An Introduction to the Mach-II Event Object and Listeners

Mach-II 1.1.1 for ColdFusion

Last Updated on November 15, 2006

Matt Woodward
Release Coordinator

Overview

1. Introduction
2. The Mach-II Event Object
3. URL Variables
4. Form Variables
5. What's a Listener?
6. Building Your First Listener: LoginListener
7. The Login Form and Login Events
8. Summary

Introduction

In the Mach-II Quick Start Guide (available at mach-ii.com), you got your hands dirty with Mach-II and, whether you realize it or not, you also learned a great deal about how Mach-II works. In this guide we're going to extend that knowledge by introducing you to two of the most fundamental concepts in building more real-world Mach-II applications: the Event object and Listeners. The Event object is at the very heart of Mach-II, and as we saw in the Quick Start Guide everything in Mach-II revolves around announcing and handling events.

In the Hello World application you built in the Quick Start Guide your "sayHello" event simply displayed a view using the view-page command. While displaying views is of course going to be a very common occurrence in your application (yes, those pesky users are rearing their ugly heads again, and they actually expect to see things on the screen when they're using an application), displaying simple views isn't going to get you very far with respect to building a real-world web application.

In order to take the next step with Mach-II you'll need to grasp the Event object and Listeners, both of which are extremely simple concepts. If you think of the Quick Start Guide as laying the foundation of your Mach-II knowledge, you can think of this guide as starting to add the framing upon which you'll build your own Mach-II applications very quickly.

Since we're going to be building upon the simple "Hello World" application that is outlined in the Quick Start Guide, if you haven't created that application you can either refer back to the Quick Start Guide and work through that or download the completed example from mach-ii.com.

The Mach-II Event Object

As mentioned above, the Event object is at the heart of Mach-II. At their most basic level Mach-II applications are nothing more than a series of event announcements. These event announcements can be as simple as "sayHello" that simply renders a view and displays it to the user, or can be as complex as "solveTheWorldsProblems," which may involve a huge amount of backend processing, massive payoffs, creative accounting, and out and out voodoo to get the job done.

While I won't go so far as to say you can use Mach-II to solve the world's problems, it is a darn handy framework for building ColdFusion applications. One of the things that makes it so handy is this Event object to which I keep referring without actually telling you what it is. So let's get that bit of business out of the way.

The Event object in Mach-II is, not surprisingly, involved with every event in Mach-II applications. When an event is announced via the URL (e.g. index.cfm?event=sayHello) an Event object is created automatically, and for each event a new Event object is created. In the simple view example from the Quick Start Guide you didn't even realize this was happening (I didn't want to throw everything at you all at once!), but nonetheless an event object did get created.

What the Event object does for you is keep track of all the event details and package them up into one tidy little package. The "details" are things like form variables, URL variables, database queries, bits and pieces of composite views, you name it—if data is involved with or generated as the result of an event announcement, it will either exist in the Event object or you can easily put it there. This gives you a very easy way to manage even the most complex sequence of actions in a single object. Powerful stuff.

URL Variables

"Sounds great in theory," you're probably saying at this point, "but how do I actually use this supposedly fantastic Event object?" Let's start with a simple example, then we'll dig into Listeners to do more complex (as well as more real-world) tasks. For our first foray into using the Event object, the Hello World application you built in the Quick Start Guide will serve as a great starting point.

The Hello Mach-II application (yes, calling it an application is a bit of a stretch, but bear with me) worked fine, but it's not very personable. In actuality, by saying "Hello Mach-II!" it's actually just talking to itself, which left us with a bit of lingering loneliness. Why won't Mach-II talk to me? Does it just not like me?

