Captiva 7 Web Services Output Tutorial

M. Scott Roth

Director of Technology

May 2014

Contents

Preface

I was recently asked to create a simple Captiva solution that allowed a client to scan documents, verify the value of a barcode with a database, and export the scanned images to a content management system. This is a capture process I have created many times using a combination of output modules and enterprise export modules. However, the catch this time around was that all database and content management system interaction had to be accomplished through web services and not the ODBC Export module or an enterprise export module. In addition, it had be accomplished with no or minimal client-side scripting.

Having never used the Captiva web service modules, nor created web services for Captiva to consume, I started reading the Captiva documentation, the *WSOutputScan* sample Captiva Designer project, the online Captiva forums, and Google hoping to find a comprehensive example. It didn't take long to exhaust these resources and gain no useable knowledge to help me get started.

This tutorial lays out, in a step-by-step fashion, my successful experience with Captiva's Web Service Output module. It is my contribution to the Captiva community to fill the need for a simple, web services starter project. Hopefully you will find it helpful.

1 Introduction

Captiva 7 (as well as Captiva 6) includes the Web Services Output (WSO) module which allows Captiva capture processes (a.k.a., CaptureFlows) to interface with external systems using SOAP-based web services. This tutorial documents my experience using Captiva 7's WSO module. In particular, it provides examples and best practices, and examines nuances for using web services and WSO.

The scenario for this tutorial is that of a law office that scans documents related to cases, verifies the index data retrieved from an external system, and releases the scanned documents to a content management system. The hardcopy documents are received by the office and barcode labels are affixed to the documents to identify their case number in a pre-processing step. Captiva is used to scan the documents, read the case number from the barcode, and retrieve case information from an external case management system via web services. Case information is displayed to the Scan Operator for verification. If it is incorrect, the Scan Operator can change the case number and retrieve the case information again. After the case information is verified, Captiva exports the scanned documents to the firm's case management system and updates a simple log file recording the date and time of the scan. Figure 1 depicts the high-level flow of this scenario.

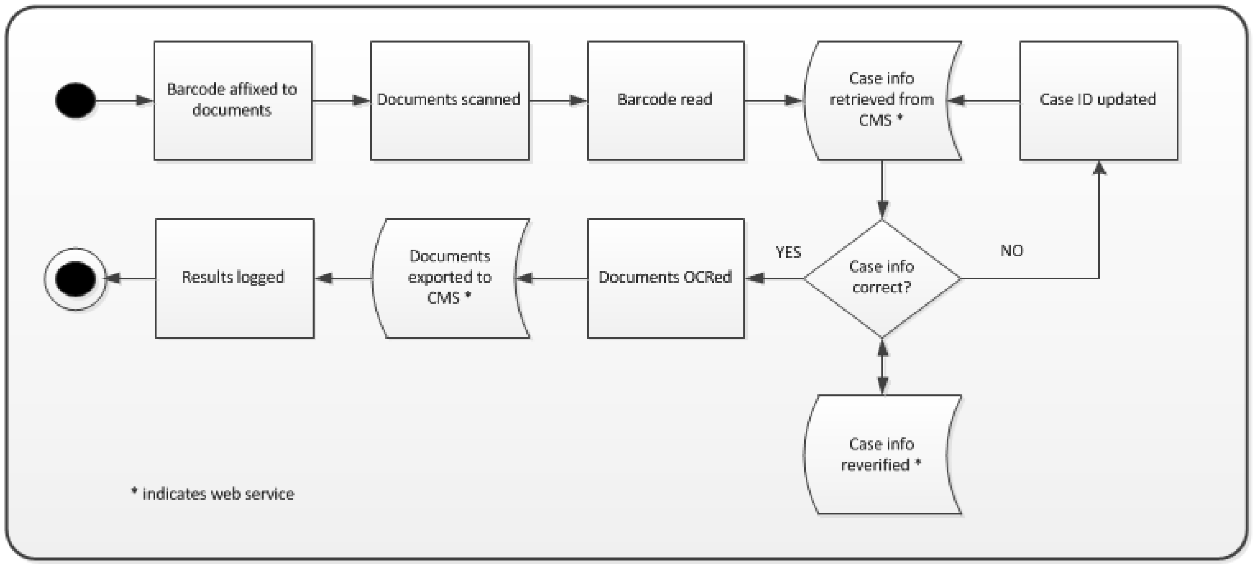


Figure 1 Incoming Case Document Process Flow

The web services employed in this scenario simulate the actions they imply and simply return reasonable values to the Captiva WSO module. What the web services do is irrelevant; the important aspects of the tutorial are how to design, build, and configure the web services to be consumed by the Captiva WSO module and how to configure the WSO module in the CaptureFlow.

This tutorial assumes competency with Eclipse, Java, Captiva modules, and Captiva Designer. I do not explain how to use these tools other than to highlight important aspects or nuances concerning implementation of the solution.

2 Environment

The computing environment I used to develop and test this tutorial consisted of a single, virtualized server running: Windows 2008 Server, Microsoft SQL Server 2008, Captiva 7, Apache TomEE+ 1.6, Java 1.7, and Eclipse for Java EE developers 4.3 (Kepler).

Captiva was installed in an out-of-the-box configuration using SQL Server for its database. The following Captiva modules were installed:

- EMC Captiva Designer
- EMC Captiva ScanPlus
- EMC Captiva Desktop
- EMC Captiva Administrator
- EMC Captiva InputAccel Server
- EMC Captiva Image Processor
- EMC Captiva NuanceOCR
- EMC Captiva Web Services Output
- EMC Captiva Standard Export

I used Apache TomEE+ as my application server for web services hosting. TomEE+ is Apache Tomcat pre-configured to host web services and dynamic web applications. See the References section for links to more information regarding TomEE+.

3 Web Services

When designing web services to use with Captiva WSO, there are a few important things to keep in mind. First, Captiva WSO can only consume SOAP web services; RESTful web services are not supported.

Second, you need to know that the Captiva WSO module can only consume *anonymous* web services that do not require Basic or Windows authentication before accessing the service. This does not mean you can't send credentials as input parameters to web services and have the service logic do authentication. I simulate this idea later in the tutorial. Anonymous access web services mean any user/process can request access to the web service. Think of it like allowing access to a public web page. You have used anonymous web services and probably haven't even realized it. For example, anonymous web services can be used to report weather conditions and look up ZIP codes. Many of the apps on your smartphone use these kinds of services to find nearby restaurants and movie listings.

You can safeguard your web services to some degree by using network security techniques like IP filtering, so only connections from the Captiva WSO server are accepted by the web services server. See the References section for more information regarding anonymous web services, and secure connections using SSL. By default, TomEE+ hosts anonymous web services that do not require authentication.

Lastly, the Captiva WSO module has no problem consuming web services that return primitive types: `Boolean`, `String`, `int`. However, it does not seem to consume collections well, which can present a problem if your services return something more complex than a primitive type. For example, in my scenario, the *getCaseInfo()* web method[1] returns an array of `Strings` containing a case's ID, name, plaintiffs, and defendants. Returning these `Strings` as a `List<String>` (which is a valid JAX-WS type), results in the WSO module's inability to even map the result to IA values. My solution to this limitation is to wrap all web methods that return a complex type in a Plain Old Data class[2].

By having all of my web methods return an object, I can return multiple variable types to the Captiva WSO module from a single web method. As noted above, the *getCaseInfo()* method returns several `String`s, one of which is a status message from the web service that can be mapped to an IA value. For example, if the method encounters an error and can't return the case info, instead of "silently failing", it returns a message that is mapped to an IA value that can be discovered by the scan operator or the Captiva administrator.

The following sections discuss the web services I built for this tutorial to simulate interaction with external systems. They all return an object as a result. The *importFileToCMS()* method even simulates sending the scanned documents to a case management system using MTOM (Message Transmission Optimization Mechanism).

3.1 Web Service Design

Table 1 defines the web method interfaces designed to meet the requirements of the scenario.

Table 1 IncomingCaseDocs Web Services Definitions

Web Method	Input Arguments	Return Values	Purpose
getCaseInfo	`String caseId`	`WSOCaseInfo Result`	Simulates retrieving data from the case management system
validateCaseId	`String caseId`	`WSOValidateCaseId Result`	Simulates the validation of the case Id with the case management system
importFileToCMS	`String username, String password, String caseId, String filename, byte[] filedata`	`WSOImportFile Result`	Simulates importing the scanned document into the case management system using MTOM

The logic for each interface, method, and data class are described in the following section.

1 In general, I consider a web service to be an interface which describes a collection of operations that can be accessed through SOAP messages. A web method is a component of a web service. The web service is called via SOAP, where the web method is called by proxy from the web service. I try to use these terms in their proper manner in this tutorial, but may occasionally use them synonymously.

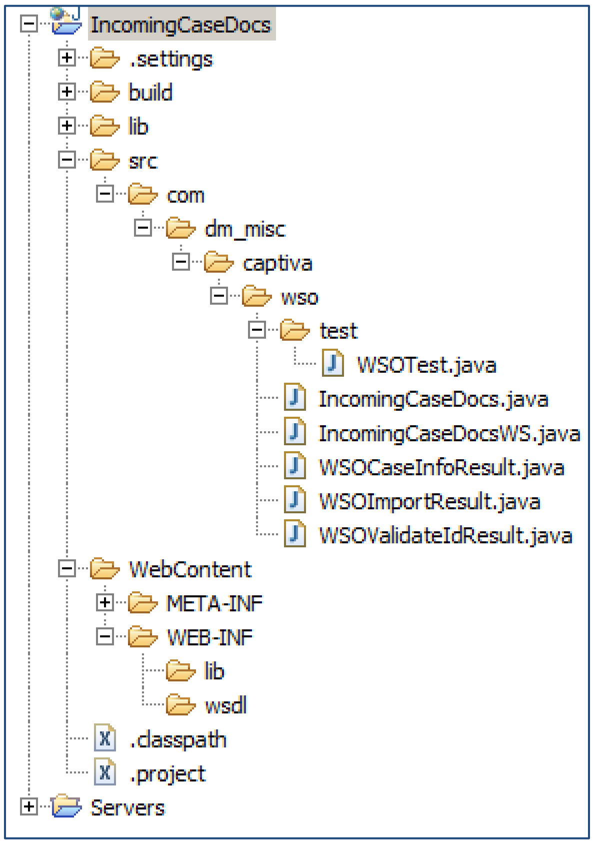
2 A Plain Old Data (POD) class is nothing more than a wrapper around a data structure with getters and setters. The limitations addressed by these POD objects can also be addressed by using additional JAX-WS annotations in the method definitions, but it seemed easier to use PODs and let JAX-WS generate the necessary XML automatically.

3.2 Web Service Implementation

The web services are implemented using an Eclipse Dynamic Web Project. The project contains six classes:

- `IncomingCaseDocs` – This class contains the implementation and the simulated logic for all of the methods listed in Table 1 . The details of each class follow in subsequent sections of this tutorial.
- `IncomingCaseDocsWS` – This is the interface class for the web services and contains the interface contracts listed in Table 1 as well as all of the necessary annotations to make the services and data elements visible to Captiva WSO.
- `WSOCaseInfoResult` – This data class contains the results that are returned when the *`getCaseInfo()`* method is called.
- `WSOValidateCaseIdResult` – This data class contains the results that are returned when the *`validateCaseId()`* method is called.
- `WSOImportFileResult` – This data class contains the results that are returned when the *`importFileToCMS()`* method is called.
- *`WSOTest`* – This class is a simple unit test class for the methods in `IncomingCaseDocs`.

Figure 2 depicts the Navigator view of the Eclipse project.



```
IncomingCaseDocs
    .settings
    build
    lib
    src
        com
            dm_misc
                captiva
                    wso
                        test
                            WSOTest.java
                        IncomingCaseDocs.java
                        IncomingCaseDocsWS.java
                        WSOCaseInfoResult.java
                        WSOImportResult.java
                        WSOValidateIdResult.java
    WebContent
        META-INF
        WEB-INF
            lib
            wsdl
    .classpath
    .project
Servers
```

Figure 2 Eclipse Dynamic Web Project Structure

3.2.1 IncomingCaseDocsWS Interface Class

The interface class contained in Listing 1 describes the web service method contracts as specified in Table 1 . It also contains the annotations necessary to make the web methods visible and consumable by Captiva WSO.

Listing 1 IncomingCaseDocsWS Interface Class

```
package com.dm_misc.captiva.wso;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;

@WebService(targetNamespace = "http://dm_misc.com/wsdl")
public interface IncomingCaseDocsWS {

   @WebMethod(operationName="getCaseInfo",
      action="http://dm_misc.com/wsdl/getCaseInfo")
   @WebResult(name="caseInfo")
   public WSOCaseInfoResult getCaseInfo(
      @WebParam(name="caseId") String caseId);

   @WebMethod(operationName="validateCaseId",
      action="http://dm_misc.com/wsdl/validateCaseId")
   @WebResult(name="isValid")
   public WSOValidateIdResult validateCaseId(
      @WebParam(name="caseId") String caseId);

   @WebMethod(operationName="importToCMS",
      action="http://dm_misc.com/wsdl/importToCMS")
   @WebResult(name="importResult")
   public WSOImportResult importToCMS(
         @WebParam(name="username") String username,
         @WebParam(name="password") String password,
         @WebParam(name="caseId")   String caseId,
         @WebParam(name="filename") String filename,
         @WebParam(name="filedata") byte[] filedate);
}
```

Note the `@annotations` used in describing the interfaces:

- `@WebService` – This annotation declares the interface to be a JAX-WS web service. It also defines a name space for the service. You will see this value used in the WSDL file to scope each element.
- `@WebMethod` – Each public, callable method is declared as a web method, and given an operation name and an action URI. You will see the operation names defined here when we access the web service WSDL from Captiva WSO (Section 4.3.6). The action URIs are used in the WSDL to map the operations to the web methods.
- `@WebResult` – This annotation gives a meaningful name to the result value. Again, you will see this in Section 4.3.6 when we map the web method results to IA values.

- @WebParam – This annotation gives each web method's input parameters a descriptive name that is visible when mapping the IA values to the web method calls (see Section 4.3.6). Without these annotations, the input parameters are simply labeled as arg0, arg1, etc.

3.2.2 IncomingCaseDocs Implementation Class

The IncomingCaseDocs class contains the implementation logic for the web methods described by the IncomingCaseDocsWS interface class discussed in Section 3.2.1. Mostly these are "mock methods" that simply return reasonable values without actually doing anything. However, some error checking is performed to ensure Captiva is sending and receiving valid data and to test the use of the message result value in each POD class.

Listing 2 contains the first few lines of the IncomingCaseDocs class file with the necessary web service annotations. Each method of this file is discussed individually in subsequent sections.

Listing 2 IncomingCaseDocs Implementation Class

```
package com.dm_misc.captiva.wso;

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import javax.jws.WebService;
import javax.xml.ws.soap.MTOM;

@MTOM
@WebService(
      portName = "IncomingCaseDocsPort",
      serviceName = "IncomingCaseDocsService",
      targetNamespace = "http://dm_misc.com/wsdl",
      endpointInterface = "com.dm_misc.captiva.wso.IncomingCaseDocsWS")

public class IncomingCaseDocs implements IncomingCaseDocsWS {
```

There are only two annotations required in this file:

- @MTOM – This annotation tells the web service to expect binary data to be sent unencoded. Generally, this is a more efficient way to transport binary data (i.e., file content) than just plain SOAP. This annotation is required for the *importFileToCMS()* method discussed in Section 3.2.2.3.
- @WebService – This is the same annotation we saw in the definition of the incomingCaseDocsWS interface class (Section 3.2.1), but with a few new arguments. The arguments shown here are all used to properly generate the WSDL file.
    - portName – The name of the web service port as it will appear in the WSDL. You can choose any name you like for your port, but common convention dictates that you add the word 'Port' to the end of the class name.
    - serviceName – The name of the web service as it will appear in the WSDL. You can choose any name you like for your web service, but common convention dictates that you add the word 'Service' to the end of the class name.

- `targetNamespace` – It is critical that the namespace defined here matches the namespace defined in the `incomingCaseDocsWS` interface class. Note: I found that any `targetNamespace` that did not end with '`/wsdl`' would not produce a WSDL file readable by Captiva WSO.
- `endpointInterface` – This is the fully qualified name of the web services interface.

3.2.2.1 getCaseInfo Method

The *`getCaseInfo()`* method is shown in Listing 3 . The method's logic is straightforward:

- it receives a case ID, which it validates (the details of validation are discussed in Section 3.2.2.2);
- after validation, it defines a set of mock case data depending upon whether the case ID is an even or odd number;
- it then returns a `WSOCaseInfoResult` object containing the case data (see Section 3.2.3.1).

If the case ID is valid, the message returned in the `WSOCaseInfoResult` object is "Success". If the case ID was not valid, the message returned is the message generated by the *`validateCaseId()`* method.

Listing 3 getCaseInfo Method

```
public WSOCaseInfoResult getCaseInfo(String caseId) {
   int id;
   String name;
   String plaintiff;
   String defendant;

   // make sure case Id is valid
   if (validateCaseId(caseId).isValidId()) {

      // return one of two sets of data based on even/odd case id
      id = Integer.parseInt(caseId);

      if (id % 2 == 0) {
         name = "Allegiance Insurance Co. vs. Molly Peters";
         plaintiff = "Allegiance Insurance Co.";
         defendant = "Molly Peters";
      } else {
         name = "Rate Hikes by Allegiance Insurance Co.";
         plaintiff = "Molly Peters, John Spivy, and the Arizona Insurance
            Board.";
         defendant = "Allegiance Insurance Co.";
      }
   } else {
      return new WSOCaseInfoResult(caseId, "", "", "",
         validateCaseId(caseId).getMessage());
   }
   return new WSOCaseInfoResult(caseId, name, plaintiff, defendant,
      "Success");
}
```

3.2.2.2 validateCaseID Method

Listing 4 contains the code for the `validateCaseId()` method. A valid case ID meets three simple criteria:

- it is not null or empty,
- it is greater than 4 digits long (leading zeroes are ignored),
- it contains only numeric characters.

Listing 4 validateCaseId Method

```
public WSOValidateIdResult validateCaseId(String caseId) {

   // if case id is longer than 4 digits and contains only numbers
   // it is valid
   if ((caseId == null) || (caseId.length() == 0))
      return new WSOValidateIdResult(false,"Case Id is null");

   try {
      long i= Integer.parseInt(caseId);
      if (i > 9999) {
         return new WSOValidateIdResult(true,"valid");
      } else {
         return new WSOValidateIdResult(false,"Case Id too short");
      }
   } catch (Exception e) {
      return new WSOValidateIdResult(false,"Case Id is not numeric: " +
         caseId);
   }
}
```

Using *`Integer.parseInt()`* kills two birds with one stone: if the conversion produces an exception, the case ID contains a non-numeric character; and if the converted number is less than 10,000 it is too short.

3.2.2.3 importFileToCMS Method

The *`importFileToCMS()`* method in Listing 5 simulates importing the scanned files into a case management system and returns a unique ID for each file imported. The method implements the following logic:

- validate the case ID,
- validate the password (“captiva”), and
- save the file streamed to the method using MTOM to the `c:/temp` directory.

Listing 5 importFileToCMS Method

```
public WSOImportResult importToCMS(String username, String password,
         String caseId, String filename, byte[] filedata) {

   String filePath = "c:/temp/" + filename;
   String fileId = "";

   // validate case Id
   if (! validateCaseId(caseId).isValidId()) {
      return new WSOImportResult(false,null,"Invalid case ID.");
   }

   // simulate logging in with username and password
   if (! password.equalsIgnoreCase("captiva")) {
      return new WSOImportResult(false,null,"Invalid user password");
   }

   if (filedata != null)  {
      fileId = "" + System.currentTimeMillis();

      try {
         FileOutputStream fos = new FileOutputStream(filePath);
         BufferedOutputStream outputStream = new BufferedOutputStream(fos);
         outputStream.write(filedata);
         outputStream.close();
      } catch (Exception e) {
         return new WSOImportResult(false,null,"Error importing " + filename
            + " to case " + caseId);
      }

      return new WSOImportResult(true,fileId,"Successfully imported " +
         filename + " to case " + caseId + " as " + fileId);
   } else {
      return new WSOImportResult(false,null,"No file transfered");
   }
}
```

In this method, I simply compare the password passed into the method to the string "captiva" to authenticate the user. This is obviously not a practice you want to implement in real life; however, I hope it gives you some ideas for how you could implement real authentication. For example, you could pass in a security token or encrypted credentials and have the method perform real authentication with Active Directory or the case management system. The *EMC Captiva Capture Web Services Guide* also discusses how you can use Captiva WSO with SSL, further securing your credentials.

The file content is passed to the web method using the MTOM protocol mentioned in Section 3.2.2. The incoming parameter type is `byte[]`. Nothing else needs to be done to transfer the file. As you will see in Section 4.3.6.3, when we configure the Captiva WSO module to use this web method, we simply map a binary output IA value from Captiva to the `byte[]` input parameter of the web method and SOAP/MTOM takes care of the rest.

A unique file ID is fabricated to simulate interaction with the case management system by getting the system time. The file ID is logged later in the process for auditing purposes (see Section 4.2.3).

3.2.3 IncomingCaseDocs Result Classes (POD)

The most important aspect of these web services are the POD classes they return. I have created a POD class for each web method, primarily so I could return complex results and a message to the calling Captiva WSO module. These classes contain `String` data fields with getters/setters, and their purpose is to simply encapsulate the values passed into them and make them accessible via the WSDL.

3.2.3.1 WSOCaseInfoResult Data Class

The `WSOCaseInfoResult` class returns five `Strings` (see Listing 6):

- `caseId` – The ID for the case. This is the same value that was passed into the method from the document's barcode.
- `caseName` – The name of the case in the case management system.
- `casePlaintiff` – The list of case plaintiffs in the case as a single `String`.
- `caseDefendant` – The list of case defendants in the case as a single `String`.
- `message` – A message to be returned to the Captiva WSO module.

Listing 6 WSOCaseInfoResult Data Class

```
package com.dm_misc.captiva.wso;

public class WSOCaseInfoResult {

   private String caseId;
   private String caseName;
   private String casePlaintiff;
   private String caseDefendant;
   private String message;

   public WSOCaseInfoResult(String id, String name, String plaintiff,
      String defendant, String msg) {

      caseId = id;
      caseName = name;
      casePlaintiff = plaintiff;
      caseDefendant = defendant;
      message = msg;
   }

   public String getCaseId() {
      return caseId;
   }

   public String getCaseName() {
      return caseName;
   }

   public String getCasePlaintiff() {
      return casePlaintiff;
   }

   public String getCaseDefendant() {
      return caseDefendant;
   }

   public void setCaseId(String caseId) {
      this.caseId = caseId;
   }

   public void setCaseName(String caseName) {
      this.caseName = caseName;
   }
   public void setCasePlaintiff(String casePlaintiff) {
      this.casePlaintiff = casePlaintiff;
   }

   public void setCaseDefendant(String caseDefendant) {
      this.caseDefendant = caseDefendant;
   }

   public String getMessage() {
      return message;
   }

   public void setMessage(String message) {
      this.message = message;
   }
}
```

3.2.3.2 WSOValidateIdResult Data Class

The data fields contained in the `WSOValidateIdResult` class (see Listing 7) are:

- `validId` (`boolean`) – This indicates whether the case ID is valid or not.
- `message` (`String`) – A message to be returned to the Captiva WSO module.

Listing 7 WSOValidateCaseIdResult Data Class

```
public class WSOValidateIdResult {

   private boolean validId;
   private String message;

   public WSOValidateIdResult (boolean result, String msg){
      validId = result;
      message = msg;
   }

   public boolean isValidId() {
      return validId;
   }

   public String getMessage() {
      return message;
   }

   public void setValidId(boolean validId) {
      this.validId = validId;
   }

   public void setMessage(String message) {
      this.message = message;
   }
}
```

3.2.3.3 *WSOImportResult Data Class*

The `WSOImportResult` class (shown in Listing 8) contains data fields for:

- `importSuccess` (`boolean`) – This indicates whether the import to the case management system was successful.
- `importedFileId` (`String`) – This is a fabricated ID to simulate interaction with the case management system.
- `message` (`String`) – This is an error or success message to be returned to the Captiva WSO module.

Listing 8 WSOImportFileResult Data Class

```
package com.dm_misc.captiva.wso;

public class WSOImportResult {

   private boolean importSuccess;
   private String message;
   private String importedFileId;

   public WSOImportResult (boolean result, String fileId, String msg){
      importSuccess = result;
      message = msg;
      importedFileId = fileId;
   }

   public boolean isImportSuccess() {
      return importSuccess;
   }

   public String getMessage() {
      return message;
   }

   public String getImportedFileId() {
      return importedFileId;
   }

   public void setImportSuccess(boolean importSuccess) {
      this.importSuccess = importSuccess;
   }

   public void setMessage(String message) {
      this.message = message;
   }

   public void setImportedFileId(String importedFileId) {
      this.importedFileId = importedFileId;
   }

}
```

3.3 Web Services Deployment

To deploy the web service, simply export the `IncomingCaseDocs` project from Eclipse as a `WAR` file, and place it in the TomEE+ `/webapps` directory. The content of the `WAR` file is listed in Listing 9 .

Listing 9 IncomingCaseDocs WAR File

```
IncomingCaseDocs
IncomingCaseDocs\META-INF
IncomingCaseDocs\WEB-INF
IncomingCaseDocs\META-INF\MANIFEST.MF
IncomingCaseDocs\WEB-INF\classes
IncomingCaseDocs\WEB-INF\lib
IncomingCaseDocs\WEB-INF\wsdl
IncomingCaseDocs\WEB-INF\classes\com
IncomingCaseDocs\WEB-INF\classes\com\dm_misc
IncomingCaseDocs\WEB-INF\classes\com\dm_misc\captiva
IncomingCaseDocs\WEB-INF\classes\com\dm_misc\captiva\wso
IncomingCaseDocs\WEB-INF\classes\com\dm_misc\captiva\wso
   \IncomingCaseDocs.class
IncomingCaseDocs\WEB-INF\classes\com\dm_misc\captiva\wso
   \IncomingCaseDocsWS.class
IncomingCaseDocs\WEB-INF\classes\com\dm_misc\captiva\wso
   \WSOCaseInfoResult.class
IncomingCaseDocs\WEB-INF\classes\com\dm_misc\captiva\wso
   \WSOImportResult.class
IncomingCaseDocs\WEB-INF\classes\com\dm_misc\captiva\wso
   \WSOValidateIdResult.class
IncomingCaseDocs\WEB-INF\classes\com\dm_misc\captiva\wso\test
IncomingCaseDocs\WEB-INF\classes\com\dm_misc\captiva\wso\test\WSOTest.class
```

3.4 WSDL

Notice that there is no WSDL file listed in Listing 9 . The Captiva WSO module requires a WSDL file to map input and output parameters. So, where is the WSDL file? JAX-WS web services do not require a WSDL file to be created at compile time; one is generated automatically upon request at runtime. JAX-WS generates the WSDL file based upon reflection and the `@annotations` in the class files. That said, it can be difficult to determine the WSDL URL required by Captiva WSO.

The easiest way to find the WSDL URL is to look in the TomEE+ `Catalina.log` file. Find an entry similar to this after you deploy the `WAR` file and start the TomEE+ server:

```
INFO: Webservice(wsdl=http://localhost:8080//IncomingCaseDocs/
IncomingCaseDocsService, qname={http://dm_misc.com/wsdl}
```

From this log entry, you can discern that the WSDL URL is:

```
http://localhost:8080//IncomingCaseDocs/IncomingCaseDocsService?wsdl
```

Keep this value handy for configuring the Captiva WSO module discussed in Section 4.3.6.

4 Captiva Designer Project

Now that the web services are in place, we can turn our attention to the Captiva CaptureFlow required for this tutorial. This section describes the Captiva Designer project and the module configurations that support the scenario discussed in Section 1. The project contains the following components:

- Image Processing Profile
  - *ReadBarCode* – The Image Processor profile for reading the barcodes from the incoming documents.
- Document Types Profile
  - *CaseDocument* – The Document type defining metadata fields to support the CaptureFlow and document export process.
- Export Profile
  - *LogCaseDocumentExportToCMS* – The Standard Export profile to log metadata regarding scan and export.
- CaptureFlow
  - *IncomingCaseDocument* – The CaptureFlow implementing the process described in Figure 1 and containing the necessary logic and IA values to integrate the modules.

Each of these components is described in the following sections.

4.1 CaptureFlow

The CaptureFlow is essentially the heart and soul of the capture process and gives context to the rest of tutorial's discussion. The following sections describe the CaptureFlow, the modules used, the Custom Values defined, and all of the IA value assignments necessary to implement the business process discussed in Section 1.

4.1.1 Flow

The Captiva Designer project contains a single CaptureFlow named `IncomingCaseDocument`, depicted in Figure 3 . Table 2 lists the process steps and corresponding Captiva modules used to implement each step depicted in the figure.

Table 2 Process Step Definitions

Process Step	Module Type	Trigger Level	Remarks
ScanPlus	ScanPlus	Batch (7)	See Section 4.3.1 for configuration details.
ReadBarCode	Image Processor	Page (0)	Profile: *ReadBarCode.* See Section 4.2.1 for configuration details.
GetCaseInfo	WSO	Document (1)	See Section 4.3.6.1 for configuration details.
Desktop	Desktop	Document (1)	See Section 4.3.3 for configuration details.
ValidateCaseInfo	WSO	Document (1)	See Section 4.3.6.2 for configuration details.
Is Case Invalid	Decision	Document (1)	The decision uses the test: `CustomValues:1.isCaseIdValid = false.`
Jump To: Desktop	Jump	Document (1)	No additional configuration.
NuanceOCR	Nuance OCR	Document (1)	See Section 4.3.4 for configuration details.

Process Step	Module Type	Trigger Level	Remarks
ImportToCMS	WSO	Document (1)	See Section 4.3.6.3 for configuration details.
LogResults	Standard Export	Batch (7)	Profile: *LogCaseDocumentExportToCMS.* See Section 4.2.3 for configuration details.

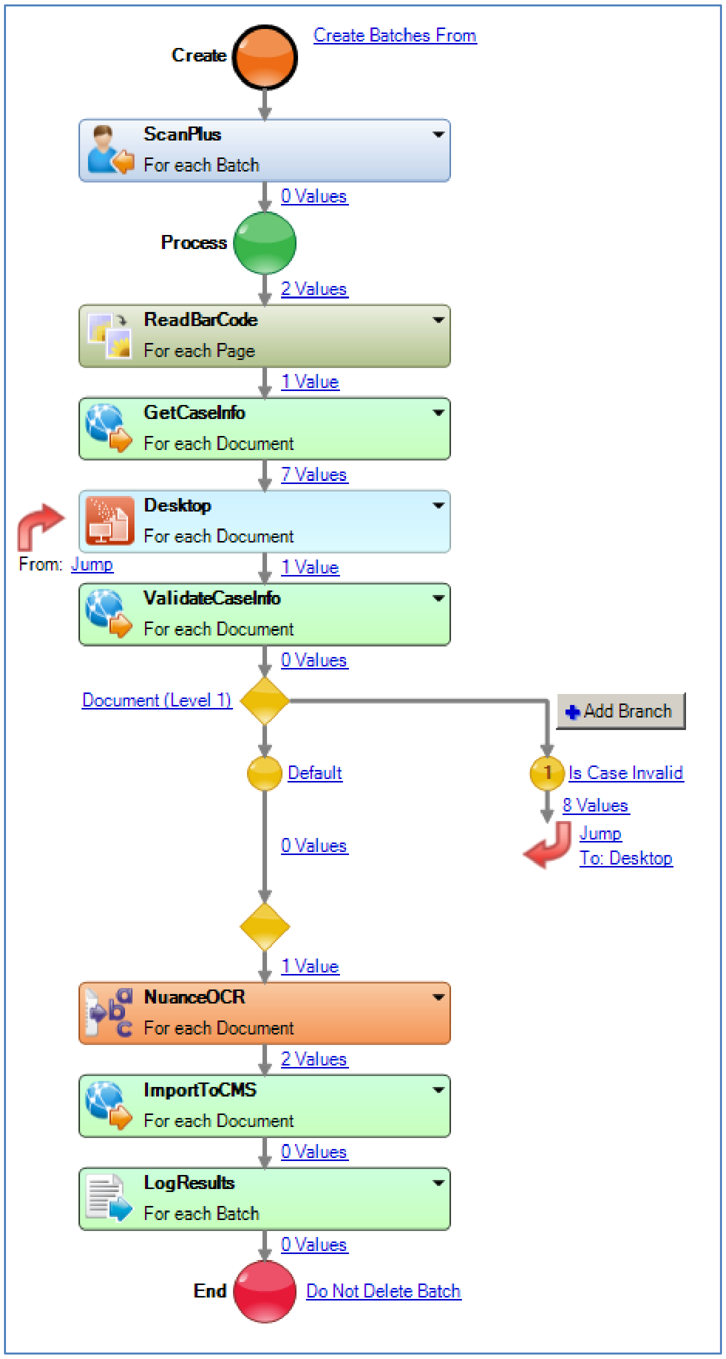


Figure 3 IncomingCaseDocs CaptureFlow

4.1.2 Custom Values

Table 3 describes the Custom Values defined in the CaptureFlow and their purposes.

Table 3 Custom Value Definitions

Custom Value	Module	Type	Level	Remarks
`CaseNumber`	CustomValues	String	1	Holds the case ID read from the barcode.
`CaseName`	CustomValues	String	1	Holds the case name retrieved by the web service.
`Plaintiff`	CustomValues	String	1	Holds the plaintiffs' names retrieved by the web service.
`Defendant`	CustomValues	String	1	Holds the defendants' names retrieved by the web service.
`FileName`	CustomValues	String	1	Holds the name of the file as it will appear in the case management system.
`isCaseIdValid`	CustomValues	String	1	Holds the result of the *ValidateCaseInfo* process step.
`isExportOk`	CustomValues	String	1	Holds the result of the *ImportToCMS* process step.
`ImportedFileId`	CustomValues	String	1	Holds the file ID returned from the case management system as a result of the import.
`Password`	CustomValues	String	1	Holds the password for the username required to import file to case management system.
`ServerMsg`	Custom Values	String	1	Place holder for informational messages returned by web services.
`Username`	CustomValues	String	1	Holds the scan operator's name.

4.1.3 IA Value Assignments

Table 4 describes the Custom and IA value assignments made in the CaptureFlow. As you will see, the purpose of these assignments is to capture metadata about the process and the scanned documents, and to link the various CaptureFlow components together by passing data among them.

Table 4 IA Value Assignments

Where Assignment Occurs	Assignment	Remarks
ScanPlus – ReadBarCode	`ReadBarCode:0.InputImage = ScanPlus:0.OutputImage`	Send each scanned page to the *ReadBarCode* module.
	`CustomValues:1.Username = ScanPlus:0.ScanOperator`	Capture the scan operator's name.
ReadBarCode - GetCaseInfo	`CustomValues:1.CaseNumber = ReadBarCode:0.Barcode0_Text when _node:0.NodeIndexFromL1 = 0`	Get the case number (case ID) from the barcode. Ensure it is the first barcode on the first page only.

Where Assignment Occurs	Assignment	Remarks
GetCaseInfo – Desktop	`Desktop:0.Image = ReadBarCode:0.OutputImage`	Send each page to *Desktop* for viewing and processing.
	`Desktop:1.UimDocumentType = "CaseDocument"`	Manually assign the document type since we are not using classification and don't want the scan operators to have to do it manually in *Desktop*.
	`Desktop:1.UimDataImportMode = 1`	Tell *Desktop* to accept the following data.
	`Desktop:1.$Runtime.InUimData_CaseNumber = CustomValues:1.CaseNumber`	Pre-populate the **Case Number** field with the case number read from the barcode.
	`Desktop:1.$Runtime.InUimData_CaseName = CustomValues:1.CaseName`	Pre-populate the **Case Name** field with the case name retrieved by the *GetCaseInfo* step.
	`Desktop:1.$Runtime.InUimData_Defendant = CustomValues:1.Defendant`	Pre-populate the **Defendant** field with the defendant retrieved by the *GetCaseInfo* step.
	`Desktop:1.$Runtime.InUimData_Plaintiff = CustomValues:1.Plaintiff`	Pre-populate the **Plaintiff** field with the plaintiff retrieved by the *GetCaseInfo* step.
Desktop - ValidateCaseInfo	`CustomValues:1.ServerMsg = ""`	Clear any message from a previous WSO call.
ValidateCaseInfo – NuanceOCR	`NuanceOCR:0.Level0_InputImage = Desktop:0.Image`	Pass each page from the *Desktop* to be OCRed and converted to PDF.
Is_Case_Invalid – Jump To: Desktop	`Desktop:1.UimDataImportMode = 1`	Tell *Desktop* to accept the following data.
	`Desktop:1.$Runtime.InUimData_ServerMsg = CustomValues:1.ServerMsg`	Pre-populate the **Error Msg** field with any messages returned from the previous WSO call.
	`Desktop:1.$Runtime.InUimData_CaseNumber = CustomValues:1.CaseNumber`	Pre-populate the **Case Number** field with the case number previously entered.
	`Desktop:1.$Runtime.InUimData_CaseName = CustomValues:1.CaseName`	Pre-populate the **Case Name** field with the case name previously entered.
	`Desktop:1.$Runtime.InUimData_Defendant = CustomValues:1.Defendant`	Pre-populate the **Defendant** field with defendant previously entered.

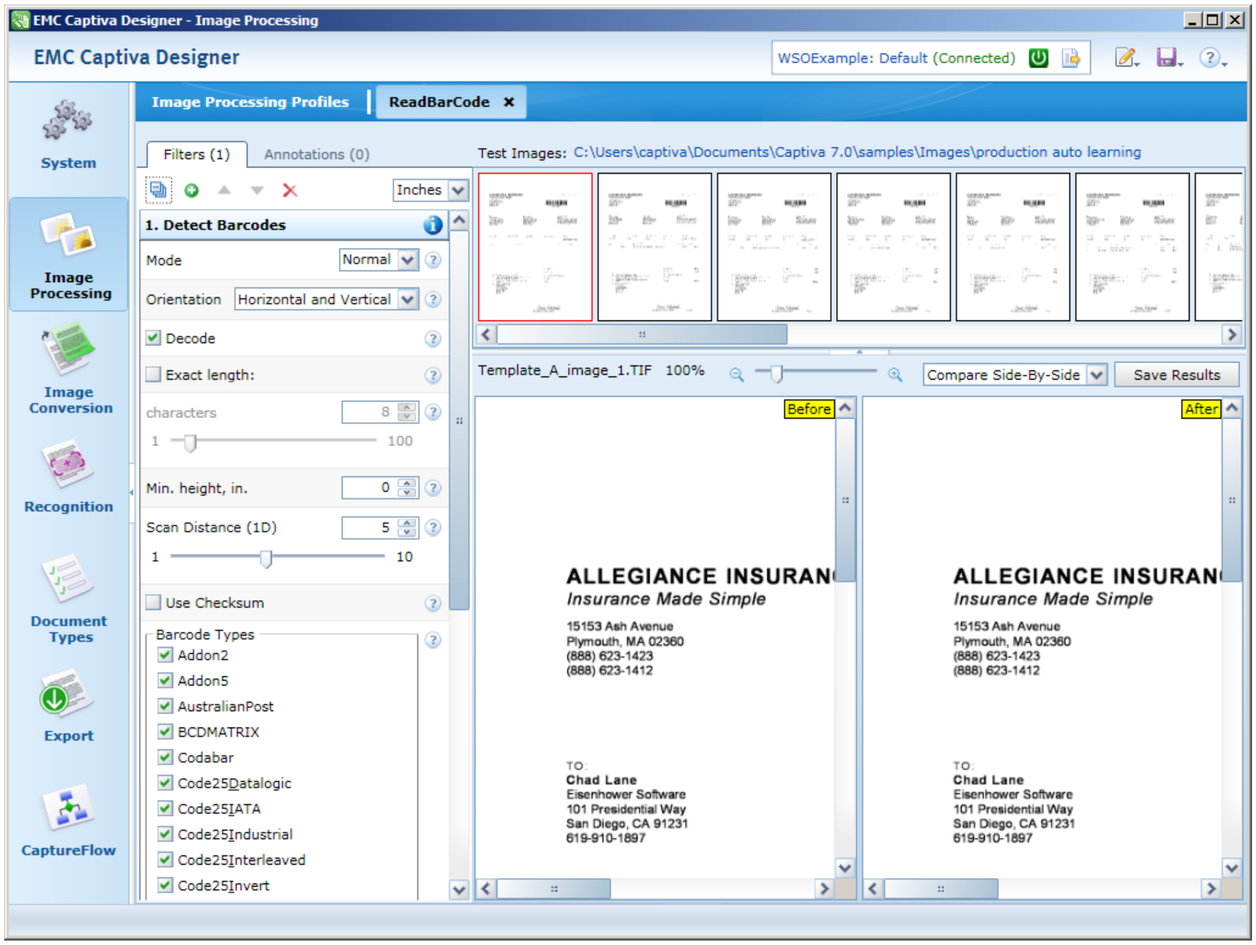
Where Assignment Occurs	Assignment	Remarks
	`Desktop:1.$Runtime.InUimData_Plaintiff = CustomValues:1.Plaintiff`	Pre-populate the **Plaintiff** field with the plaintiff previously entered.
	`Desktop:1.$Runtime.InUimData_Password = “”`	Reset the password on the *Desktop* screen.
	`CustomValues:1.Password = “”`	Reset the password in the Custom Values forcing the user to re-enter it.
NuanceOCR – ImportToCMS	`CustomValues:1.FileName = "CaseDoc-" & CustomValues:1.CaseNumber & "_" & _Batch.BatchID & "_" & _Node:1.NodeIndexFromL7 + 1 & ".pdf"`	Build the filename to be used during the import to the CMS.
	`CustomValues:1.serverMsg = “”`	Clear any message from a previous WSO call.

4.2 Profiles

Some of the modules in this CaptureFlow are configured using profiles. Profiles are new in Captiva 7 and allow you to setup configurations for certain modules that can be reused in other CaptureFlows. For this tutorial, we will setup profiles for the Image Processor, the definition of the Document Type (this includes the index fields the user will fill), and Standard Export. Each of these profiles is discussed in more detail in the following sections.

4.2.1 Image Processor Profile

The Image Processor profile, *ReadBarCode*, contains a single filter for reading barcodes, as depicted in Figure 4 .



Test Images: C:\Users\captiva\Documents\Captiva 7.0\samples\Images\production auto learning

Template_A_image_1.TIF 100%

Inches

1. Detect Barcodes

Mode: Normal

Orientation: Horizontal and Vertical

☑ Decode

☐ Exact length:

characters: 8 (1 – 100)

Min. height, in.: 0

Scan Distance (1D): 5 (1 – 10)

☐ Use Checksum

Barcode Types
- ☑ Addon2
- ☑ Addon5
- ☑ AustralianPost
- ☑ BCDMATRIX
- ☑ Codabar
- ☑ Code25Datalogic
- ☑ Code25IATA
- ☑ Code25Industrial
- ☑ Code25Interleaved
- ☑ Code25Invert

Figure 4 ReadBarCode Profile

4.2.2 Document Type

This tutorial's single document type, *CaseDocument*, contains six metadata fields, as defined in Table 5 .

Table 5 CaseDocument Fields

Field Name	Data Type	Input Mode	Index Level	Remarks
CaseNumber	String / Text	Editable / Required	1	Contains the case ID as read from the barcode.
CaseName	String / Text	Read-Only	1	Contains the case name as returned by the *GetCaseInfo* step.
Plaintiff	String / Text	Read-Only	1	Contains the plaintiffs' names as returned by the *GetCaseInfo* step.
Defendant	String / Text	Read-Only	1	Contains the defendants' names as returned by the *GetCaseInfo* WSO step.
Password	String / Text	Editable / Required	1	Password used to import the scanned documents into the case management system.

Field Name	Data Type	Input Mode	Index Level	Remarks
`ServerMsg`	`String / Text`	`Read-Only`	`1`	Place holder a message returned by any of the WSO steps.

These fields all map directly to the Custom Values discussed in Section 4.1.2. To ensure smooth exchange between document field values in *Desktop* and Custom Values in the CaptureFlow, their names and index levels are exactly the same in both locations.

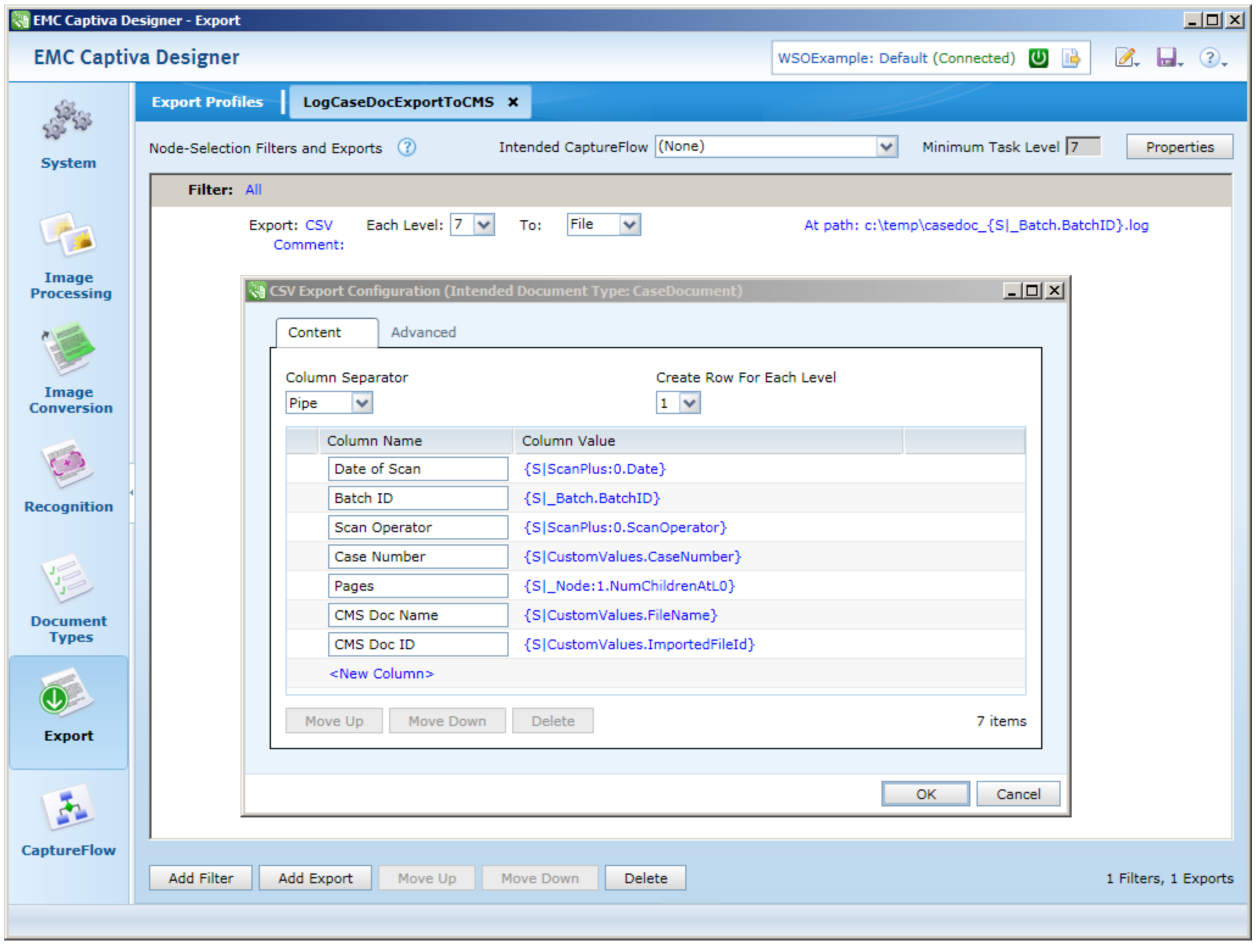
The first four fields, `CaseNumber`, `CaseName`, `Plaintiff`, and `Defendant`, are obvious as to their purpose. The *`getCaseInfo()`* web method will fill these fields with the results of its query. The last two fields, `Password` and `ServerMsg`, are perhaps a little less obvious.

I included the `Password` field as a field the user must complete in order to successfully import the document into the case management system. If the user enters anything other than "`captiva`" the import will fail with an "Invalid password" error. This is part of the simulation of passing user credentials to a web service and having the web service do authentication with an external system. Obviously this is not a realistic implementation, but allows you to easily trace the user's entry here, to the web service call, and ultimately to the success of the process.

The purpose of the read-only `ServerMsg` field is to display informational messages returned from the web service. For example, it the `caseId` contains a non-numeric character, the document will be returned to the Desktop with the error "`Case ID is not numeric.`" I have included this capability as another example to highlight the value of returning complex types from your web methods. In doing so, I can return the `Strings` that populate the *Desktop* fields, as well as a `Boolean` to indicate whether the `caseId` was valid, and an informational message. Figure 18 in Section 5.5 contains a depiction of the *Desktop* displaying an error message.

4.2.3 Standard Export Profile

The *IncomingCaseDocs* CaptureFlow uses one Standard Export profile, *LogCaseDocumentExportToCMS*. The *LogCaseDocumentExportToCMS* profile writes metadata values to a CSV file as a record of what was scanned and what was imported into the case management system. Figure 5 depicts this profile's configuration in Captiva Designer.



Node-Selection Filters and Exports

Intended CaptureFlow (None)

Minimum Task Level 7

Filter: All

Export: CSV Each Level: 7 To: File At path: c:\temp\casedoc_{S|_Batch.BatchID}.log

Comment:

CSV Export Configuration (Intended Document Type: CaseDocument)

Column Separator: Pipe

Create Row For Each Level: 1

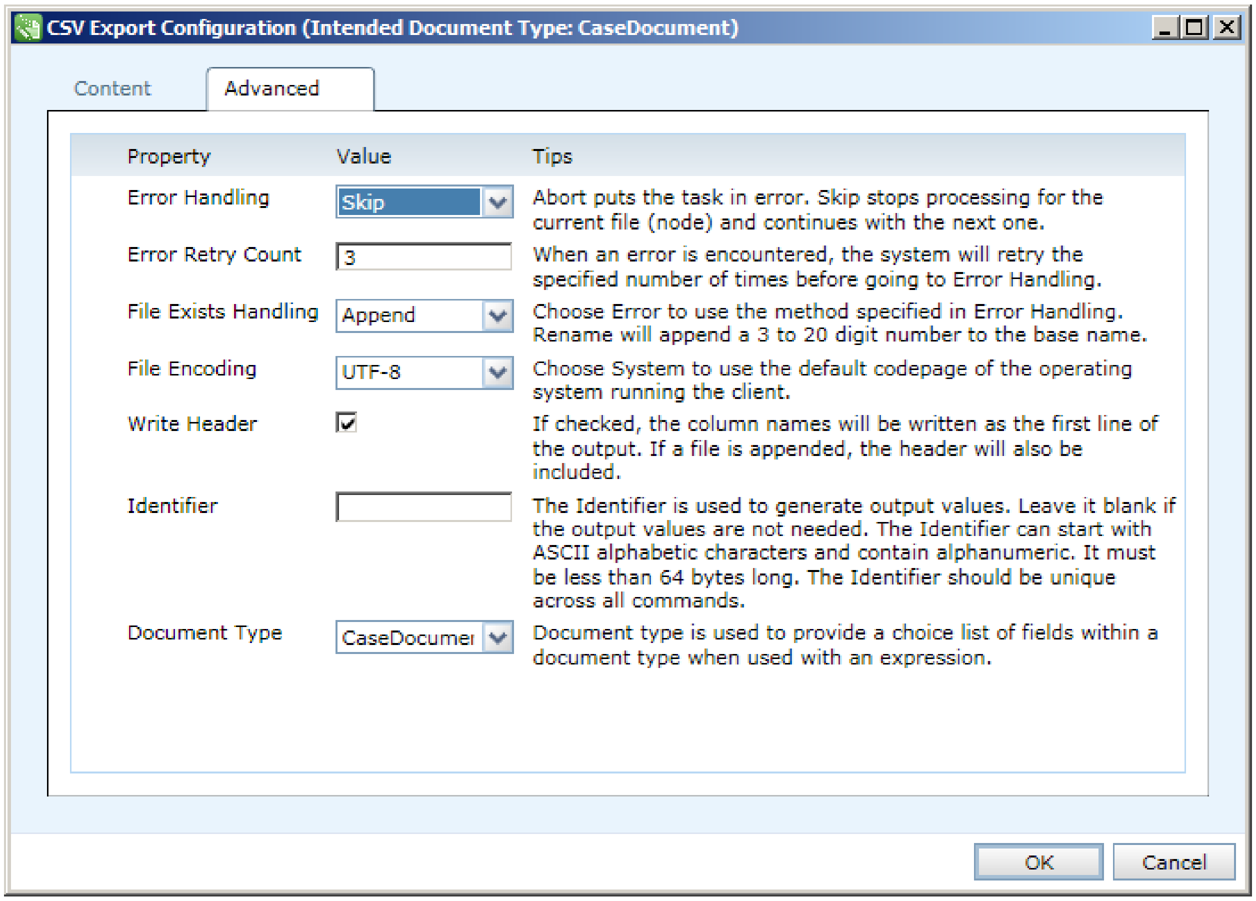
Column Name	Column Value
Date of Scan	{S|ScanPlus:0.Date}
Batch ID	{S|_Batch.BatchID}
Scan Operator	{S|ScanPlus:0.ScanOperator}
Case Number	{S|CustomValues.CaseNumber}
Pages	{S|_Node:1.NumChildrenAtL0}
CMS Doc Name	{S|CustomValues.FileName}
CMS Doc ID	{S|CustomValues.ImportedFileId}
<New Column>	

7 items

1 Filters, 1 Exports

Figure 5 LogCaseDocumentExportToCMS Standard Export Profile

Figure 6 depicts the error handling for the *LogCaseDocumentExportToCMS* profile.


CSV Export Configuration (Intended Document Type: CaseDocument)

Property	Value	Tips
Error Handling	Skip	Abort puts the task in error. Skip stops processing for the current file (node) and continues with the next one.
Error Retry Count	3	When an error is encountered, the system will retry the specified number of times before going to Error Handling.
File Exists Handling	Append	Choose Error to use the method specified in Error Handling. Rename will append a 3 to 20 digit number to the base name.
File Encoding	UTF-8	Choose System to use the default codepage of the operating system running the client.
Write Header	✓	If checked, the column names will be written as the first line of the output. If a file is appended, the header will also be included.
Identifier		The Identifier is used to generate output values. Leave it blank if the output values are not needed. The Identifier can start with ASCII alphabetic characters and contain alphanumeric. It must be less than 64 bytes long. The Identifier should be unique across all commands.
Document Type	CaseDocumer	Document type is used to provide a choice list of fields within a document type when used with an expression.

Figure 6 LogCaseDocumentExportToCMS Advanced Tab

See Figure 20 in Section 5.5 for an example of this profile's output.

4.3 Module Configurations

Now that we have created the necessary Profiles, we will assign them to the appropriate CaptureFlow process modules, and configure those modules that do not utilize profiles. Configuring modules is accomplished using the CaptureFlow Designer in Captiva Designer. Simply click the down error on a process icon and select **Module Settings** from the pop-up menu. See Figure 7 for an example.

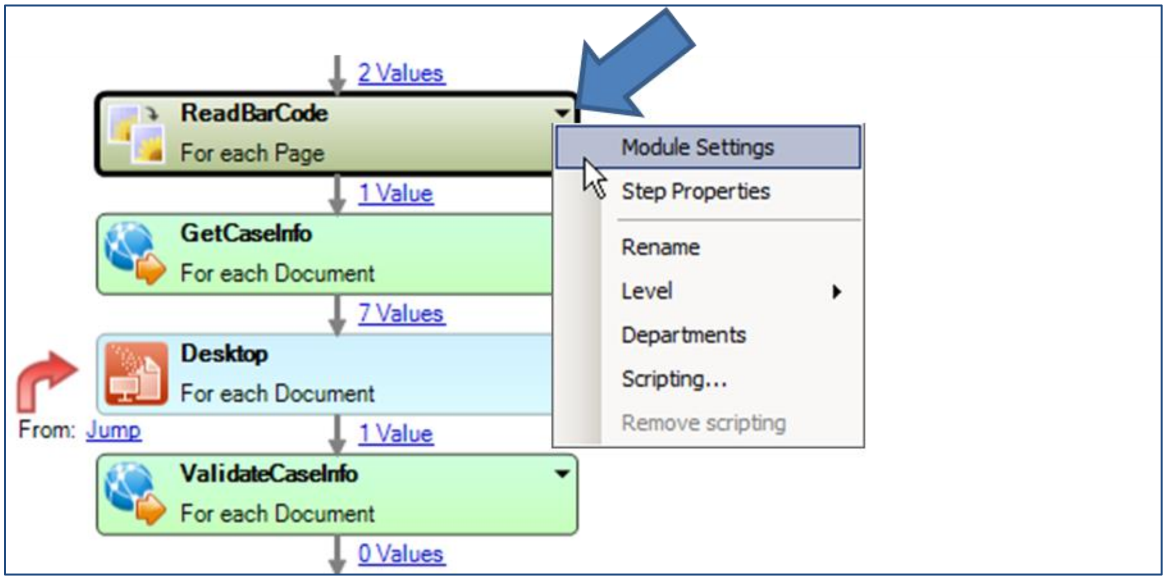


Figure 7 CaptureFlow Module Configuration

The following sections discuss the configuration for each of the process modules listed in Table 2 in Section 4.1.1. It is important to note that for the modules that do not use profiles, the module will be launched in "Setup Mode" to facilitate configuration. That said, the modules must reside on the computer from which they are being configured. This is not an issue in an environment such as mine where everything is self-contained. However, this can cause real headaches in a multi-server environment.

4.3.1 ScanPlus

The *ScanPlus* module is used out-of-the-box with no special configuration except the following:

- A scanner was configured.
- On the **Auto Batch Creation** tab:
  - The **Batch name schema** was defined as follows: `IncomingCaseFile_@(Name)_@(Now)`
  - **Process schema:** `IncomingCaseDocument`

4.3.2 ReadBarCode (Image Processor)

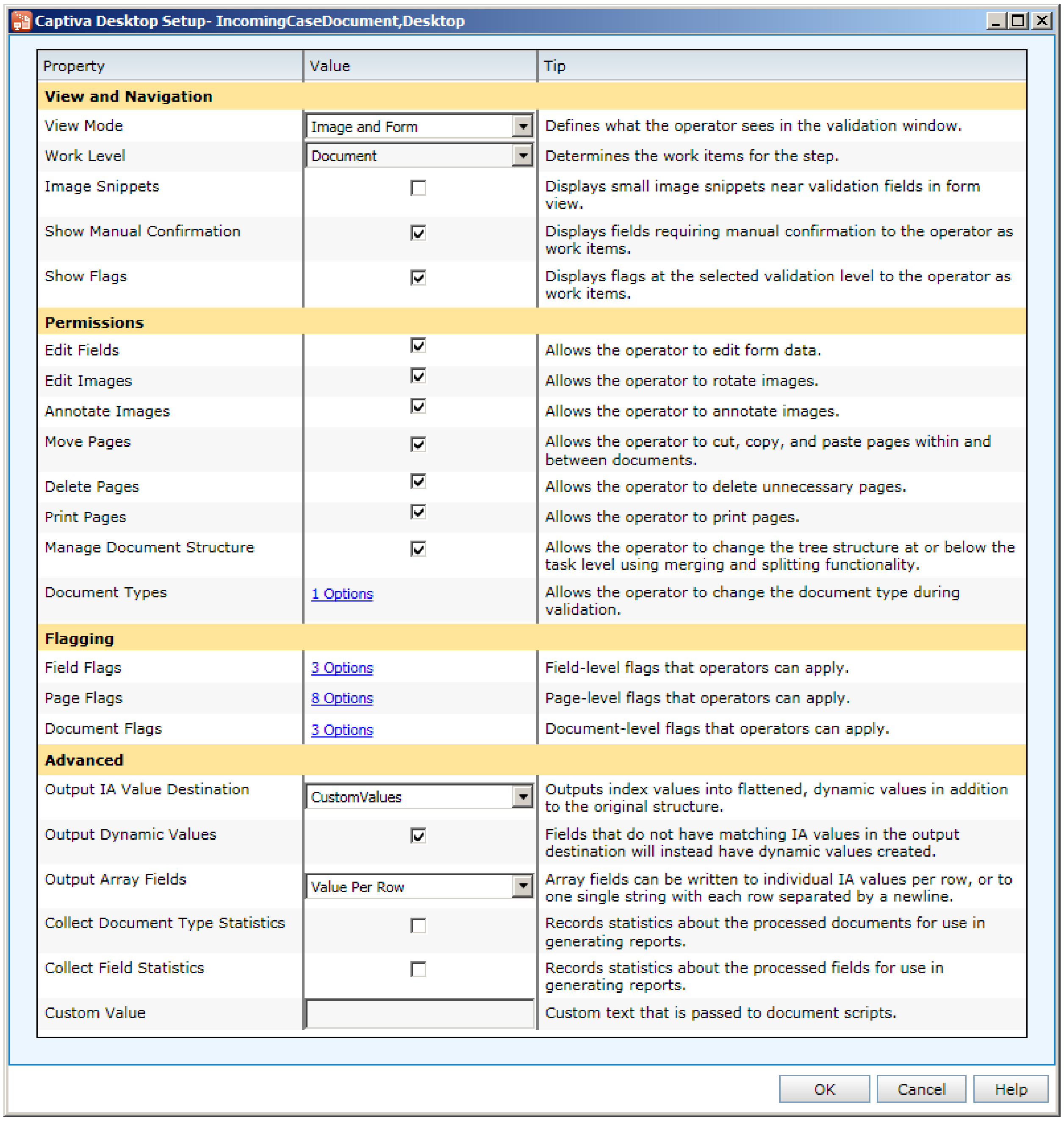
The *Image Processor* module is configured to use the `ReadBarCode` Image Processing profile we created in Section 4.2.1.

4.3.3 Desktop

The *Desktop* module is configured as depicted in Figure 8 Note the following settings:

- **View Mode**: `Image and Form`
- **Work Level**: `Document`
- **Output IA Value Destination**: `CustomValues`
- **Output Dynamic Values**: `checked`

- **Output Array Fields**: `Value Per Row`



Captiva Desktop Setup- IncomingCaseDocument,Desktop

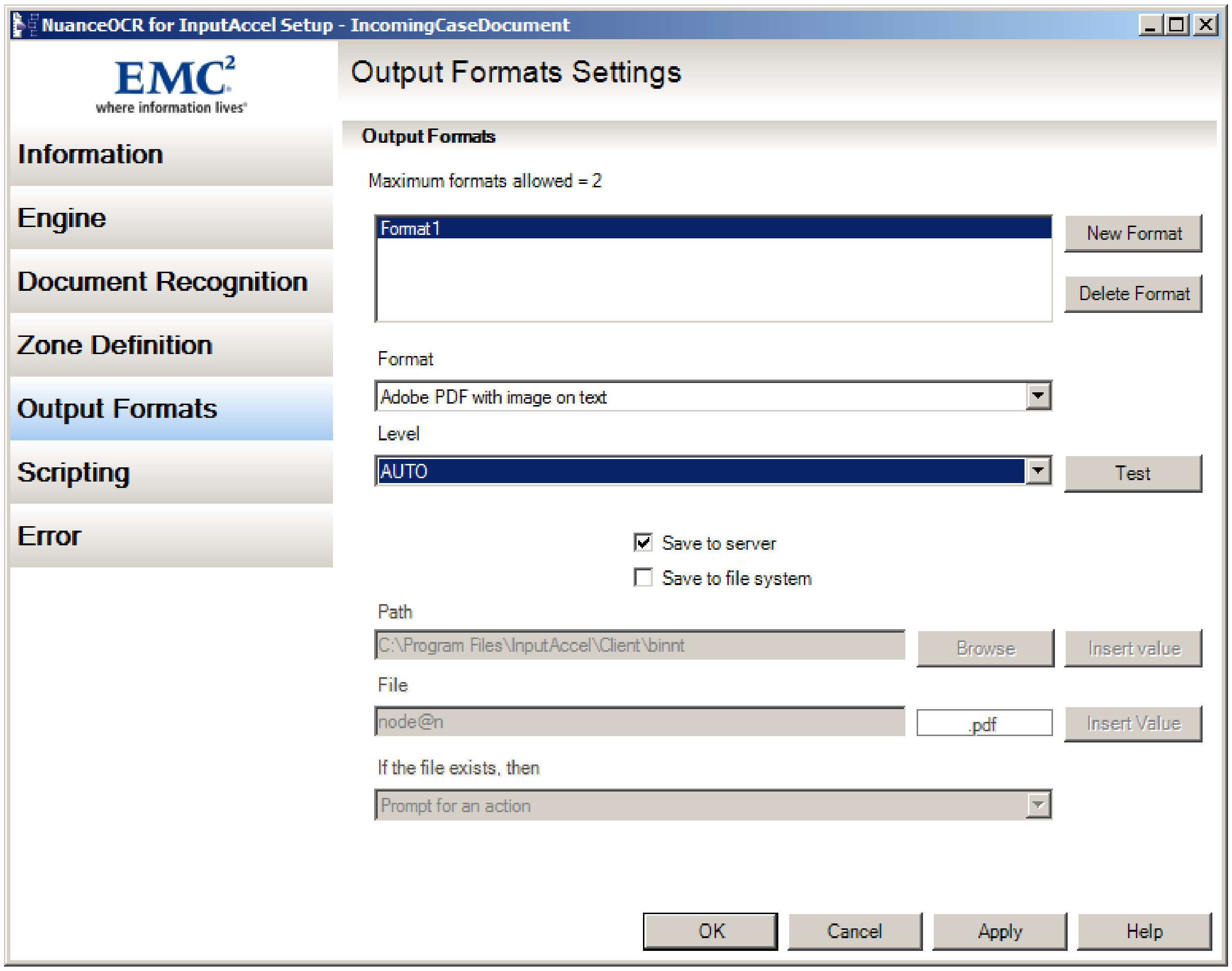
Property	Value	Tip
View and Navigation		
View Mode	Image and Form	Defines what the operator sees in the validation window.
Work Level	Document	Determines the work items for the step.
Image Snippets	☐	Displays small image snippets near validation fields in form view.
Show Manual Confirmation	☑	Displays fields requiring manual confirmation to the operator as work items.
Show Flags	☑	Displays flags at the selected validation level to the operator as work items.
Permissions		
Edit Fields	☑	Allows the operator to edit form data.
Edit Images	☑	Allows the operator to rotate images.
Annotate Images	☑	Allows the operator to annotate images.
Move Pages	☑	Allows the operator to cut, copy, and paste pages within and between documents.
Delete Pages	☑	Allows the operator to delete unnecessary pages.
Print Pages	☑	Allows the operator to print pages.
Manage Document Structure	☑	Allows the operator to change the tree structure at or below the task level using merging and splitting functionality.
Document Types	1 Options	Allows the operator to change the document type during validation.
Flagging		
Field Flags	3 Options	Field-level flags that operators can apply.
Page Flags	8 Options	Page-level flags that operators can apply.
Document Flags	3 Options	Document-level flags that operators can apply.
Advanced		
Output IA Value Destination	CustomValues	Outputs index values into flattened, dynamic values in addition to the original structure.
Output Dynamic Values	☑	Fields that do not have matching IA values in the output destination will instead have dynamic values created.
Output Array Fields	Value Per Row	Array fields can be written to individual IA values per row, or to one single string with each row separated by a newline.
Collect Document Type Statistics	☐	Records statistics about the processed documents for use in generating reports.
Collect Field Statistics	☐	Records statistics about the processed fields for use in generating reports.
Custom Value		Custom text that is passed to document scripts.

Figure 8 Desktop Configuration

4.3.4 NuanceOCR

The *NuanceOCR* module is configured as depicted in Figure 9 . Note that a new output format was created, `Format1`, and was assigned `Adobe PDF with image on text` in the **Format** dropdown.

This format definition will produce a text searchable PDF file and store it in the `OutputFile1_OutputFile` IA value that will be used by the web service to import the content to the case management system (see Section 4.3.6.3).



Maximum formats allowed = 2

Format1

Format: Adobe PDF with image on text

Level: AUTO

☑ Save to server

☐ Save to file system

Path: C:\Program Files\InputAccel\Client\binnt

File: node@n .pdf

If the file exists, then: Prompt for an action

Figure 9 NuanceOCR Configuration

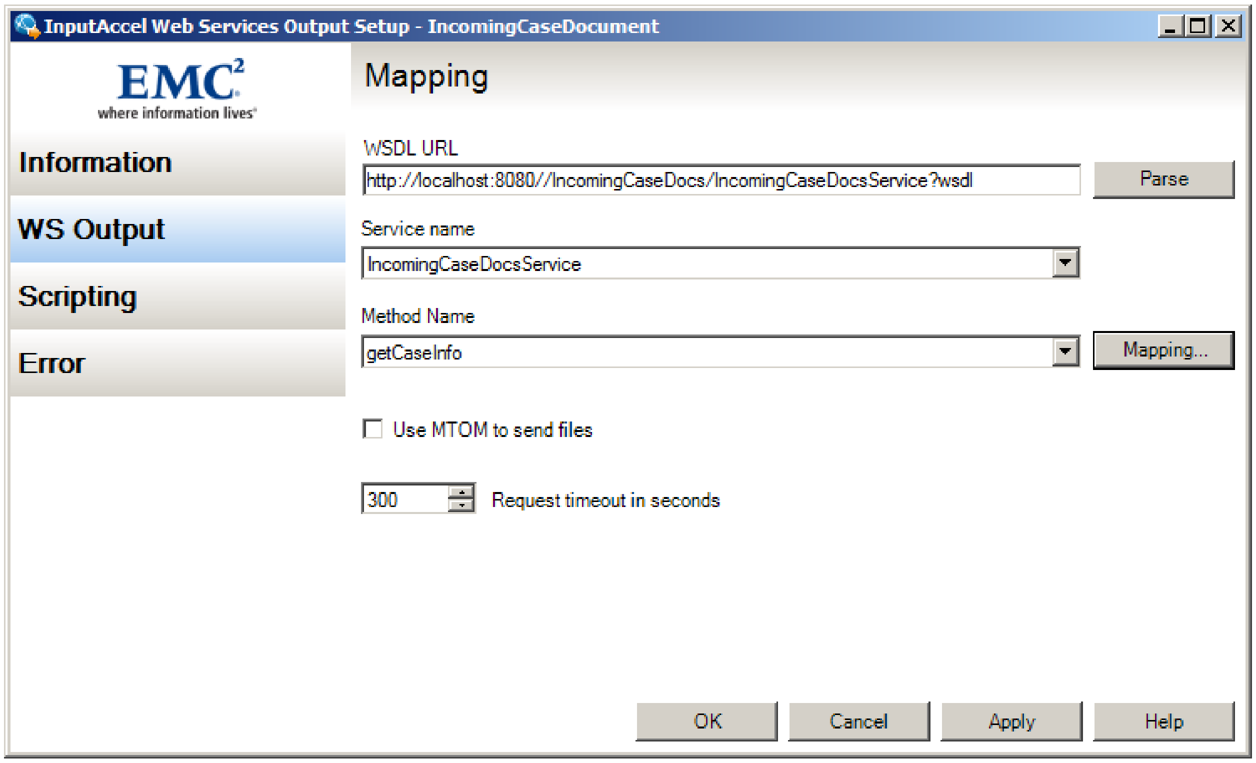
4.3.5 LogResults (Standard Export)

The *Standard Export* module was configured to use the `LogCaseDocExportToCMS` Standard Export profile we created in Section 4.2.3.

4.3.6 WebService Output

Configuring the three WSO modules discussed in this section is really where you see the web services described in Section 3 come together with the CaptureFlow and IA values described in Section 4.1. It is also the place where you see how some of the design decisions made in those sections come into play.

Common to all three configurations will be the **WSDL URL**. Copy and paste the value obtained from the Catalina log file (as discussed in Section 3.4) into the **WSDL URL** field of the WSO setup screen and click the **Parse** button. The result should be `IncomingCaseDocsService` in the **Service name** field (the service name was defined in the code in Listing 2 Section 3.2.2), and one of the method names from Table 1 Section 3.1 in the **Method Name** field (see Figure 10).


InputAccel Web Services Output Setup - IncomingCaseDocument
EMC2 where information lives
Information
WS Output
Scripting
Error
Mapping
Parse
Mapping...
OK
Cancel
Apply
Help

WSDL URL: http://localhost:8080//IncomingCaseDocs/IncomingCaseDocsService?wsdl

Service name: IncomingCaseDocsService

Method Name: getCaseInfo

Use MTOM to send files

300 Request timeout in seconds

Figure 10 WSO Setup Screen

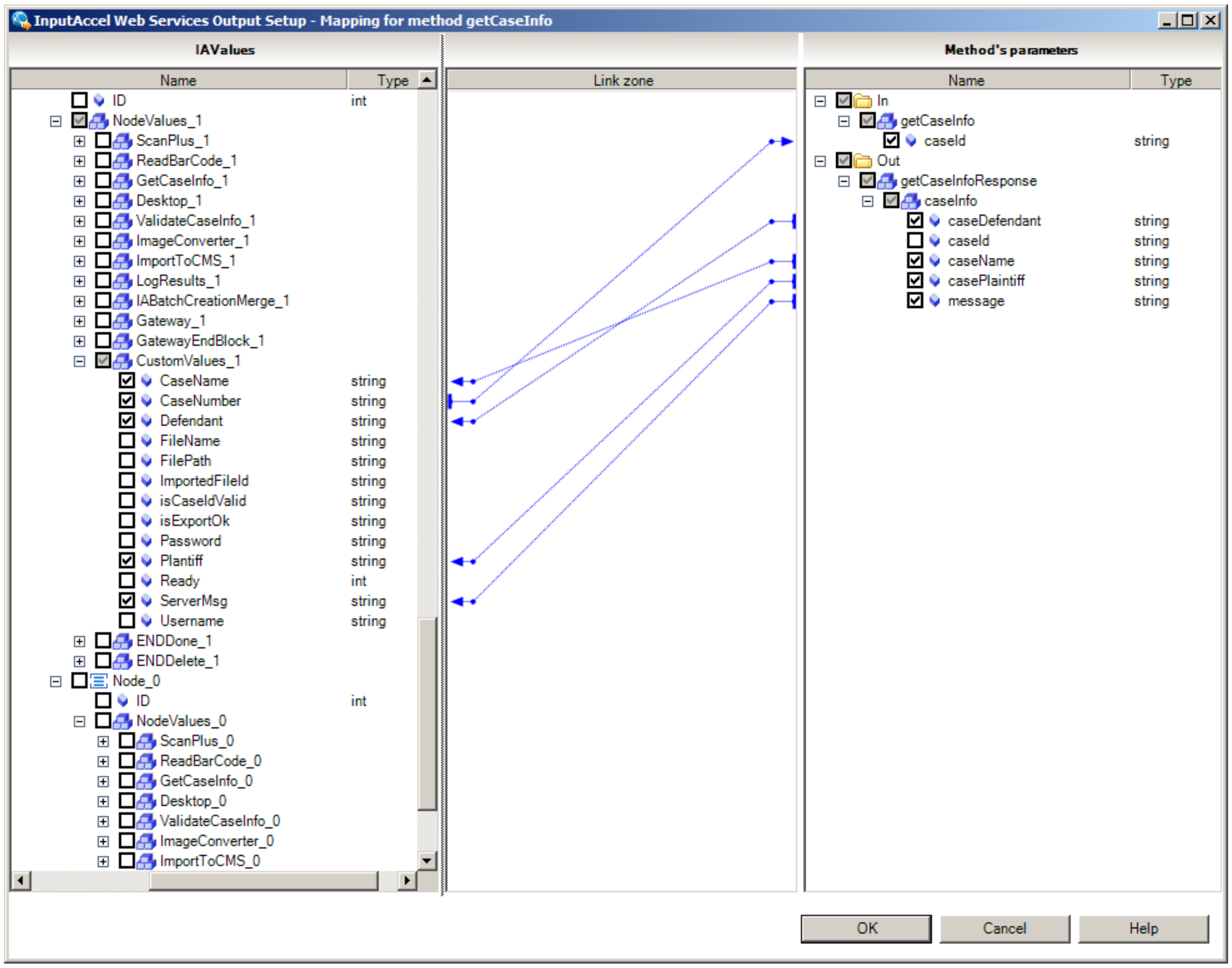
For each WSO module in the CaptureFlow, choose the appropriate method name to configure, and click the **Mapping** button. Table 6 contains the mapping between the process steps and the web methods.

Table 6 WSO Process Step Names Mapped To Web Method Names

WSO Process Step Name	Web Method Name
GetCaseInfo	*getCaseInfo()*
ValidateCaseInfo	*validateCaseId()*
ImportToCMS	*importFileToCMS()*

4.3.6.1 GetCaseInfo

Figure 11 depicts the mapping between the Custom and IA values and the web method parameters in the *GetCaseInfo* setup screen.


InputAccel Web Services Output Setup - Mapping for method getCaseInfo
IAValues
Link zone
Method's parameters
OK
Cancel
Help

```
IAValues                                   Method's parameters
Name                         Type          Name                         Type
☐ ID                         int           ☑ In
☑ NodeValues_1                                ☑ getCaseInfo
  ☐ ScanPlus_1                                   ☑ caseId                 string
  ☐ ReadBarCode_1                          ☑ Out
  ☐ GetCaseInfo_1                             ☑ getCaseInfoResponse
  ☐ Desktop_1                                    ☑ caseInfo
  ☐ ValidateCaseInfo_1                              ☑ caseDefendant     string
  ☐ ImageConverter_1                                ☐ caseId            string
  ☐ ImportToCMS_1                                   ☑ caseName          string
  ☐ LogResults_1                                    ☑ casePlaintiff     string
  ☐ IABatchCreationMerge_1                          ☑ message           string
  ☐ Gateway_1
  ☐ GatewayEndBlock_1
  ☑ CustomValues_1
    ☑ CaseName               string
    ☑ CaseNumber             string
    ☑ Defendant              string
    ☐ FileName               string
    ☐ FilePath               string
    ☐ ImportedFileId         string
    ☐ isCaseIdValid          string
    ☐ isExportOk             string
    ☐ Password               string
    ☑ Plantiff               string
    ☐ Ready                  int
    ☑ ServerMsg              string
    ☐ Username               string
  ☐ ENDDone_1
  ☐ ENDDelete_1
☐ Node_0
  ☐ ID                       int
  ☐ NodeValues_0
    ☐ ScanPlus_0
    ☐ ReadBarCode_0
    ☐ GetCaseInfo_0
    ☐ Desktop_0
    ☐ ValidateCaseInfo_0
    ☐ ImageConverter_0
    ☐ ImportToCMS_0
```

Figure 11 getCaseInfo Parameter Mapping

Table 7 contains a summary of the mappings depicted in Figure 11 .

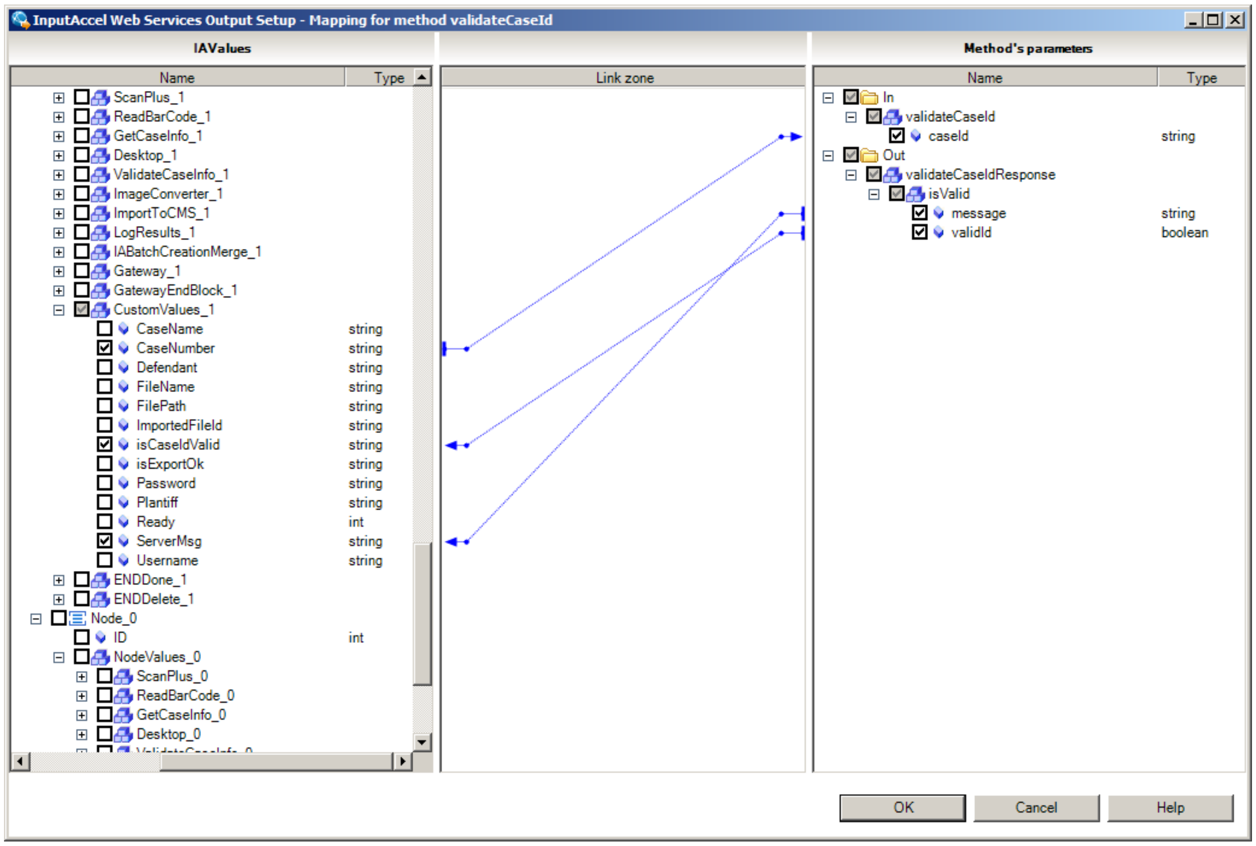
Table 7 getCaseInfo Parameter Mapping

IA Values		Method Parameters
`CustomValues_1.CaseName`	←	`Out.getCaseInfoResponse.caseInfo.caseName`
`CustomValues_1.CaseNumber`	→	`In.getCaseInfo.caseId`
`CustomValues_1.Defendant`	←	`Out.getCaseInfoResponse.caseInfo.caseDefendant`
`CustomValues_1.Plaintiff`	←	`Out.getCaseInfoResponse.caseInfo.casePlaintiff`
`CustomValues_1.ServerMsg`	←	`Out.getCaseInfoResponse.caseInfo.message`

Notice how the web method's parameters displayed in Figure 11 match the `@annotations` made in the web services code in Listing 1 Section 3.2.1. The `@WebMethod(operationName="getCaseInfo")` caused the operation name to be "`getCaseInfo`". `@WebResult(name="caseInfo")` caused the name of the resulting complex type to be “caseInfo”. Without these @annotations in the web services code, the method’s parameters in WSO would display as arg0, arg1, etc. Interestingly, the names of the actual web service output variables are the names of the *private* class variables of the WSOCaseInfoResult object (see Listing 6 , Section 3.2.3.1). This phenomenon is a result of the JAX-WS serialization of the web services objects. Because of this, it is critical that you use properly cased variable names in your class files and create properly cased getters/setters also.

4.3.6.2 ValidateCaseInfo

Figure 12 depicts the mapping between the Custom and IA values and the web method parameters in the *ValidateCaseInfo* setup screen.


InputAccel Web Services Output Setup - Mapping for method validateCaseId
IAValues
Link zone
Method's parameters
Name
Type
OK
Cancel
Help

Name	Type
☐ ScanPlus_1	
☐ ReadBarCode_1	
☐ GetCaseInfo_1	
☐ Desktop_1	
☐ ValidateCaseInfo_1	
☐ ImageConverter_1	
☐ ImportToCMS_1	
☐ LogResults_1	
☐ IABatchCreationMerge_1	
☐ Gateway_1	
☐ GatewayEndBlock_1	
☑ CustomValues_1	
☐ CaseName	string
☑ CaseNumber	string
☐ Defendant	string
☐ FileName	string
☐ FilePath	string
☐ ImportedFileId	string
☑ isCaseIdValid	string
☐ isExportOk	string
☐ Password	string
☐ Plantiff	string
☐ Ready	int
☑ ServerMsg	string
☐ Username	string
☐ ENDDone_1	
☐ ENDDelete_1	
☐ Node_0	
☐ ID	int
☐ NodeValues_0	
☐ ScanPlus_0	
☐ ReadBarCode_0	
☐ GetCaseInfo_0	
☐ Desktop_0	

Name	Type
☑ In	
☑ validateCaseId	
☑ caseId	string
☑ Out	
☑ validateCaseIdResponse	
☑ isValid	
☑ message	string
☑ validId	boolean

Figure 12 validateCaseInfo Parameter Mapping

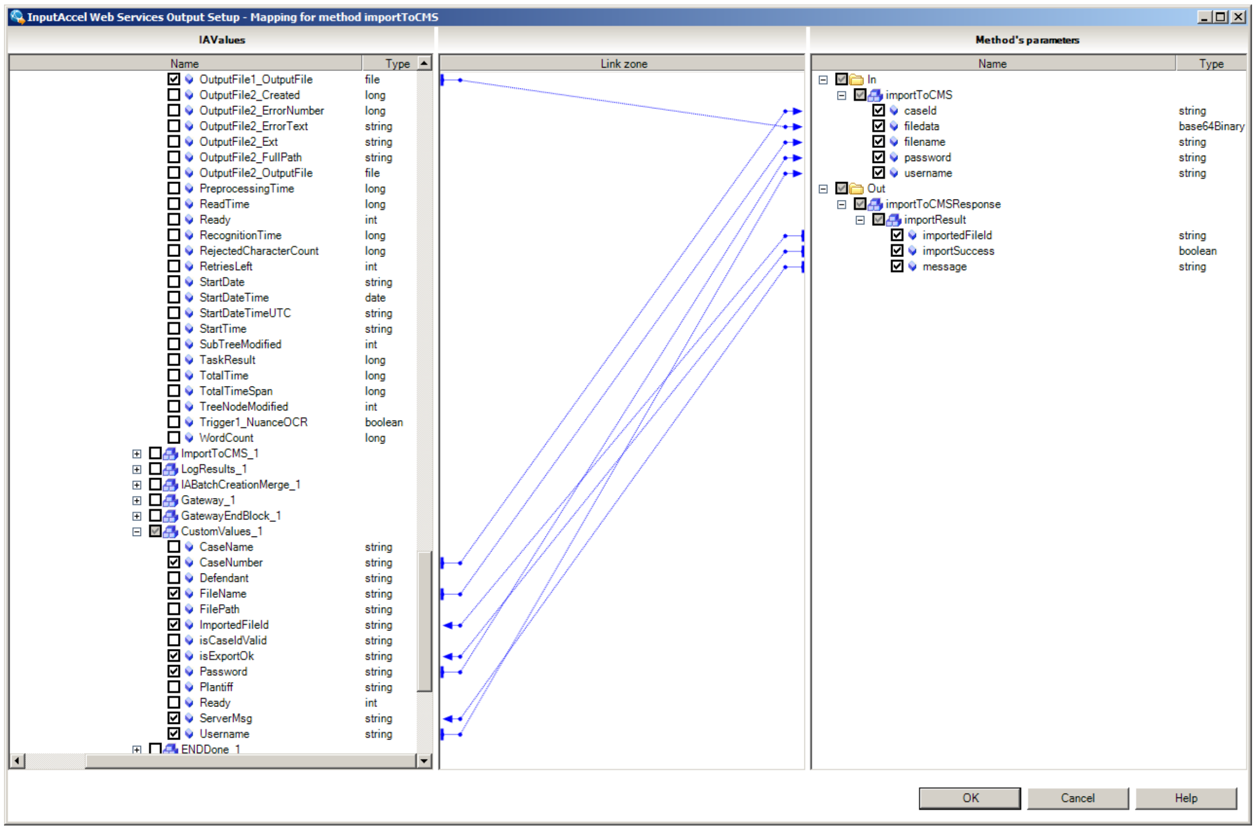
Table 8 contains a summary of the mappings depicted in Figure 12 .

Table 8 validateCaseInfo Parameter Mapping

IA Values		Method Parameters
CustomValues_1.CaseNumber	→	In.validateCaseInfo.caseId
CustomValues_1. isCaseIdValid	←	Out.validateCaseIdResponse.isValid. validId
CustomValues_1.ServerMsg	←	Out.validateCaseIdResponse.isValid. message

4.3.6.3 ImportToCMS

Figure 13 depicts the mapping between the Custom and IA values and the web method parameters in the *importToCMS* setup screen.



InputAccel Web Services Output Setup - Mapping for method importToCMS

IAValues

	Name	Type
☑	OutputFile1_OutputFile	file
☐	OutputFile2_Created	long
☐	OutputFile2_ErrorNumber	long
☐	OutputFile2_ErrorText	string
☐	OutputFile2_Ext	string
☐	OutputFile2_FullPath	string
☐	OutputFile2_OutputFile	file
☐	PreprocessingTime	long
☐	ReadTime	long
☐	Ready	int
☐	RecognitionTime	long
☐	RejectedCharacterCount	long
☐	RetriesLeft	int
☐	StartDate	string
☐	StartDateTime	date
☐	StartDateTimeUTC	string
☐	StartTime	string
☐	SubTreeModified	int
☐	TaskResult	long
☐	TotalTime	long
☐	TotalTimeSpan	long
☐	TreeNodeModified	int
☐	Trigger1_NuanceOCR	boolean
☐	WordCount	long
☐	ImportToCMS_1	
☐	LogResults_1	
☐	IABatchCreationMerge_1	
☐	Gateway_1	
☐	GatewayEndBlock_1	
☑	CustomValues_1	
☐	CaseName	string
☑	CaseNumber	string
☐	Defendant	string
☑	FileName	string
☐	FilePath	string
☑	ImportedFileId	string
☐	isCaseIdValid	string
☑	isExportOk	string
☑	Password	string
☐	Plantiff	string
☐	Ready	int
☑	ServerMsg	string
☑	Username	string
☐	ENDDone_1	

Link zone

Method's parameters

	Name	Type
☑	In	
☑	importToCMS	
☑	caseId	string
☑	filedata	base64Binary
☑	filename	string
☑	password	string
☑	username	string
☑	Out	
☑	importToCMSResponse	
☑	importResult	
☑	importedFileId	string
☑	importSuccess	boolean
☑	message	string

Figure 13 importToCMS Parameter Mapping

Table 9 contains a summary of the mappings depicted in Figure 13 .

Table 9 importToCMS Parameter Mapping

IA Values		Method Parameters
NuanceOCR_1.OutputFile1_ OutputFile	→	In.importToCMS.filedata
CustomValues_1. CaseNumber	→	In.importToCMS.caseId
CustomValues_1.FileName	→	In.importToCMS.filename
CustomValues_1. ImportedFileId	←	Out.importToCMSResponse.importResult. importFileId
CustomValues_1.isExportOk	←	Out.importToCMSResponse.importResult. importSuccess
CustomValues_1.Password	→	In.importToCMS.password
CustomValues_1.ServerMsg	←	Out.importToCMSResponse.importResult. Message
CustomValues_1.Username	→	In.importToCMS.username

Two parameters stand out here. The first is the `NuanceOCR_1.OutputFile1_OutputFile` mapped to `In.importToCMS.filedata`, and the second is `CustomValues_1.Password` mapped to `In.importToCMS.password`. In the case of the `NuanceOCR_1.OutputFile1_OutputFile`, this is the in-memory copy of the PDF created by the OCR process step that is mapped as input to the *`importToCMS`* web method. Specifically, I mapped it to the web method's `byte[]` input parameter, `filedata`. A little bit of MTOM magic happens here, and the binary content of the PDF file is transferred to the web service and ingested by the case management system (simulated).

Notice that during the configuration of this WSO module, I did not select the **Use MTOM to send files** checkbox. Intuitively, you would think checking this box would be necessary for sending file using MTOM; however, with the box checked, the module throws a runtime error citing an incorrect MIME type was used in the SOAP header.

The second parameter is the `CustomValues_1.Password` variable. As discussed previously in Section 4.2.2, capturing and passing around a password in this manner is not practical. This example is merely supposed to show you that you can send login credentials to web services and have the services to the authentication if necessary.

4.4 Deployment

Deploy the CaptureFlow and all of the profiles to the InputAccel server using the Captiva Designer.

5 Testing and Results

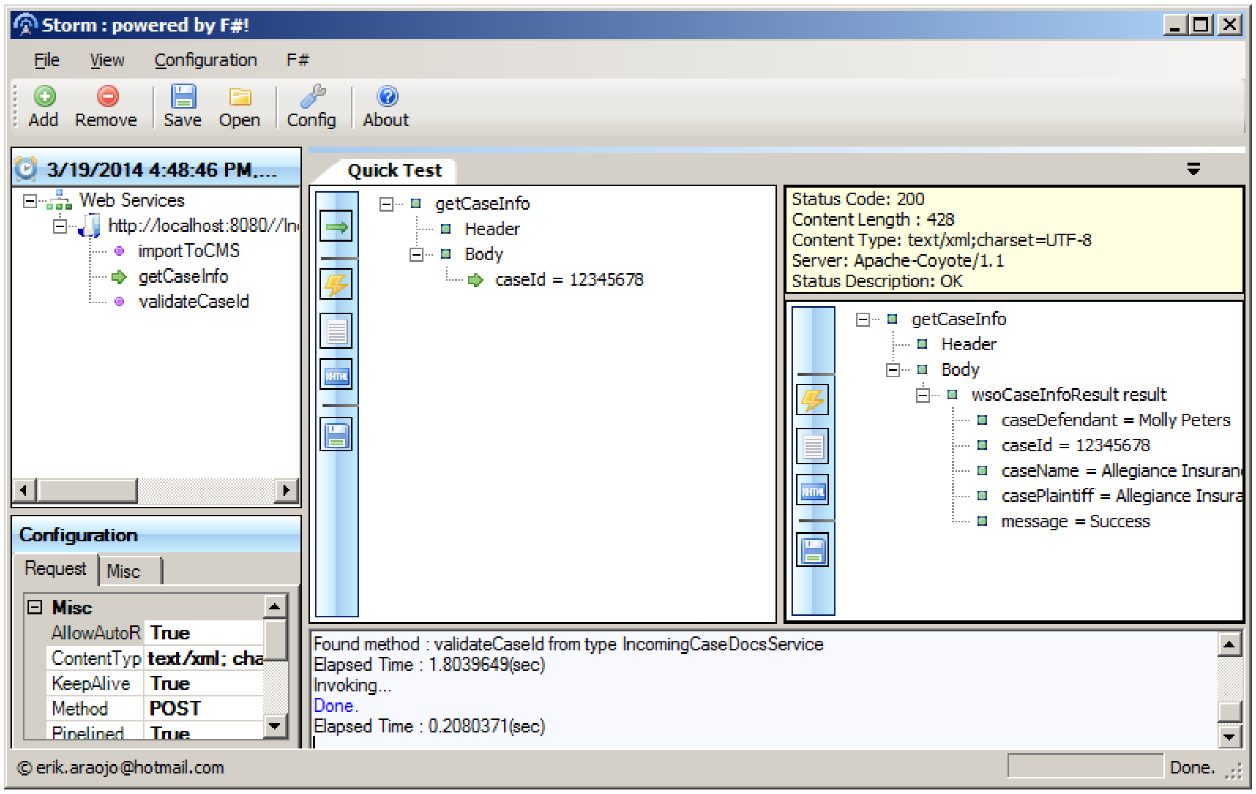
Now that the web services and the Captiva projects have been deployed, we will test them. This section briefly describes the different testing methods I used to verify the functionality of the solution at different levels.

5.1 Test Harness

The web services Eclipse project contains a test class, `com.dm_misc.captiva.wso.test.WSOTest`, that can be used to test the web service methods and result classes. This test harness does not instantiate the classes as web services to test them. Instead, it just instantiates them as POJOs and exercises their various methods and logic.

5.2 Storm

Once the web services were deployed to TomEE+, I used Storm (see References) to test each web method. Figure 14 depicts Storm testing the *`getCaseInfo()`* web method. As you can see, the input and output parameters are properly labeled (thanks to all those `@annotations`), and with the proper input, the expected output is returned.



```
Web Services
  http://localhost:8080//In
    importToCMS
    getCaseInfo
    validateCaseId
```

```
getCaseInfo
  Header
  Body
    caseId = 12345678
```

```
Status Code: 200
Content Length : 428
Content Type: text/xml;charset=UTF-8
Server: Apache-Coyote/1.1
Status Description: OK
```

```
getCaseInfo
  Header
  Body
    wsoCaseInfoResult result
      caseDefendant = Molly Peters
      caseId = 12345678
      caseName = Allegiance Insuran
      casePlaintiff = Allegiance Insura
      message = Success
```

```
Misc
AllowAutoR  True
ContentTyp  text/xml; cha
KeepAlive   True
Method      POST
Pipelined   True
```

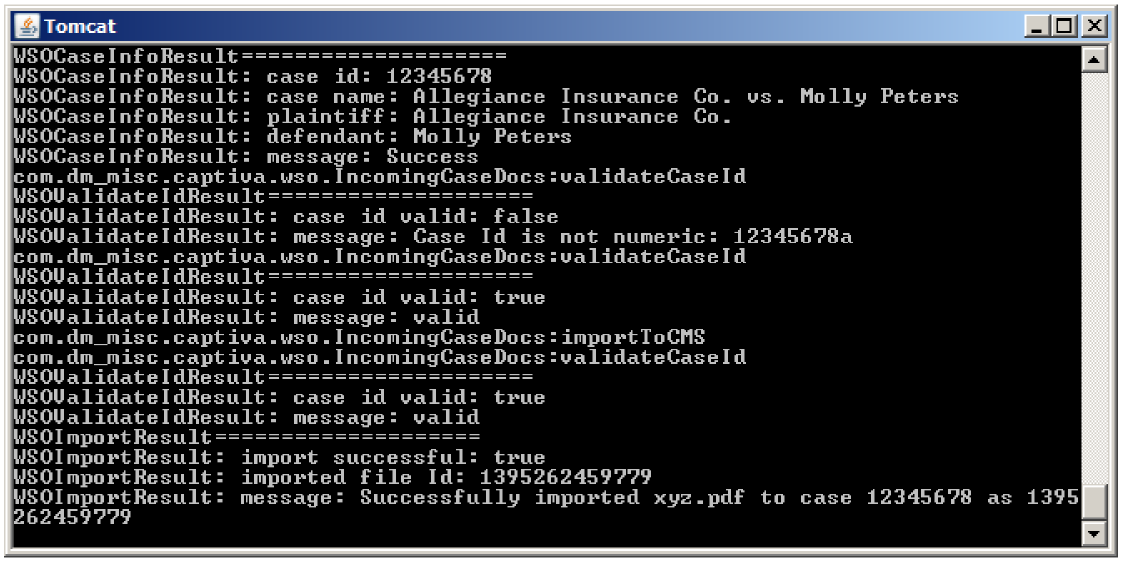
```
Found method : validateCaseId from type IncomingCaseDocsService
Elapsed Time : 1.8039649(sec)
Invoking...
Done.
Elapsed Time : 0.2080371(sec)
```

© erik.araojo@hotmail.com Done.

Figure 14 Storm Web Services Test Client

5.3 Standard Out

All of the web service result classes (see Section 3.2.3) contain simple `System.out.println()` statements that output messages to the TomEE+ console window. For brevity, these statements were omitted from the code listings in Section 3.2.3. Figure 15 depicts the output messages displayed on the TomEE+ console. These messages help trace the processing of the web service.

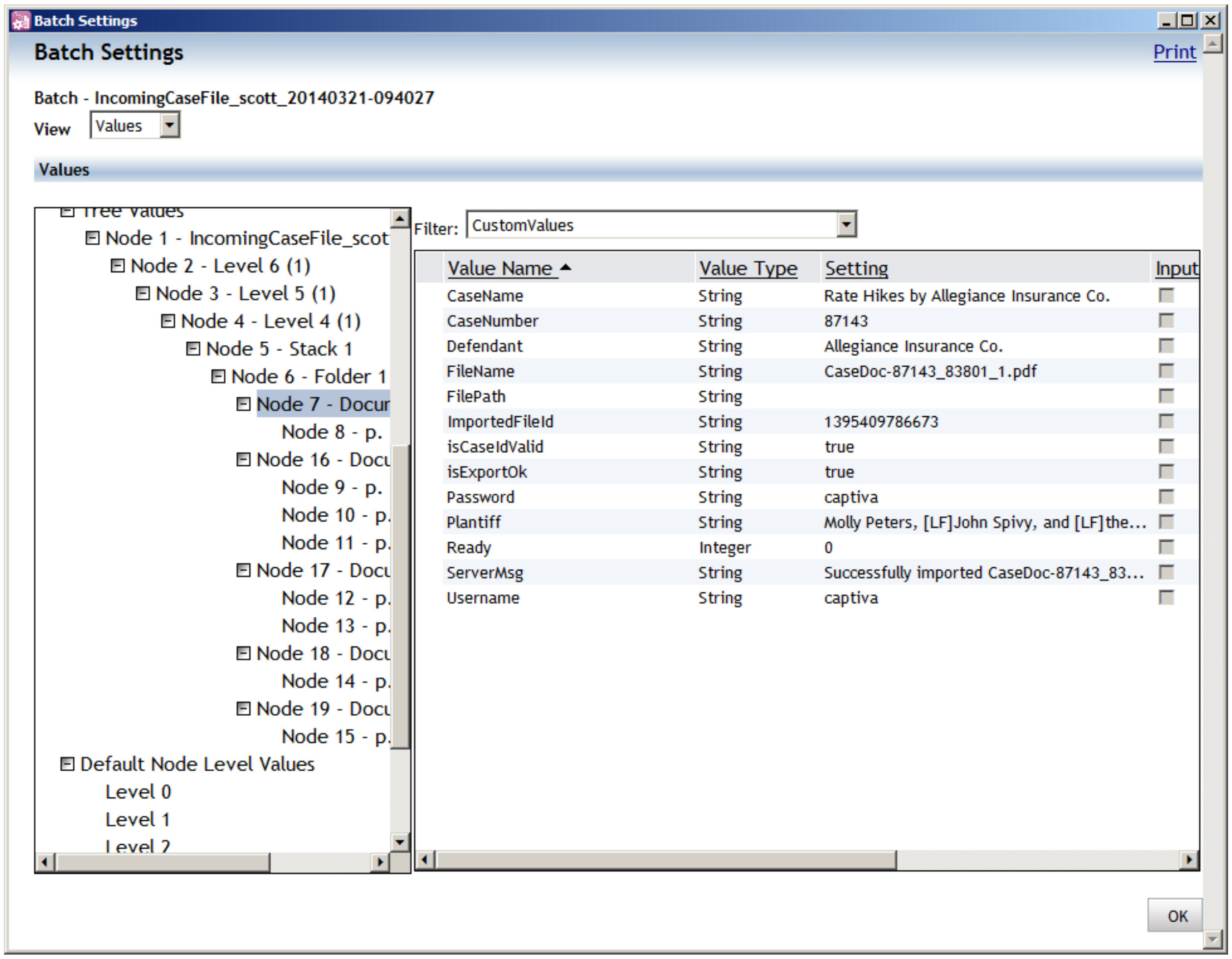


```
WSOCaseInfoResult===================
WSOCaseInfoResult: case id: 12345678
WSOCaseInfoResult: case name: Allegiance Insurance Co. vs. Molly Peters
WSOCaseInfoResult: plaintiff: Allegiance Insurance Co.
WSOCaseInfoResult: defendant: Molly Peters
WSOCaseInfoResult: message: Success
com.dm_misc.captiva.wso.IncomingCaseDocs:validateCaseId
WSOValidateIdResult===================
WSOValidateIdResult: case id valid: false
WSOValidateIdResult: message: Case Id is not numeric: 12345678a
com.dm_misc.captiva.wso.IncomingCaseDocs:validateCaseId
WSOValidateIdResult===================
WSOValidateIdResult: case id valid: true
WSOValidateIdResult: message: valid
com.dm_misc.captiva.wso.IncomingCaseDocs:importToCMS
com.dm_misc.captiva.wso.IncomingCaseDocs:validateCaseId
WSOValidateIdResult===================
WSOValidateIdResult: case id valid: true
WSOValidateIdResult: message: valid
WSOImportResult===================
WSOImportResult: import successful: true
WSOImportResult: imported file Id: 1395262459779
WSOImportResult: message: Successfully imported xyz.pdf to case 12345678 as 1395
262459779
```

Figure 15 TomEE+ Console Output

5.4 IA Administrator

Finally, the IA Administrator is an invaluable tool for debugging and monitoring your Captiva CaptureFlows. I am not going to cover all of the aspects of using the IA Administrator, but want to point out one helpful feature. Open a batch's settings (double-click the batch in the IA Administrator's **Batch Traffic** window), change the view to **Values**, and the Filter to **CustomValues** (see Figure 16). Choose a document node from the node tree on the left and examine the values in the right-hand pane. Here you see all of the Custom Values used by our CaptureFlow, including all of the values sent to and received from the web service. Of special note is the `ServerMsg` value which contains the informational message from the web method.



Batch Settings

Batch - IncomingCaseFile_scott_20140321-094027

View: Values

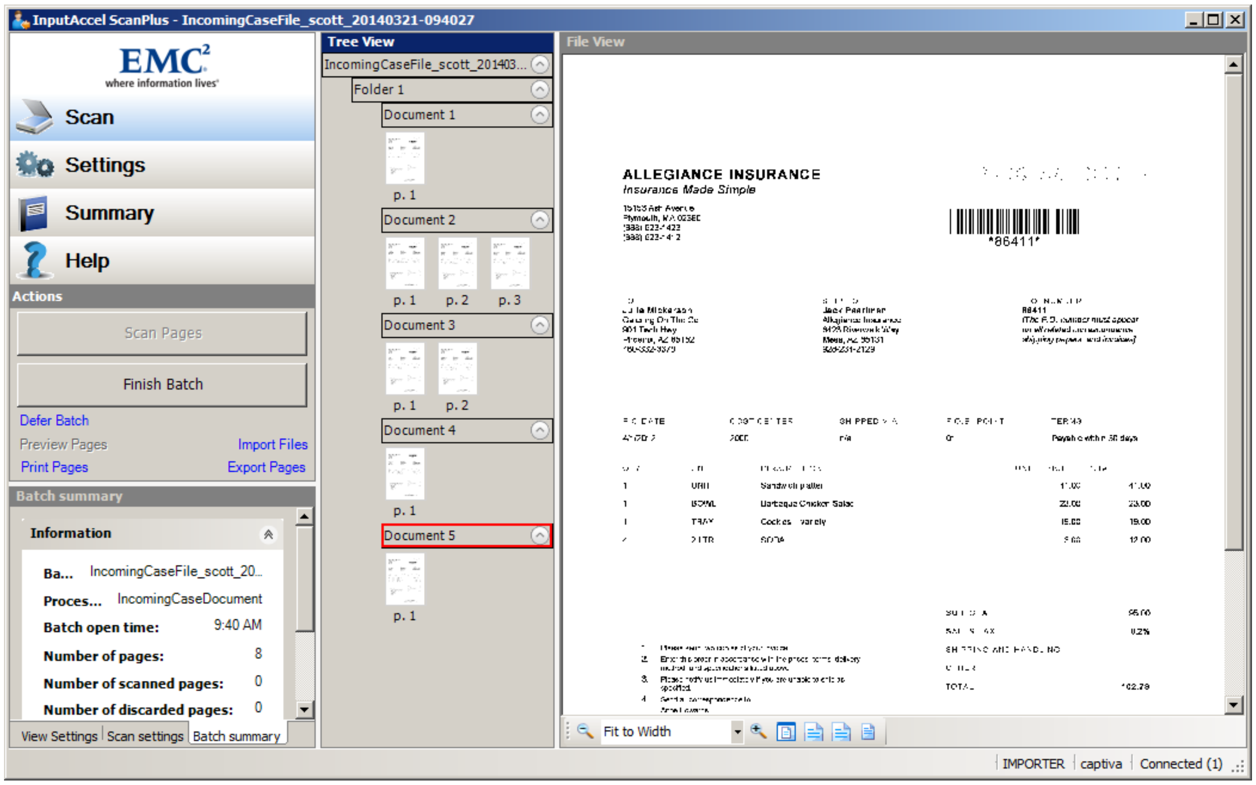
Filter: CustomValues

Value Name	Value Type	Setting	Input
CaseName	String	Rate Hikes by Allegiance Insurance Co.	☐
CaseNumber	String	87143	☐
Defendant	String	Allegiance Insurance Co.	☐
FileName	String	CaseDoc-87143_83801_1.pdf	☐
FilePath	String		☐
ImportedFileId	String	1395409786673	☐
isCaseIdValid	String	true	☐
isExportOk	String	true	☐
Password	String	captiva	☐
Plantiff	String	Molly Peters, [LF]John Spivy, and [LF]the...	☐
Ready	Integer	0	☐
ServerMsg	String	Successfully imported CaseDoc-87143_83...	☐
Username	String	captiva	☐

Figure 16 IA Administrator Values View

5.5 Complete Run Through

This last section depicts a complete run through of the CaptureFlow. For simplicity, I am importing files into the CaptureFlow instead of scanning them. Specifically, I am using the sample images distributed with Captiva, located on the IA server at: `C:\Users\captiva\Documents\Captiva 7.0\samples\Images\production auto learning`. Figure 17 depicts the ScanPlus screen after importing eight sample images as five documents. Clicking the **Finish** button starts the CaptureFlow process.

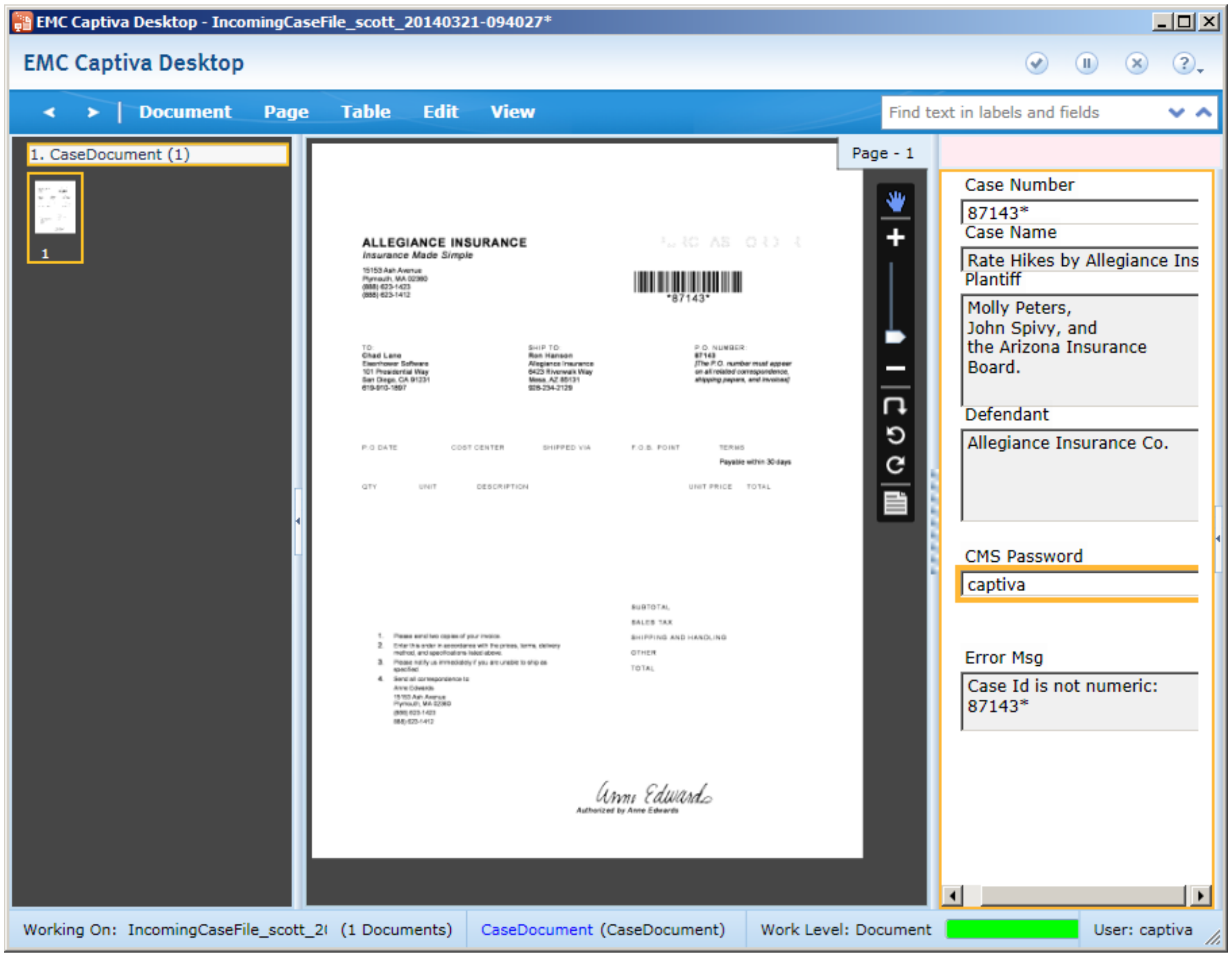


Ba...	IncomingCaseFile_scott_20...
Proces...	IncomingCaseDocument
Batch open time:	9:40 AM
Number of pages:	8
Number of scanned pages:	0
Number of discarded pages:	0

Figure 17 ScanPlus

After a moment, the sample images should show up in the Captiva Desktop queue for processing, having had their barcodes read and their case info retrieved via the WSO module. Figure 18 depicts the Captiva Desktop screen. Note the field values in the right-hand pane of the window. In this example, I deliberately append an asterisks (*) to the **Case Number** in order to force an error. The result is the screen you see. Note the **Error Msg** that was returned from the web service.

After correcting the **Case Number** and entering the **Password** again, the document is processed successfully.



Case Number
87143*

Case Name
Rate Hikes by Allegiance Ins

Plantiff
Molly Peters,
John Spivy, and
the Arizona Insurance
Board.

Defendant
Allegiance Insurance Co.

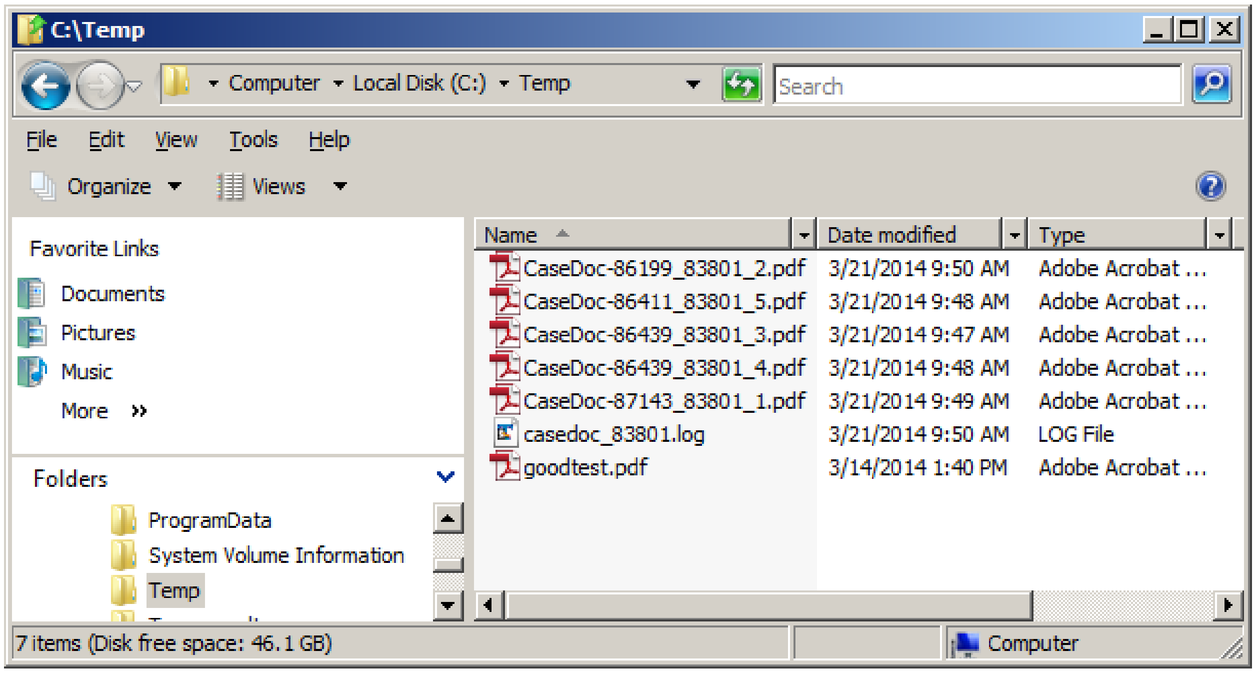
CMS Password
captiva

Error Msg
Case Id is not numeric:
87143*

Working On: IncomingCaseFile_scott_2((1 Documents) | CaseDocument (CaseDocument) | Work Level: Document | User: captiva

Figure 18 Captiva Desktop

There are two primary results of a successful run of the CaptureFlow: PDF files are saved in the `c:\temp` directory, and a log file is created. Figure 19 depicts the `c:\temp` directory after a successful run. Note the following: The files are all PDF, thanks to the *OCR* step; the file names all follow the correct naming convention; and a log file exists. The existence of the PDF files in this directory is proof that the web service is working: the PDF file was passed from the *NuanceOCR* step to the *`importToCMS()`* web method, and saved here to simulate interaction with a case management system.

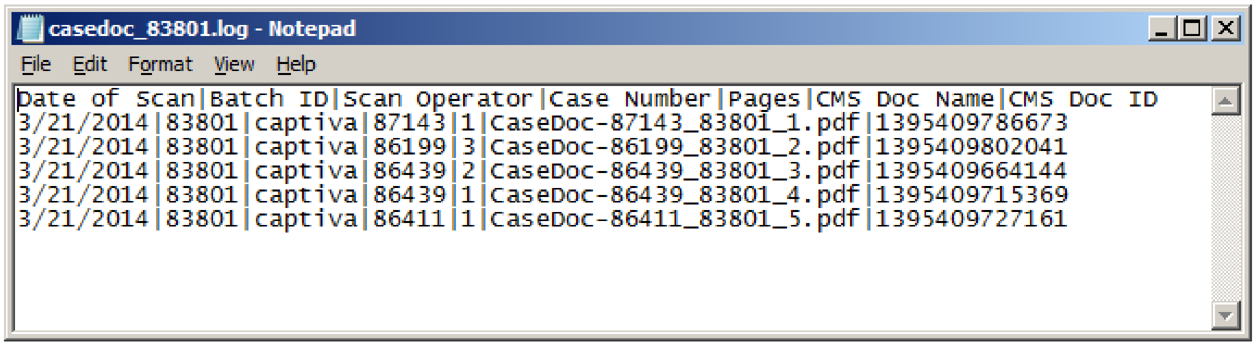


Name	Date modified	Type
CaseDoc-86199_83801_2.pdf	3/21/2014 9:50 AM	Adobe Acrobat ...
CaseDoc-86411_83801_5.pdf	3/21/2014 9:48 AM	Adobe Acrobat ...
CaseDoc-86439_83801_3.pdf	3/21/2014 9:47 AM	Adobe Acrobat ...
CaseDoc-86439_83801_4.pdf	3/21/2014 9:48 AM	Adobe Acrobat ...
CaseDoc-87143_83801_1.pdf	3/21/2014 9:49 AM	Adobe Acrobat ...
casedoc_83801.log	3/21/2014 9:50 AM	LOG File
goodtest.pdf	3/14/2014 1:40 PM	Adobe Acrobat ...

7 items (Disk free space: 46.1 GB)

Figure 19 Case Docs in Temp Folder

Figure 20 depicts the log file generated by the Standard Export profile created in Section 4.2.3. This file is used by the scan operators to verify and track the successful import of scanned files into the case management system.



```
Date of Scan|Batch ID|Scan Operator|Case Number|Pages|CMS Doc Name|CMS Doc ID
3/21/2014|83801|captiva|87143|1|CaseDoc-87143_83801_1.pdf|1395409786673
3/21/2014|83801|captiva|86199|3|CaseDoc-86199_83801_2.pdf|1395409802041
3/21/2014|83801|captiva|86439|2|CaseDoc-86439_83801_3.pdf|1395409664144
3/21/2014|83801|captiva|86439|1|CaseDoc-86439_83801_4.pdf|1395409715369
3/21/2014|83801|captiva|86411|1|CaseDoc-86411_83801_5.pdf|1395409727161
```

Figure 20 Case Doc Log File

6 Conclusion

The purpose of this tutorial was to provide a practical, hands-on example of using Captiva's Web Services Output module. The tutorial was presented in two primary parts: creating web services that can be consumed by WSO, and creating a CaptureFlow that consumes the web services.

I hope this tutorial has demonstrated how interfacing Captiva to external systems via web services expands Captiva's capabilities considerably. Though the constraint of only consuming anonymous, SOAP web services, is limiting in some environments, there are techniques for securing – or at least improving the security of – these web services. The ability to return complex types to WSO from web services further expands WSO's usefulness and ease of integration. When web services return complex results to the CaptureFlow (e.g., collections or multiple type results), they can be assigned to Custom Values or IA values to augment or affect the capture process.

There are other, and perhaps more effective ways to achieve the same results demonstrated in this tutorial. However, I don't believe that there are any simpler ways. One of the goals of this tutorial was to implement a solution that used no Captiva client-side scripting to implement a WSO solution. This tutorial has achieved that goal by using only Custom and IA value assignments, CaptureFlow logic, and module configuration.

Here are some key points from this tutorial:

- *Anonymous web services* – The Captiva WSO module can only consume anonymous, SOAP-based web services. These two limitations constrain how your web services can be implemented. The tutorial makes suggestions for alleviating some of these constraints (e.g., using IP filtering) as does the *EMC Captiva Capture Web Services Guide* (e.g., using SSL). The tutorial provides examples of creating and implementing anonymous web services.
- *Returning complex types* - If you have any control over the format of the web service return values, I suggest that you always return POD objects as complex types. This affords you the option of returning multiple and various data types from a single web method. As demonstrated in the tutorial, it is often necessary to return collections of items (e.g., arrays of `Strings`), or `Strings` and a `Boolean`, etc. Another good practice is to return informational messages from the web methods to WSO for inclusion in the Custom Values and/or display to the scan operator. This will assist in troubleshooting problems in the future.
- *Level of triggering web services* – Ensure the trigger level of the WSO is appropriate for both the service and the data passed to it. For example, in the tutorial the *ImportToCMS* step triggered for each document in the batch. If the batch contained 50 documents, it would trigger 50 times. Depending upon your situation, this could cause a performance bottleneck. It wouldn't be too difficult to change the *ImportToCMS* trigger level to `Batch`, and update the web service to receive the entire batch's documents all at once, thus reducing the number of web service calls by 49.
- *Assignment of values* – Be very careful of your assignment of values to Custom and IA values both in the CaptureFlow and in the WSO mapping tool. Even though the tools will allow you to make virtually any assignment you want, not all assignments will work as expected due to value access across levels of the Captiva node tree.
- *WSDL changes* - Any time you make a change to the web service code that results in a change to the WSDL file, you will need to reconfigure ALL WSO modules in the CaptureFlow.
- *Using MTOM to transfer files* – I mentioned it in the tutorial, but it bears repeating: if you use MTOM to transport binary content to a web service, DO NOT check the **Use MTOM to send files** checkbox on the WSO setup screen, it causes an error in the web service.

I hope you have enjoyed this tutorial and have found it helpful. If you are interested, the Eclipse project and the Captiva Designer project can be downloaded here:

- Both projects (2.6 MB) – https://app.box.com/s/hrsog9afxg947kxd667i
- Eclipse project (1.9 MB) – https://app.box.com/s/0khzu76fniokkmop9gdf
- Captiva Designer project (724 KB) – https://app.box.com/s/1nj8h5v8po4gcxw28vot

7 References

Following are links to web sites and documentation I found useful while developing this tutorial.

- *EMC Captiva Capture Web Services Guide* - Web Services Guide > Setup > Securing Web Services Communications
- TomEE+ pre-configured application servers - http://tomee.apache.org/
- Eclipse IDE – http://www.eclipse.org
- Simple web services tutorial using Eclipse and TomEE - http://blog.sortedset.com/step-by-step-web-services-with-tomcat-tomee-apache-cxf-eclipse/
- Web service tutorial - http://www.yourepeat.com/watch/?v=mGlPXKJo_6U
- http://java.dzone.com/articles/creating-and-deploying-jax-ws
- http://www.myeclipseide.com/documentation/quickstarts/webservices_jaxws/
- http://tomee.apache.org/examples-trunk/simple-webservice/README.html
- MTOM - http://cxf.apache.org/docs/mtom.html,
- http://www.mkyong.com/webservices/jax-ws/jax-ws-attachment-with-mtom/
- JAX-WS annotations http://pic.dhe.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=%2Fcom.ibm.websphere.wsfep.multiplatform.doc%2Finfo%2Fae%2Fae%2Frwbs_jaxwsannotations.html
- Web Services FAQ: http://www.coderanch.com/how-to/java/WebServicesFaq

8 Acknowledgements

I extend special thanks and recognition to Eric Chen, Brian Yasaki, and Rachael Roth for their assistance with this tutorial.

<SDG><