



Cognitier SimpleIPC

Creating MS Word Doc from Java Servlet

Sample

Version 1.0.0.1

May 13, 2009

Copyright © 2009 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.

Creating MS Word Doc from Java Servlet Sample

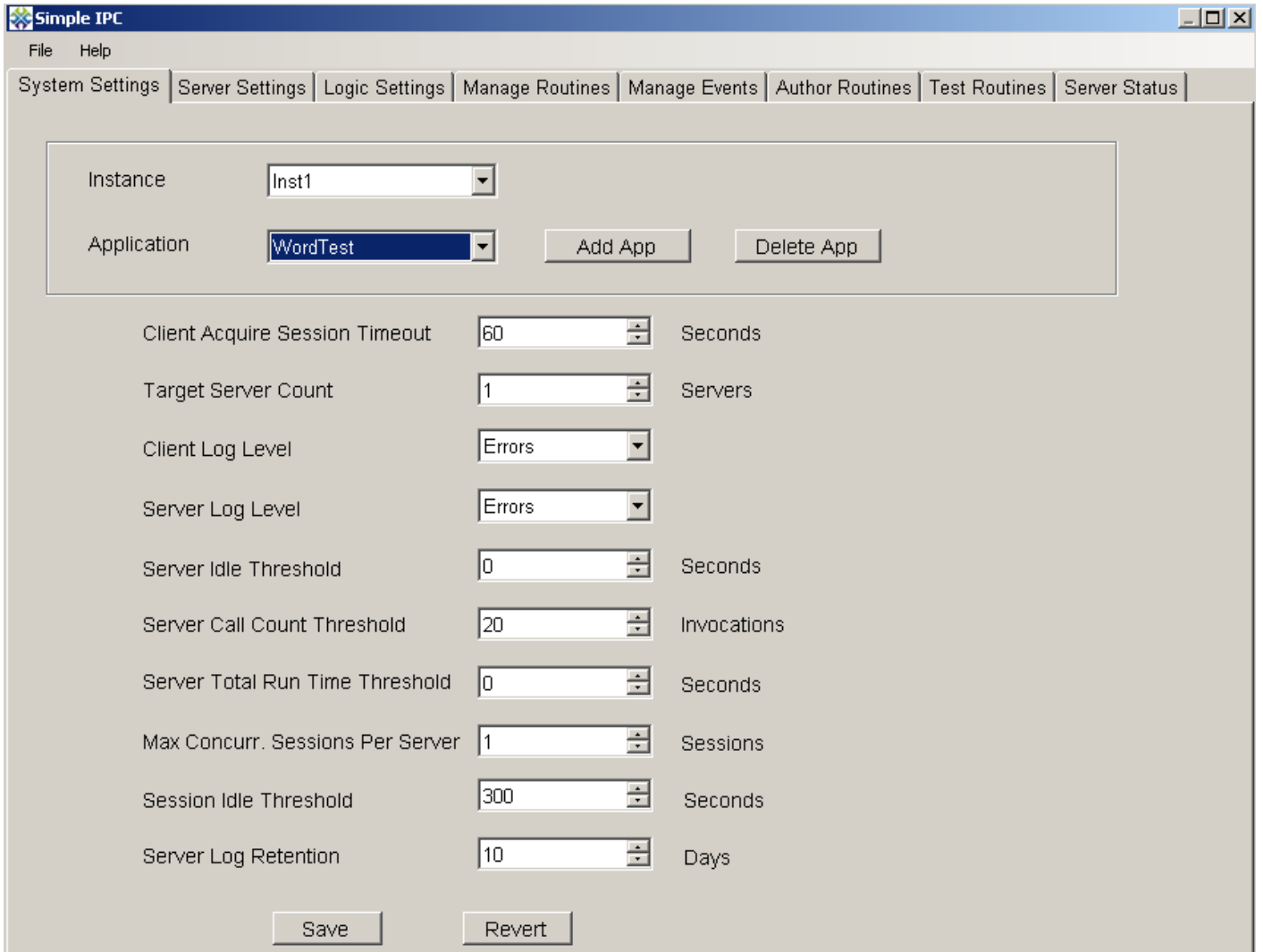
Because of the wealth of existing applications based on Java, .NET, and COM, there is frequently a need to achieve interoperability between these languages. With SimpleIPC, your server code must be written in a .NET programming language, but within your .NET code you can take advantage of COM objects, just as you can in any other .NET application. Additionally, you can call into your IPC servers from a variety of languages, including Java. In this example, we will demonstrate making the call from a Java application, in this case a Java servlet running inside a Tomcat web server, into our custom .NET code which is running inside a SimpleIPC server.

Within the .NET code, we will use COM automation to create, save, and return a Microsoft Word document which will then be returned to the servlet and will be made available for the web user to save or view. In this example, we will impose the restriction that we want just one instance of the COM object in memory, one instance of the winword.exe executable running on the server, and that we want to keep re-using the same Word document on the server, rather than saving a new file for every visitor. Despite these restrictions, our web application will still be able to service multiple visitors – we just disallow concurrent access to the COM object and Word document. In this example, our servlet will collect the web visitor's name and pass it to the IPC server. The IPC server code will incorporate the visitor's name into the Word document before returning that Word document.

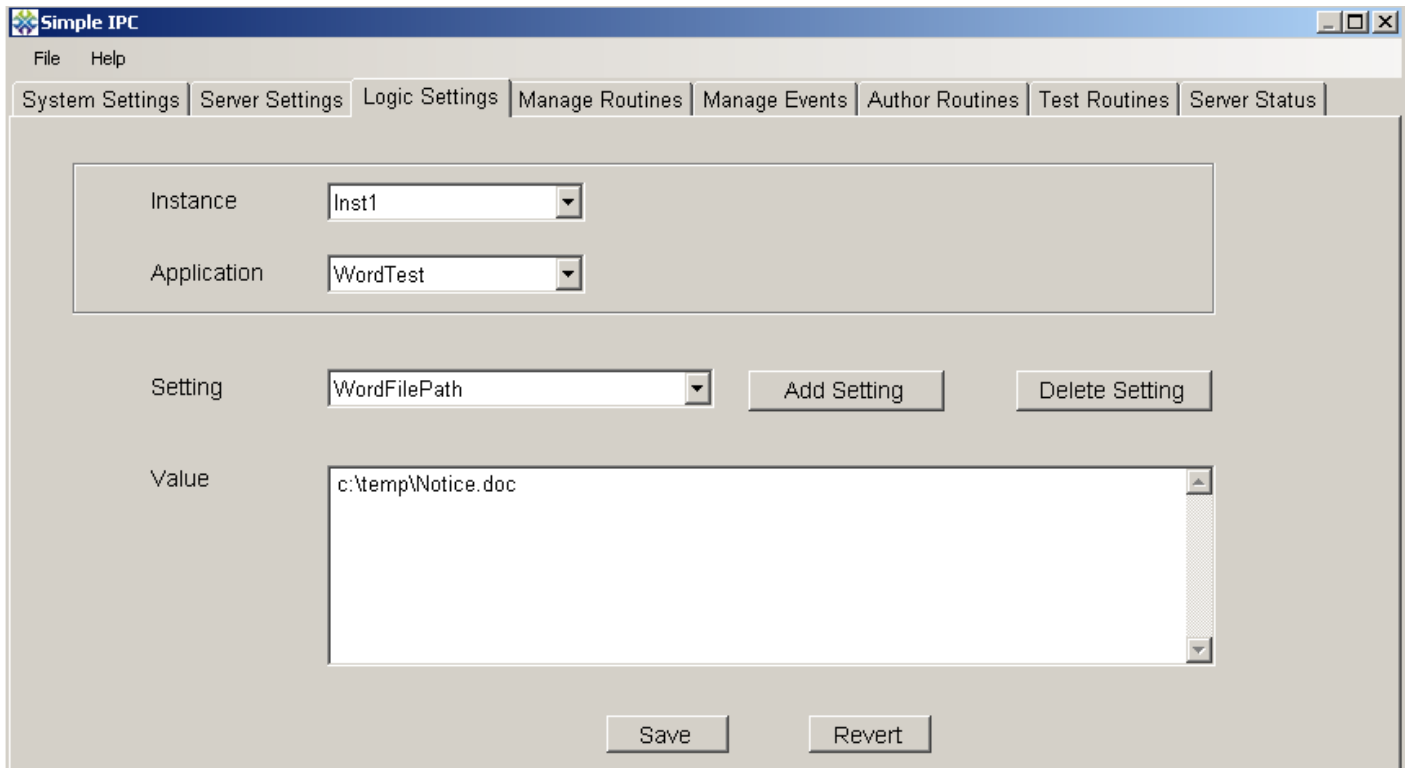
The sample code omits many error-checking operations in order to make the code more readable. Be aware of word-wrapping if you copy and paste sample code from this document.

Step 1: Go to the System Settings tab in the CTConsole and create a new application in SimpleIPC for this exercise. Call it WordTest. Set the following performance parameters for the application:

Target Server Count	1
Server Idle Threshold	0 (infinite)
Server Call Count Threshold	20
Server Total Run Time Threshold	0 (infinite)
Max Concurr. Sessions Per Server	1
Session Idle Threshold	300



Step 2: Create a Logic Setting for this new application with the path and file name to use when saving the Word document. Go to the Logic Settings tab and toggle the Instance selection in order to refresh the list of Applications. Select the WordTest application and enter the new Logic Setting. Call the Logic Setting WordFilePath, enter a value appropriate for your environment, and save the value.



