



Anatomy of a Central Application - Part 1

By: [Paul Newman](#)

You have to admit, there's a certain "gee whiz" quality about [Macromedia Central](#) (<http://www.macromedia.com/go/central/>). The Central components are much slicker than the V1 components released for Flash MX, and even Flash MX 2004 doesn't have an accordion tab component. But Flash is not just about cool Flash intros (*ugh*) or rich user interfaces anymore. With the emergence of Central and [Flex](#) (<http://www.macromedia.com/go/flex/>), Flash has become an incredibly practical and efficient means of retrieving, storing, and displaying information.

Sometimes, the best way to learn is by example. That's why we've decided to release the complete source code for our Central application, [Community MXtra](#) (<http://www.communitymx.com/content/article.cfm?cid=A2D2A836CDB0353C>). In this two-part article, I'll examine some of the key features of CMXtra and explain how certain effects were achieved. You'll learn about agents, pods, interfaces, tooltips, Flash Remoting, the WebService object, LCService, LCDDataProvider, Shared Objects, custom data grids, and much more. To obtain the source code, click **Download Support Files** at the bottom of this page. I encourage you to poke around the files to familiarize yourself with the application. If you're new to Central, you might want to read "[Your First Central App](#) (<http://www.communitymx.com/content/article.cfm?cid=A1CAC646D95BD130>)" before proceeding.

CMXtra: The Big Picture

The CMXtra application enables users to search for articles on CommunityMX.com, browse content using a calendar, and maintain a list of favorites. Because Central is authored using Flash and ActionScript 1.0, it's fairly easy to migrate existing apps to Central.

TIP: Many of the Central components will look familiar to Flash developers (DataGrid, ComboBox, PushButton, ScrollPane, etc.). Keep in mind, however, that because Central is currently an ActionScript 1.0 solution, its component methods correspond to the V1 components for Flash MX — not the V2 components for Flash MX 2004. The V1 components were distributed with the Flash UI Component sets 1 through 5.

In other words, if you can create apps for Flash, you can create apps for Central. The Central API introduces new components and classes, as well as functions required to initialize various parts of the application (agent, pods, etc.). Once you get up to speed on the Central API, developing Central apps is fairly straightforward.











(http://www.communitymx.com/content/source/A122EDE548C81F25/Figure1_lg.gif)

Figure 1 CMXtra is comprised of an app, and agent, and two pods ([enlarge](http://www.communitymx.com/content/source/A122EDE548C81F25/Figure1_lg.gif) (http://www.communitymx.com/content/source/A122EDE548C81F25/Figure1_lg.gif))

The CMXtra application consists of four FLA files and eight ActionScript files:

File	Description
Application (Client) <i>The main application that is loaded into the Shell</i>	
cmxtra.fl	<ul style="list-style-type: none"> Includes the following components: AccordionTab, Calendar, DataGrid, DialogBox, IconButton, PushButton, ScrollPane, TextInputField.
cmxtra.as	<ul style="list-style-type: none"> Consists of button and event handlers, utility functions (showDialog, updateStatus), functions to add and remove favorites, and code to initialize and call the Flash Remoting service. Callable LCSERVICE Client functions: gCom.formatSearchResult, gCom.setQuery.
init.as	<ul style="list-style-type: none"> Contains required Central initialization functions such as onActivate, getMaximumSize, getMinimumSize. Creates LCSERVICE Client instance (gCom) and LCDataProvider Client instances (gFavorites, gLastSearch).
Favorites Pod (Client) <i>Enables user to view or remove favorites from the Console</i>	
favePod.fl	<ul style="list-style-type: none"> Includes the following components: DataGrid, IconButton, TossButton.

