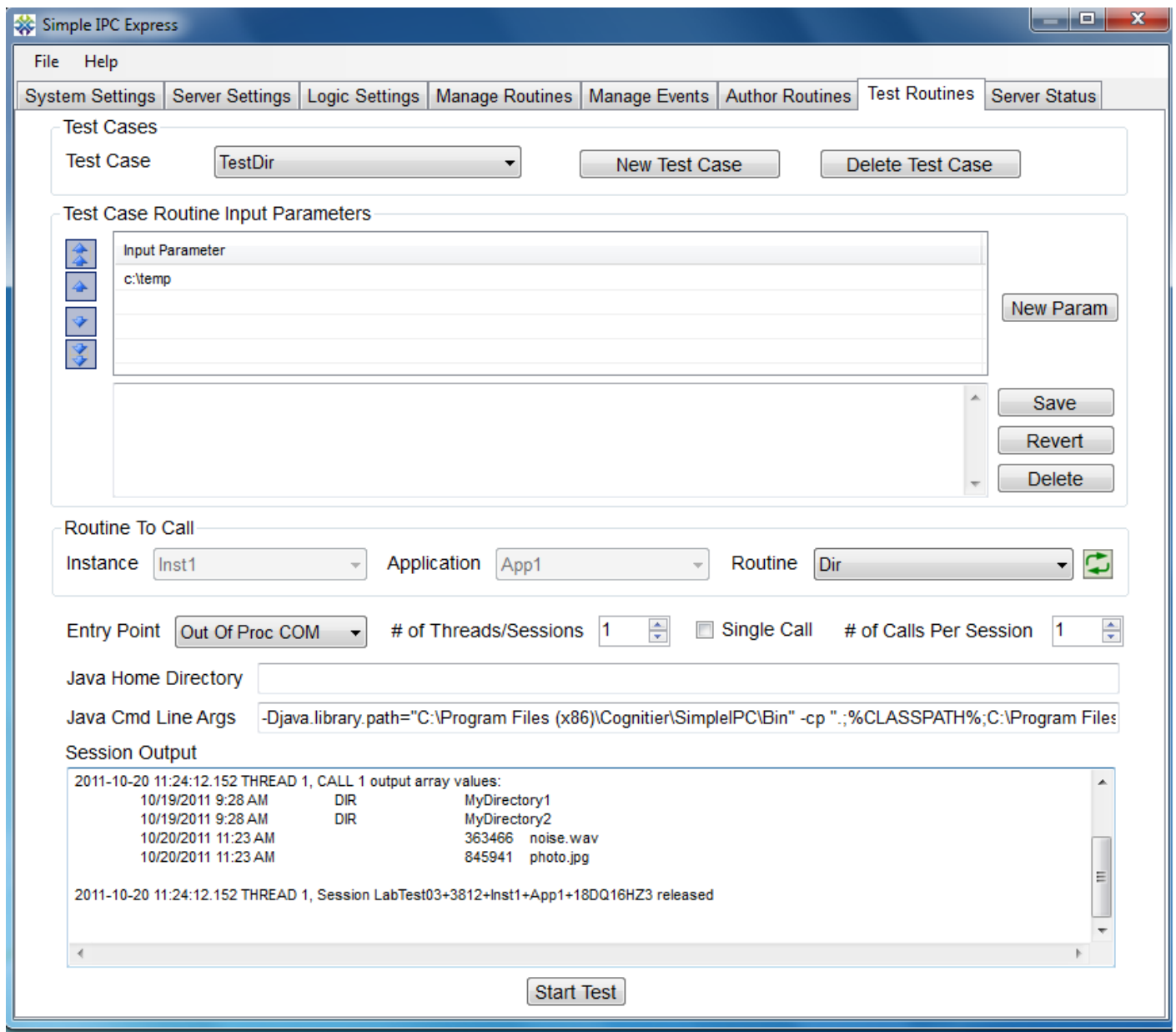
Step 7: A test case is a collection of parameters. For our new routine, we need to pass the starting point directory as a parameter. Click the “New Param” button and change the default parameter value to “c:\temp”. Then click the “Save” button to save that parameter value.



Step 8: Click the refresh button (circular arrows) to refresh the contents of the “Routine” drop-down. Select “Dir” from the “Routine” drop-down and click the “Start Test” button at the bottom of the form.



Step 9: Depending on the current state of your system, a command prompt window may appear displaying output from the SimpleIPC server created to handle the request. There should then be a list of files and directories in the “Session Output” text area at the bottom of the form.



Step 10: The new routine has now been tested locally, and so the next step will be to test it from over the internet. Ensure that you have properly installed the Messaging Peer For PHP, that the CT Messaging Peer For PHP service is running on your computer, and that you have entered the settings you were sent via email. Ensure you have your HomeWeb username and password available. Open a web browser on a computer – it need not be the same computer where you have installed the Messaging Peer. Navigate to [https://homeweb.cognitier.com/\[username\]/interactive/RegisterRoutine1.php](https://homeweb.cognitier.com/[username]/interactive/RegisterRoutine1.php) - replace “[username]” with the username you selected when you signed up for the HomeWeb service (e.g. <https://homeweb.cognitier.com/jsmith123/interactive/RegisterRoutine1.php>). Note that this address is case-sensitive. If you are reaching this page for the first time, you will be prompted for a username and password. Enter the username and password you selected when you signed up for the HomeWeb service.

Note: The web pages under the /interactive directory are assigned to the HomeWeb account holder, and these pages do not support use by multiple concurrent users.

Step 11: This web page is used to save “Routine Sets”. A routine set is a routine name and a collection of input parameters. You need not create a routine set in order to call routines from the “interactive” web pages, but if you want to save several test cases with different parameters it is convenient to do it this way. Enter a “Set Name” of “Dir1” and a “Routine Name” of “Dir” and click “Create”.

Save a routine set to call later

You can save a routine name along with some input arguments as a "routine set" (test case) to run later. This simply saves time re-entering the information.

Select an existing routine set to edit or create a new one

Existing Routine Set Name:

OR New Routine Set:

Set Name

Routine Name

Routine: *HelloWorld*

Parameters:

[Call a routine that returns string values](#)

[Call a routine that renders a file to the browser](#)

Step 12: Create a new parameter for this routine set. If “Dir1” is not already selected in the “Existing Routine Set Name” drop-down, then select it. Then click the “Create New” button. After this, change the value in the text box in the lower-left of the screen to “c:\temp” and click “Update”.