Thankfully this is easy enough to resolve and will be a great way to show a very simple example of how the Event object works. Using the Hello Mach-II code as a starting point let's make things a bit more friendly. If you have the code handy (or if not, download it from mach-ii.com) please open the hello.cfm file in your views directory. In its original form it looks like this:

```
<html>
    <head>
        <title>Hello Mach-II!</title>
    </head>
    <body>
        <h3>Hello Mach-II!</h3>
    </body>
</html>
```

When you call the sayHello event (http://localhost/index.cfm?event=sayHello), your application dutifully displays "Hello Mach-II!" and ends. Now it's time to tell Mach-II a little about ourselves, so just append your name to the end of the URL as follows: http://localhost/index.cfm?event=sayHello&myName=Matt

Now you may notice that the final result didn't change yet, and you'd be entirely correct. Behind the scenes, however, your name is already getting stored in the Event object. Let's modify the hello.cfm view page and see if we can get Mach-II to actually talk to us instead of itself for a change:

```
<html>
    <head>
        <title>Hello <cfoutput>#event.getArg("myName")#</cfoutput>!</title>
    </head>
    <body>
         <h3>Hello <cfoutput>#event.getArg("myName")#</cfoutput>!</h3>
    </body>
</html>
```

Hit refresh and you'll find that Mach-II is a very personable framework after all. Now how did this happen and what's with all that event.getArg() stuff above? As mentioned above, Mach-II automatically takes your form and URL variables and puts them in the Event object for each event that is announced in your application. So when you appended your name to the end of the URL, Mach-II put that variable in the Event object.

The Event object is then available throughout the course of the request, so it can be called from the view page. By default when you hit your view page there will be an instance of the Event object available called "event," and this instance of the Event object allows you to access the variables it's storing via the getArg() method. As you can see in the example above, "myName" is the name of the argument you're looking for because that's what you called it in the URL.

The same scenario would apply if we built a form in which you entered your name into a form field and subsequently posted this form to a Mach-II event. Let's create another couple of events in our ever-growing application (they always grow so quickly ...) so you can see this in action.

Form Variables

First, open up the mach-ii.xml file in the config directory and declare the necessary events and views. In the event-handlers section of your file, add the following event handler (just put it after the sayHello event-handler):

```
<event-handler event="showInputForm" access="public">
    <view-page name="inputForm" />
</event-handler>
```

Now to make this actually work, we'll also need a page view called, you guessed it, inputForm. First let's add the page-view in the page-views section right after the hello page view:

```
<page-view name="inputForm" page="/views/inputForm.cfm" />
```

And last but not least, we have to create the actual inputForm.cfm file, which should look like this:

```
<html>
    <head>
        <title>Tell Me Your Name</title>
    </head>
    <body>
        <form action="index.cfm?event=sayHello" method="post">
            Enter your name: <input type="text" name="myName" size="30" /><br />
            <input type="submit" value="Say My Name!" />
        </form>
    </body>
</html>
```

After you complete these steps, go to
http://localhost/HelloMachII/index.cfm?event=showInputForm

Enter your name and hit the "Say My Name!" button.

"Whoa. Wait a minute," you might be thinking. "I thought there'd be more steps involved." Nope, it's as simple as that. Now why does this work? Why can we submit the form to the exact same page we were using for our URL variables and not change a thing? That's the beauty of the event object. Since Mach-II puts both form and URL variables into the event object, and since we named our form input field myName (which was the same as the URL variable), the view page can just output the result of the event.getArg("myName") call regardless of where it comes from (URL or form).

One small diversion here since I'm sure some of you are thinking, "But what if I have form and URL variables that have the same name?" That's obviously a situation you'll want to avoid since it's confusing in general, but in the properties section of the mach-ii.xml configuration file (which starts on line 4 by default) you'll see a property called parameterPrecedence on line 8. By default in the Mach-II application skeleton this is set to "form," meaning that if there is a conflict, form variables will override URL variables. Not a common situation and one that you should avoid anyway, but this is how Mach-II handles any conflicts between form and URL variables.

Now that you're starting to understand the power of the Event object, let's move on to Listeners and see how you can use Listeners in combination with the Event object to achieve some extremely powerful results.