Step 3: Go to the Author Routines tab and create the routine that will execute when an IPC server starts. Create a new routine called WordTestAppStart using the BasicServerOp1 template. Enter the following code to start a worker thread which establishes an instance of the Word Automation COM object and then processes incoming client requests.

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

[assembly: AssemblyVersion("1.0.0.0")]

public class WordTestAppStart implements CTSHost.IBasicServerOp1
{
    static var mServerContextAccessor: CTSHost.ServerContextAccessor;
    public function RunRoutine(oServerContextAccessor:
CTSHost.ServerContextAccessor) : System.Boolean
    {
        WordTestAppStart.mServerContextAccessor = oServerContextAccessor;
        var oLogUtil: CTCommon.LogUtil;
```

```

var bRet: System.Boolean;

oLogUtil = CTCommon.LogUtil.Instance;
bRet = false;
try
{
    //Start a worker thread to keep running through the server's lifespan
    var oWorkerThread: System.Threading.Thread = new
System.Threading.Thread(System.Threading.ThreadStart(ProcessInboundRequests));
    oWorkerThread.SetApartmentState(System.Threading.ApartmentState.STA);
    oWorkerThread.Start();
    bRet = true;
}
catch (e)
{
    //Write errors to the server log file
    oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, e);
}
return bRet;
} //end function

static function ProcessInboundRequests()
{
    var oLogUtil:
CTCommon.LogUtil; oLogUtil = CTCommon.LogUtil.Instance;
    //Instantiate the MS Word COM Object
    var oWordApp = new ActiveXObject("Word.Application");
    var oDoc = null;
    var oSelection = null;

    try
    {
        //Cast the argument object back to the ServerContextAccessor
        var oServerContextAccessor: CTSHost.ServerContextAccessor =
WordTestAppStart.mServerContextAccessor;

```

```

    //Put a boolean variable in the context to tell us when to end thread
execution
    var bServerShuttingDown: System.Boolean = false;
    oServerContextAccessor.SetServerContextItem("ServerShuttingDown",
bServerShuttingDown);
    //Get the Logic Setting value with the file path
    var sFilePath: System.String =
oServerContextAccessor.GetLogicSetting("WordFilePath");

    //Loop while the server doesn't appear to be shutting down
var sRequestID: System.String = String.Empty;
var sRequestPayload: System.String = String.Empty;
var bFileBytes: System.Byte[];
var oFileStream: System.IO.FileStream = null;
var iFileLength: int = 0;
var iCount: int = 0;
var oStringBuilder: System.Text.StringBuilder = new
System.Text.StringBuilder();
    var htServerContextItems: Hashtable;
    htServerContextItems = oServerContextAccessor.GetServerContextItems();
    bServerShuttingDown =
System.Boolean(htServerContextItems["ServerShuttingDown"]);

while (bServerShuttingDown== false)
{
    //Look for new requests queued in the server context
sRequestID = String.Empty;
for (var sKeyName in htServerContextItems.Keys)
{
    if (sKeyName.StartsWith("REQ:"))
    {
        sRequestID = sKeyName;
        break;
    }
}
if (sRequestID.Length > 0)
{

```

```

//Type some text into a Word document and save it
sRequestPayload = System.String(htServerContextItems[sRequestID]);
oServerContextAccessor.RemoveServerContextItem(sRequestID);
//Set initial Word document settings
oDoc = oWordApp.Documents.Add();
oSelection = oWordApp.Selection;
oSelection.Font.Name = "Verdana";
oSelection.Font.Size = "12";
oDoc.Select();
oSelection.TypeText("Dear ");
oSelection.TypeText(sRequestPayload);
oSelection.TypeText(",");
oSelection.TypeParagraph();
oSelection.TypeText("Thank you for using our application.");
oSelection.TypeParagraph();
oSelection.TypeText("Sincerely,");
oSelection.TypeParagraph();
oSelection.TypeText("The app support team");
oDoc.SaveAs(sFilePath);
oDoc.Close();
//Put a response corresponding to the request into the server
context with the resulting Word doc bytes
oFileStream = new System.IO.FileStream(sFilePath,
System.IO.FileMode.Open);
iFileLength = int(oFileStream.Length);
bFileBytes = new System.Byte[iFileLength];
oFileStream.Read(bFileBytes, 0, iFileLength);
oFileStream.Close();
oStringBuilder.Remove(0, oStringBuilder.ToString().Length);
iCount = 0;
while (iCount < iFileLength)
{
    oStringBuilder.Append(GetHexStringFromByte(bFileBytes[iCount]));
    iCount++;
}
oServerContextAccessor.SetServerContextItem("RESP" +
sRequestID.Substring(3, sRequestID.Length - 3), oStringBuilder.ToString());

```

```

        }
        System.Threading.Thread.Sleep(400);
        htServerContextItems =
oServerContextAccessor.GetServerContextItems();
        bServerShuttingDown =
System.Boolean(htServerContextItems["ServerShuttingDown"]);
    } //end while
}
catch (e)
{
    //Write errors to the server log file
    oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, "Error in ProcessInboundRequests
thread: " + e);
}
//When the loop exits, perform cleanup operations - in production, do this
when an exception is thrown as well
oSelection = null;
oDoc = null;
oWordApp.Application.Quit();
} //end function

static function GetHexStringFromByte(bByte: System.Byte) : System.String
{
    var oLogUtil:
CTCommon.LogUtil; oLogUtil = CTCommon.LogUtil.Instance;
    var sRet: System.String = String.Empty;
    var sHexChars: System.String = "0123456789ABCDEF";
    var cHexChars: System.Char[] = sHexChars.ToCharArray();
    var iByteVal: int = int(bByte);
    if (iByteVal < 16)
    {
        sRet = "0";
    }
    else
    {
        sRet = sRet + cHexChars[int(System.Math.Floor(iByteVal / 16))];
    }
}

```

```

    }
    sRet = sRet + cHexChars[iByteVal % 16];
    return sRet;
}
} //end class

```

Step 4: Register the routine we just created. Go to the Manage Routines tab of the CTConsole and click the Refresh File List button under the “File (Assembly)” drop-down. Select the WordTestAppStart.dll dll, enter a name of WordTestAppStart and click Save New Routine Registration.

The screenshot shows the 'Simple IPC' application window. The 'Manage Routines' tab is active, displaying a table of existing routines and a form for registering a new one.

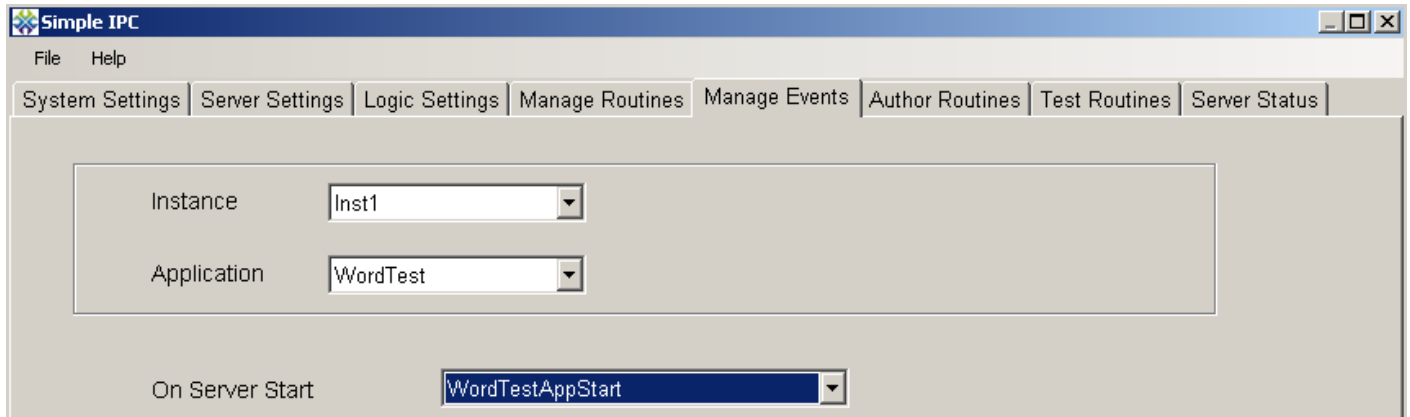
RoutineName	FileName	Class Name	Interface Name	Used w/ Events?
MCSessionEnd	MCSessionEnd.dll	MCSessionEnd	IBasicSessionOp1	no
MCSessionStart	MCSessionStart.dll	MCSessionStart	IBasicSessionOp1	no
PooledObjBasicCall	PooledObjBasicCall.dll	PooledObjBasicCall	IBasicRoutineRun1	no
PooledObjSvrEnd	PooledObjSvrEnd.dll	PooledObjSvrEnd	IBasicServerOp1	yes
PooledObjSvrStart	PooledObjSvrStart.dll	PooledObjSvrStart	IBasicServerOp1	yes
TestRoutine1	TestRoutines.dll	TestRoutines.TestRunRou...	IBasicRoutineRun1	no
WordTestAppEnd	WordTestAppEnd.dll	WordTestAppEnd	IBasicServerOp1	yes
WordTestAppStart	WordTestAppStart.dll	WordTestAppStart	IBasicServerOp1	yes
XMLTest1	XMLTest1.dll	XMLTest1	IBasicRoutineRun1	no

Buttons: Refresh List, Delete Routine Record

Register New Routine

File (assembly): WordTestAppStart.dll
 Refresh File List (D:\Formal_runtime\Development\Bin\)
 Type (class name): WordTestAppStart
 Interface: IBasicServerOp1
 Routine Name: WordTestAppStart
 Save New Routine Registration

Step 5: Associate the server start-up routine with the “On Server Start” event for our application. Go to the Manage Events tab and toggle the Instance selection to refresh the list in the Applications drop-down. Select the WordTest application and associate the WordTestAppStart routine with the “On Server Start” event.



Step 6: Go to the Author Routines tab and create a new source code file with a name of InvokeWordTestApp. Use the BasicRoutineRun1 template. Add a reference to C:\WINNT\Microsoft.NET\Framework\v2.0.50727\Microsoft.VisualBasic.dll (needed for some date/time functions). For this exercise, enter the following code to set a request into the server context to be picked up by the worker thread, and then periodically check for a response and return the text of the response in the output array. In this exercise, we want to ensure that there is only one IPC server for this Instance/Application combination, so we will terminate the server when we are within one call of the call count threshold.