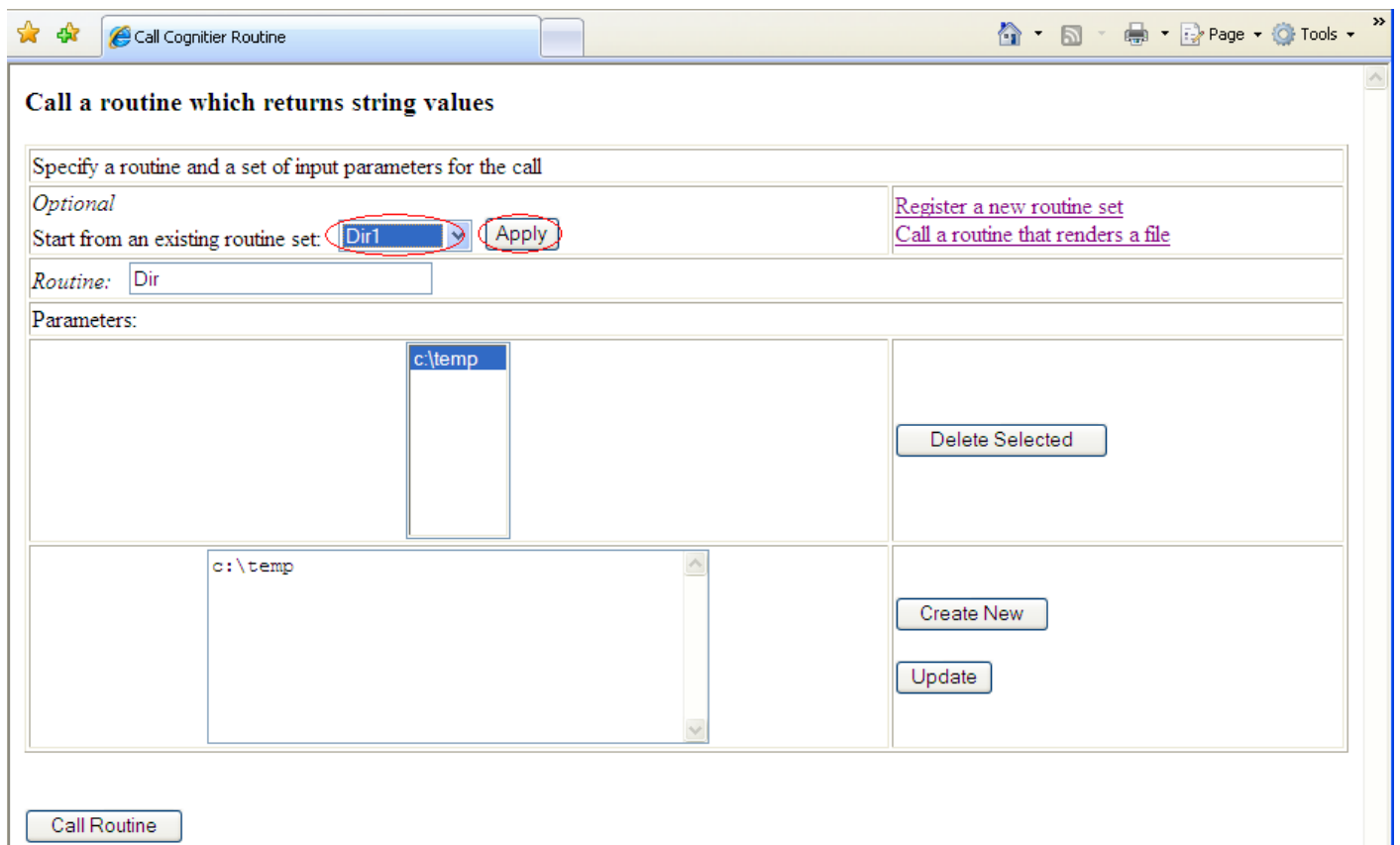
Save a routine set to call later

You can save a routine name along with some input arguments as a "routine set" (test case) to run later. This simply saves time re-entering the information.

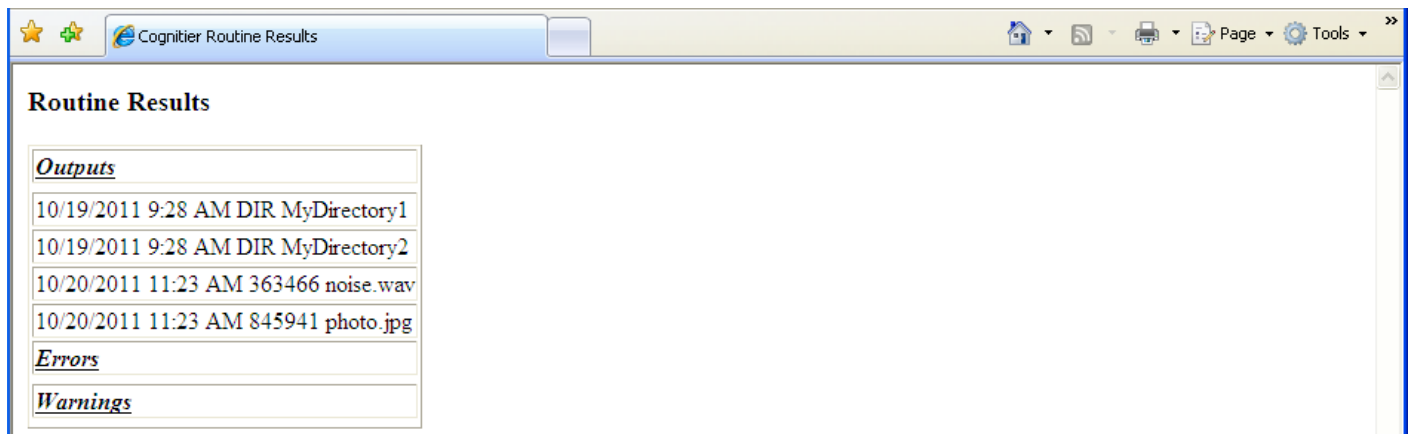
Select an existing routine set to edit or create a new one

Existing Routine Set Name: Dir1 <input type="button" value="Delete"/>	OR New Routine Set: Set Name <input type="text"/> Routine Name <input type="text"/> <input type="button" value="Create"/>
Routine: Dir	
Parameters:	
<input type="text" value="##New Param To Update##"/>	<input type="button" value="Delete Selected"/>
<input type="text" value="c:\temp"/>	<input type="button" value="Create New"/> <input type="button" value="Update"/>
Call a routine that returns string values	
Call a routine that renders a file to the browser	

Step 13: At the bottom of the form, click the link that reads "Call a routine that returns string values". On the resulting screen, select "Dir1" from the "Start from an existing routine set" drop-down and click "Apply".



Step 14: At this point, we could change the routine or the parameters, but we will leave them as-is. Click the “Call Routine” button to invoke the “Dir” routine and list the files and directories in the “c:\temp” directory.

