

Secure Authentication with Flash Communication Server: Using Tickets and Flash Remoting MX to Transmit Secure Information

JAN 31, 2003 By [Kevin Towes](#).

*Excerpted from Macromedia **Flash Communication Server MX, 0735713332s**; **Kevin Towes**, Copyright [2002]. Reproduced by permission of New Riders (www.newriders.com).*



Kevin Towes, author of *Macromedia Flash Communication Server MX*, shows you how to construct a basic authentication schema for Flash Communication Server Applications.

Other articles by [Kevin Towes](#).

There has been a lot of discussion about Security and Macromedia Flash Communication Server MX. The most common topic being addressed in the discussion boards and the newsgroups is that the RTMP Protocol cannot operate using a secure encryption key, which is commonly used in the web's HTTPS protocol. *Real Time Message Protocol* (RTMP) is Flash Communication Server's proprietary method for persistent communication between the server and Flash Player 6. It was developed by Macromedia specifically for the Flash Communication Server, to transport multi-way audio and video streams as well as data and broadcast messaging using the Action Message Format (AMF).

This discussion has led many people to believe that a Flash Communication Server Application cannot protect the transmission of secure data, such as passwords and credit card information. It is true that RTMP is not a secure transport, but the Flash player and the Flash Communication Server *can* communicate independently over a *secure* HTTPS protocol using Flash Remoting MX.

Flash Remoting MX was introduced in the Spring of 2002, with the release of Macromedia ColdFusion MX and Macromedia JRUN 4. It enables Flash applications to communicate easily and securely with Application servers that are connected to Databases. Flash Remoting MX uses the HTTP protocol or the secure HTTPS protocol to transmit AMF packets that contain ActionScript objects, such as arrays, recordset, or any other valid ActionScript data values such as Boolean, string, or numeric. To use Flash Remoting MX, you must be running a Macromedia MX Application Server such as ColdFusion or JRUN 4, or have Flash Remoting MX installed for J2EE or Microsoft .NET.

This article will give you a hands-on introduction to constructing a basic authentication schema for Flash Communication Server Applications. [Figure 1](#) is an overview of the process this technique will follow. Below the diagram is a list of the six steps.

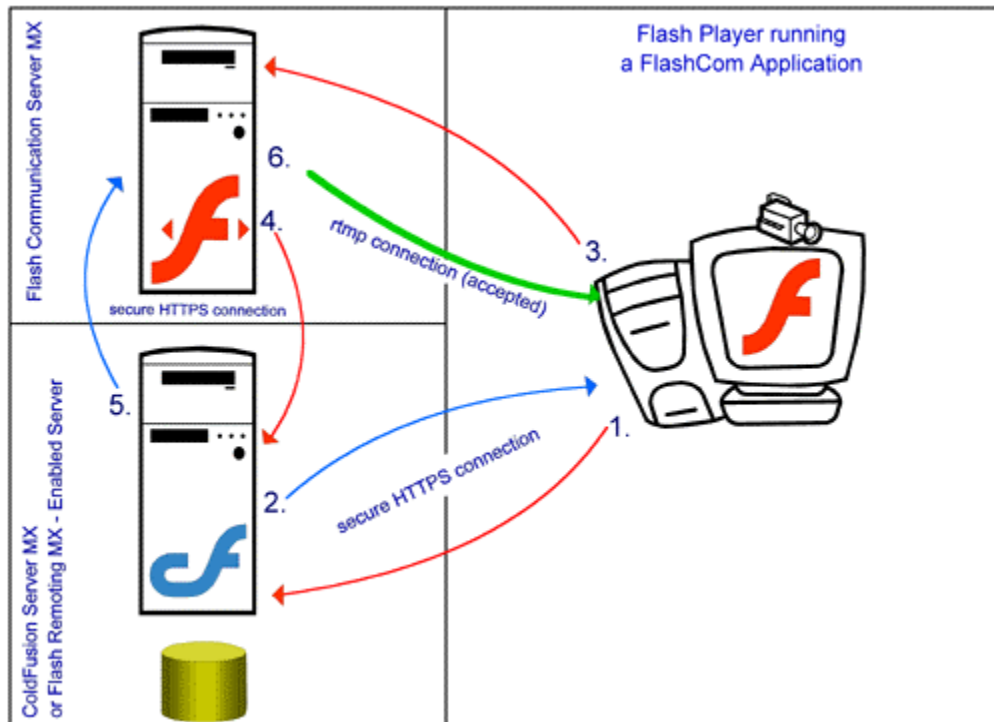


Figure 1 The communication flow from Player to Flash Communication Server to ColdFusion.