```
import System;
import System.Collections;
import System.Reflection;
import Microsoft.VisualBasic;

[assembly: AssemblyVersion("1.0.0.0")]

public class InvokeWordTestApp implements CTSHost.IBasicRoutineRun1
{
    public function RunRoutine(oServerContextAccessor:
CTSHost.ServerContextAccessor, oSessionContextAccessor:
CTSHost.SessionContextAccessor, oCallParamsAccessor:
CTSHost.CallParamsAccessor) : System.Boolean
```

```

{
    var REQUESTTIMEOUT: int = 120;
    var oLogUtil: CTCommon.LogUtil;
    var bRet: System.Boolean;

    oLogUtil = CTCommon.LogUtil.Instance;
    bRet = false;
    try
    {
        //Check if the server is within one call of being at the call count limit.
        //oServerContextAccessor.ServerCallCount is the current cumulative number
of calls for the server.
        //oServerContextAccessor.ServerCallCountLimit is the System Setting for
the call count threshold
        if (oServerContextAccessor.ServerCallCount >=
(oServerContextAccessor.ServerCallCountLimit - 1))
        {
            //Set the boolean context variable to true to let the worker thread
know the server is exiting
            var bServerShuttingDown: System.Boolean = true;
            oServerContextAccessor.SetServerContextItem("ServerShuttingDown",
bServerShuttingDown);
            //Wait for two seconds to let the worker thread dispose of the COM
object and then terminate the server
            System.Threading.Thread.Sleep(2000);
            System.Diagnostics.Process.GetCurrentProcess().Kill();
        }
    }
    else
    {
        ///Make up a request id from the session id and the tick count
        //Put it in the server context with the first input parameter to this
routine call if there is one
        var sBaseRequestID: System.String = oSessionContextAccessor.SessionID
+ "::" + System.Environment.TickCount;
        var sRequestID: System.String = "REQ:" + sBaseRequestID;
        var sRequestExtraInfo: System.String = String.Empty;
        if (oCallParamsAccessor.InputArgs.Count > 0)
        {

```

```

        sRequestExtraInfo = oCallParamsAccessor.InputArgs[0];
    }
    oServerContextAccessor.SetServerContextItem(sRequestID,
sRequestExtraInfo);
    //Wait for the server's worker thread to process the request and put
the response in the server context
    var sResponseID: System.String = "RESP::" + sBaseRequestID;
    var sResponsePayload: System.String = String.Empty;

    var dStart: System.DateTime = System.DateTime.Now.ToUniversalTime();
    var iDateDiffSecs: int = 0;
    var htServerContextItems: Hashtable;
    while ( iDateDiffSecs < REQUESTTIMEOUT)
    {
        htServerContextItems =
oServerContextAccessor.GetServerContextItems();
        if (htServerContextItems[sResponseID] != null)
        {
            //If the response is present, remove it from the context and
exit the loop
            sResponsePayload =
System.String(htServerContextItems[sResponseID]);
            oServerContextAccessor.RemoveServerContextItem(sResponseID);
            break;
        }
        System.Threading.Thread.Sleep(400);
        iDateDiffSecs = int(DateAndTime.DateDiff(DateInterval.Second,
dStart, System.DateTime.Now.ToUniversalTime()));
    }
    //Return the response to the calling program - it could be empty if
the request timed out
    oCallParamsAccessor.OutputArgs.Add(sResponsePayload);

    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
    oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, e);
    }
    return bRet;
} //end function
} //end class

```

Step 7: Register the routine we just created. Go to the Manage Routines tab of the CTConsole and click the Refresh File List button under the “File (Assembly)” drop-down. Select the InvokeWordTestApp.dll dll, enter a name of InvokeWordTestApp and click Save New Routine Registration.

Step 8: Go to the Author Routines tab and create the routine that will execute when an IPC server exits. Create a new routine called WordTestAppEnd using the BasicServerOp1 template. Enter the following code to set a Boolean variable in the server context which indicates to the worker thread that it should quit the Microsoft Word application and exit.

```

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

[assembly:AssemblyVersion("1.0.0.0")]

public class WordTestAppEnd implements CTSHost.IBasicServerOp1
{
    public function RunRoutine(oServerContextAccessor:
CTSHost.ServerContextAccessor) : System.Boolean
    {
        var oLogUtil: CTCommon.LogUtil;
        var bRet: System.Boolean;

        oLogUtil = CTCommon.LogUtil.Instance;
        bRet = false;
        try
        {

```

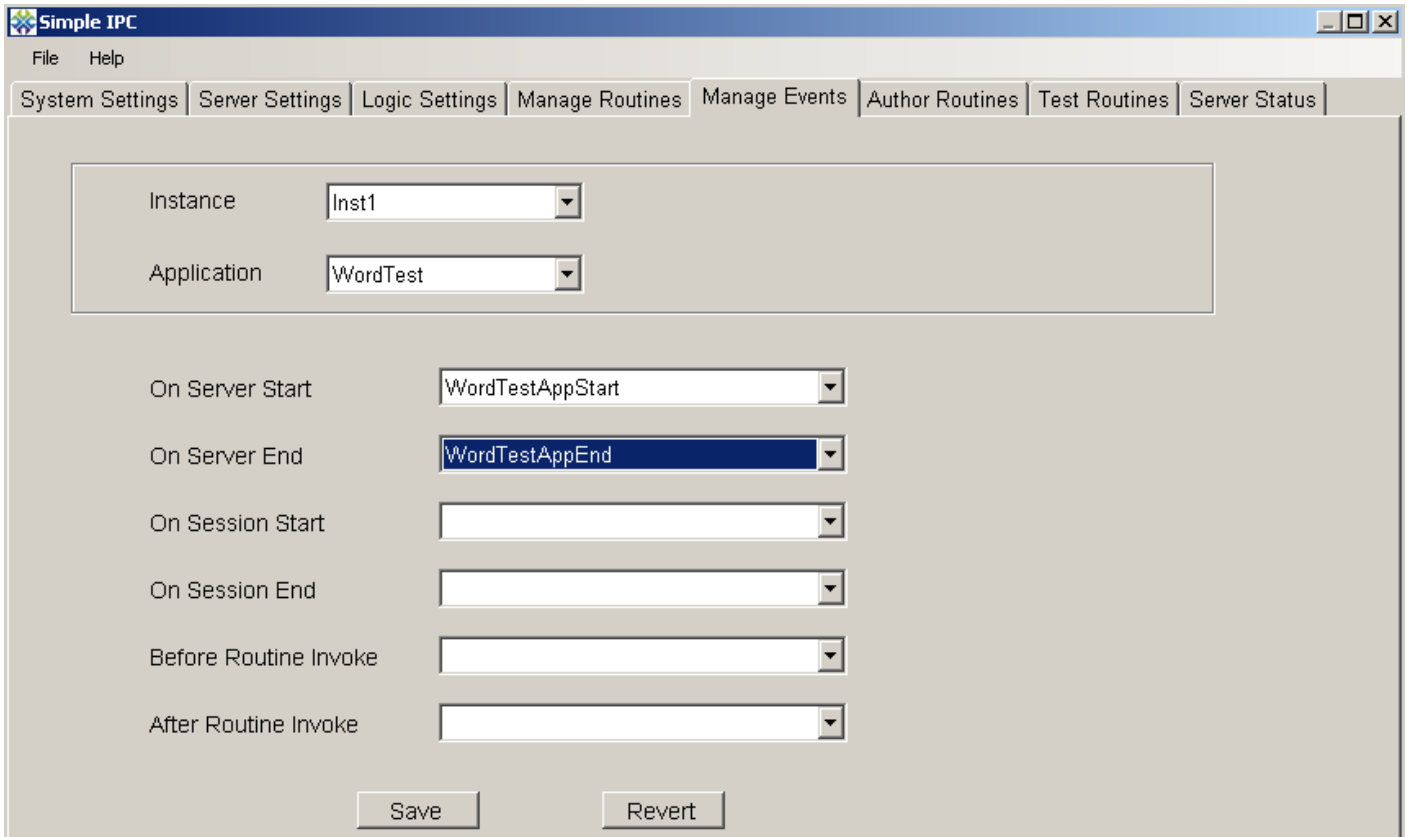
```

        //Set the boolean context variable to true to let the worker thread know
the server is exiting
        var bServerShuttingDown: System.Boolean = true;
        oServerContextAccessor.SetServerContextItem( "ServerShuttingDown",
bServerShuttingDown);
        //Wait for two seconds
        System.Threading.Thread.Sleep(2000);
        bRet = true;
    }
    catch (e)
    {
        //Write errors to the server log file
        oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, e);
    }
    return bRet;
} //end function
} //end class

