



Document Library

Die InterLake GmbH bietet Ihren Kunden in der Document Library Zugriff auf interessante Informationen. Wir stellen Ihnen Inhalte unserer Partner und aus frei zugänglichen Online Quellen zur Verfügung, die mit unserer Tätigkeit und der IT- und Medienbranche zu tun haben. Bitte beachten Sie, dass die Inhalte dem Copyright der jeweiligen Herausgeber unterliegen und diese Inhalte nicht ohne Referenz auf das jeweilige Copyright weitergegeben werden dürfen.

Weitere Informationen zu InterLake und weitere Dokumente finden Sie unter

www.interlake.net

Die InterLake Document Library steht den Nutzern der InterLake.Network zur Verfügung:



InterLake engagiert sich ehrenamtlich beim Münchener IT- und Medienverband FIWM, dem unser Geschäftsführer Sven Slazenger als Vorstandsmitglied angehört, sowie in der Macromedia ColdFusion User Group Central Europe, die unter unserer Leitung seit 1998 ein deutschsprachiges Forum für über 700 Macromedia Entwickler in Deutschland, Österreich und der Schweiz bietet.

Weitere Informationen zu diesen beiden Initiativen erhalten Sie bei Sven Slazenger (slazenger@interlake.net)

InterLake offers its customers and business partners access to an extensive document library. We are offering information from our partners and have also compiled freely available papers from the internet that we consider of value to you. Please be advised that the copyright belongs to the issuer of the information and that you may not distribute this content without referring to these copyrights.

Additional information about InterLake and more documents can be accessed free of charge at

www.interlake.net

The InterLake Document Library is available through the websites of the InterLake.Network:

InterLake is also an active member of the non-profit Munich IT- and Media Association FIWM. With InterLake CEO Sven Slazenger we provide one of the board members of the FIWM. We are also the founders (1998) of the Macromedia ColdFusion User Group Central Europe, the German language forum for Macromedia Developers from Austria, Germany and Switzerland and one of the largest Macromedia Communities worldwide.

For further information please contact Sven Slazenger (slazenger@interlake.net)



Persistent Data Communications

Looking at the Flash Communication Server in a whole new light

With the recent addition of the Flash Communication Server to our already powerful arsenal of tools, ColdFusion MX developers now have the opportunity to create applications that maintain a sustained communication channel with other applications. So what does this mean, and how can it affect you?

As an exercise, I decided to put CF and the Communication Server to the test and develop a simple application that allows multiple people to interact with a shared database in real time. As a bonus, I'll discuss continuing to use the database without being connected to it. That's right, pull that Ethernet cable from the back of your laptop, and take your database on the road – without requiring the server! Seem impossible? Read on.

In this article, I'll make extensive use of the Flash SharedObject programming model, Flash Remoting, and the Flash RecordSet Object. You will also learn useful techniques to help you understand what persistent server communications can offer an application.

To make it all happen I use a real mixed bag of Macromedia MX tools and servers – think of it as an MX soup:

- **Servers:** ColdFusion MX Server, Flash Remoting MX, Flash Communication Server MX
- **Tools:** Flash MX and Dreamweaver MX

Flash Remoting MX is used to transfer data between the Flash Communication Server and a ColdFusion database.

The Remote SharedObject is used to transfer database data between the Communication Server and each player connected to it. It is also how the real-time synchronization process occurs. Because the connection between a Flash player and the Communication Server is



By Kevin Towes

persistent, synchronization messages are easily delivered to each connected player and the server. When a change to a record is made, the record can be immediately locked on all players until the change has been completed and recorded.

The RecordSet Object is used to manage and maintain the integrity of the “rows”

returned from the database. Using this object's native methods, you can easily manipulate and interact with the data collections, transferring them back and forth between ColdFusion, the Communication Server, and the Flash players.

Let's first set the stage with a fundamental statement that most of you should know: “Connecting with ColdFusion via a Web browser or using Flash Remoting is done over the HTTP or HTTPS protocols. HTTP is a nonpersistent protocol, meaning that when you request something using HTTP, a server will respond once. The connection is open only when the client requests something from the server.”

This is great for a lot of IP-based applications, but is limited when you want the server to “message” a client that something has happened – without the client “asking” for an update. Consider today's instant messaging programs. Instant messengers like Yahoo!, MSN, and AOL maintain a persistent connection with the messenger servers and are updated immediately when someone comes online or goes offline. This is how you can see when a friend comes online.