 favePod.as	<ul style="list-style-type: none"> ■ Contains event handlers, utility function, and required Central functions such as onActivate. ■ Creates an LCDDataProvider Client instance (gFavorites).
Search Pod (Client) <i>Enables user to search CMX and add favorites from the Console</i>	
 searchPod.fla	<ul style="list-style-type: none"> ■ Includes the following components: DataGrid, IconButton, PushButton, TextInputField, TossButton.
 searchPod.as	<ul style="list-style-type: none"> ■ Contains event handlers, utility functions (doSearch, enableIcons), and required Central functions such as onActivate. ■ Creates LCService Client instance (gCom) and LCDDataProvider Client instances (gFavorites, gLastSearch). ■ Callable LCService Client functions: gCom.formatSearchResult, gCom.setQuery.
Agent (Server) <i>Handles communication among application and pods</i>	
 agent.fla	<ul style="list-style-type: none"> ■ Contains no graphic elements. This "headless SWF" is simply a container for the ActionScript in agent.as.
 agent.as	<ul style="list-style-type: none"> ■ Contains required Central initialization functions such as onActivate, getMaximumSize, getMinimumSize. ■ Defines agent as the LCService Server (gCom) and LCDDataProvider Server (gFavorites, gLastSearch). ■ Instantiates the WebService object used to search Community MX. ■ Callable LCService Server functions: gCom.searchContent, gCom.getQuery.
Favorites <i>Saves favorites to disk using Shared Object</i>	
 favorites.as	<ul style="list-style-type: none"> ■ Initializes gStorage Shared Object. ■ Contains three functions: initStorage, saveFavorites, deleteStorage.
Shared <i>Functions and global methods shared by the app, agent, and pods</i>	

 functions.as	<ul style="list-style-type: none"> Contains a single function: formatDate.
 interface.as	<ul style="list-style-type: none"> Defines global methods for the LCService object.

The app, in the context of this article, is the SWF file that's loaded into the Shell (see Figure 1). The agent is a "headless SWF" — one without any graphic elements — that you can use to retrieve and store data. By centralizing your data services and global methods in an agent, you can use it to broadcast events and trigger updates in your app and pods. The pod is a smaller SWF file that's usually a miniaturized or condensed version of the main app. Pods enable users to access the core functionality of your main app without consuming as much desktop real estate.

I know this seems like an awful lot of files for a relatively simple application, but keep in mind that most of these files are required to implement the pods. Without the pods, the CMXtra app can be generated with two files: cmxtra.fla and cmxtra.as.

Points of Interest

Macromedia Central boasts a number of unique features unavailable to regular Flash applications. Some of the standouts include the LCService object, the LCDDataProvider object, OS-level tooltips, and local file caching.

LCService Object

The LCService (or Local Communication Service) object greatly simplifies communication among the agent, app, and pods. Using a client/server metaphor, the LCService object enables you to create a single named service (e.g., gCom) with shared global methods. In Central, you create the LCService Server in your agent, and the LCService Clients in your app and pods. The following code uses the `LCService.createServer` and `LCService.createClient` methods to define a server (agent.as) and a client (searchPod.as):

```
// Define LCService Server (agent.as)
gCom = Central.LCService.createServer(CMXService, this,
true);

// Define LCService Client (searchPod.as)
gCom = Central.LCService.createClient(CMXService, uniqueID,
this, true);
```

To enable global LCService methods, you must create an interface that defines the Server methods and the Client methods. If you open **shared\interface.as** in a text editor, you'll find the following code (this file is *included* by agent.as, cmxtra.as, and searchPod.as):

```
// Interface across the agent (server) and application/pods
(clients)

if(CMXService == null)
{
    CMXService = new Object();
    CMXService.name = "CMXService";
    CMXService.interfaces = new Object();

    // callable functions in clients (app, search pod)
    CMXService.interfaces.Client = ["formatSearchResult",
    "setQuery"];

    // callable functions in server (agent)
    CMXService.interfaces.Server = ["searchContent",
    "getQuery"];
}
```

In the case of CMXtra, this means that the app and the search pod can both call the `searchContent` and `getQuery` functions in the agent, and the agent can call the `formatSearchResult` and `setQuery` functions in the app and search pod. Outside of Central, this would be considerably more difficult using the Local Connection object. What's nice about LService is that the clients can each have their own implementation of a global method. For example, when the `gCom.searchContent` method is called in the agent, it uses the WebService object to search CommunityMX.com. When a result is returned, the agent calls the `gCom.formatSearchResult` method, which invokes the method in both the app and the search pod.

Yes, you heard right. With LService, you can call functions in the agent from your pod, and vice versa. You can even pass in parameters. That's how CMXtra remembers the most recent search string: Its value is stored in the agent as a global variable (`gQuery`), and is easily accessible via the `gCom.getQuery` method.

LCDataProvider Object

If you thought the LService object was cool, the LCDataProvider object is like LService on steroids.

The LCDataProvider object, like the LService object, uses a client/server metaphor, but its purpose is not to share methods, but data! The LCDataProvider object enables you to store data in the agent (server) that immediately propagates to your app and pods (clients). For example, whenever favorites are added or removed in the app, these changes are immediately reflected in the Favorites pod, without any additional coding. The key is that the app and pods share the same LCDataProvider object (`gFavorites`):

```
// Favorites data grid in app (lines 48-51 of init.as)

// create LCDDataProvider Client for favorites
gFavorites = Central.LCDDataProvider.createClient(uniqueID,
"cmx_faves");
// populate fave_dg w/ LCDDataProvider
fave_mc.fave_dg.setDataProvider(gFavorites);

// Favorites data grid in pod (lines 26-29 of favePod.as)

// create LCDDataProvider Client for favorites
gFavorites = Central.LCDDataProvider.createClient(uniqueID,
"cmx_faves");
// populate fave_dg w/ LCDDataProvider
fave_dg.setDataProvider(gFavorites);
```

The agent is the LCDDataProvider Server (lines 45-48 of agent.as):

```
// create LCDDataProvider Server for favorites
gFavorites =
Central.LCDDataProvider.createServer("cmx_faves");
// get initial data from local Shared Object
gFavorites.setData(gStorage.data.favorites);
```

In the case of CMXtra, the source of the data is a Shared Object (the user's favorites are saved to disk). Managing favorites in Central then becomes fairly straightforward:

- Create LCDDataProvider Server in agent
- Create LCDDataProvider Clients in app and pod(s)
- Use the `LCDDataProvider.setData` method to define the data for the data provider
- Use the `setDataProvider` method to populate Central components with the LCDDataProvider object

The advantage of using LCDDataProvider is you don't have to worry about synchronizing data among two or more components — Central handles all of it for you. In CMXtra, this saves a great deal of code because we don't have to update the Shared Object every time the user clicks **Add to Favorites** or **Delete Favorite**. Instead, we update the LCDDataProvider (`gFavorites`).

So how do the favorites get saved to disk before the user's session ends? Simple. We make a call to the `saveFavorites` function when the agent is deactivated (see lines 51-70 of `agent.as`):

```
function onDeactivate()
{
    // save gFavorites to Shared Object (favorites.as)
    saveFavorites();
}
```

This is the `saveFavorites` function in `favorites.as`:

```
function saveFavorites(){
    // delete all favorites in Shared Object
    gStorage.data.favorites.splice(0);

    var len = gFavorites.getLength();

    for(var i=0; i<len; i++){
        var item = gFavorites.getItemAt(i);
        // add gFavorites LCDDataProvider to Shared Object
        gStorage.data.favorites.push({dDate: item.dDate...});
    }
    // write Shared Object to disk
    gStorage.flush();
}
```

Another useful feature of `LCDataProvider` is the `getIndexByKey` method. Ordinarily, to determine if a favorite has already been added, you would loop through the Shared Object and check for a match:

```
function findArticle(title){
    var len = gStorage.data.favorites.length;
    for(var i=0; i<len; i++){
        // if the favorite already exists...
        if (gStorage.data.favorites[i].title == title)
            return i;
    }
    // favorite hasn't been added yet
    return -1;
}
```

With the `getIndexByKey` method, you simply identify the column name (or key) and the value

you want to compare it to. This method returns the *index* of the item that matches. If no item matches, `getIndexByKey` returns `-1` (see lines 127-140 of `searchPod.as`):

```
function onAddFave(evtObj){
    var item = search_dg.getSelectedItemAt();
    var index = gFavorites.getIndexByKey("title", item.Title);

    // if not already added...
    if(index == -1){
        // add new favorite to LCDataProvider
        gFavorites.addItemAt(0,{dDate: item.dDate...});
        output_txt.text = "Favorite added";
    }
    else{
        output_txt.text = "Already added";
    }
}
```

On the whole, the `LCDataProvider` object is a great feature. However, on occasion, a search initiated in the pod doesn't fully propagate to the Search data grid in the main app. I've yet to determine what's responsible.

Another drawback of `LCDataProvider` is that it doesn't enable you to define custom sort functions. Prior to implementing `LCDataProvider`, I was able to use `MGridColumn.setSortFunction` to specify a custom sort function for a particular grid column. Unfortunately, when you use `LCDataProvider` in conjunction with the `DataGrid` component, the `MGridColumn.setSortFunction` method is ignored. Until a workaround is devised, or the Central team adds an `LCDataProviderColumn.setSortFunction` method, sorting by date is disabled in the Search and Favorites data grids.

OS-Level Tooltips

What's so exciting about OS-level tooltips? With OS-level tooltips, your tooltips extend beyond the boundaries of your SWF files (and Central itself):

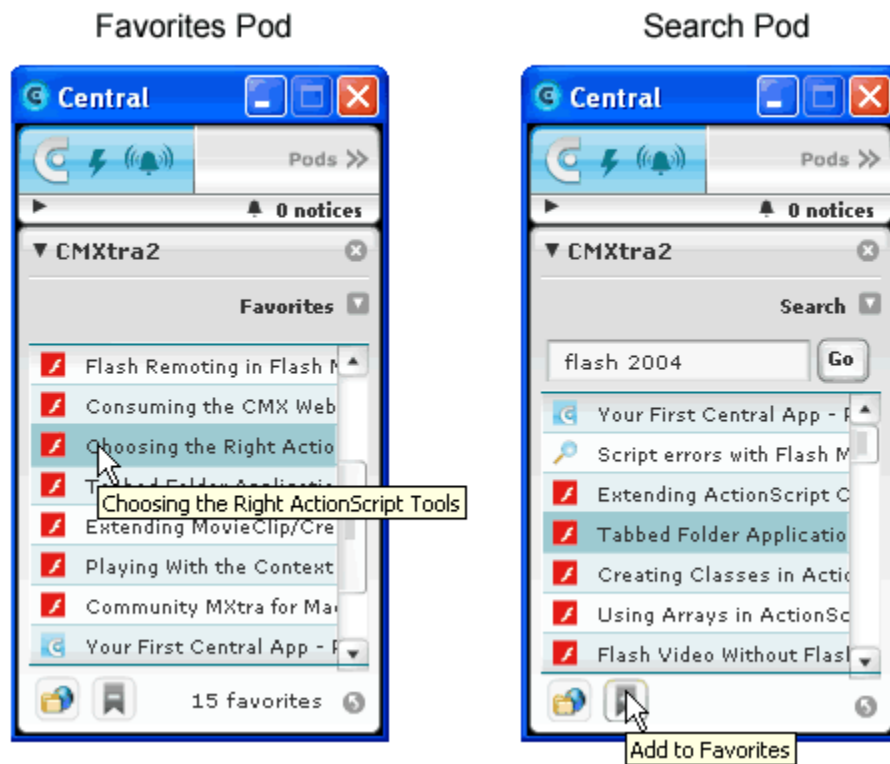


Figure 2 OS-level tooltips extend beyond the boundaries of Central

This means you don't have to worry about the tooltips being cropped. In addition, tooltips are very easy to implement in Central. Simply use the `MovieClip.tooltipText` property:

```
search_mc.getURL_ic.tooltipText = "View Article";
```

To remove a tooltip, set its value to an empty string:

```
search_mc.getURL_ic.tooltipText = "";
```

Central automatically handles the `onMouseOver` and `onMouseOut` events for you. To see how tooltips were added to the data grid in the Favorites pod, open **favePod.fla** in Flash and explore the **title** and **titlecell** movie clips in the library.

ScrollPane Component

Like Flash MX 2004, the Central API includes a scroll pane component. The ScrollPane component is simply a container for movie clips.

NOTE: To distinguish themselves from Flash MX 2004 components, the names of Central components are preceded by an *M*: MDataGrid, MDialogBox, MScrollPane, etc. In addition, many of the Central component methods differ from their Flash counterparts. For instance, to get a reference to the movie clip loaded into the scroll pane, you use `MScrollPane.getScrollContent` in Central, but `ScrollPane.content` in Flash MX 2004. For a better understanding of these discrepancies, read "OOP Sacrilege (<http://www.flash-remoting.com/articles/oopsacrilege.cfm>)" by Tom Muck.

Like most of the Central (and Flash) components, there are two ways to interact with the scroll pane: you can define its content programmatically, using the `MScrollPane.setScrollContent` method, or you can enter a value in the Property or Component inspector. In either case, the value must be a valid linkage ID for a movie clip in your Flash library.

In CMXtra, the ScrollPane component is used to display the currently selected articles on the Articles tab:



Figure 3 The ScrollPane component loads a movie clip whose linkage ID is *container*

The `populateScrollPane` function (lines 374-462 of `cmxtra.as`) uses the `MScrollPane.setScrollContent` method to load a movie clip whose linkage ID is `container`. The `setScrollPosition` method resets the scrollbar to its top position, so the beginning of the first article is visible by default. In addition, we grab a reference to the `container` movie clip by using the `getScrollContent` method:

```
// clear scrollpane content by reloading container mc
sp.setScrollContent("container");
// move scrollbar to top
sp.setScrollPosition(0, 0);
// get reference to container movie clip
var ref = sp.getScrollContent();
```

Later in the function, we can use the `ref` variable to target movie clips inside the `container` movie clip:

```
if(i>0){
    // if more than 1 article, duplicate cont movie clip
    ref.cont0.duplicateMovieClip("cont"+i, i);
}
```

Another useful method is `MScrollPane.refreshPane`. This method automatically recalculates the height of the `container` movie clip — after all its content has been dynamically generated — and resizes the scrollbar accordingly. That's why `refreshPane` is called at the very end of the `populateScrollPane` function (line 460):

```
// refresh ScrollPane component after changing its size
sp.refreshPane();
```

Of course, most of the ActionScript in the `populateScrollPane` function would be unnecessary in Flash Player 7, since FP7 offers partial support for CSS classes and the `` tag in HTML text fields. But until Central supports Flash Player 7, you can accomplish a great deal with the `ScrollPane` component.

Local File Caching

You may have noticed the `Shell.addToLocalInternetCache` method (line 423) in the `populateScrollPane` function. This method enables you to cache files on the user's hard drive:

```
// add category icon to local cache:
var gifUrl = "http://www.communitymx.com/images/large" +
    ➔ article.prefix + "icon.gif";

if(!gshell.inLocalInternetCache(gifUrl)){
    gshell.addToLocalInternetCache(gifUrl);
}
```

In this example, we first check if the GIF image is already in the local cache. If not, we use the

`addToLocalInternetCache` method to save the image to the user's hard drive. Once the image is cached, Central uses the cached version, if available, the next time the URL is requested. Where does Central save the files? On Windows XP, the CMXtra category icons are saved to the following folder:

C:\Documents and Settings\<username>\Application
Data\Macromedia\Central\#Central\<uniqueid>\Local Internet\communitymx.com\images

Also notice that Central, unlike Flash MX 2004, supports dynamically loading GIF files in addition to JPEG, SWF, MP3, XML, etc.:

```
// line 427 of cmxtra.as  
temp.loader_mc.loadMovie(gifurl);
```

For a complete list of file types that are *not* supported by Central (e.g., EXE, REG, WSH), see *Developing Central Applications* in the SDK.

Conclusion

I hope the source code for CMXtra helps you with your Central applications. If you have any questions or suggestions, please use the feedback form below. In Part 2 of this article, I'll share with you some Central tips and tricks, such as customizing the DataGrid component and placing icons on the AccordionTab component.

NOTE: When you extract the zip file that accompanies this article, you'll find two folders: **source** and **for_webserver**. The **source** folder contains the FLAs and ActionScript files required to publish the application. The **for_webserver** folder contains the files required to *install* the application. Copy the **for_webserver** folder to a web server (e.g., IIS, ColdFusion, Apache), browse to **installer.swf**, and click the installation badge. This will install a new instance of **CMXtra*** in Macromedia Central using the developer product ID. You can use this version to experiment with the application without overwriting or uninstalling CMXtra 1.0 (<http://www.communitymx.com/content/article.cfm?cid=A2D2A836CDB0353C>) .

Approximate download size: 480k

Keywords

macromedia central, cmxtra, flash mx 2004, LCService, LCDDataProvider, data grid, ActionScript, accordion tab, shared object