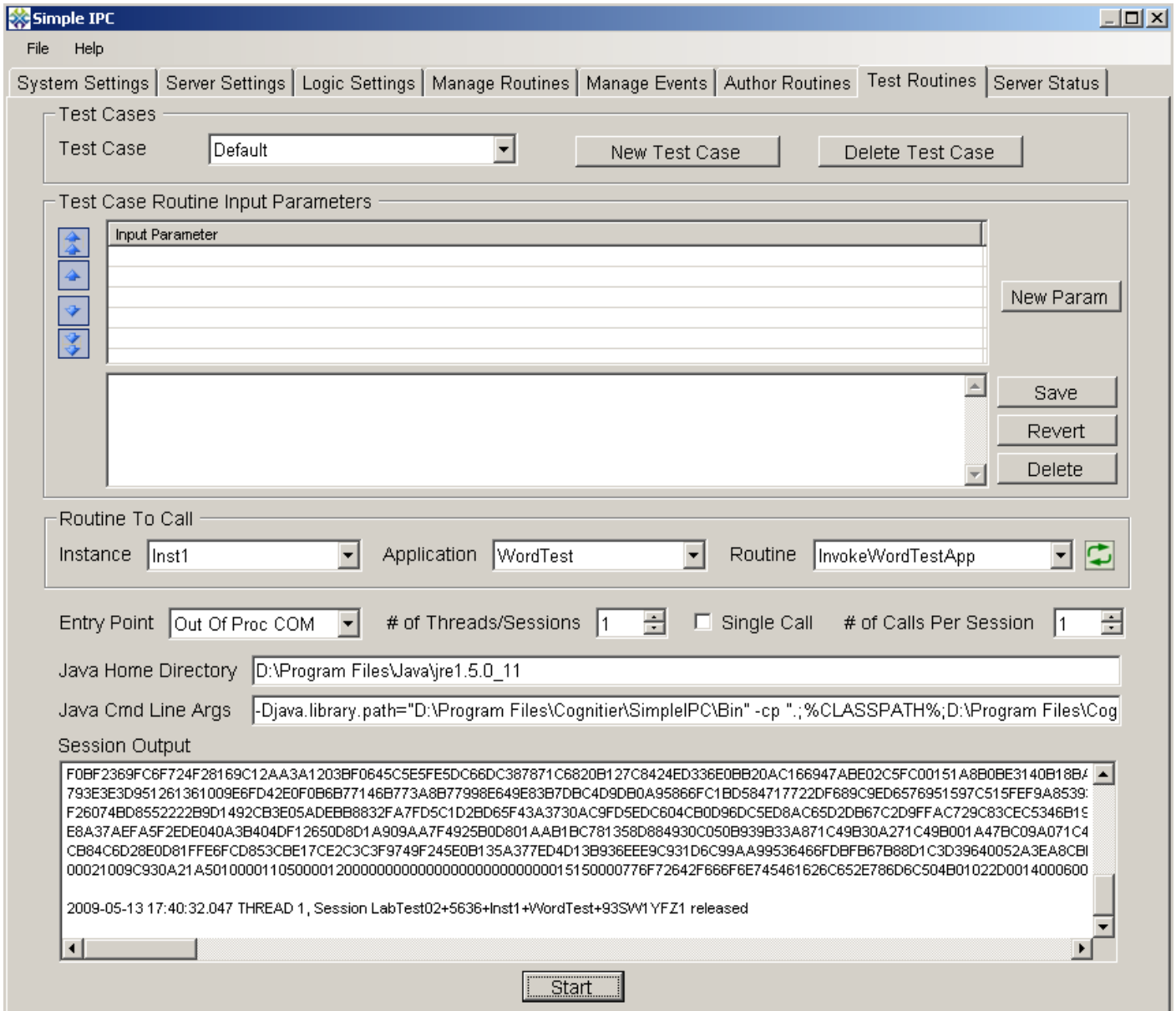
```

Step 9: Register the routine we just created. Go to the Manage Routines tab of the CTConsole and click the Refresh File List button under the “File (Assembly)” drop-down. Select the WordTestAppEnd.dll dll, enter a name of WordTestAppEnd and click Save New Routine Registration.

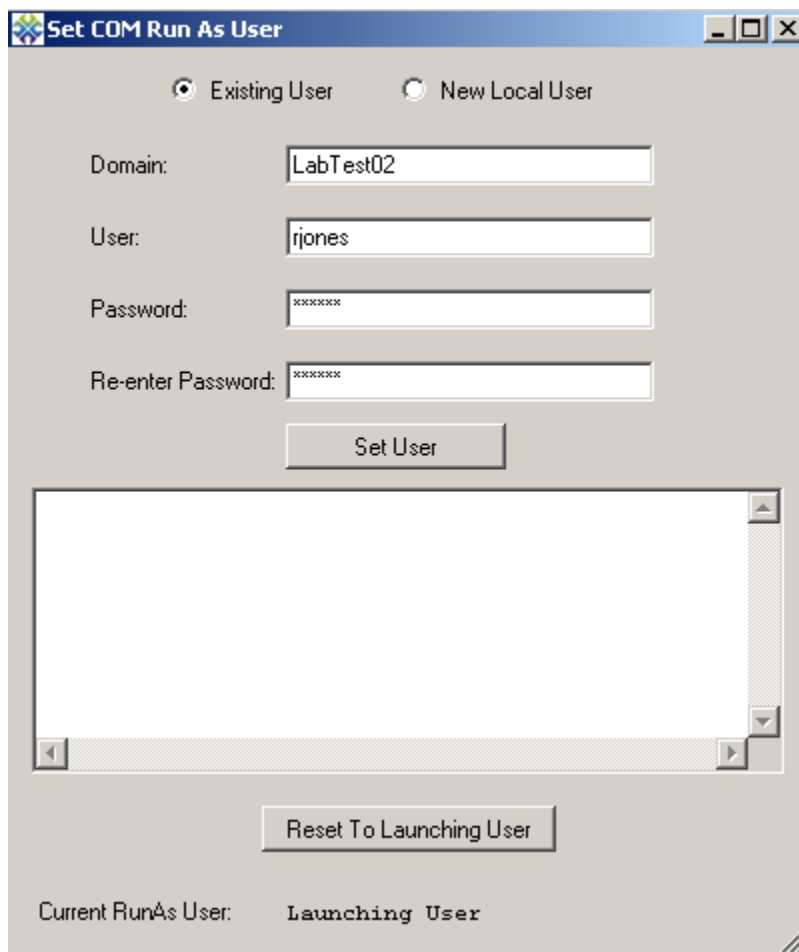
Step 10: Associate the server exit routine with the “On Server End” event for our application. Go to the Manage Events tab and toggle the Application selection to “App1” and then “WordTestApp” to refresh the selections in the routine drop-downs. Associate the WordTestAppEnd routine with the “On Server End” event of the WordTestApp application.



Step 11: Test your work up to this point. Go to the Test Routines tab and toggle the Instance selection from Inst1 to Inst2 and back to Inst1 to refresh the list of Applications. Select the WordTestApp application. Click the refresh button to the right of the Routines drop-down and then select the InvokeWordTestApp routine. Designate one session making one call to the routine. The other selections on the form are not important. Click the Start button to run the test and make sure you obtain a long string of characters as the result (i.e. a string representation of the bytes of the Word document).

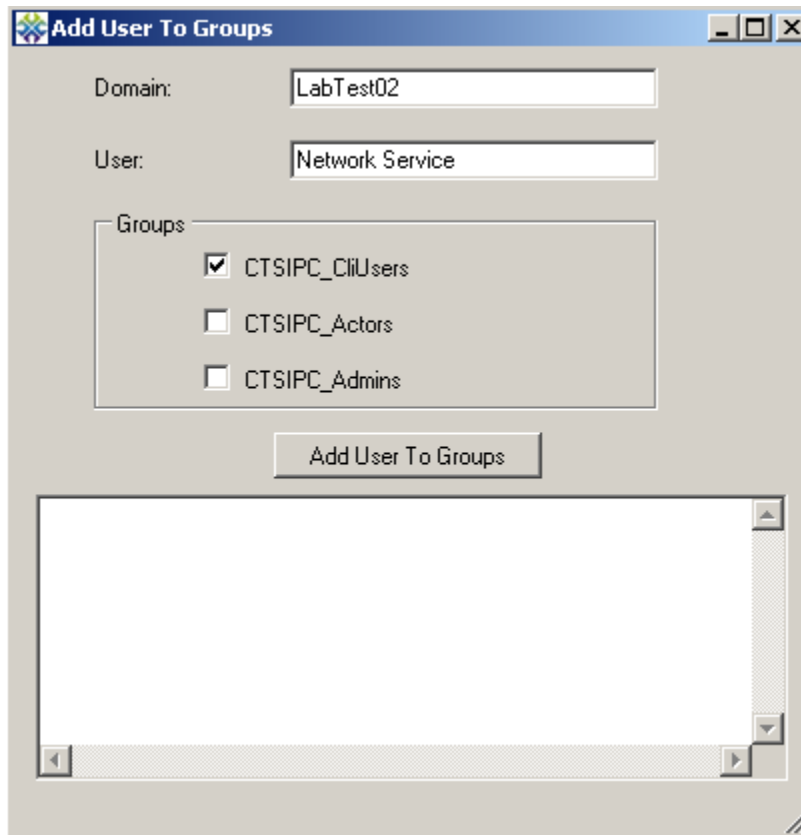


Step 12: A web application is an example of a situation where we want the client application and the IPC server running under different user accounts. We want the IPC server to run under a fixed account with membership in the CTSIPC_Actors group, and we want the user account under which the web application is running to be a member of the CTSIPC_CliUsers group. If you have not already done so, configure the out-of-process COM servers to run under a fixed identity. Select File>>Set COM Run As User from the CTConsole menu. Enter the credentials for either an existing user account, or create a new local user.

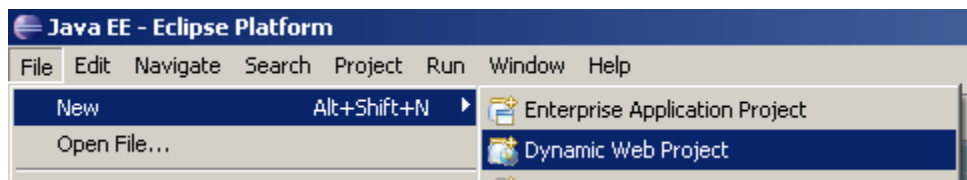


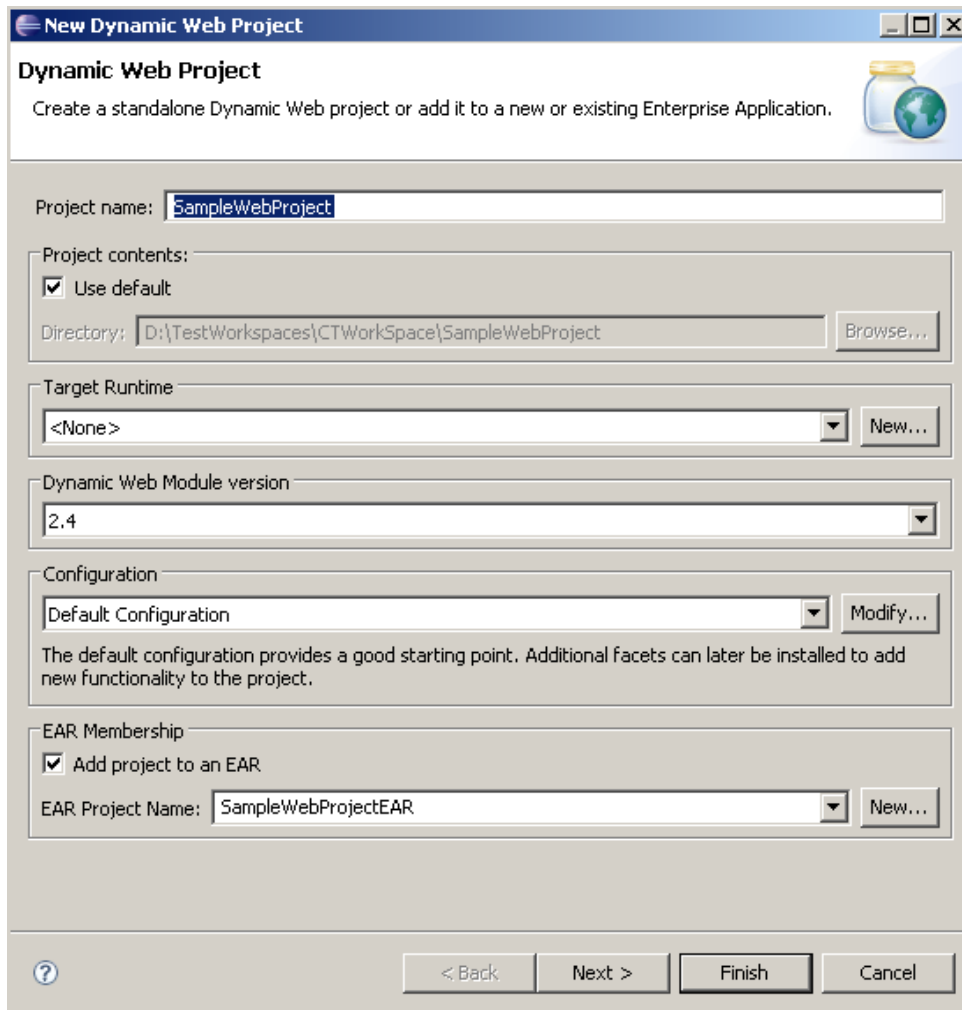
Since we have changed the COM server configuration, the computer should be rebooted at this point. When the computer is booted up again, go to the Test Routines tab of the CTConsole and select the HelloWorld routine. Run a test with the Out Of Proc COM Entry Point with the Inst1 instance, and then repeat the test with the Inst2 instance. This will verify that the COM server registration information is intact after the configuration change.