What's a Listener?

The Listener object in Mach-II is one that you'll end up using a great deal as you begin to build more real-world Mach-II applications. Unlike the Event object, which exists in the Mach-II framework code and that you more or less just use without giving it too much thought, Listeners are objects within your application that extend the Listener object from the Mach-II framework code. What this means is that the Listener objects you build will inherit all the functionality of the base Listener object provided in the Mach-II framework.

More on those fancy OO terms in a moment. First let's talk about what a Listener does.

As the name implies, Listeners listen for things and subsequently take additional action as needed. Your application will notify Listeners of events they need to know about, and the Listener will then do what it's told and either return something (e.g. query data) or announce another event.

That probably sounds rather nebulous at this point, so let's consider a concrete example. In a blog application, for instance, unless you want everyone on the planet posting entries to your blog, you're going to want to require a login before allowing a user to post. As you might gather from your ever-growing Mach-II knowledge, when the login form is submitted this announces an event, and let's say this event is called processLoginAttempt. The obvious next step is to validate the login information provided by the user, but how does this happen? All we've looked at so far in our examples is displaying information to the user.

This is where Listeners come in. After the form data is submitted, as we saw above the form data becomes part of the Event object for this event. Within our event handler for the processLoginAttempt event, we can notify a Listener using the notify command that someone's trying to login. Think of this as the Mach-II application yelling "Hey you! Is this login valid?" to a Listener, which in this case we'll call the LoginListener.

The LoginListener's job is to respond to notifications from the application, but how does it know what to do once it's been notified? This is handled by the method attribute of the notify command. So first we notify a particular Listener, in this case the LoginListener, and then we tell it which method to execute, which in this case we'll call validateLogin. Remember that the Event object will be available to the LoginListener, so we don't need to worry about explicitly passing the LoginListener the form data. The LoginListener will then announce another event, either loginSucceeded if the login credentials are validated, or loginFailed if the user name and password aren't valid.

With the basics out of the way, let's look at the LoginListener in greater detail.

Building Your First Listener: LoginListener

As I mentioned above, the Listener objects that you build will extend the base Listener object from the Mach-II framework code. The reason for this is so you can gain a whole lot of native functionality by building upon what's available in this base Listener object. This does get into a bit of OO theory, but for now just know that it's necessary to have the Listeners in your application extend the Mach-II base Listener object in order to work properly within Mach-II applications.

First, create a new file in your favorite editor called LoginListener.cfc and save this in the model directory of your application. The outer-most tag of your CFC will look like this (go ahead and type this in now):

```
<cfcomponent name="LoginListener" displayname="LoginListener"
        output="false" extends="MachII.framework.Listener"
        hint="LoginListener for Hello Mach-II sample application">
</cfcomponent>
```

Note the extends attribute of the cfcomponent tag. This is what tells ColdFusion that you're going to be building upon functionality that's available in another object through a process known as inheritance. This is super-powerful OO stuff that we won't get into in any detail here, but please feel free to consult any of the numerous OO resources available in print and on the web for additional details.

Next we need to implement a specific function within our Listener in order to be able to take full advantage of our Listener. This function is called configure, and one of the important things this does is tell Mach-II to load our Listener into RAM when the application loads, which makes it available to our entire Mach-II application and also makes it perform darn fast since it's in RAM. (We'll get more into application configuration and object loading in a future guide.) You'll also use the configure function to handle any startup necessities (setting variables that you want available to your entire listener right away, etc.), but in our case we don't need to do anything in this function. The configure function still needs to be implemented even though it's empty in order for Mach-II to load it and make it available. Your configure function should look like this, and it goes right after your opening cfcomponent tag.

```
<cffunction name="configure" access="public" output="false" returntype="void"
        hint="Configures this listener as part of the Mach-II framework">
    <!--- do nothing for now --->
</cffunction>
```

With the initialization function out of the way, we now need to create the function that will validate user login attempts, and we'll call this function validateLogin. Don't worry, this is extremely simple and will allow you more insight into how the Event object gets tossed about within Mach-II applications.

