Flash Communication Server-Side framework (by Fernando Florez)

Posted At : February 1, 2004 5:35 PM | Posted By : Stefan Richter
Related Categories: Tutorials

This subject sounds really advanced but it is not. There isn't much info about the Flash Communication Server Server-Side framework but along this tutorial I'll try to document some interesting parts of it that hopefully will help you with some common tasks you could have on your daily development.

I think this framework was created not as a utility but as a manager for flashcom server-side components (something like the UIComponent class for Client-Side components) but it also has some nice utilities we can take advantage of.

The normal .asc structure
Normally an .asc (ActionScript Communication) file will have this structure:

```
application.onAppStart = function(){
                // this will be executed when the application instance it
}</p><p>application.onConnect = function(clientRef){
                // this will be executed each time a new user connects t
nstance
                // "clientRef" is a Client instanceof the new connected u
                // here we need to acceptConnection or rejectConnectio
                // if any of these commands are called then the client is
                // and can't interact with the application instance or cli
}</p><p>application.onDisconnect = function(clientRef){
                // this will be executed each time a user disconnects fro
 instance
                // this could be because a NetConnection.close call or if
e browser
                // "clientRef" is a Client instance of the disconnected us
}
```

This is the basic structure, we have more server-side events but they are used in specific occasions and are not really common as these ones are.

Server-side event declaration
We can only specify one server-side event declaration, if we specify more than one then only the last one will remain. So is it possible to have more than one server-side event declaration and make them all work?
Not really but this is when the server-side framework is useful.
This feature is not supported and I sincerely haven't tested it much so it may have some wrong behaviors.

First of all we need to load the framework into our .asc file.

```
load("framework.asc");
```

The framework.asc file is on our scriptlib folder so we can load it this way without specifying any path (I extremely recommend not modifying or moving this file if we

don't have an idea of what we are doing exactly).

Then just create your server-side events like you normally do but instead of making them inside your application object create them inside other objects created by you. For example:

```
this.myFirstObject = new Object();
this.myFirstObject.onConnect = function(newClient){
trace ("onConnect: myFirstObject");
}
this.myFirstObject.onDisconnect = function(oldClient){
trace ("onDisconnect: myFirstObject");
}
this.mySecondObject = new Object();
this.mySecondObject.onConnect = function(newClient){
trace ("onConnect: mySecondObject");
}
this.mySecondObject.onDisconnect = function(oldClient){
trace ("onDisconnect: mySecondObject");
}
```

You can have as many objects as you want/need and each one of these objects can have any of the server-side methods. We have the "onConnectAccept" and "onConnectReject" events too if you have already used them with components.

Now we only need to subscribe our objects into the gFrameworkFC object which is the base instance of the framework.

```
gFrameworkFC.addListener(this.myFirstObject);
gFrameworkFC.addListener(this.mySecondObject);
```

Run the code and Voila!

The framework was intended to work with components, it's our base class for server-side components but there are so many nice things in there so why not use it with non-component application?
We use it a lot here at **funciton communications.**

Why have a Server-side framwork
The first question that came to my mind was the purpose of the server-side framework, in flash remoting we don't have a server-side framework and we still have components for it so what's the real purpose of this framework?

The reason for this is that flashcom is pure event callbacks, we have the onConnect, the onDisconnect, the onStatus, etc, etc. but we only have one "application" object and the possibility for only one main.asc file. I bet the flashcom team asked themselves this question: "How can we execute these events for different objects?" and the answer is with an event manager that should be able to broadcast the events to all subscribed objects.

The first part of this tutorial explains on how to use it (see above). This is done by the framework each time we subscribe our component class via the FCComponent constructor.
Now the problem is that each component should have it's own room in where it can rest, this room depends on the component type (the type can be a simpleconnect, peoplelist, etc.).

The object gFrameworkFC object has another object named "components" and this is the *hotel* for all the components.
Inside this object new objects are created for each component type (what we call rooms before, remember?)

For example, if we use the FCChat component then it will be placed like this in the framework: gFrameworkFC.components.FCChat.componentInstance

"componentInstance" is the instance of our client-side component on stage. Any symbol on stage has a unique instance name even if we didn't give it one Flash will give one for us to it.
Each "room" can have any amount of components subscribed to it, each of those properties point to our server-side component class instance. Now everything is getting clear, huh?

Anyway, let's continue by reviewing one framework method that appeared in one of the dot releases for flashcom 1.5 (it was for flashcom 1.5.1 if I remember right).

We all know that "arguments" in flashcom is not really an array but an object. "What a great bug!" you may say but not really, it's not Macromedia's fault or even decision it's something specified on the ECMA in which the flashcom engine is based. To avoid this uncomfortable thing the framework has this method that will convert any object into an array. It can convert any object into an array but it's used only for arguments conversion (this does not mean that you can't use it for other things).

The method is named "__toarray__" and is placed in the gFrameworkFC object.

An example of it could be:

```
var obj = new Object();
trace (obj.length); //Error! I don't have that property, that is an array property
obj = gFrameworkFC.__toarray__(obj);
trace (obj.length); // Look Ma'! It's working
```

Now we can do whatever we want with "obj", this means we can apply an splice, pop, shift, etc. to it.

Ok, last thing in this tutorial before I get too boring. How many times you had the need to give each client instance a unique id or how many times have you seen people doing it? Lots, right?

Normally a developer will do something like: (Editor: yes, I do it all the time...)

```
application.onAppStart = function(){
```

```
this.uniqueID = 0;
}
application.onConnect = function(newClient){
newClient.id = this.uniqueID++;
}
```

So every client instance will have a unique identifier. “uniqueID” will return to 0 when the application is restarted.

We can avoid these steps and just include the framework.asc file on our main.asc file (if we haven’t done it yet). The framework creates a unique identifier that you can access with the “__ID__” property. For example:

```
load("framework.asc");
  application.onConnect = function(newClient){
  trace (newClient.__ID__);
}
```

That’s all for this tutorial, I hope you like it.
Don’t forget to visit our blog at: **http://blog.funciton.com/en/**

Take care, have fun and keep flashing.