Step 13: Later in this exercise, we will be starting the Tomcat web server via a batch file. The server will run under the account of the user who executed the batch file. In production, you may configure the Tomcat server to start by another means and under a different set of credentials (presumably an account without elevated privileges). The account under which the Tomcat server will be running must be a member of the CTSIPC_CliUsers group. If, for example, you will have Tomcat run under the "Network Service" account, then choose File>>Add User To Groups from the CTConsole menu and add the "Network Service" account to the CTSIPC_CliUsers group.

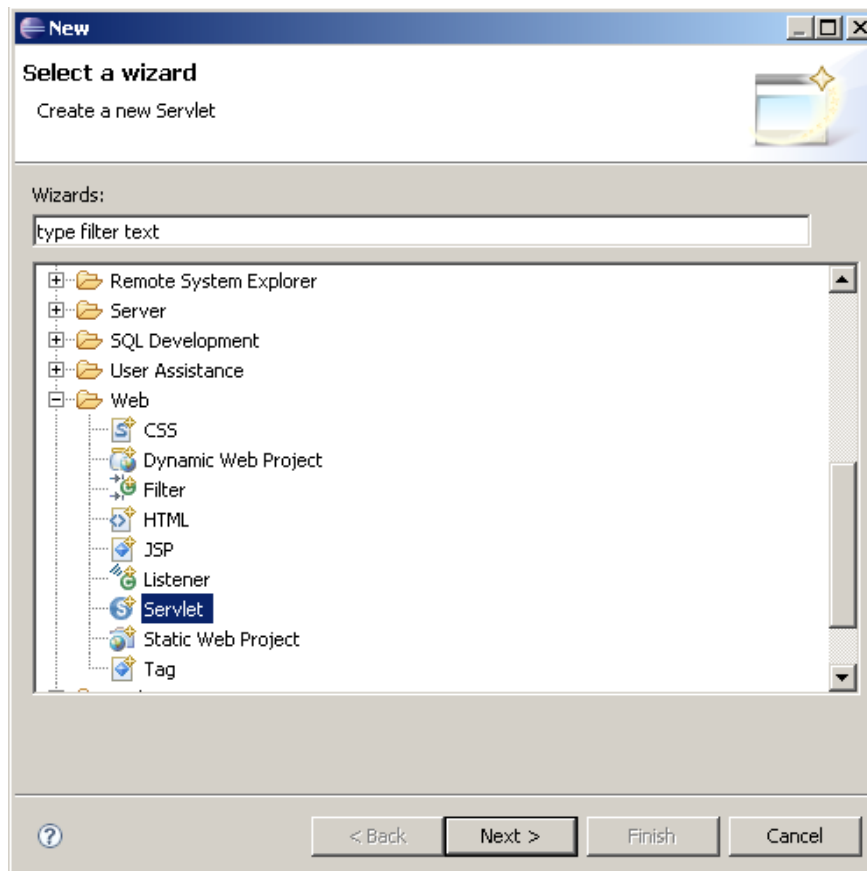
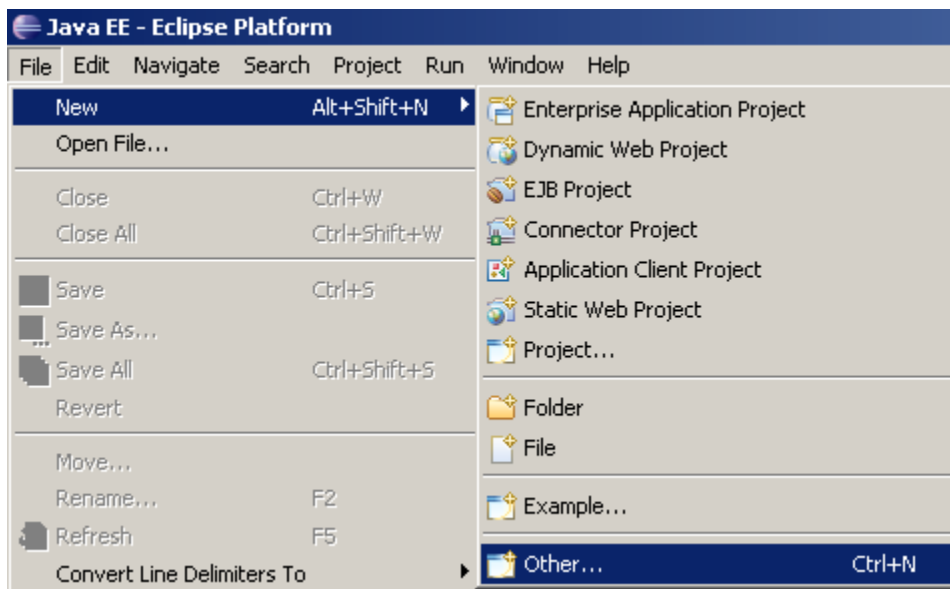


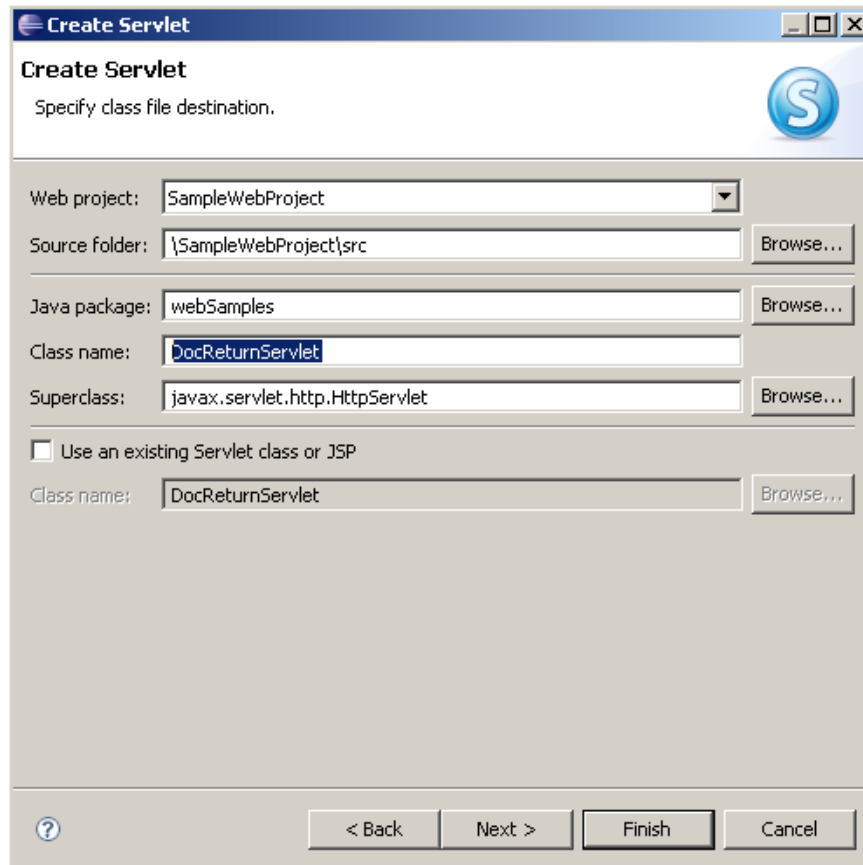
Step 14: We will now need to use Eclipse (or another Java development environment) to build the Java servlet which will be interacting with SimpleIPC. Launch Eclipse and select File>>New>>Dynamic Web Project from the menu. In the resulting dialog, name the project "SampleWebProject" and check the checkbox to add the project to an EAR. Then click Finish.





Step 15: From the Eclipse menu, select File>>New>>Other. In the resulting dialog, select Web>>Servlet and click Next. On the next page, name the package “webSamples” and name the class “DocReturnServlet” and click Finish.





Step 16: Eclipse should next open the source code file DocReturnServlet.java. If the resulting generated code shows errors regarding resolving class names starting with “javax.servlet”, then right-click on SampleWebProject and select Properties. Then select “Java Build Path” and select the Libraries tab. Click the “Add External JARs...” button and browse to the \tomcat\common\lib directory on your machine and select servlet.jar. Regardless of whether or not there are errors resolving these class names, click the “Add External JARs...” button and browse to the \Cognitier\SimpleIPC\bin directory and select CTJObjects.jar. This is necessary because the servlet code will incorporate classes defined in CTJObjects.jar.

Step 17: For this example, use the following Java code as the contents of DocReturnServlet.java. The code will accept a string from a web form, it will pass the string to SimpleIPC, and SimpleIPC will return a string representation of an array of bytes. The servlet code will convert the string into a byte array and write it to the output stream. The visitor’s web browser will receive the byte array as the MS Word document, and the visitor will have the option to save or open the file.

```
package webSamples;

import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.lang.*;
```

```

import com.cognitier.*;

/**
 * Servlet implementation class DocReturnServlet
 */
public class DocReturnServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Default constructor.
     */
    public DocReturnServlet() {
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        // TODO Auto-generated method stub
        //super.doGet(request, response);
        try
        {
            //Instantiate the CTJClient object, which provides access to make
routine calls
            CTJClient oCTJClient = new CTJClient();

            //Instantiate the CTJSessionReturnStruct object, which bundles the
return values from the call to obtainSession
            CTJSessionReturnStruct oSessionRetStruct = new
CTJSessionReturnStruct();

            //Instantiate the CTJBasicCall1RetStruct object, which bundles the
return values from the call to basicRoutineCall1
            CTJBasicCall1RetStruct oCall1RetStruct = new
CTJBasicCall1RetStruct();

            //Declare variables
            boolean bRet = false;
            String sInstanceName = "Inst1";
            String sAppName = "WordTest";
            String sRoutineName = "InvokeWordTestApp";
            int iNumCallsToMake = 1;
            int iCallCount = 0;
            String SESSIONINVALIDORTIMEOUT = "12001";
            String SERVERCOMMUNICATIONERR = "12004";
            String ERRRESUMINGSESSION = "12005";
            boolean bIncrementCallCount = true;

            //Initialize input arguments array
            String[] inputArray = null;
            inputArray = new String[1];
            inputArray[0] = request.getParameterValues("VisitorName")[0];

```

```

        //Make the call to obtainSession
        bRet = oCTJClient.obtainSession(sInstanceName, sAppName,
oSessionRetStruct);
        if (bRet == true)
        {
            while (iCallCount < iNumCallsToMake)
            {
                bIncrementCallCount = true;
                //Make call to BasicRoutineCall1
                bRet =
oCTJClient.basicRoutineCall1(oSessionRetStruct.SessionID, sInstanceName, sAppName,
sRoutineName, inputArray, oCall1RetStruct);
                if (bRet == false)
                {
                    if (oCall1RetStruct.errorsVector.size() > 0)
                    {
                        if
(((String)oCall1RetStruct.errorsVector.elementAt(0)).equals(SESSIONINVALIDORTIMEOUT) ||
((String)oCall1RetStruct.errorsVector.elementAt(0)).equals(SERVERCOMMUNICATIONERR) ||
((String)oCall1RetStruct.errorsVector.elementAt(0)).equals(ERRRESUMINGSESSION))
                        {
                            //The call to basicRoutineCall1 failed,
but the error code indicates there might be a problem with the session, so try to get a
new session

                                oSessionRetStruct.RejectCode = 0;
                                oSessionRetStruct.SessionID = "";
                                bRet =
oCTJClient.obtainSession(sInstanceName, sAppName, oSessionRetStruct);
                                if (bRet == false)
                                {
                                    break;
                                }
                                else
                                {
                                    bIncrementCallCount = false;
                                }
                            }
                        }
                    }
                }
            }
        }
        else
        {
            break;
        }
    }
    else
    {
        if (oCall1RetStruct.outVector.size() > 0)
        {
            String sByteArrayAsString =
(String)oCall1RetStruct.outVector.elementAt(0);
            char cByteArrayChars[] =
sByteArrayAsString.toCharArray();

            char cNextTwoChars[] = new char[2];
            int iCharPos = 0;
            byte bNextByte;
            OutputStream oOutStream = null;

```

```

        response.setHeader("Content-
Disposition", "attachment;filename=\"MyWordDoc.doc\"");
        response.setHeader("Cache-Control", "no-
store");

        response.setHeader("Pragma", "no-cache");
        response.setDateHeader("Expires", 0);
        response.setContentType("application/octet-
stream");

        response.setContentLength(sByteArrayAsString.length() / 2);
        oOutputStream = response.getOutputStream();
        while (iCharPos <
(sByteArrayAsString.length() - 1))
        {
            cNextTwoChars[0] =
cByteArrayChars[iCharPos];
            cNextTwoChars[1] =
cByteArrayChars[iCharPos+1];
            bNextByte =
(byte)GetIntFromHexString(cNextTwoChars);
            oOutputStream.write(bNextByte);
            iCharPos = iCharPos + 2;
        }
        oOutputStream.flush();
        oOutputStream.close();
    }
}
//Increment the call count unless the previous call
failed and we had to get a new session
    if (bIncrementCallCount == true)
    {
        iCallCount++;
    }
}
//Release the session
oCTJClient.releaseSession(oSessionRetStruct.SessionID);
}
}
catch (Exception e)
{
    log(e.getMessage());
}
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // TODO Auto-generated method stub
    super.doPost(request, response);
}

private static int GetIntFromHexString(char[] sSubstr)
{

```

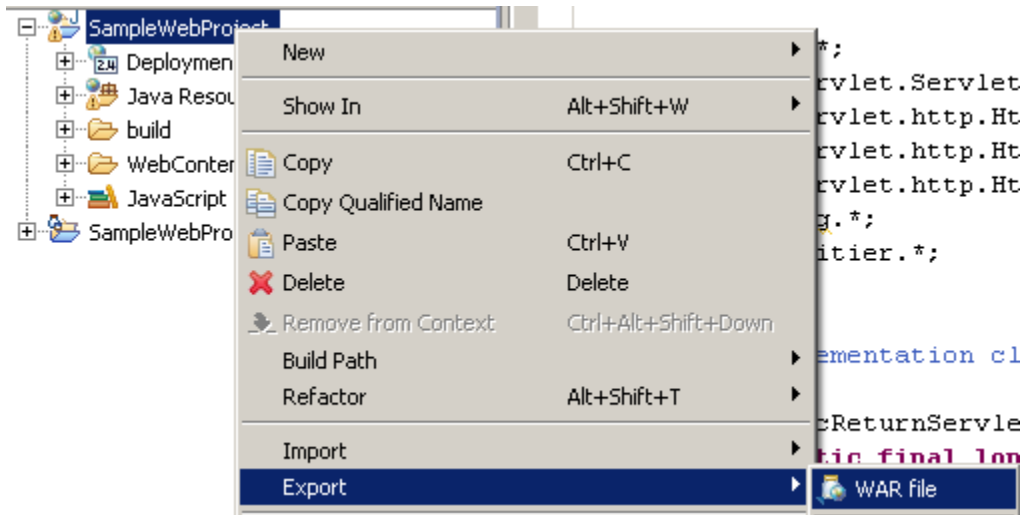
```
        int retVal = 0;
        retVal = (GetIntFromHexChar(sSubstr[0]) * 16) +
GetIntFromHexChar(sSubstr[1]);