You may have seen Flash Communication Server examples that include a live chat, or a shared whiteboard. These are simple invocations of the real power of persistent server connections. A few years back, a company called PointCast introduced “push” technology with its PointCast screensaver. This screensaver opened and maintained a channel to the PointCast server. This channel was used by the server to send real-time data to each screensaver that was on.

Flash Communication Server applications operate in a way that's very similar to the PointCast technology, except they make use of the ubiquitous Flash player. The user is not required to download and install any additional technology to start using it.

This is where the Flash Communication Server (Communication Server) can come in handy for applications that require instant notifications of events. The Communication Server doesn't use HTTP. It uses RTMP, a unique protocol that operates like a Java socket connection. It opens a persistent transportation channel between a Flash player and the Communication Server. The channel can be used to stream video and audio, or my favorite, to share data! Yes, the Communication Server is about much more than just audio/video streaming. This is where I want to help you see it in a new light.

Consider a shared address book database. When a database record is inserted, updated, or deleted, everyone using the database must reflect the new changes. With traditional ColdFusion applications, the user would be required to “reload” the browser page to show the updated data. This is because ColdFusion can't communicate the change to the user without the user first requesting an update. There are two problems here. First, the user must be connected to the application server and second, the user must be notified that there is a change in the database.

I'm going to attempt to solve these two problems using the Flash Communication Server and some new techniques for making

offline data available. There is a working example of the application at <http://flash.com.pangaeaNewMedia.com>. You can also download the source code and the sample database. I'm not going to get into every detail of the working application in this article, but I have heavily documented the source code so you can see what's going on.

For this exercise, I have a Communication application called "CFDJ" with all files (CFC, ASC, SWF, HTML) located in the folder c:\inetpub\wwwroot\flashcom\applications\CFDJ. I have a ColdFusion MX Server running on the same computer as my Flash Communication Server MX. This is a standard "Developer" installation of the Flash Communication Server. Your workspace may be different.

Flash Remoting, ColdFusion, and the Communication Server

This shared address book application will use Flash Remoting in a way that's different from what you're used to. You may have used Flash Remoting to transfer information between ColdFusion and a Flash application running in the Flash 6 player. This time, the same techniques will be used to transfer data from ColdFusion to the Flash Communication Server. There is no difference in the ColdFusion Component (CFC) code, nor is there a difference in the Flash ActionScript used to communicate with ColdFusion. The only difference is that the ActionScript will not exist within the Flash Movie, but on the Communication Server in Server-Side ActionScript (SSAS).

The application calls a CFC on the ColdFusion Server that returns a simple RecordSet Object to the Communication Server. Getting comfortable with the Flash RecordSet Object is important; by the end of this article, you will have a good idea of what you can do with it. Once the Communication Server receives the RecordSet from ColdFusion, the RecordSet items (records) are copied into a Remote SharedObject. I'll explain this shortly, but for now, simply understand that to share synchronized data across multiple Flash players, you must use Remote SharedObjects.

The ColdFusion Component (dataControl.cfc)

Before we get knee-deep into Flash, let's start with something familiar, the ColdFusion Component (CFC). There are two functions that ColdFusion will be responsible for in this project. The CFC's

first function returns a full collection of all records from the table, "Contacts," in the data source, "articleDb". The function name will be called "getAllRecords". (FYI, this table contains the following fields: fName, lName, emailAddr, phoneNum, faxNum, notes, isLocked [type: yes/no], editTime [type:date/Time]). A field called recordID is the autonumber key/identity field. All fields are text except the ones identified differently. The second function in the CFC, doUpdate, is a simple data update method. There's nothing spectacular about these two functions. Just make sure the access attribute in the CFFUNCTION tag is set to "remote" (see Listing 1).

In the second function, "doUpdate", each field in the database is declared in a CFARGUMENT tag. Flash will be sending values for these fields using Remoting. Field naming in this exercise is important, as I will use looping to generate lists and text objects in Flash. The file will be saved into the folder %wwwroot%/flashcom/applications/cfdj. The Flash gateway path to this CFC will be flashcom.applications.cfdj.dataControl.