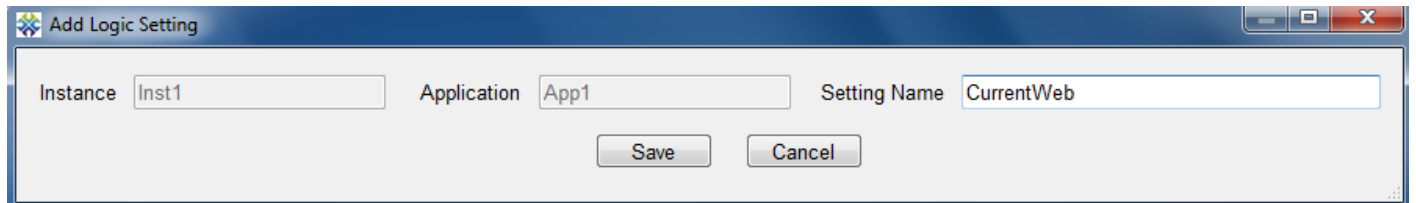


6.2 Exercise 2 – Retrieve a File From Your Computer

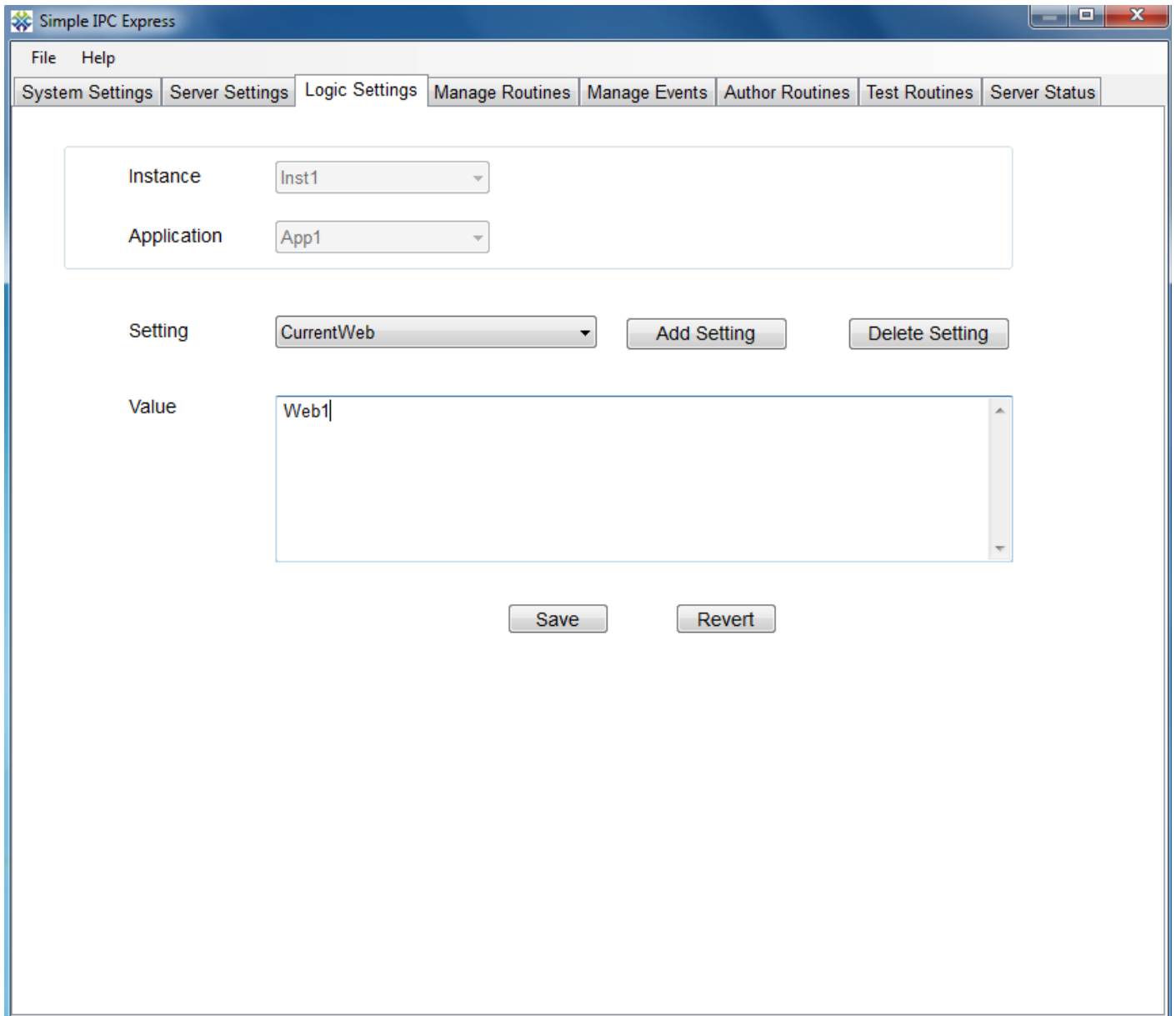
The following procedure will establish a command that can be used to retrieve a file from your computer by rendering the file to the web browser. There is a ten megabyte limit on the file size that can be retrieved. Also, not every type of file is suitable for rendering in a browser. Images and Microsoft Office documents are examples of files that generally make good candidates for rendering in a browser.

Step 1: As before, open Task Manager and view all of the processes running on your computer. Use the “Show processes from all users” option. If there are any occurrences of CTServerInst1.exe running, then terminate them.

Step 2: In this exercise, the routine will explicitly interact with the Message Peer service in order to upload a file. To do this, the routine will need to reference the collection of settings referred to as the “Web” name in the Messaging Peer console. We will create a global variable to hold the name of the “Web”. Go to the “Logic Settings” tab and click the “Add Setting” button. In the resulting dialog, enter a name of “CurrentWeb” and click “Save”.