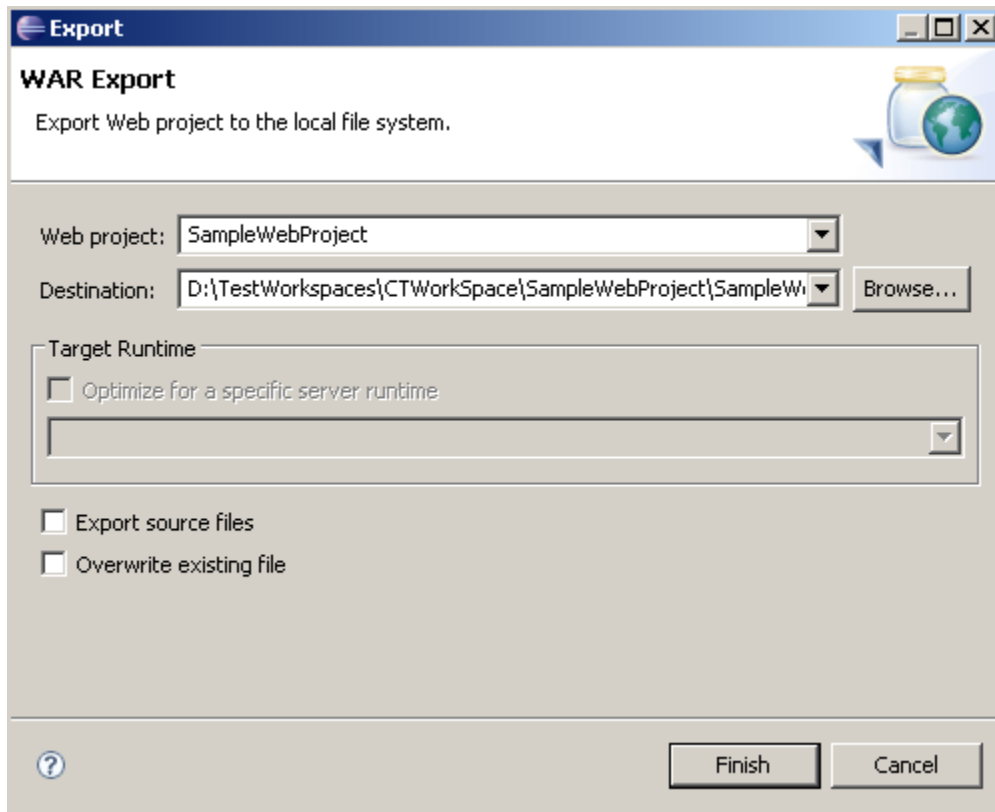
        return retVal;
    }

    private static int GetIntFromHexChar(char c)
    {
        int retVal = 0;
        switch (c)
        {
            case '0':
                retVal = 0;
                break;
            case '1':
                retVal = 1;
                break;
            case '2':
                retVal = 2;
                break;
            case '3':
                retVal = 3;
                break;
            case '4':
                retVal = 4;
                break;
            case '5':
                retVal = 5;
                break;
            case '6':
                retVal = 6;
                break;
            case '7':
                retVal = 7;
                break;
            case '8':
                retVal = 8;
                break;
            case '9':
                retVal = 9;
                break;
            case 'A':
                retVal = 10;
                break;
            case 'B':
                retVal = 11;
                break;
            case 'C':
                retVal = 12;
                break;
            case 'D':
                retVal = 13;
                break;
            case 'E':
                retVal = 14;
                break;
        }
    }
}
```

```
        break;
    case 'F':
        retVal = 15;
        break;
    }
    return retVal;
}
}
```

Step 18: Eclipse will normally compile the code automatically. Save your work and ensure that there are no syntax errors. Right-click on the SampleWebProject and select Export>>WAR file. On the resulting dialog, select a destination location (name the file SampleWebProject.war) and click Finish.





Step 19: Copy the resulting SampleWebProject.war file into your \tomcat\webapps directory. When the Tomcat server is started, the files within the archive will be extracted as necessary.

Step 20: Just as Eclipse needed access to CTJObjects.jar at design time, Tomcat will need access to it at runtime. Copy \Cognitier\SimpleIPC\bin\CTJObjects.jar into the \tomcat\shared\lib directory.

Step 21: In addition to the Java classes in CTJObjects.jar, Tomcat will need to know the location of the SimpleIPC JNI libraries. There are several options for providing this information to Tomcat. For simplicity, we will create a batch file in which we set the information into the environment before starting the Tomcat server. In Windows Explorer, navigate to the \tomcat\bin directory and create a new batch file in that directory. Use the following as the contents of the batch file:

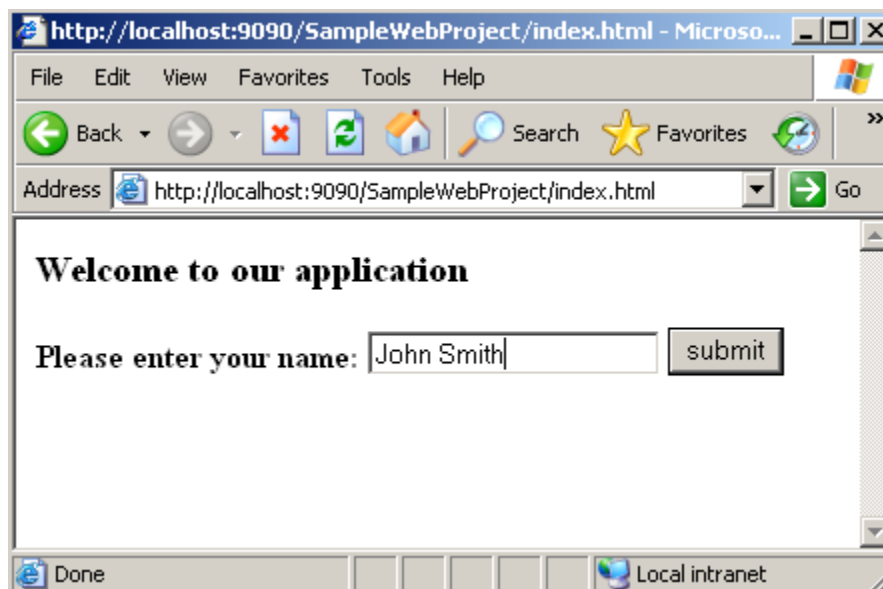
```
set JAVA_OPTS=-Djava.library.path="D:\Program Files\Cognitier\SimpleIPC\Bin"  
startup.bat
```

Double-click on the batch file you just created to start the Tomcat server. Since you started the server interactively, Tomcat will run under the identity of your user account. When Tomcat starts, it will extract the contents of your .war file. This will create a \tomcat\webapps\SampleWebProject directory.

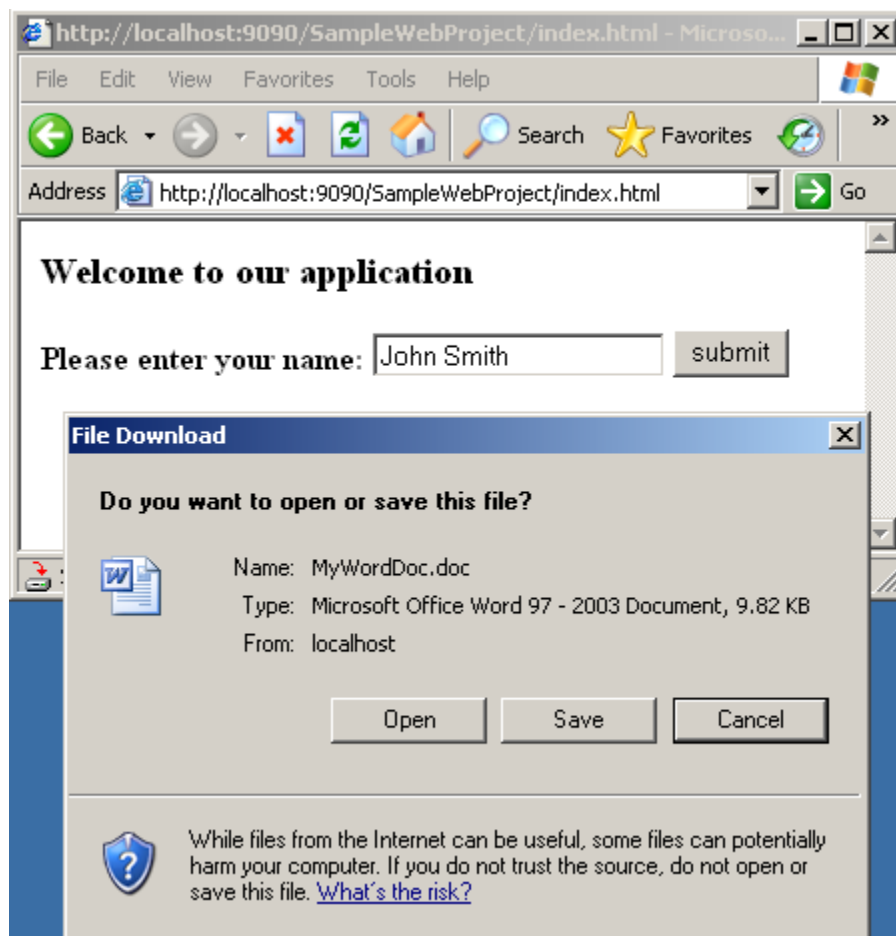
Step 22: Create a new file called index.html in the \tomcat\webapps\SampleWebProject directory. Within this new html file, we will define a form with one text field. The website visitor can enter his or her name and submit the form to the servlet. For this exercise, use the following as the contents of index.html:

```
<HTML>
<BODY>
<H3>Welcome to our application</H3>
<form action=DocReturnServlet>
<B>Please enter your name:</B>
<input type=text name=VisitorName>
<input type="submit" value="submit">
</form>
</BODY>
</HTML>
```

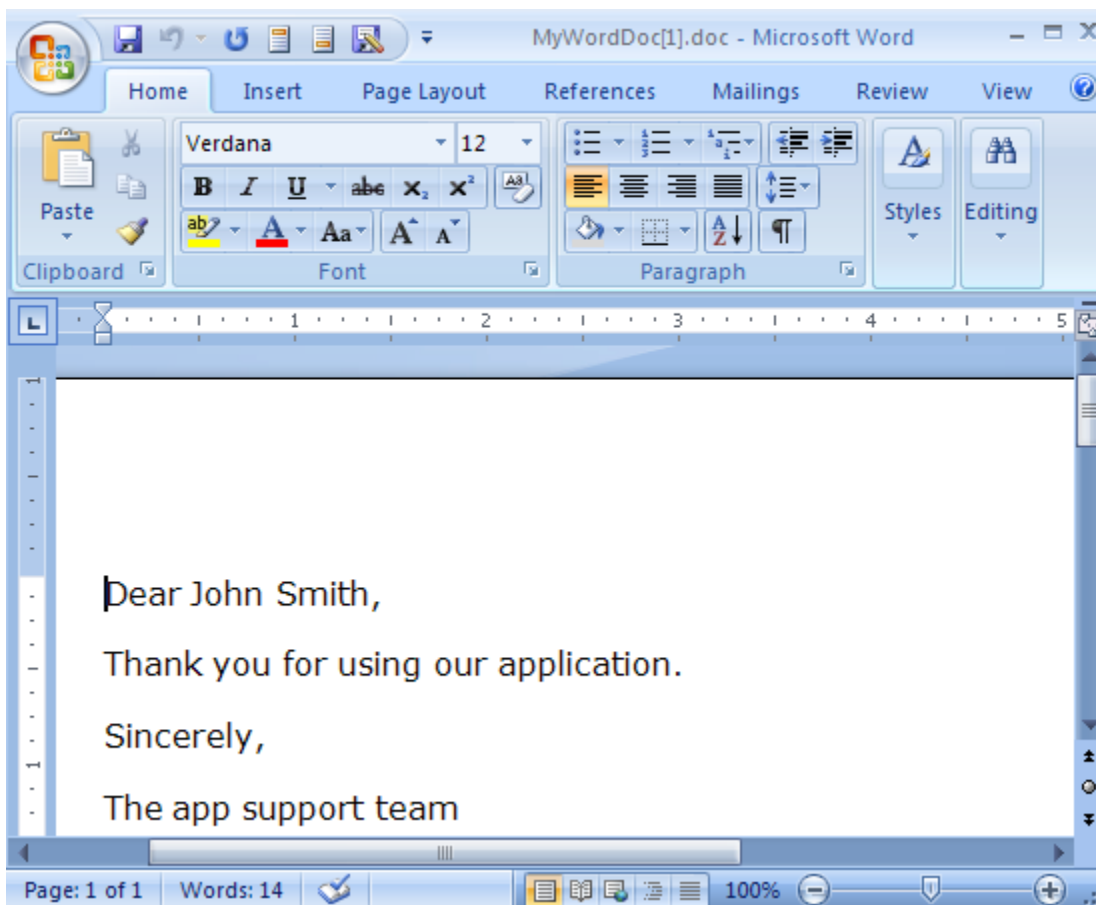
Step 23: Open Internet Explorer or another web browser and navigate to the location of your new web page as hosted by Tomcat. Enter a name in the form field and submit the page.



Step 24: When you submit the page, you should be prompted to either save or open the Word document.



Step 25: Select the option to open the file. View the contents and ensure the value you entered for the visitor's name is present in the document.



Step 26: If you have access to more than one machine for testing, it is best to access the web page from a different machine. The reason is that when you close the Word document, the next time the Word Automation COM object is used it will throw an exception. This will terminate the host IPC server. However, you can still repeat the process of producing and closing Word documents and new IPC servers will be spawned to replace those that exit. Again, if possible, go to a different machine and access the web page several times. Enter a new visitor name each time and ensure that the Word document returned has the correct visitor name. On the machine running SimpleIPC, ensure that only one Word document file is created. Also check memory consumption and CPU utilization via Task Manager.

Step 27: We will want to be able to simulate many users invoking our servlet at nearly the same time. Create a new routine for this purpose. Go to the Author Routines tab of the CTConsole and create a new source code file. Call the new routine `InvokeServlet` and use the `BasicRoutineRun1` template. For this exercise, enter the following code to invoke the servlet and put a string representation of the Word document bytes into the output array.

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

[assembly:AssemblyVersion("1.0.0.0")]
```

```

public class InvokeServlet 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;

        var oResponse: System.Net.HttpWebResponse = null;
        var oBinaryResponseReader: System.IO.BinaryReader = null;
        try
        {
            //Create a web request object for invoking our servlet
            var oRequest: System.Net.HttpWebRequest = null;
            var sURL: System.String =
"http://localhost:9090/SampleWebProject/DocReturnServlet?VisitorName=John+Smith
";
            var bDocBytes: System.Byte[];
            oRequest = System.Net.HttpWebRequest(System.Net.WebRequest.Create(sURL));
            oRequest.KeepAlive = false;
            oRequest.Timeout = 180000;

            //Invoke the servlet and read the document bytes into a byte array
            oResponse = System.Net.HttpWebResponse(oRequest.GetResponse());
            oBinaryResponseReader = new
System.IO.BinaryReader(oResponse.GetResponseStream());
            var iContentLength: int = int(oResponse.ContentLength);
            bDocBytes = oBinaryResponseReader.ReadBytes(iContentLength);
            oBinaryResponseReader.Close();
            oResponse.Close();

            //Convert the byte array to a string and return it to the calling program
            var iCount: int = 0;
            var oStringBuilder: System.Text.StringBuilder = new
System.Text.StringBuilder();
            while (iCount < iContentLength)
            {

```

```

        oStringBuilder.Append(GetHexStringFromByte(bDocBytes[iCount]));
        iCount++;
    }
    oCallParamsAccessor.OutputArgs.Add(oStringBuilder.ToString());

    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
    oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, e);

    //If we caught an exception, confirm that we're closing the response and
response reader
    if (oBinaryResponseReader != null)
    {
        oBinaryResponseReader.Close();
    }
    if (oResponse != null)
    {
        oResponse.Close();
    }
}
return bRet;

} //end function

static function GetHexStringFromByte(bByte: System.Byte) : System.String
{
    var sRet: System.String = String.Empty;
    var sHexChars: System.String = "0123456789ABCDEF";
    var cHexChars: System.Char[] = sHexChars.ToCharArray();
    var iByteVal: int = int(bByte);
    if (iByteVal < 16)
    {
        sRet = "0";
    }
}

```

```
    else
    {
        sRet = sRet + cHexChars[int(System.Math.Floor(iByteVal / 16))];
    }
    sRet = sRet + cHexChars[iByteVal % 16];
    return sRet;
}

} //end class
```

Step 28: Register the routine we just created. Go to the Manage Routines tab and click the "Refresh List" button so that our new dll will appear in the dlls drop-down. Select InvokeServlet.dll, enter a routine name of InvokeServlet, and save it.

Step 29: Test your work. Go to the Test Routines tab of the CTConsole. Select the "App1" application. It is important not to select the WordTest application because we want to simulate a user invoking the servlet and we do not need to have the server start-up routine execute for this part of the test. Click the Refresh button to the right of the Routines drop-down and select the InvokeServlet routine. For this exercise, it does not matter which Entry Point is selected. Designate one thread/session making one call per session. Click the "Start" button and observe the output in the text area beneath the button. The output should be a string representation of the bytes in the Word document. If this test produces the expected results, then increase the load such that 10 threads are each making 5 invocations of the routine (a total of 50 requests for the servlet). Review the output displayed in the text area in the CTConsole, and also check the IPC server logs to see if any errors occurred. Likewise check memory and CPU utilization on the machine. There are some obvious bottlenecks in this solution, and it would not be suitable for a very high-volume web application. However, the load in this test should not be overwhelming.