Flash Communication Server SSAS (Main.asc)

I'm not going to get into all the details of building Communication Server applications. You should know that each Communication Server application maintains a unique file called "main.asc". This file contains SSAS and is responsible for handling connection requests and managing server-side objects. The SSAS language is very similar to Flash ActionScript. Just remember that it's case-sensitive.

As an event-based language, the Flash Communication Server has some automatic events that require custom event handlers. You should be aware of at least two event handlers, specifically:

- application.onAppStart(); called only once, when the Communication Server application instance first runs
- application.onConnect(); called each time a Flash player tries to connect to the Communication Server

The onAppStart() handler will contain the initial Flash Remoting call to ColdFusion. It will transfer the returned RecordSet into a persistent SharedObject that will be accessed later by the Flash player. Inside the Main.asc file, you must first load the NetServices class library. This ActionScript class library contains the objects required to connect with

ColdFusion and handle the server response.

```
// Load the NetServices Class Libraries
load("netservices.asc");
```

The next sequence of code executes only once because it is placed within the onAppStart() handler. This handler is called only when the Communication Server application starts. Please refer to the source code available at www.sys-con.com/coldfusion/sourcecode.cfm for a detailed explanation of the project.

Step 1: Flash Remoting Initialization

Just as I promised, if you're familiar with Flash Remoting, there is nothing different about the following lines of code (again, other than that it's running on the Communication Server, not the client). I have encapsulated this code within a function so I can control when it is called.

```
flashRemoting_init = function () {
    NetServices.setDefaultGatewayUrl("http://127.0.0.1/flashservices/gateway");
    gatewayConnection =
    NetServices.createGatewayConnection();
    cf_service =
    gatewayConnection.getService("flashcom.applications.cfdj.dataControl", this);
}
```

Step 2: Server Response Handler (getAllRecords_Result)

When the ColdFusion function getAllRecords is called (see Step 3), a function within SSAS is created to handle any objects returned from the call. Just like in Flash, the name of this result handler is the combination of the service-function name with a suffix of "_Result". Remember to capitalize the "R", as SSAS is case-sensitive.

The result handler does two things:

- First, using the getColumnNames() function, the names of the database fields are copied as an array into the columnDb_columnNames slot of a remote SharedObject called "serverData_so". (If you're new to SharedObjects and the concept of slots, this will be discussed further.)
- Second, the handler copies each row of the RecordSet into a unique slot in another SharedObject, contact_recordSet_so. A for-in loop is used to create the slots using the index value of the loop as the name of the new slot. The index value ("item") counts from zero to the number of records. It will also act as the Flash

identifier for the rows in the SharedObject. Figure 1 clearly shows how the original rows have been copied into the data property of the SharedObject.

```

this.getAllRecords_Result =
function(result_rs) {

serverData_so.setProperty("contactDb_columnNames", result_rs.getColumnNames());

    for (item in result_rs.items) {
        isRecord = result_rs.items[item].__ID__ !=
undefined;
        if (isRecord) {
            contact_recordSet_so.setProperty(item,
result_rs.items[item]);
        }
    }
}

```

The untouched `result_rs` RecordSet Object is returned from ColdFusion via Flash Remoting. The `__ID__` field is a Flash index key. The RecordID field is the database identifier or unique key. The items (or database rows) are “slots” within the `contact_recordSet_so` data property. Each row is a unique slot, which is important for synchronizing messages between the server and client.

Compare the structure of the original RecordSet Object with the Remote SharedObject structure. Notice how each data “row” is a “slot” within the data property of the SharedObject.

For this article, consider the Remote SharedObject as a memory area that stores data, accessible by every Flash movie (client) that is connected to it. The Remote SharedObject can be used to access and change data as well as send and receive messages. A message is the invocation of a function on all remote connections. Messages can originate from a Flash player or from the Communication Server.