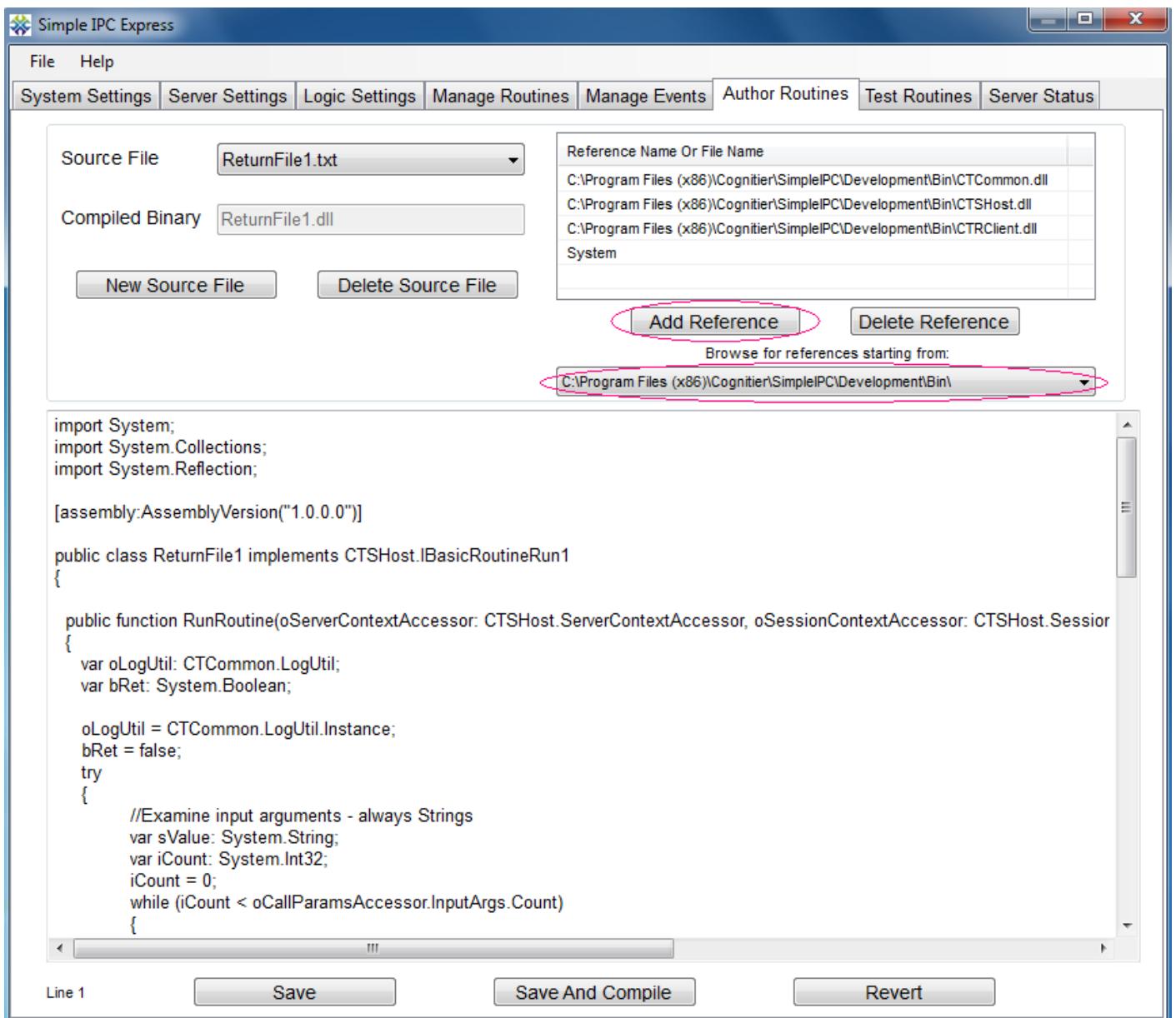


Step 3: Give the “CurrentWeb” logic setting a value of “Web1” and click the “Save” button.

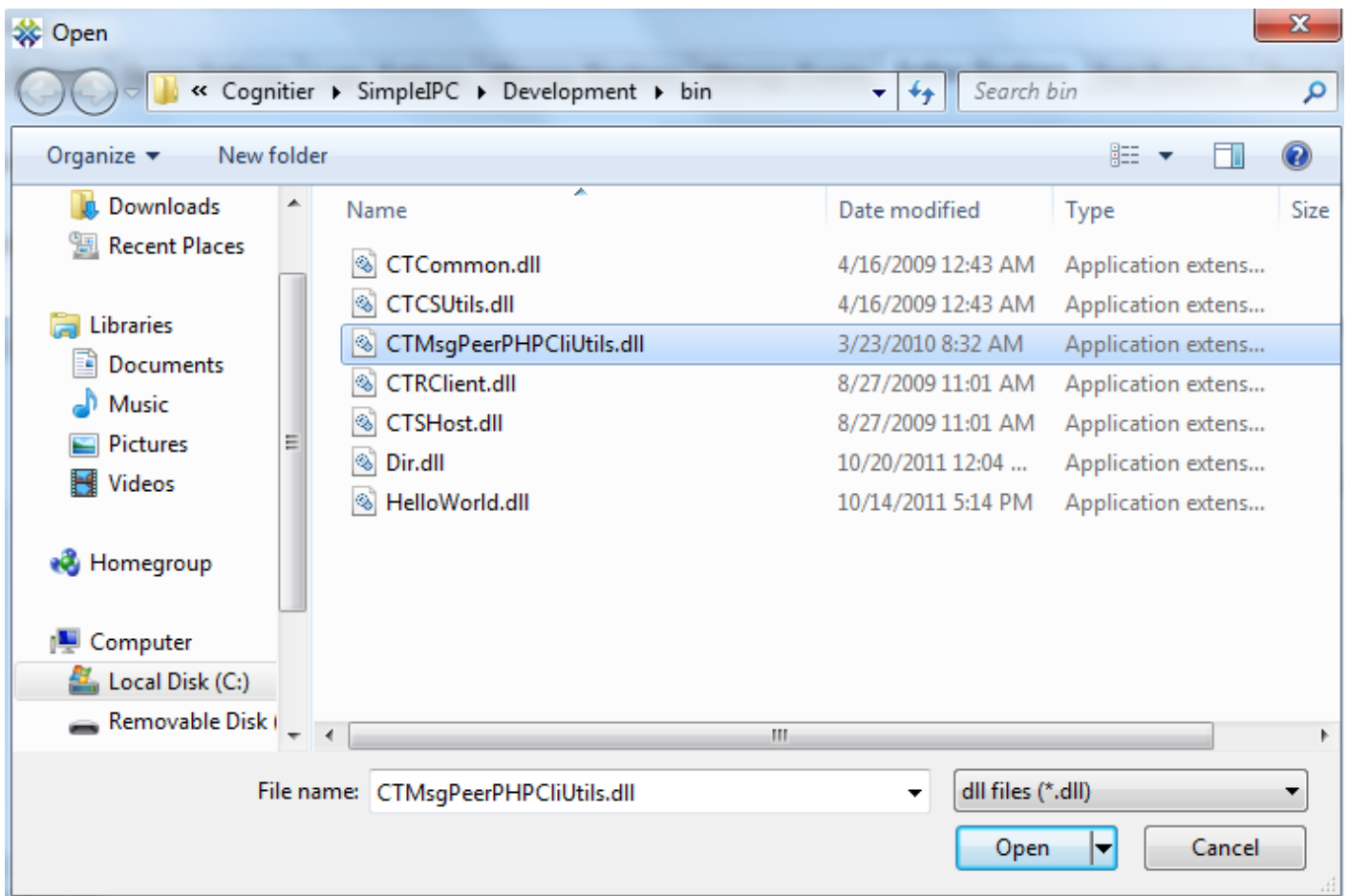


Step 4: Go to the “Author Routines” tab to enter the code for the routine that will return the contents of a file. Create a new source file with a name of “ReturnFile1”.