1. The Flash Player sends a login and password challenge over a secure HTTPS connection to a web service method on a ColdFusion MX Application Server connected to a database or LDAP Server. This action is performed using Flash Remoting MX.
2. The ColdFusion MX Server receives the call and processes the login and password. If successful, ColdFusion returns a unique HASH "Ticket" to the Caller (the Flash Player). The Ticket and an expiry date are also stored in a table within the database.
3. When the Ticket is received by the Flash Player, a NetConnection request is sent to the Flash Communication Server. The Ticket received from ColdFusion is sent along with the request.
4. The Flash Communication Server calls a remote method on the ColdFusion server using a secure HTTPS connection. It sends the Ticket received from the Flash Player to the server for validation.
5. The ColdFusion Server validates the ticket, and returns a true or false response with (if available) the user's full name.
6. The Flash Communication Server accepts the connection based on the response from ColdFusion and registers the user's name in the UI Component Framework.

At this point the connection has been successfully made to the Flash Communication Server. At no point during this process was there any insecure transmission of sensitive information.

Let's now take a detailed walkthrough of the processes involved.

Getting Started

For this exercise, you will need the following components installed and operational. This article will assume that all these components are installed and running on the SAME workstation.

- Flash Communication Server MX (any edition)
- Flash MX Authoring Environment (*with the Communication and Remoting Components installed*)
- Macromedia Dreamweaver MX or ColdFusion Studio
- ColdFusion MX (Any Edition)
- Webserver (IIS or ColdFusion Stand-Alone)

Like most Flash Communication Server applications, there are a number of components that you will need to build separately. The completed source code can be downloaded from the *Macromedia Flash Communication Server MX* book's web site: <http://flashcom.PangaeanewMedia.ca/>. You can also see a working version of the example application on that web site. I've tried to keep it as simple as possible, but there are four parts to his application.

- **"myFlashComDb.mdb"**: The Microsoft Access Database that contains user information
- **"FCS_Security.cfc"**: ColdFusion Component that contains the remote service methods.
- **"FCS_Secure fla"**: Flash MX source File that is used within Flash MX
- **"main.asc"**: the SSAS file on the Flash Communication Server

Before you begin, create a folder within the [%webRoot%]/**flashcom/applications** folder called **"informIT"**. All files for this project will be stored within that folder. Make sure your Flash Communication Server recognizes this folder as a Flash Communication Server Application folder. Also make sure that your default web server has access to this folder. When you are ready to move your files to a live (public) server, make sure the FlashCom/Applications folder is not accessible through your web server.

Exercise 1: The Database, "myFlashComDb.mdb"

The Database for this project is simple. It uses two database tables, "Users" and "Users_Ticket". Using Microsoft Access, or your favourite database software, create these tables similar to the ones shown in [Figure 2](#) and [Figure 3](#). Create the database with the name "myFlashComDb.mdb".

Field Name	Data Type	Description
UserID	AutoNumber	
fname	Text	The Users First Name
lname	Text	Their Last Name
login	Text	A Login to access the sytem
password	Text	A Password to access the system
emailAddr	Text	Email Address
phoneNum	Text	Phone Number
streetAddr	Text	Street Address

Figure 2 Users Table; Design View, MSAccess.

Field Name	Data Type	Description
ticketID	AutoNumber	
ticket	Text	the Ticket Generated By ColdFusion
UserID	Number	The Related User Record ID
expiry	Date/Time	The Expiry Date of the Ticket

Figure 3 Users_Ticket Table; Design View, MSAccess.