A very unique feature of the Remote SharedObject is its synchronization messaging. Imagine a scenario that has multiple Flash clients able to change data within a common RecordSet. A method to inform each participant that a change has been made becomes very important. The Communication Server will do this for you automatically. All you have to do

Name	Value
result_rs	
gateway_conn	
items	
0	
_ID	0
editTime	null
emailAddr	"ktowers@pangaeaNet...
faxNum	"416-922-0799"
fName	"Kevin"
isLocked	false
lName	"Towers"
notes	"Nice Guy"
phoneNum	"416-922-1697"
RecordID	1
1	
2	
3	
4	
5	
6	
7	
mRecordsAvailable	8
mTitles	
serverInfo	
uniqueID	0
views	

Name	Value
contact_recordSet_so	
data	
0	
_ID	7
editTime	null
emailAddr	"ktowers@pangaeaNetMedia.c...
faxNum	"416-922-0799"
fName	"Kevin"
isLocked	false
lName	"Towers"
notes	"Nice Guy"
phoneNum	"416-922-1697"
RecordID	1
1	
2	
3	
4	
5	
6	
7	
onStatus	undefined

Figure 1: The untouched “`result_rs`” object (left) returned from ColdFusion compared to the same records mapped as “slots” within the SharedObject, “`contact_recordSet_so`” data property

is listen for it and code what happens when a change notification is made. This is the reason we have mapped the RecordSet to the SharedObject.

Applying this procedure allows an application to leverage this automatic messaging. Each time any data slot (record) is changed, the server will notify every Flash client which slot (record) has been changed. It does this by sending information objects containing what happened, and which slot (or in our example, which record) was affected.

Two key information object codes are “change” and “success”. A value of “success” will be received by the connection that made the change, and a value of “change” will be received by every client connected to the SharedObject that needs to be updated. ActionScript handlers must be constructed to manage these messages. You will see some information object handlers shortly.

Step 3: Retrieve the Data

Request the database collection by calling the service-function, `getAllRecords()`, on the ColdFusion Server. That will finish off the Flash Remoting initialization function.

```

callTheServer = cf_service.getAllRecords();
};

```

Step 4: Shared Object Initialization

With the Flash Remoting function ready to go, let’s initialize the SharedObjects. Just as the Flash Remoting script was encapsulated into a function, the SharedObject initializations will also be encapsulated so we can control when it happens.

This function will do four main tasks when it’s called:

- The first responsibility for this function is to connect the two local variables (`serverData_so` and `contact_recordSet_so`) to the Remote SharedObjects. These are the two SOs described earlier: one to track miscellaneous data, and the other to store the address book data. The way SharedObjects work (in Communication Server as well as in Flash) is that when they are first used, if they don’t exist, they are created. Think of it as how CFPARAM works.
- The second responsibility is to clear any existing data that may exist in the SharedObjects.
- Third, a custom synchronization handler is applied only to the address book data (the recordSet SharedObject). This `onSync` script will be explained later.
- Last, after the SharedObjects have been set up, the Flash Remoting initialization script we defined above is called.

```

SO_init = function () {
    serverData_so =
SharedObject.get("serverData", true);
    contact_recordSet_so =
SharedObject.get("contact_recordSet", true);
    serverData_so.clear();
    contact_recordSet_so.clear();
    contact_recordSet_so.onSync = SO_onSync;
    // Initialize Flash Remoting (above)
    flashRemoting_init();
};

```

Step 5: SharedObject Synchronization Monitor (onSync)

This next function is one of the most important functions in the entire application. Its role is to monitor all changes to any data within the SharedObject. (We'll see how the data can be changed later.) When a change is detected, the object indicating the slot changed and a "change" or "success" code (as described at the end of Step 2) will be passed to the function describing what's changed. Assuming it's detecting a change, the changed record is copied into a local temporary variable, where its `isLocked` property, an actual field within the database table, can be accessed. It is used primarily to disallow users to edit a

record if someone else is editing it, as described below.

Its value will also be used to trigger the server. Two values make the server respond: "save" will trigger the server to call the `doUpdate()` service function (CFC method) on the ColdFusion Server, and "true" will trigger the server to refresh every Flash Client by calling a function, `SO_messages`. A value of "false" is used as a null value that will force the server to ignore the change (see Listing 2). This function and these values will be reviewed when we look at the Flash ActionScript after Step 12.

Step 6: Communication Server Automatic Event Handlers

These next two steps are simple and use the application-level handlers that we discussed near the beginning: `onConnect` and `onAppStart`. Each Flash Communication application runs within an application scope (similar to the ColdFusion Application Scope) and uses its own unique SSAS. Because it is a true object-oriented language, applications actually run under application instances. Each instance of the application maintains its own variables,

SharedObjects, and other communication objects. It's not important for this project that you have a complete understanding of application instances; however, it is good to know. Application handlers exist within the application object's instance within the Communication Server.