Step 5: In this routine, we will need to interact with the Messaging Peer facilities, and so we will need to establish a “reference” to one of the Messaging Peer assemblies. In the upper-right section of the screen, find the drop-down labeled “Browse for references starting from:” and select the \SimpleIPC\Development\Bin directory. Then click the “Add Reference” button.



Step 6: Select the CTMsgPeerPHPCliUtils.dll assembly from the list of files and click the “Open” button.



Step 7: Copy and paste the following code into the source code text area:

```
import System;
import System.Collections;
import System.Reflection;

[assembly: AssemblyVersion("1.0.0.0")]

public class ReturnFile1 implements CTSHost.IBasicRoutineRun1
{
    public function RunRoutine(oServerContextAccessor: CTSHost.ServerContextAccessor,
oSessionContextAccessor: CTSHost.SessionContextAccessor, oCallParamsAccessor:
CTSHost.CallParamsAccessor) : System.Boolean
    {
        var oLogUtil: CTCommon.LogUtil;
        var bRet: System.Boolean = false;

        oLogUtil = CTCommon.LogUtil.Instance;
        try
        {
            //We expect one input argument - the directory path and file name
            if (oCallParamsAccessor.InputArgs.Count == 0)
            {
```

```

        oCallParamsAccessor.OutputErrors.Add("No file path provided");
    }
    else
    {
        var sFilePath: System.String;
        sFilePath = oCallParamsAccessor.InputArgs[0];

        //Instantiate the client utilities object and upload the file to the server
        var oCliUtil: CTMsgPeerPHPCliUtils.ClientUtility = new
CTMsgPeerPHPCliUtils.ClientUtility();
        var oRetStruct: CTMsgPeerPHPCliUtils.UploadFileRetStruct =
oCliUtil.UploadFile(oServerContextAccessor.GetLogicSetting("CurrentWeb"), sFilePath,
"");

        //If the call to upload the file returned true, add the dynamically-
assigned remote file name to the output arguments;
        //otherwise, add the errors to the errors arguments
        if (oRetStruct.ReturnValue == true)
        {
            bRet = true;
            oCallParamsAccessor.OutputArgs.Add(oRetStruct.RemoteFileName);
        }
        else
        {
            oCallParamsAccessor.OutputErrors.Add(oRetStruct.Errors);
        }
    }
}
catch (e)
{
    //Set errors into errors collection - must be strings
    oCallParamsAccessor.OutputErrors.Add(e.Message);

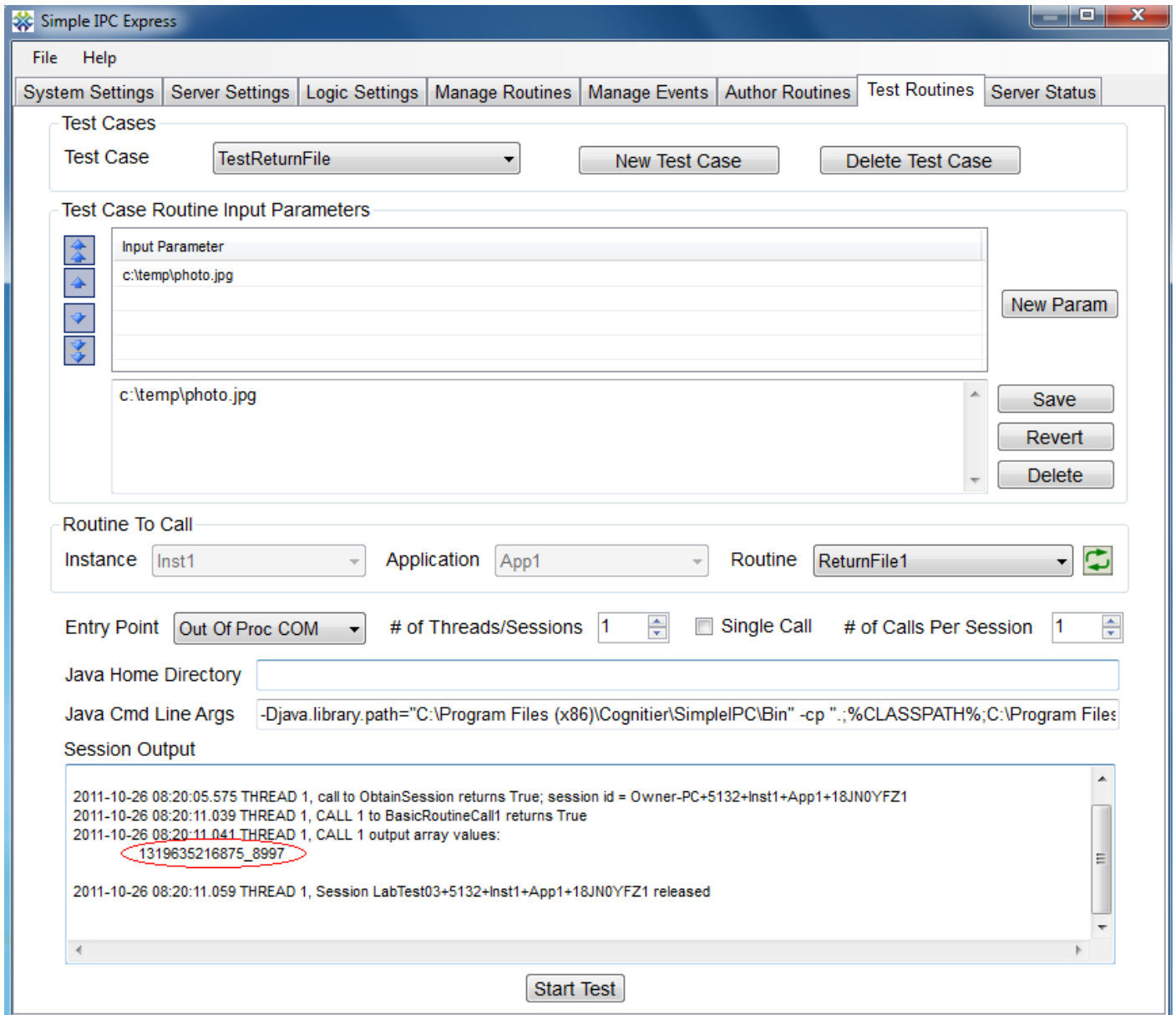
    //Also write errors to the server log file
    //Log levels are Errors (1), Warnings (2), Info (3), Verbose (4)
    oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, e);
}
return bRet;
} //end function
} //end class

```

Step 8: Click the “Save And Compile” button to compile the code. Accept the prompts to copy the compiled assembly into the alternate directory.