After you have created the database, add some values to the Users table, particularly login, password, first name, and last name. Add a couple rows so you have something to test with.

Next, connect the database to the ColdFusion MX Server (see [Figure 4](#)). Log into the ColdFusion Administrator and add the datasource "myFlashComDb.mdb" to the Data Sources control panel.

Microsoft Access Data Source: myFlashComDb.mdb

CF Data Source Name: myFlashComDb.mdb

Database File: om|applications|informIT|myFlashComDb.mdb [Browse Server]

System Database File: [Browse Server]

Use Default Username:

Description: [Text Area]

[Show Advanced Settings] [Submit] [Cancel]

If you downloaded the example applications, you will need to manually register the database.

`C:\Inetpub\wwwroot\flashcom\applications\informIT\myFlashComDb.mdb`

With the database set up, let's move on to the fun stuff. We'll start with the ColdFusion Component, and then we'll move onto Flash and writing some Server-Side ActionScript (SSAS).

Exercise 2: The ColdFusion Component, "FCS_Security.cfc"

In this exercise you construct the ColdFusion service methods used by both the Flash Communication Server and Flash Player 6. You will build these service methods inside a ColdFusion Component (not to be confused with the Flash UI Components—they are completely different). These service methods connect with the database you built in Exercise 1. There are three principle methods (or CFFUNCTION objects) required for this project; however, a fourth function is included to help you grow this application.

1. Create a new file in Dreamweaver MX, or your favourite ColdFusion IDE called "FCS_Security.cfc", and place it within the folder **flashcom/applications/informIT**. Make sure this folder is mapped from the default web server. (This will be the same web server that you use to access the ColdFusion Administrator.)
2. Add the following CFCOMPONENT tags to the file. The methods following will be placed within these tags:

```
<cfcomponent>
```

```
<!--- place the cffunction definitions between these tags --->
```

```
</cfcomponent>
```

Method #1: authenticateUser

The authenticateUser function is used by Flash MX to make the initial login/password challenge to the database. It requires two parameters, login and password. This function returns an object to Flash, so the object (ret_obj) is setup, as a ColdFusion structure.

Initial keys (or properties) are set, and then the database query is challenged. This will determine if the login and password matched a record in the database. If a match was found, the function "setTicket" is called to generate a ticket.

The setTicket function is defined after this one. It returns a ColdFusion structure with two keys, TICKET and EXPIRY. When received, these keys are copied into the ret_Obj object that is returned to the Flash Player. [Figure 5](#) shows the complete structure of ret_Obj returned by the authenticateUser function.

```
<cffunction name="authenticateUser" access="remote"
returntype="any">
    <cfargument name="login"
type="string" required="yes">
```

```

<cfargument name="password"
type="string" required="yes">

<!--- Challenge the Login / Password sent from the Flash Player --
->
<CFQUERY NAME="q_checkUser" Datasource="myFlashComDb.mdb">
    SELECT UserID, fname, lname From Users
        WHERE login = '#arguments.login#'
        AND Password = '#arguments.password#';
</CFQUERY>
<CFSCRIPT>
    // Setup return object structure, and set the initial values of
the keys
    ret_Obj = structNew();
    ret_Obj.isLoginOK= false;
    ret_Obj.userName = q_checkUser.fname & " " & q_checkUser.lname;

    // Check if the Login & Password received, matched a record in
the Db
    if (q_checkUser.recordCount neq 0) {
        // call the cf function, "setTicket" passing it the UserID
        ticketObj = this.setTicket(q_checkUser.UserID);
        // Copy the ticket object into the local ret_obj scope
        ret_Obj.ticket = ticketObj.ticket;
        ret_Obj.expiry = ticketObj.expiry;
        ret_Obj.isLoginOK= true;
    }
    // RETURN the structure. The Flash Remoting Gateway will
convert this structure
    // into a valid ActionScript object.
    //The structure's keys will become object properties
    return ret_Obj;
</CFSCRIPT>
</cffunction>

```

Function: authenticateUser RETURNED Object Structure

struct	
EXPIRY	{ts '2012-12-09 20:06:55'}
ISLOGINOK	true
TICKET	2D4367DF5BBFDB03B804F1BFA84D9DF1
USERNAME	Kevin Towes

Figure 5 The ColdFusion Structure returned by the authenticateUser method is

assembled automatically by Flash Remoting MX into an ActionScript Object with four properties.