First, the `onConnect` method will be automatically called each time a Flash player makes a connection request to the server, such as a client of this shared address book application. It is called a request because, until the server application accepts the request, it will not open a connection. The function "acceptConnection", a built-in method of the application object, opens the door.

```

application.onConnect = function(clientObject)
{
    application.acceptConnection(clientObject);
};

```

The last handler, `onAppStart`, starts the ball rolling. It is only run once, the first time the application is accessed. Its only job is to call for the initialization of the SharedObject that we created in Step

HOSTMYSITE.COM

www.hostmysite.com

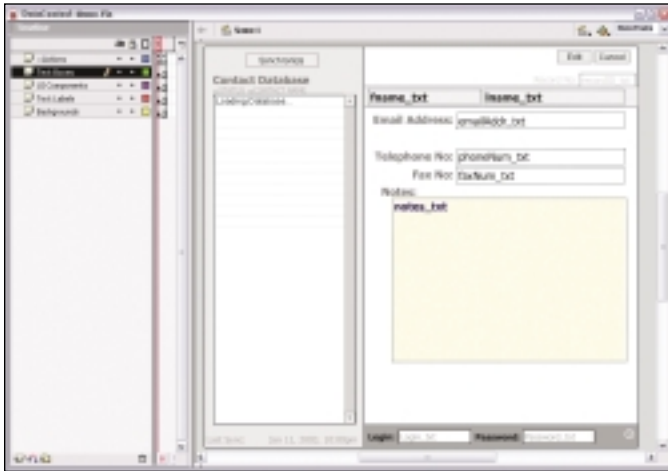


Figure 2: The Flash application interface

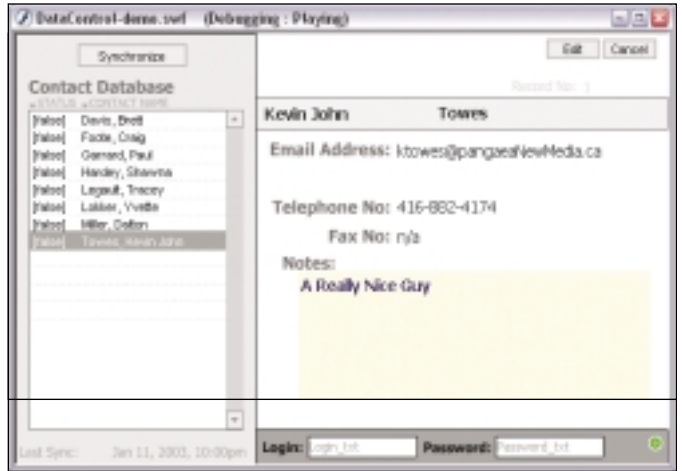


Figure 3: Editing a user record

4. This starts a chain reaction that includes initializing the Flash Remoting.

```
application.onAppStart = function() {
    SO_init();
};
```

A key point to remember is that each time you make a change to SSAS and your Main.asc file, you must reload your application using the Communication App Inspector. The App Inspector is a debugging tool that installs with the Communication Server. It is used to monitor the applications and the instances running on a single server. It is also used as a debugging Watch tool. Just like Flash has the trace() function, so does SSAS. The trace output is displayed in the App Inspector. It's a good idea to put some trace actions in your script so you can watch the startup sequence in the Live Log window of the App Inspector, a useful debugging tool you should learn about.

The Flash application interface (see Figure 2) is quite simple. It consists of a series of dynamic text boxes each named exactly the same as the database field where it will look for values. Each name has a suffix of "_txt" identifying it as a Flash text object. There is a combo box UI component stretched to allow for a long list of contact names. A synchronize button is simply used to keep our ActionScript simple. It's used here to advance the Timeline by one frame and execute additional ActionScript when the user is ready to continue. The

edit/cancel buttons allow the user to lock a record, edit it, then save it.

That's about it for an explanation of the interface; refer to the online source code for a deeper view. We'll move on to the client-side ActionScript that makes it all work.