Step 9: Go to the “Manage Routines” tab. Click the “Refresh File List” button and select “ReturnFile1.dll” from the “File (assembly)” drop-down. Enter a Routine Name of “ReturnFile1” and click the “Save New Routine Registration” button.

Step 10: Go to the “Test Routines” tab and enter a test case with a specific file name as a parameter. Specify “ReturnFile1” as the routine to be called. Run the test and observe the output. The output should be a dynamically-generated file name for the file that was uploaded from the local computer to the “web” (i.e. a substitute file name was generated to ensure uniqueness of file names on the remote machine).



Step 11: Return to your web browser and register another routine set. Call the routine set “ReturnPhoto1” and specify the “ReturnFile1” routine. Specify the path and file name for an image file on your computer as the input parameter.

Save a routine set to call later

You can save a routine name along with some input arguments as a "routine set" (test case) to run later. This simply saves time re-entering the information.

Select an existing routine set to edit or create a new one	
Existing Routine Set Name: <input type="text" value="ReturnPhoto1"/> <input type="button" value="Delete"/>	OR New Routine Set: Set Name <input type="text"/> Routine Name <input type="text"/> <input type="button" value="Create"/>
Routine: <i>ReturnFile1</i>	
Parameters:	
<input type="text" value="c:\temp\photo.jpg"/>	<input type="button" value="Delete Selected"/>
<input type="text" value="c:\temp\photo.jpg"/>	<input type="button" value="Create New"/> <input type="button" value="Update"/>
Call a routine that returns string values	
Call a routine that renders a file to the browser	

Step 12: Click the link at the bottom of the screen that reads "Call a routine that renders a file to the browser". In the drop-down labeled "Start from an existing routine set", select "ReturnPhoto1" and click the "Apply" button. In the "File Name" field, enter the name of the file as you would like the name to be displayed in the web browser. The file name you enter here need not be the same as the name of the file on your computer, but you should give it the same file extension. In most cases, you can leave the "Content Type" field blank unless your browser is not able to determine the type of content based on the file name extension. Click the "Call Routine" button at the bottom on the form.

Call a routine which renders a file to the browser

Specify a routine and a set of input parameters for the call

Optional

Start from an existing routine set: [Register a new routine set](#)
[Call a routine that returns string values](#)

Routine:

File

Name: *This is the file name you want to see displayed by the browser. It need not be the same as the source file name.*

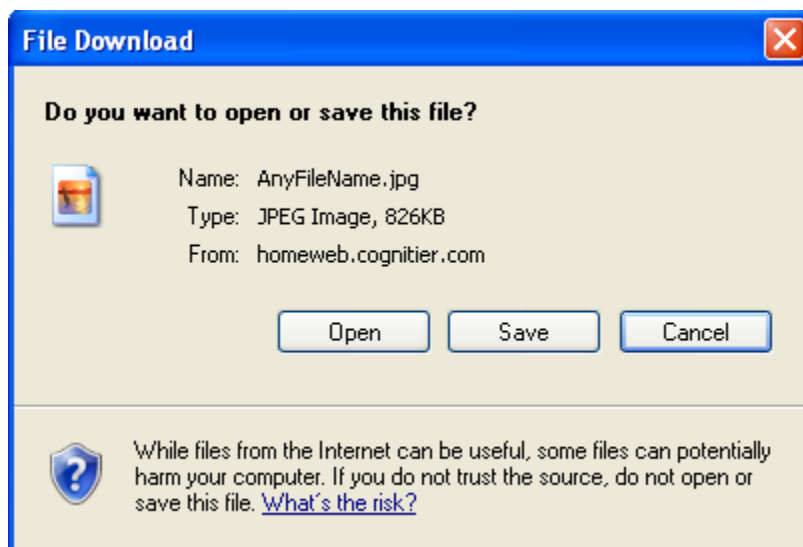
Content

Type: *This is the file content format. If blank, the content type defaults to "application/octet-stream"*

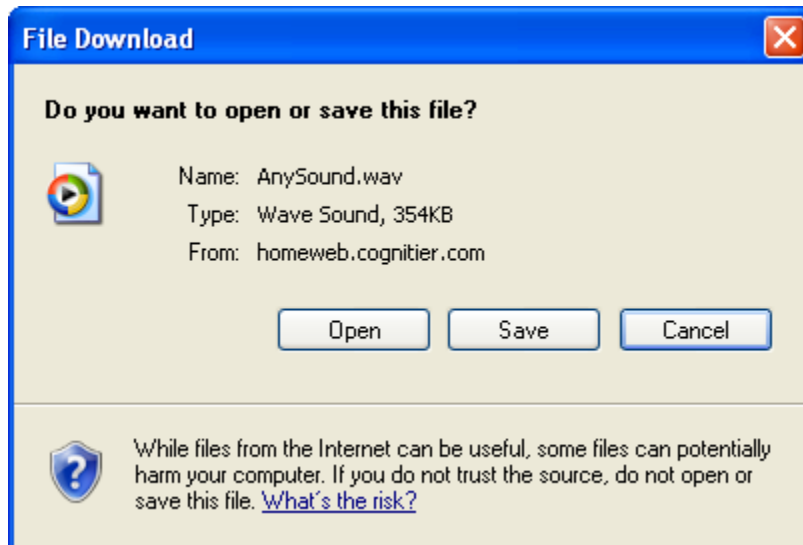
Parameters:

<input type="text" value="c:\temp\photo.jpg"/>	<input type="button" value="Delete Selected"/>
<input type="text" value="c:\temp\photo.jpg"/>	<input type="button" value="Create New"/>
	<input type="button" value="Update"/>

Step 13: Your browser should give you the option to save or download the file. An image file can be displayed directly in the browser.



Step 14: Repeat this exercise, but specify a sound file (.wav, etc.). Depending on your local computer configuration, you will most likely be prompted to save the file or play the sound with an installed media player application.



6.3 Exercise 3 – Email a File to Yourself

The following procedure will establish a command that can be used to retrieve a file from your computer by emailing the file as an attachment to a specified email address. This exercise relies on using a mail server hosted by your Internet Service Provider (ISP). Your ISP, or the recipient’s ISP may impose a limit on the size of a file that can be sent or received via email.