As you learned above, when the user submits the login form the form data gets put into the event object. Just as we saw on our personalized hello.cfm page above this means that these details can be pulled from the Event object by anyone participating in this event, and we get these details within our listener in a very similar way. We simply declare that our function will expect the Event object as an argument and then we can pull the form data out of the Event object and validate it.

After our validateLogin function determines whether or not the login attempt is valid, it then needs to either return something or announce another event, otherwise the request would just stop within our Listener. In this case we're going to announce a success or failure event, which kicks off this whole event announcement process with which we're becoming intimately familiar all over again. We'll see what the implications of this are in a moment.

Let's get down to it and add the validateLogin function to our LoginListener; just put it right after the configure function:

```
<cffunction name="validateLogin" access="public" output="false" returntype="void"
    hint="Validates a login attempt and announces a success or failure event">
    <cfargument name="event" type="MachII.framework.Event" required="true" />

    <cfscript>
        // need to var scope ALL VARIABLES for thread safety!!!
        var success = false;

        // check hard-coded user name and password; in real life
        // we'd be hitting a database, LDAP, or something similar
        if (arguments.event.getArg("username") EQ "matt"
            AND arguments.event.getArg("password") EQ "machii") {
            success = true;
        }

        // announce the appropriate event based on the success or
        // failure of the login
        if (success) {
            announceEvent("loginSucceeded");
        } else {
            // put a message in the event argument so we can tell
            // the user their login failed
            arguments.event.setArg("message", "Your login failed.  Please try again.");
            announceEvent("loginFailed", arguments.event.getArgs());
        }
    </cfscript>
</cffunction>
```

Much of that is probably pretty straight-forward, but let's walk through it. As you can see this function takes the Event object in as an argument, and within this Event object are the form fields username and password.

The next chunk in the cfscript block does the work of this function. First and foremost, be aware that within your functions you absolutely, positively, must var scope any and all variables that are used within the function (or are generated by anything within the function) in order to make things thread safe. This is probably more common knowledge now than it was when CFMX was first released, but if you're new to CFCs it definitely will bite you if you don't do this. Queries, cffile, cfhttp, loop counters, anything and everything that either is a variable or generates a variable needs to be var scoped.

Following the default setting of the success variable to false we do the validation of the login attempt, and in this case we're just checking to see if the user name is matt and the password is machii. Obviously this isn't the way you'd validate a real login, but the principle would be the same; you'd just swap out this logic and make a database call to validate the login. (You can see this in action in some of the Mach-II sample applications.)

Let's review how we get the form variables out of the Event object. In the view example above we simply called event.getArg("argName"). The procedure is exactly the same here, but since we're taking the Event object in as an argument, we append the arguments scope to the beginning of the method call, which gives us arguments.event.getArg("username") and arguments.event.getArg("password"). If these are valid, which in this case means the username is matt and the password is machii, then the success boolean is set to true, otherwise it will remain false since that's the default we set earlier in the function.

Next, if we don't want things to stop within this function, we either need to return something or announce another event. In this case we'll do the latter, announcing loginSuccess if the login is valid, or loginFailed if the login is invalid. Events are announced by simply calling the Mach-II announceEvent method, which takes in a string representing the name of the event you wish to announce as its argument.

Bear in mind that when you announce a new event this is separate and distinct event, which means the new event does not share the Event object with the old event. Each event announced in Mach-II has its own Event object.

That doesn't mean, however, that you can't insert information into the Event object for the new event. This is actually quite easy to do, and if you take a look at the event announcement when the login fails, you'll see that we insert a message into the Event object so we can easily inform the user that their login failed.
arguments.event.setArg("message", "Your login failed. Please try again.");

Then when we announce the next event, we just need to tell Mach-II that we want to include all the current Event data in the new event, which is done like so:
announceEvent("loginFailed", arguments.event.getArgs());

The arguments.event.getArgs() call as the second argument of the announceEvent() method injects the data from the current Event object into the new Event object. This gets really handy for doing things like repopulating forms, because not only does this include our new message about the login failing, this actually includes the user name and password as well, so we could easily repopulate the form if that was desired.