Step 7: Include the Flash Remoting Class Libraries

Setting up a Communication Server application is quite simple in the Flash MX authoring environment. You will need to include the NetServices and DataGlue class libraries, so make sure you have Flash Remoting installed in Flash MX. Although we aren't using Remoting in the Flash player, we will use the RecordSet and DataGlue objects to interact with the data arriving from the Communication Server. These objects are included in the Remoting classes.

```
#include "NetServices.as"
#include "DataGlue.as"
```

Step 8: Initialize the SharedObject: initSO()

Much like the SO_init() function used in SSAS, this function connects two local variables to the two persistent Remote SharedObjects created on the server. The server location of those sharedObjects is held in a my_nc object that will be defined in Step 12. The function also assigns a local function, SO_messages (created in the next step), to handle messages from the server. For the server to access any functions on the Flash player(s), they must be declared within the SharedObject variable.

```
initSO = function () {
    remote_so =
    SharedObject.getRemote("serverData",
    my_nc.uri, true);
    contact_recordSet_so =
    SharedObject.getRemote("contact_recordSet",
    my_nc.uri, true);
    remote_so.connect(my_nc);
    contact_recordSet_so.connect(my_nc);
    contact_recordSet_so.SO_messages = SO_mes-
    sages;
};
```

Step 9: A SharedObject Handler Called by the Server to Update the Screen – SO_messages()

A SharedObject function can be called from any Flash player or server connected to the Remote SharedObject. This function has two parameters: the command and the source. The variable, doServerUpdate(), contains a true/false value determined if the source of the message is the server. This will help trap erroneous calls to this function.

The value, "updateLocal", was sent from the server's message in the onSync function defined in Step 5. You can easily build on this example code by expanding the switch cases. When called, it will update the localRecordSet with the values of the SharedObject using a function to be defined in the next step.

```
SO_messages = function (command, source) {
    var doServerUpdate = source == "Server";
    if (doServerUpdate) {
        switch (command) {
            case "updateLocal" :
                createLocalRecordSet();
        }
    }
};
```

```

break;
}
}
};

```

Because of the nature of the Remote SharedObject architecture, you don't need to send the updated RecordSet to each Flash player. Once a change is made to the Remote SharedObject, the change is immediately available to everything that is connected to it.

Step 10: Create a RecordSet Object Pointer to the Remote Data: createLocalRecordSet()

Our next step will perform a local version of the process from Step 2, in which the Remote SharedObject was populated with data from the database. In this step, we'll take that data as passed from the Communication Server, and will create a local RecordSet to hold the data.

When you use a Remote SharedObject as a source to populate a new RecordSet object, something happens that is very powerful and may take you a while to wrap your head around. The new RecordSet is actually a virtual pointer to the Remote SharedObject

data. This means that each time the local RecordSet is updated, the Remote SharedObject is updated, and so is every Flash player connected to it. If that isn't enough to excite you, the reason we transfer the SharedObject data back into a RecordSet is so we can use the massive collection of functions. Most importantly, DataGlue allows us to easily populate and format Flash UI components.

```

createLocalRecordSet = function () {
    fieldNames_array =
remote_so.data["contactDb_columnNames"];
    my_rs = new RecordSet(fieldNames_array);
    my_rs.removeAll();
    for (slot in contact_recordSet_so.data) {

my_rs.addItem(contact_recordSet_so.data[slot])
    ;
    }
    updateList();
};

```

It's important to note that while the Remote SharedObject will change when the local RecordSet changes, it does not work in reverse. The RecordSet is not changed automatically and neither is the screen. When the Flash player is informed

of a change to one of the records, it will use this function to update the local RecordSet object, and the screen.

Step 11: Update the Screen – updateList()

This function populates the ComboBox UI component with the names of the common address book. Because the data is available as a RecordSet, it is one single line of code.

```

updateList = function () {
    DataGlue.bindFormatStrings(contactList_cb,
my_rs, "[#isLocked#] #lname#, #fname#",
"#_ID_#");
};

```

Step 12: Connect the Flash Player to the Flash Communication Server

The final major piece to the puzzle is really the first step run when the user begins running the movie. We want to connect the Flash player to the Flash Communication Server, which means requesting that the local client be synchronized with the application managing the Remote SharedObject. Note that when we connect, we're connecting not using HTTP, but instead RTMP, and we're con-

PAPERTHIN

www.paperthin.com

necting not to a CFC, but to our communication application on the Communication Server.

This application is just an example, so I have omitted any security authentication here. We're also using the connectionLight, which is one of the Communication UI components installed to support Communication Server applications in the Flash authoring environment, and that shows a traffic light icon indicating the success of our connection with the Communication Server. The onStatus function is another automatic function that's called when an event occurs on the NetConnection Object. In this example, we're monitoring for a code value of "NetConnection.Connect.success". When this occurs, we know that the connection request was accepted, so we can then start connecting to the SharedObjects and get the party started, calling the initSO function we created in Step 8.

```
my_nc = new NetConnection();
connectionLight_mc.connect(my_nc);

my_nc.onStatus = function(infoObj) {
    if (infoObj.code ==
        "NetConnection.Connect.Success") {
        initSO();
    }
};

my_nc.connect("rtmp://127.0.0.1/CFDJ/myInstance");
stop();
```

Frame 2, User Interface Controls

When we created the user interface (see Figure 2), we created a synchronize button which, when pressed, passed control to the second frame of the movie. This is where we actually get the initial set of data from the remote server. Normally, you would just put this into another function, but to keep this example as simple as possible, I took this approach.

I am not going to go into great detail for the ActionScript in Listing 3, but there are a few things you should be aware of. When invoked, this script handles the management of the database data. It allows the user to edit a record and manage locking. The first responsibility is to call the createLocalRecordSet() function that we created in Step 10, in the previous Frame. From this point, the movie

will run, responding to various actions such as the user selecting data records to view from the contact name list (see Figure 3), possibly editing and/or saving them. The user interface controls will call upon these functions to handle their events. Additional details on these functions can be found in the source code package online.

In several of these functions, you will notice the usage of a loop, for (pos in fieldNames_array){}. This ActionScript loops through the database column names (as obtained in Step 10) and returns the name of the field over each loop. It is used to assign values to dynamic text objects and to manage the data objects. This is why the user interface component instances were named the same as the server-side database columns. A benefit of this dynamic approach is that if the interface and database are changed to add or remove columns, this code doesn't need to change.

Note also that this is the code that sets and uses the isLocked property to decide whether to mark a record as being edited, or prevent the user from editing an already locked record (see Figure 3). Notice also that the label for the save button is changed dynamically based on the status of actions performed, just as the input fields are being dynamically enabled/disabled (see Listing 3). It's all coming together!

A Database to Go, Please!

As promised, here is the code to take your RecordSet on the road! How can you extend this application to allow you even to work with the address book when you're not connected to the server? With Local SharedObjects (as opposed to the Remote SharedObject we've been working with to this point).

Cookies are frequently used to store data on the client persistently, even if the browser is shut down. The Local SharedObject is becoming known as the Flash cookie. Except it is much more effective than an HTML cookie, in that not only is it persistent over browser shut-downs, you can store complex data within it – like a RecordSet Object. Hmm...where might we have a RecordSet we'd want to store persistently on the client?

Here's some homework for you. Use the example in this article and create a

local persistent version of the RecordSet that can be accessed without an Internet connection or a connection to a Web server. Here is a leg up:

```
// copy the RecordSet into the Local
SharedObject
local_so = SharedObject.getLocal("contactDb");
local_so.data["contactDb_rs"] = my_rs;

// copy the RecordSet out of the Local
SharedObject
my_rs = new RecordSet();
my_rs = local_so.data["contactDb_rs"];
```

Allowing access to the application (and the data) without a connection to the server is definitely cool, but this wouldn't work without the record-locking and synchronization approaches described throughout this article. This is another frequent problem in Web applications that can be solved with Flash and Communication Server, and it's just one approach to solving such problems. Very powerful stuff!

Summary

Clearly, there are a lot of omissions with this example, including the most obvious features of adding and deleting records. Consider this as a foothold into leveraging the persistent connectivity power of the Flash Communication Server. At a basic level you could use a Flash widget to simply notify users, or force the browser to request an updated HTML page from the ColdFusion Server. An extreme use of these techniques would produce an extremely rich working environment for the user. Watch for my future articles on Flash Remoting and Flash Communication Server here and on Macromedia.com. For more information on the Flash Communication Server, check out my book, *Flash Communication Server MX*, published by Macromedia Press. 

About the Author

Kevin Towes is the cofounder of Pangaea NewMedia in Toronto, and author of the book, *Flash Communication Server MX* published by Macromedia Press. He is a regular speaker and writes articles for a variety of journals including the *Macromedia Designer Developer Center*.

kevin@kevintowes.ca