Step 1: As before, open Task Manager and view all of the processes running on your computer. Use the “Show processes from all users” option. If there are any occurrences of CTServerInst1.exe running, then terminate them.

Step 2: The mail server name (SMTP server) and the sender’s email address (i.e. your email address) make good candidates for global variables. Go to the “Logic Settings” tab and click the “Add Setting” button. In the resulting dialog, enter a name of “MailServer” and click “Save”. Enter the value of the mail server name (“mail.myisp.com”, etc.) and click “Save”. Create another logic setting called “SenderEmail”. Set the value to your email address (“myname@myisp.com”, etc.).

Step 3: Go to the “Author Routines” tab to enter the code for the routine that will email a file as an attachment. Create a new source file with a name of “EmailFile1”.

Step 4: Copy and paste the following code into the source code text area:

```
import System;
import System.Collections;
import System.Reflection;
```

```

[assembly:AssemblyVersion("1.0.0.0")]
public class EmailFile1 implements CTSHost.IBasicRoutineRun1
{
    public function RunRoutine(oServerContextAccessor: CTSHost.ServerContextAccessor,
oSessionContextAccessor: CTSHost.SessionContextAccessor, oCallParamsAccessor:
CTSHost.CallParamsAccessor) : System.Boolean
    {
        var oLogUtil: CTCommon.LogUtil;
        var bRet: System.Boolean;

        oLogUtil = CTCommon.LogUtil.Instance;
        bRet = false;
        try
        {
            //We expect two input arguments - the directory path and file name, and the
recipient email address
            if (oCallParamsAccessor.InputArgs.Count < 2)
            {
                oCallParamsAccessor.OutputErrors.Add("Expected two arguments");
            }
            else
            {
                var sFilePath: System.String = oCallParamsAccessor.InputArgs[0];
                var sRecipientEmail: System.String = oCallParamsAccessor.InputArgs[1];

                var iStart: int = sFilePath.LastIndexOf("\\");
                var sSubject = "Emailing: " + sFilePath.Substring(iStart + 1,
sFilePath.Length - (iStart + 1));

                var oMailClient: System.Net.Mail.SmtpClient = new
System.Net.Mail.SmtpClient();
                oMailClient.Host = oServerContextAccessor.GetLogicSetting("MailServer");

                var oMailMsg: System.Net.Mail.MailMessage = new
System.Net.Mail.MailMessage();
                oMailMsg.From = new
System.Net.Mail.MailAddress(oServerContextAccessor.GetLogicSetting("SenderEmail"));
                oMailMsg.To.Add(sRecipientEmail);
                oMailMsg.Subject = sSubject;

                var oAttachment: System.Net.Mail.Attachment = new
System.Net.Mail.Attachment(sFilePath);
                oAttachment.ContentDisposition.DispositionType =
System.Net.Mime.DispositionTypeNames.Attachment;
                oMailMsg.Attachments.Add(oAttachment);

                oMailClient.Send(oMailMsg);

                oAttachment.Dispose();
                oMailMsg.Dispose();
                bRet = true;
            }
        }
        catch (e)
        {
            //Set errors into errors collection - must be strings

```

```

oCallParamsAccessor.OutputErrors.Add(e.Message);

//Also write errors to the server log file
//Log levels are Errors (1), Warnings (2), Info (3), Verbose (4)
oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, e);
}
return bRet;
} //end function
} //end class

```

Step 5: Click the “Save And Compile” button to compile the code. Accept the prompts to copy the compiled assembly into the alternate directory.

Step 6: Go to the “Manage Routines” tab. Click the “Refresh File List” button and select “EmailFile1.dll” from the “File (assembly)” drop-down. Enter a Routine Name of “EmailFile1” and click the “Save New Routine Registration” button.

Step 7: Go to the “Test Routines” tab and enter a test case with a specific file name (the complete path and file name) as a parameter. Specify “EmailFile1” as the routine to be called. Run the test and observe the output. Wait a few minutes and check your inbox for an email message with the specified file as an attachment. Note that there are a number of reasons why the email message may not be delivered. The mail server might block the message because the attachment is too large, or it may perceive the message as being either a virus or spam. If the first message is not delivered, then try a few more times with files of different types and sizes.

If the email message is not delivered after repeated attempts, then check the antivirus configuration for the local computer. Certain antivirus applications may block outgoing email with attachments to prevent the spread of viruses. You can determine if this is the issue by temporarily turning off the antivirus application while you run the test.

Step 8: Return to your web browser and register another routine set. Call the routine set “EmailPhoto1” and specify the “EmailFile1” routine. Specify the path and file name for an image file on your computer as the first input parameter. Specify the recipient email address as the second input parameter.

Step 9: Click the link at the bottom of the screen that reads “Call a routine that returns string values”. In the drop-down labeled “Start from an existing routine set”, select “EmailPhoto1” and click the “Apply” button. Click the “Call Routine” button at the bottom on the form. Since we did not specify any output arguments in the routine, the resulting page should show headers for “Outputs”, “Errors” and “Warnings”, but there will likely be no contents. Wait a few minutes and check your email.

6.4 Exercise 4 – Call a Routine From Your Website

The previous exercises relied on a publicly-accessible website employed by the HomeWeb service to allow remote access to your computer. However, if you currently have your own website and wish to delegate certain functions to your home computer, you can invoke routines on your home computer from your own website. The

mechanism for doing this depends upon the programming language employed on your website. This exercise demonstrates the procedure for the PHP programming language.

Step 1: Go to the Cognitier web site and download the "CTWebUtils1.zip" file. Unzip this file to "CTWebUtils1.php". Upload "CTWebUtils1.php" to the computer hosting your web site. This file contains the functions you will need to interact with the HomeWeb service.

Step 2: For demonstration purposes, create a web page containing a form that can be used to request a file from your home computer. Use the following HTML:

```
<HTML>
<BODY>
<HEAD>
<TITLE>Call Cognitier Routine</TITLE>
</HEAD>
<H3>Get a file from the remote computer</H3>
<BR>
<FORM ACTION="RoutineResults1.php" METHOD="POST">
<TABLE>
<TBODY>
<TR>
<TD>File And Path:</TD><TD><INPUT TYPE="TEXT" NAME="fileandpath" SIZE="100" /></TD>
</TR>
<TR>
<TD>Recipient Email:</TD><TD><INPUT TYPE="TEXT" NAME="recipient" SIZE="100" /></TD>
</TR>
<TR>
<TD></TD><TD><INPUT TYPE="SUBMIT" VALUE="Submit" /></TD>
</TR>
</TBODY>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Step 3: Create a PHP file (specified as "RoutineResults1.php" above) which is called when the HTML form is submitted. Use the following file contents:

```
<html>
<head>
<title>File Request Results</title>
</head>
<body>
<?php
include("CTWebUtils1.php");

//Update the URL and credentials as appropriate
$url = "https://homeweb.cognitier.com/jsmith123/CTRetXML1.php";
$filepath = $_POST["fileandpath"];
if (get_magic_quotes_gpc())
{
    $filepath = stripslashes($filepath);
}
}
```

```

$postdata = array("CT_1" => $filepath, "CT_2" => $_POST["recipient"]);
$port = 443;
$timeout = 300;
//Credentials are required to interact with the HomeWeb service
//Hard-code the credentials here
$username = "jsmith123";
$password = "1mp0ss1bl3";
//Hard-code the routine to be called here
$routinename = "EmailFile1";
//Create empty arrays as by-reference arguments to the function call
$oOutargs = array();
$oErrors = array();
$oWarnings = array();
$returnerrormsg = "";

//Call the function defined in CTWebUtils1.php to invoke the private computer routine
PostReturnStrings1($url, $postdata, $port, $timeout, $username, $password,
$routinename, $oOutargs, $oErrors, $oWarnings, $returnerrormsg);

if (strlen($returnerrormsg) > 0)
{
    echo $returnerrormsg;
}
else
{
    echo "<BR><B>Outputs</B>";
    foreach($oOutargs as $oOutarg)
    {
        echo "<BR>" . $oOutarg;
    }
    echo "<BR><B>Errors</B>";
    foreach($oErrors as $oError)
    {
        echo "<BR>" . $oError;
    }
    echo "<BR><B>Warnings</B>";
    foreach($oWarnings as $oWarning)
    {
        echo "<BR>" . $oWarning;
    }
}
?>
</body>
</html>

```

Step 4: Open a web browser and navigate to the HTML file we just created.

Step 5: If the routine invocation is successful, you should receive no errors in the output. Wait a few minutes and then check your email.

Step 6: Create another HTML file for the purpose of rendering a file directly to the browser. Use the following HTML:

```
<html>
<body>
<h3>Render a file from the remote computer</h3>
<form method="post" action="FileRoutineResults1.php">
<table>
<tbody>
<tr>
<td>Source File And Path:</td>
<td><input name="fileandpath" size="100" /></td>
</tr>
<tr>
<td>File Name as Rendered:</td>
<td><input name="browserfilename" size="100" /></td>
</tr>
<tr>
<td>Content Type:</td>
<td><input name="contenttype" value="application/octet-stream" size="100" /></td>
</tr>
<tr>
<td></td>
<td><input value="Submit" type="submit" /></td>
</tr>
</tr>
</tbody>
</table>
</form>
</body>
</html>
```

```
</tbody>
</table>
</form>
</body>
</html>
```

Step 7: Create a PHP file (specified as “FileRoutineResults1.php” above) which is called when the HTML form is submitted. Use the following file contents:

```
<?php
error_reporting(0);
include("CTWebUtils1.php");

//Update the URL and credentials as appropriate
$url = "https://homeweb.cognitier.com/jsmith123/CTRetFile1.php";
//The name of the file we want is the routine input argument for ReturnFile1
$filepath = $_POST["fileandpath"];
if (get_magic_quotes_gpc())
{
    $filepath = stripslashes($filepath);
}
$postdata = array("CT_1" => $filepath);
$port = 443;
$timeout = 300;
//Credentials are required to interact with the HomeWeb service
//Hard-code the credentials here
$username = "jsmith123";
$password = "1mp0ss1bl3";
//Hard-code the routine to be called here
$routinename = "ReturnFile1";
$targetfilename = $_POST["browserfilename"];
$contenttype = $_POST["contenttype"];
$returnerrormsg = "";

//Call the function defined in CTWebUtils1.php to invoke the private computer routine
PostRenderFile1($url, $postdata, $port, $timeout, $username, $password, $routinename,
$targetfilename, $contenttype, $returnerrormsg);

if (strlen($returnerrormsg) > 0)
{
    echo $returnerrormsg;
}
?>
```

Step 8: Open a web browser and navigate to the HTML file we just created.

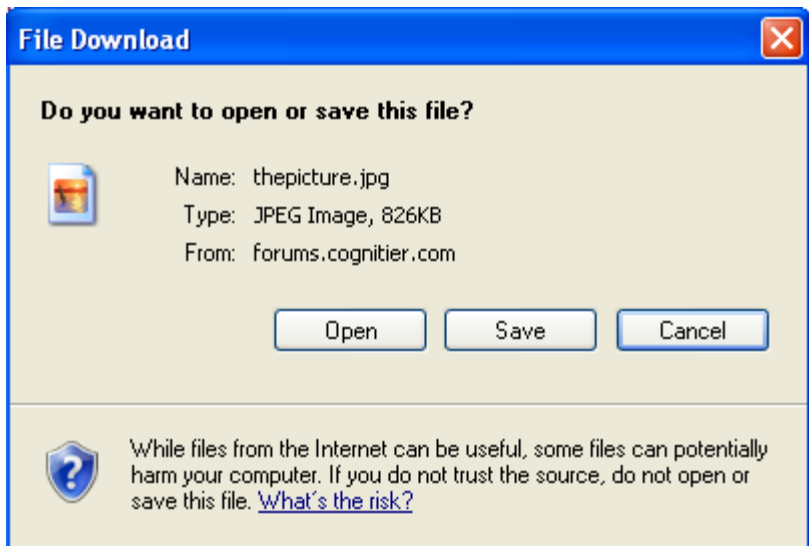
Render a file from the remote computer

Source File And Path:

File Name as Rendered:

Content Type:

Step 9: If the routine invocation is successful, you should be prompted by your web browser to open or save the file.



6.5 Sample Exercise Conclusions

The sample exercises focused on file transfer because it is a common need to obtain a file from your computer when you do not have physical access to the computer. However, the types of routines that can be authored are not limited to file transfer operations. If you have system administration operations you wish to run remotely, such as executing batch files, then this can be accomplished via the HomeWeb service. Likewise, if you are a website developer and wish to augment the functionality of your website by delegating certain operations to a private computer, then the HomeWeb service can provide a means to do so – comparable to calling web services against the private computer. The sample exercises demonstrated the code for the case where your website’s programming language is PHP. However, any programming language capable of performing an HTTP POST operation may interact with the HomeWeb service.

7 Getting Additional Support

Please send email to support@cognitier.com in the event that you run into difficulties getting signed up for the HomeWeb service, or if you have trouble installing the products or getting them configured. If you need developer assistance, such as sample code for writing a routine to perform a specific function on your computer, you may send email or post a question on the discussion forum.