Last but not least, we have to tell Mach-II that our listener exists, which is accomplished via a simple addition to the mach-ii.xml file. Go ahead and add this inside the <listeners> node of mach-ii.xml:

```
<listener name="loginListener" type="HelloMachII_v3.model.LoginListener" />
```

Make sense? Good! To review quickly, here's what happens in the validateLogin function:

1. The Event object is taken in as an argument.
2. We var scope the success variable and set it to a default value of false.
3. We check to see if the username and password variables in the Event object are set to "matt" and "machii" respectively. This is done by pulling the form data from the Event object using arguments.event.getArg("username") and arguments.event.getArg("password").

4. If the login data is valid, we set the success variable to true.
5. Based on the value of the success variable we announce the next event, either announceEvent("loginSucceeded") or announceEvent("loginFailed").
6. If the login fails, we set a message in the Event object (arguments.event.setArg("message", "Your login failed. Please try again.")), then announce the loginFailed event and put the data from the current Event object into the new Event object (announceEvent("loginFailed", arguments.event.getArgs())).
7. We told Mach-II our LoginListener exists by adding information about it to the listeners node of the mach-ii.xml file.

Now let's see how our Listner interacts with the application.

The Login Form and Login Events

With the LoginListener created and all the backend logic in place, next we have to build the front-end pieces so the users (yes, them again) can actually use the application. First, let's create a simple login form. Create a new file called loginForm.cfm and put it in your views directory.

```
<html>
    <head>
        <title>Login Form</title>
    </head>
    <body>
        <cfif event.isArgDefined("message")>
            <p><cfoutput>#event.getArg("message")#</cfoutput></p>
        </cfif>
        <form action="index.cfm?event=processLoginAttempt"
            method="post">
            <table border="0" width="500" cellpadding="2"
                cellspacing="1" bgcolor="#999999">
                <tr bgcolor="#dedede">
                   <td colspan="2" align="center">
                       <strong>Please Login</strong>
                   </td>
                </tr>
                <tr bgcolor="#ffffff">
                    <td>User Name:</td>
                    <td>
                        <input type="text" name="username" size="20" />
                    </td>
                </tr>
                <tr bgcolor="#ffffff">
                    <td>Password:</td>
                    <td>
                        <input type="password" name="password"
                            size="20" />
                    </td>
                </tr>
                <tr bgcolor="#dedede">
                    <td colspan="2" align="center">
                        <input type="submit" value="Login" />
                    </td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

Note that as with our "Say My Name!" form earlier, the form action points to a Mach-II event (index.cfm?event=processLoginAttempt), and recall that the form variables username and password will get put into our Event object when the form is submitted.

The next logical step is to create an event-handler we can use to show the login form, and another event-handler to process the login attempt. Go to your mach-ii.xml file and add the following two new event-handlers below the showInputForm event-handler:

```
<event-handler event="showLoginForm" access="public">
    <view-page name="loginForm" />
</event-handler>

<event-handler event="processLoginAttempt" access="public">
    <notify listener="loginListener" method="validateLogin" />
</event-handler>
```

The first event-handler should look very familiar to you. This simply displays the view-page called loginForm to the user. To complete everything we need to do for this event, go ahead and add a page-view for the loginForm in the page-views section of mach-ii.xml (after the inputForm view will be a good place for it):