Method #2: setTicket

The setTicket method is not directly called by either the Flash Communication Server or the Flash Player. It is intended to be called by the authenticateUser() method. Its role is to simply generate a ticket and the expiry date. It also stores these values in a separate table in the database. This storage bridges the gap between the Flash Player and Flash Communication Server.

The internal ColdFusion function **CreateUUID()** will generate a unique value. The **Universally Unique Identifier (UUID)** is a 35-character string representation of a unique 128-bit integer. ColdFusion uses the unique time-of-day value, the IEEE 802 Host ID, and a cryptographically strong random number generator to generate UUIDs. It conforms to the principles laid out in the draft IEEE RFC "UUIDs and GUIDs. The ColdFusion UUID format is 8-4-4-16. This might seem a little dense, but I am sure some of you are curious.

The UUID value alone could be your "ticket", but to add one more level of complexity, compound the value with ColdFusion's **HASH** function. This function converts the UUID to a 32-byte, hexadecimal string using the MD5 algorithm. The MD5 algorithm is *not* possible to convert back to the source string, and thus provide security for transporting data from client to server.

```
<cffunction name="setTicket" access="remote" returntype="any">
  <cfargument name="userID" type="any" required="yes">
  <CFSCRIPT>
    // create a new Ticket Object that will store the ticket info
    ticket_Obj = structNew();
    // set the ticket using CreateUUID and HASH
    ticket_Obj.ticket = Hash(CreateUUID());
    // set the initial expiry Date
    ticket_Obj.expiry =
CreateODBCDateTime(DateAdd("yyyy",10,now()));
  </CFSCRIPT>
  <!-- place the ticket into the database. -->
  <CFQUERY NAME="q_setTicket" Datasource="myFlashComDb.mdb">
    INSERT INTO Users_ticket (ticket, UserID, expiry)
    VALUES('#ticket_Obj.ticket#', #arguments.userID#,
#ticket_Obj.expiry#);
  </CFQUERY>

  <CFRETURN ticket_Obj>
</cffunction>
```

This is only one method you can use to archive the persistent data. Another approach might be to store this information in a persistent server scope such as the Application, Server, Client, or Session Scope.

Method #3: "findTicket"

Now with all that security and encryption technology, a service function is required to match the ticket that will be transferred from Flash to FCS with the user data. This function will be called by the Flash Communication Server. But the API is generic enough that you could use it anywhere.

When called, this function receives a value stored in the "ticket_hash" argument. Using CFQuery, a simple lookup is all you need to do to determine if the ticket is valid. To enforce the expiry date, add a WHERE condition that ensures the ticket is not out-of-date (preventing any hackers from sending bogus tickets to the server).

The following script takes the simple lookup one step further, and using an INNER JOIN, it makes the relationship back to the Users table to capture the first and last name of the user's record.

When the CFQuery is finished, construct a return object (return_obj) structure with two keys, USERNAME and ISTICKETOK. This object with these two keys will be returned to the Flash Communication Server as an ActionScript Object. [Figure 6](#) shows the complete structure of the return_obj.

```
<cffunction name="findTicket" access="remote" returntype="any">
  <cfargument name="ticket_hash"
    type="any" required="yes">

  <CFQUERY NAME="q_getUser"
    Datasource="myFlashComDb.mdb">
    SELECT Users.fname, Users.lname
    FROM Users INNER JOIN Users_Ticket
      ON Users.UserID = Users_Ticket.UserID
    WHERE
      users_Ticket.ticket = '#arguments.ticket_hash#'
      and users_Ticket.expiry > now();
  </CFQUERY>

  <CFSCRIPT>
    return_obj = structNew();
    return_obj.userName = q_getUser.fname & " " & q_getUser.lname;
    if (q_getUser.recordCount neq 0) return_obj.isTicketOK = true;
    else return_obj.isTicketOK = false;
```

```

        /* Return the Object to the Caller */
        return return_obj;
    </CFSCRIPT>

</cffunction>

```

Function: findTicket RETURNED Object Structure

struct	
ISTICKETOK	true
USERNAME	Kevin Towes

Figure 6 The ColdFusion Structure returned by the findTicket method is assembled automatically by Flash Remoting MX into an ActionScript Object with two properties.

Method #4 (optional): "updateTicket"

This is an optional method that will let you easily update the expiry date of any ticket. Simply pass the ticket and the amount of hours (from now) that the ticket will expire. It could be used in an application.onDisconnect event in SSAS, or by an administrative application. It is not used in this project, but it was built and might be a handy script for some of you.

```

<cffunction name="updateTicket" access="remote"
returntype="boolean">
    <cfargument name="ticket_hash"
type="any" required="yes">
    <cfargument name="new_expiry"
type="numeric" required="false" default="1">

    <CFQUERY NAME="q_getUser" Datasource="myFlashComDb.mdb">
        UPDATE users_Ticket
            set Expiry =
#CreateODBCDateTime(DateAdd("h",new_expiry,now()))#
            where ticket = '#ticket_hash#'
    </CFQUERY>

    <CFReturn true>
</cffunction>

```

If you would like to test your CFC, you can do so by calling the file within a web browser using this URL: http://localhost/flashcom/applications/informIT/FCS_Security.cfc

Running this file will output a full documentation of the component. You could also test the CFC using the <CFINVOKE> tag or the CreateObject function within a ColdFusion (CFM) file. Now that the ColdFusion Component is ready, let's move on to building the Flash movie.

Exercise 3: The Flash Movie, "FCS_Secure.fla"

Let's start with the interface (keep it simple). Use [Figure 7](#) as your layout guide. Here are the elements you need:

- Two input text boxes with the names defined in the property window as "**login_txt**" and "**password_txt**".
- A Push Button UI Component with the ClickHandler entered as "**login_init**".
- A PeopleList Communication UI Component. Give it the name "**peopleList_mc**" in the property window of the object.
- A ConnectionLight Communication UI Component. Give it the name "**connectionLight_mc**" in the property window of the object.
- A Dynamic Text Box with the name "**status_txt**".
- Finally, place some text labels and any other graphics that might help people use this application on the screen.

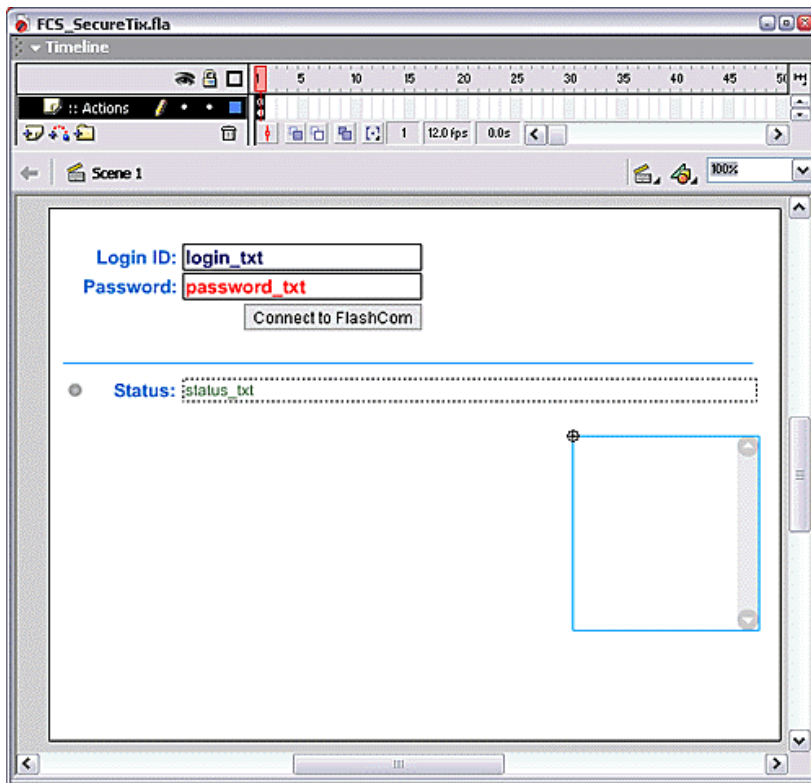


Figure 7 The Flash Interface for this Project. Use this as your layout guide for Exercise 1.

Now that your interface is set up, let's build the Flash ActionScript. Because there are a few methods to this exercise, they have been kept within the context of their function grouping.

The following ActionScript should be placed within a single frame. Don't forget to put the stop(); command at the bottom of the script; you don't want any erroneous loops!

1. First, insert the Flash Remoting Scripting objects into the Flash Movie. We'll also include the NetDebug objects so we can engage the NetConnection Debugger.
2. `// initialize the Flash Remoting Scripting Objects,`
3. `and the NetConnection Debugger`
4. `#include "NetServices.as"`
`#include "NetDebug.as"`
5. Next, instantiate the NetConnection Object (used to connect the Application to the Flash Communication Server) to a global variable called my_nc. You will not connect to the server just yet. Set up the NetConnection onStatus handler so you can watch any information objects coming in from the server. Lastly, connect the connectionLight to the NetConnection.
6. `// initialize the FlashCom NetConnection Object`
7. `_global.my_nc = new NetConnection();`
8. `_global.my_nc.onStatus = function(info_Obj) {`
9. `trace(info_Obj.code);`
10. `};`
`connectionLight_mc.connect(my_nc);`
11. Finally, set up the Flash Remoting Gateway and set the path to the ColdFusion Component you created earlier. Remember that the folders the CFC is located in are separated with a dot syntax. This ColdFusion MX call uses the secure HTTPS protocol (see red, below), so transmission of information is secure.
12. `//initialize the Flash Remoting MX Object`
13. `NetServices.setDefaultGatewayUrl("https://localhost/`
14. `flashservices/gateway");`
15. `gatewayConnection = NetServices.createGatewayConnection();`
16. `cf_service = gatewayConnection.getService(`
`"flashcom.applications.informIT.FCS_Security", this);`

Now that you have the initial objects set up, let's build the two functions required by the application. The functions are:

- **login_init:** Used by the Push Button to call the remote service function using Flash Remoting MX
- **authenticateUser_Result:** Used to handle the remote function return object, and to make the Flash Communication Server connection request

Place the following ActionScript below the code you scripted above.

Method 1: "login_init"

This method is used by the Push Button to assemble the login and password and then call the ColdFusion Component. By assembling the login and password values into an object, you can easily pass them to the ColdFusion Server. ColdFusion will see them as a normal structure.

```
login_init = function () {  
    // set the values of the login and password text fields  
    ↪ into an object that will be sent to ColdFusion  
    userInfo_Obj = ( { login:login_txt.text,  
    ↪ password:password_txt.text } );  
    // call the server function, authenticateUser,  
    ↪ passing the username and password object  
    cf_service.authenticateUser(userInfo_Obj);  
    // Update the Status Message  
    status_txt.text = "retrieving TICKET FROM SERVER";  
};
```

Method 2: "authenticateUser_Result"

This method is used to handle the return results from the Remoting function "authenticateUser". [Figure 8](#) shows the structure of the object returned to Flash by the function.

Function: authenticateUser RETURNED Object Structure

struct	
EXPIRY	{ts '2012-12-09 20:06:55'}
ISLOGINOK	true
TICKET	2D4367DF5BBFDB03B804F1BFA84D9DF1
USERNAME	Kevin Towes

Figure 8 The ColdFusion Structure returned by the authenticateUser method is assembled automatically by Flash Remoting MX into an ActionScript Object with four properties.

The method first challenges the "ISLOGINOK" property of the return object for a Boolean value of true. If successful, the Communication Server connection request is sent along with the "TICKET" property value of the returned object. Once the request is made, the PeopleList UI Component is connected to the NetConnection Object.

```
this.authenticateUser_Result = function(ret_Obj) {  
    trace("** Object Returned From the Server");  
  
    if (ret_Obj["ISLOGINOK"] == "true") {  
        // SUCCESS: Connect to the FlashCom Server
```

```

        trace("... sending Ticket Information to FlashCom Server");
        my_nc.connect("rtmp:/informIT/myInstance", ret_Obj["TICKET"]);
        // Connect the People List and other UI Components
        peopleList_mc.connect(my_nc);
        peopleList_mc.setUsername(ret_Obj["USERNAME"]);
        // Update the Status Message
        status_txt.text = "TICKET: "+ret_Obj["TICKET"];
        // Store the Ticket in the Global Scope of the Player, for future
use
        _global.Ticket_obj = ret_Obj;
    } else {
        // ERROR: Login/Password failed
        trace("!. The username and Password are incorrect");
        status_txt.text = "Login and Password are incorrect, please try
again";
    }
};

```

Your last bit of ActionScript stops the play head from looping or continuing.

```
stop();
```

It may have looked like a lot of ActionScript when you first started, but as you can see there are a lot of complex processes involved. This script successfully manages communications for both the Flash Communication Server and the ColdFusion Flash Remoting gateway service. The next exercise will set up the main.asc file for the Flash Communication Server.

Exercise 4: The Server-Side ActionScript, "main.asc"

This final exercise will construct the main.asc file on the server. This file must be placed within the InformIT folder in the FlashCom/Applications folder. Remember, SSAS is case sensitive, so your first debugging check always involves checking capitalization.

1. First, include the netservices class library to enable Flash Remoting from the Flash Communication Server. Load the components class library as well, because we are using some prebuilt Communication UI components.
2. `// Load the NetServices Class Libraries`
3. `load("netservices.asc");`
`load("components.asc");`
4. There are a number of ways you could do the next few steps; however, to keep this article short and easy, I've decided to place all actions for this application within the application.onConnect() function. This function is called whenever a client (Flash Player) requests a connection to the Flash Communication Server Application instance.

5. This application.onConnect method will require the parameter "ticket_hash" to be received. Flash Player sends this value within the connect() function; it is the same value that was received by Flash from the ColdFusion Server.

```
application.onConnect = function(clientObject, ticket_hash) {
```

6. Next, set up the SSAS Flash Remoting objects pointing to the same ColdFusion Flash gateway service you used within Flash MX. Like the ActionScript used earlier in Flash MX, this ColdFusion MX call uses the secure HTTPS protocol, so *transmission of information is secure*. In this script, we've chosen to even use the same local object name, "cf_service".

```
7.  /* --- Flash Remoting Setup --- */
8.  trace("Connecting to remoting Server..");
9.  NetServices.setDefaultGatewayUrl( "https://localhost/
10.  ➔flashservices/gateway");
11.  var gatewayConnection = NetServices.createGatewayConnection();
12.  var cf_service = gatewayConnection.getService(
13.  ➔"flashcom.applications.informIT.FCS_Security", this);
/* --- End Flash Remoting Setup */
```

14. Now that Flash Remoting is set up, call the remote service method, "findTicket", passing the value of "ticket_hash" argument that was received from the Flash Player.

```
15.  //call the server function, authenticateUser,
16.  ➔passing the username and password
    var callTheServer = cf_service.findTicket(ticket_hash);
```

17. Flash Communication Server will wait until either a successful result or a failed onStatus is received from the Flash Remoting Gateway (ColdFusion). This function handles the results returned from the ColdFusion server. The name of the function is the name of the service method (findTicket) with a "_Result" appended to the end of the name.

The ActionScript object sent from ColdFusion contains two properties (see [Figure 9](#)): ISTICKETOK, a Boolean value that tells Flash Communication Server if the ticket sent was valid, and if it was, a second property, "USERNAME", is sent containing the value of the user's name.

This function will challenge the property ISTICKETOK. If it returns true, it will accept the connection request and register the username with the Framework used by the Communication UI Components. If a value of false is received, the connection is rejected.

```
    // handle the Remoting response
this.findTicket_Result = function(result_obj) {
    trace(":: The Server Responded --> "+ result_obj["ISTICKETOK"]);
```

```

    trace(" --> User name has been set to: "+
result_obj["USERNAME"]);

    if (result_obj["ISTICKETOK"]) { // server responded TRUE
        trace("You have entered the inner sanctum");
        application.acceptConnection(clientObject);
        // register the user in the UI Framework
        gFrameworkFC.getClientGlobals(clientObject).username =
        result_obj["USERNAME"];
    } else { // Server responded FALSE
        trace("Not today, my Furry Friend...");
        application.rejectConnection(clientObject);
    }
};

```

Function: findTicket RETURNED Object Structure

struct	
ISTICKETOK	true
USERNAME	Kevin Towes

Figure 9 The ColdFusion Structure returned by the findTicket method is assembled automatically by Flash Remoting into an ActionScript Object with two properties.

It is important to remember that all properties returned from ColdFusion will always be in uppercase, and because SSAS is case sensitive, this is important.

```

// Don't forget to close the onConnect function !!!
};

```

That's it for the SSAS. Now, in most cases, you should have an onStatus function for the Remoting calls, but to save some space in this article, it was omitted.

Next up, it's time to test the full application.

Testing the Application, Some Considerations

1. Remember, every time you make a change to the main.asc file, you need to reload your application instance in the Communication App Inspector.
2. When you run the application, your output window should look similar to [Figure 10](#).

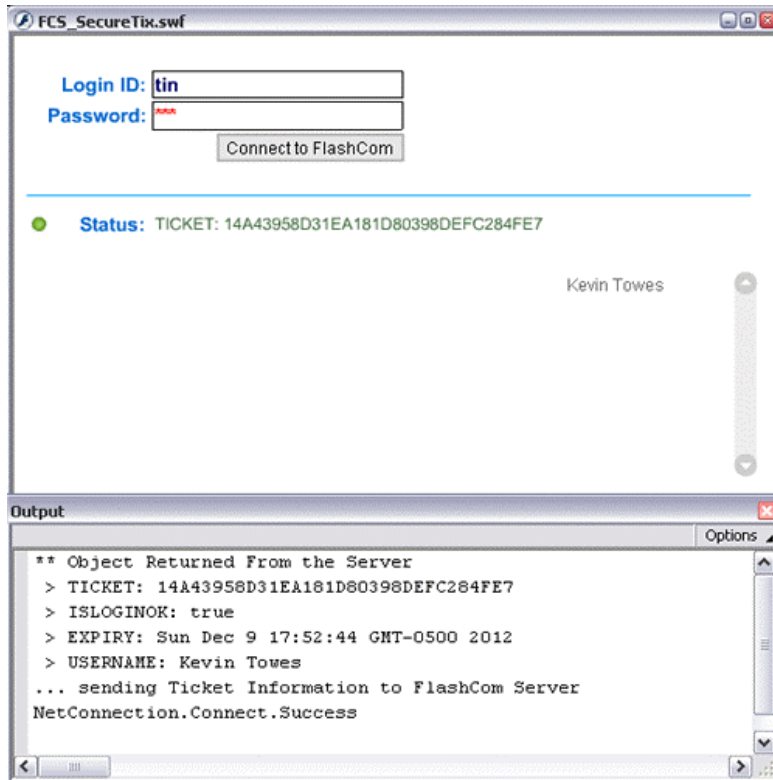


Figure 10 The Flash Communication Application showing a successful round-trip connection.

3. Ensure *both* the Flash Communication Server and the ColdFusion Server are running.
4. Test your ColdFusion Component. If you have an error in the CFC, you will not see anything in the output window. You can use the NetConnection Debugger to track error messages.

About This Article

If you want to learn more about Macromedia Flash Communication Server, check [out Macromedia Flash Communication Server MX](#) by Kevin Toves (© 2003 Macromedia Press [<http://www.newriders.com>], ISBN 0-7357-1333-2).