```
<page-view name="loginForm" page="/views/loginForm.cfm" />
```

Now back to our processLoginAttempt event. Basically here you're seeing what we outlined above in action. When the processLoginAttempt event is announced, the notify command notifies the LoginListener that someone's trying to log into the application. The first attribute of the notify command is obviously the Listener you want to notify, and the second attribute is the method within this Listener that needs to be called, which in our case is validateLogin. This corresponds to the cffunction called validateLogin in our LoginListener CFC.

When the event is announced, the username and password fields are put into the Event object, and the Event object is then passed into the validateLogin method of ListenerLogin. We then pull the username and password variables from the Event object as we discussed earlier, we match this against the hard-coded values (username of "matt" and a password of "machii"), and then announce either loginSucceeded or loginFailed as the next event.

As you might have already guessed, as a last step we need to add two more event-handlers to mach-ii.xml, one called loginSucceeded and one called loginFailed. Go ahead and add these now to the event-handlers section of mach-ii.xml now:

```
<event-handler event="loginSucceeded" access="private">
    <view-page name="mainMenu" />
</event-handler>

<event-handler event="loginFailed" access="private">
    <announce event="showLoginForm" copyEventArgs="true" />
</event-handler>
```

A couple of noteworthy things here. First, notice that as mentioned before, the access to these events has been set to private, which means they can't be announced via the URL. If an event like "loginSucceeded" doesn't ever need to be announced by anything other than your application code itself (i.e. not in the URL), then it's just good practice to make these events private. If someone does attempt to call private events from the URL an exception will be thrown.

Second, we haven't seen the announce command in the mach-ii.xml file before. Using the announce command from mach-ii.xml is essentially the same as the announceEvent() method that we saw in our CFML code. This announces a new event just as if it was announced via the URL or from within the application code. The copyEventArgs attribute of the announce command allows us to copy the current event data into the Event object for the new event; remember from our discussion above that every time an event is announced a new Event object is created, so if we need to retain data from one event to the next, we need to set the copyEventArgs attribute to true. This will allow us to output the login failed message when we route the user back to the login form.

One more thing to add to round this all out. If the login is successful, in a real-world application we'd probably route the user to a menu of some sort, and in this case you'll see we're showing the user a view called mainMenu if they log in correctly. Create a file called mainMenu.cfm and save it in your view directory:

```
<html>
    <head>
        <title>Main Menu</title>
    </head>
    <body>
        <h3>This is the main menu!</h3>
    </body>
</html>
```

Then, as you probably know by now (come on, admit it, you're getting the hang of this stuff!), you have to create a page-view entry in the page-views node of mach-ii.xml:

```
<page-view name="mainMenu" page="/views/mainMenu.cfm" />
```

With all these pieces in place, let's login! Go to this URL to show the login form: index.cfm?event=showLoginForm

First, login with incorrect login information. Enter anything other than matt for the username and anything other than machii for the password. Hit submit and you should be taken right back to the login form, and at the top you should see the error message.

Next, login with the correct login information (username of matt, password of machii) and hit submit. You should be taken to the main menu page. Now granted we aren't doing anything here to actually secure the main menu page, so at this point if you go to index.cfm?event=showMainMenu directly you won't be stopped, but that's a subject for a future guide. The main point for now is if you get the login page and the message if your login fails, and the main menu if it succeeds, you've accomplished a great deal towards building your own Mach-II applications.

Summary

Congratulations! You're now officially a Mach-II Event and Listener guru. The Event object and Listeners are at the core of every Mach-II application, so if you have a good grasp of the concepts presented in this guide, you're well on your way toward complete Mach-II enlightenment. Stay tuned for additional Mach-II guides in the not-too-distant future!

About The Author:

Matt is a Principal IT Specialist for the Office of the Sergeant at Arms at the United States Senate in Washington, D.C. He has a Bachelor of Music degree from the University of Nebraska, a Master of Music degree from Wichita State University, and a Master of Science degree in Computer Information Systems from the University of Phoenix.

Matt has been working with ColdFusion since 1996. He is the Release Coordinator for the Mach-II framework and is a member of the editorial board for the ColdFusion Developer's Journal.

You can contact Matt at matt@mach-ii.com.

Copyright: