

Mach-II How to... Develop Listeners

For Mach-II for ColdFusion version 1.0

By Ben Edwards (ben@ben-edwards.com)

The Mach-II How To series of articles provides developers technical knowledge on how to develop Mach-II applications and components.

Mach-II is an object-oriented implementation of an implicit invocation architectural framework. For these articles to be most useful you'll need at least a basic understanding of object-oriented programming ("OO") and implicit invocation ("II") architectures. You don't need to be an expert on either OO or II, but being familiar with the concepts will be helpful. For a good introduction to these topics, check out articles and documentation at www.mach-ii.com, including my article called "An Introduction to Implicit Invocation Architectures".

Okay, let's learn how to develop listeners.

Listeners in an Implicit Invocation Architecture

In practice, software architectures are commonly treated as a collection of components and connectors. Components are the system's functional elements. For example, a shopping cart, a contact manager, and a database could be components of a software architecture. Connectors are the protocols for communication between components. Examples of connectors include method calls, SQL queries, and HTTP requests. A system's chosen architecture determines both the vocabulary of components and connectors that can be used as well as the set of constraints defining how they are combined.

Two metrics important for consideration in defining the publicly exposed interfaces of an architecture's components and connectors are a system's *cohesion* and *coupling*. Cohesion is a measure of the degree to which a component has a singular purpose. The greater cohesion a component exhibits, the more focused is the component and the fewer are the assumptions about contexts for reuse.

Coupling is the degree of interdependence between components. The less a component relies on other components (the looser its coupling), the more independent and reusable it is. Maximized cohesion (simple components) and minimized coupling (fewer connectors) are hallmarks of a flexible, maintainable architecture.

Event-based, implicit invocation is an example of a well-crafted architectural style with high cohesion and loose coupling. As such, it is one of the more broadly accepted architectural styles in software engineering. Examples of implicit invocation systems abound, including virtually all modern operating systems, integrated development environments, and database management systems.

Implicit invocation systems are driven by events. Events are triggered whenever the system needs to do something—such as respond to an incoming request. Events can take many forms across different types of implementations; for Mach-II an event is an object whose properties contain any contextual information needed to process the event (similar to the way a HTTP request carries with it all its form and query-string variables).

When an event is announced, the system looks up listener components for that event. Listeners fit the same criteria for components that we've already discussed—they are functional modules of the system. Components that wish to act as listeners are registered at configuration time (by specification in an XML file) to be notified of certain events when they occur at runtime. When an event is triggered, all registered listeners of that event are passed the event by means of a dynamically-determined method call. In this way, functions are implicitly invoked. This process of notifying listeners of an event is called event announcement.

Listeners in an Object Oriented Framework

Listeners have three primary roles in Mach-II:

- Listener components contain the business logic of a Mach-II application
- Listeners are notified of events they have registered interest in
- Listeners announce new events

In Mach-II for ColdFusion, framework components, including listeners, are developed as ColdFusion Components (CFCs). All listeners developed for a Mach-II application will extend the base Listener.cfc (MachII.framework.Listener) included with the Mach-II framework code.

Listeners have several important functions used by the framework, and three methods that are particularly important for developers creating their own listeners:

- `announceEvent(string eventName, [struct eventArgs])` – used to announce a new event to the framework
- `configure()` – called after a component is initialized, override to specify configuration logic
- `getParameter(string paramName)` – used to access configuration parameters specified in the configuration XML

It may be helpful to check out the MachII.framework.Listener component in the CFMX component browser. Go to <http://localhost:8500/cfide/componentutils/componentdoc.cfm> and click on the MachII.framework package. Click on the Listener component to display the component's details.

Creating a Mach-II Application

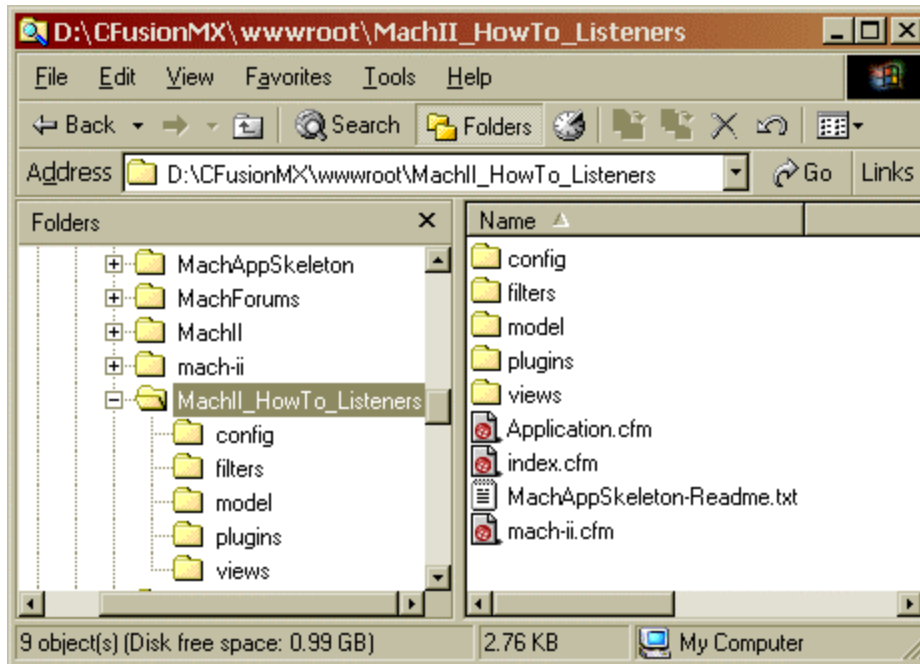
Before we begin creating listeners, let's create a Mach-II application to test the listener we are going to create. We'll create our application by using the MachAppSkeleton files downloadable from www.mach-ii.com.

The MachAppSkeleton is a group of files that can be used as a template for creating a new Mach-II application. Instructions are included with the package in a file called MachAppSkeleton-Readme.txt, and we'll summarize them here for creating our application to test our listeners.

The application we're going to create will be called MachII_HowTo_Listeners. Begin by copying the contents of the MachAppSkeleton directory to a new folder in your ColdFusion wwwroot called MachII_HowTo_Listeners.

Your new application folder should look similar to the following image:

**Mach-II How to... Develop Listeners
For Mach-II for ColdFusion version 1.0**



Next, open the Application.cfm file in the application's root directory and set the name attribute of the <cfapplication> tag to 'MachII_HowTo_Listeners'.

Next, open the mach-ii.xml file in the application's config directory. The first section in the XML file defines properties for the application. Set the applicationRoot property to the path of your application's folder, relative to the web-root and beginning with a '/'. For this application the applicationRoot value will be '/MachII_HowTo_Listeners'.

Next, set the defaultEvent property value to 'showLoginForm'. This specifies the name of our application's default event, which we haven't yet defined an event-handler for.

The <properties> section of the mach-ii.xml configuration file should look like the following code when you're done:

```
<!--- PROPERTIES --->
<properties>
  <property name="applicationRoot" value="/MachII_HowTo_Listeners" />
  <property name="defaultEvent" value="showLoginForm" />
  <property name="eventParameter" value="event" />
  <property name="parameterPrecedence" value="form" />
  <property name="maxEvents" value="10" />
  <property name="exceptionEvent" value="exceptionEvent" />
</properties>
```

Next, register two new <page-view>s: one for a loginForm.cfm page and another for a loginWelcome.cfm page. The loginForm.cfm page will simply consist of a form that submits a username and password field as a login event (which we have not yet declared an event-handler for). The loginWelcome.cfm page should consist of only a simple welcome message and a link announcing the showLoginForm event. Code for the loginForm.cfm and loginWelcome.cfm pages can be found at the end of this article.

**Mach-II How to... Develop Listeners
For Mach-II for ColdFusion version 1.0**

The <page-views> section of the mach-ii.xml configuration file should look like the following code when you're done:

```
<!-- PAGE-VIEWS -->
<page-views>
  <page-view name="loginForm" page="/views/loginForm.cfm" />
  <page-view name="loginWelcome" page="/views/loginWelcome.cfm" />
  <page-view name="exception" page="/views/exception.cfm" />
</page-views>
```

Finally, add a new <event-handler> for our default event, showLoginForm. Make access to the event-handler public, so it can be announced from a web-request. All the event-handler needs to do is show the loginForm page-view.

The <event-handler> section of the mach-ii.xml configuration file should look like the following code when you're done:

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

  <event-handler event="exceptionEvent" access="private">
    <view-page name="exception" />
  </event-handler>
</event-handlers>
```

At this point we can go to http://localhost:8500/MachII_HowTo_Listeners and see the login form. Because the URL doesn't include a parameter specifying an event, the framework will handle the default event (showLoginForm) and the loginForm.cfm page will be displayed. Submitting the login form won't yet work though because we haven't specified an event-handler for the login event.

Creating a new Listener CFC

First, let's define a new CFC called LoginListener (LoginListener.cfc) to authenticate a user's login credentials based on username and password. Create the file in the application's model directory (this is a recommended practice and not a requirement). Also, be sure the LoginListener extends MachII.framework.Listener. When a listener that we define extends Listener.cfc it *inherits* all the functionality of the base listener component.

```
<cfcomponent displayname="LoginListener"
  extends="MachII.framework.Listener"
  hint="Authenticates a user's login credentials based on username and
password.">
</cfcomponent>
```

Next, let's define a function called configure, for now leaving the function body empty. When we define this method, we are actually *overriding* a function in the base Listener.cfc. Because we are

overriding, we'll leave the arguments and attributes (access, returntype, etc) of the function the same as defined in its *supertype*.

We'll learn more about the configure() function later, but for now it should look something like this:

```
<cffunction name="configure" access="public" returntype="void">
    <!-- DO NOTHING --->
</cffunction>
```

Note that for Mach-II components, such as listeners, we do not define our own init() functions. The init() function in each base component is used during the initialization phase of the framework and should not be overridden. The configure() method defined for each component, on the other hand, is defined specifically to be overridden by custom components. The configure() function of each component is called by the framework during the framework's configuration phase, which occurs following the initialization phase.

When the framework is initialized an instance of each registered listener is also created and initialized. The same listener instance is maintained and used by the framework until it is reinitialized. Further details of the initialization and configuration phases are outside the scope of this article, what is important to know is that the configure() function should define configuration code that the listener should run before handling events.

These first few steps we've taken will be repeated almost every time we create a new Listener. We'll define the listener CFC, we'll make it extend MachII.framework.Listener, and we'll define the configure() function to hold any configuration logic.

Defining Listener Functions

Now let's start to create some business logic. The LoginListener we're defining will be responsible for authenticating a user's credentials (username and password) and determining whether or not the login attempt results in success or failure.

Let's create a function called 'attemptLogin' to handle the authentication logic. Be sure to declare the access of the function as "public" so the method can be invoked by the Mach-II framework. Define the returntype attribute of the function as "void" (meaning the function will not return a value).

Wait a minute—if the attemptLogin() function does not return a value then how do we know if the login attempt was successful or not? Remember, the Mach-II framework is driven by events, and listeners are capable of announcing events. So, to signal login success or failure, our LoginListener will announce a new event. We'll see how to announce events from a listener soon.

Speaking of events, we'll need to define the attemptLogin() function as accepting a single argument, named 'event'. The event argument will be required and will be of type MachII.framework.Event.

We've still have more to define in attemptLogin(), but for now it should look like this:

```
<cffunction name="attemptLogin" access="public" returntype="void">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <!-- TODO: GET USER INFO --->
    <!-- TODO: AUTHENTICATE USER INFO --->
    <!-- TODO: ANNOUNCE SUCCESS/FAILURE EVENT --->
```

```
</cffunction>
```

All the information we need from a user to attempt a login will be encapsulated in the event object passed to attemptLogin() when invoked by the framework. To get information from the event, such as the user's username and password, we use the event's getArg() function.

```
<cffunction name="attemptLogin" access="public" returntype="void">
  <cfargument name="event" type="MachII.framework.Event" required="true" />
  <!-- GET USER INFO -->
  <cfset var username = arguments.event.getArg('username') />
  <cfset var password = arguments.event.getArg('password') />

  <!-- TODO: AUTHENTICATE USER INFO -->
  <!-- TODO: ANNOUNCE SUCCESS/FAILURE EVENT -->
</cffunction>
```

Once we have the user's information we'll need to determine whether or not it is valid. Because there are potentially several ways to perform this validation (database, LDAP, etc) we'll separate the functionality into a new function called 'isLoginValid' and thus keeping the attemptLogin() function cohesive.

Let's make the isLoginValid() function publicly accessible and with a returntype of boolean. It will require username and password arguments (both of type string). For now, just make isLoginValid() always return true. We'll come back and finish it later.

```
<cffunction name="isLoginValid" access="public" returntype="boolean">
  <cfargument name="username" type="string" required="true" />
  <cfargument name="password" type="string" required="true" />

  <cfreturn true />
</cffunction>
```

Announcing Events in Listeners

Now that we have an isLoginValid() function (albeit unfinished), we can finish our attemptLogin() function. Use isLoginValid(), passing it the username and password we retrieved from the event, to determine whether or not the login information is valid. If the user's information is valid, we want to announce a 'loginSuccess' event. Otherwise we want to announce a 'loginFailure' event.

To announce an event from within a listener we'll use the announceEvent() function defined in the base Listener.cfc. The announceEvent() function takes two arguments. The first argument, eventName, is required and must be of type string. It specifies the name of the event to announce. The second argument, eventArgs, is optional, but if defined must be of type struct. The eventArgs argument specifies any arguments that will be *encapsulated* in the new event being announced.

When finished, attemptLogin() will look something like the following:

```
<cffunction name="attemptLogin" access="public" returntype="void">
  <cfargument name="event" type="MachII.framework.Event" required="true" />
  <!-- GET USER INFO -->
  <cfset var username = arguments.event.getArg('username') />
  <cfset var password = arguments.event.getArg('password') />
```

Mach-II How to... Develop Listeners For Mach-II for ColdFusion version 1.0

```
<!--- AUTHENTICATE USER INFO --->
<!--- ANNOUNCE SUCCESS/FAILURE EVENT --->
<cfif isLoginValid(username, password)>
    <cfset announceEvent('loginSuccess') />
<cfelse>
    <cfset announceEvent('loginFailure') />
</cfif>
</cffunction>
```

If we wanted to put the user's login information (username and password) into the loginSuccess or loginFailure events we're announcing, we can use the following code:

```
<cfset announceEvent('loginSuccess', arguments.event.getArgs()) />
```

The above code gets all the event-args of the event passed to attemptLogin() and puts them into the new event being announced.

Registering and Configuring Listeners

Now that we've got our LoginListener developed we'll need to register it in the mach-ii.xml config file. The following code in the <listeners> section will register the LoginListener:

```
<!-- LISTENERS -->
<listeners>
    <listener name="LoginListener"
        type="MachII_HowTo_Listeners.model.LoginListener">
        <invoker type="MachII.framework.invokers.CFCInvoker_Event" />
    </listener>
</listeners>
```

The name attribute specified in the <listener> declaration can be anything, it is just a key used to reference the listener within the XML. The type attribute must be a fully-qualified, dot-delimited path to the CFC.

Be sure to note the required <invoker> tag we've included in the listener registration code. The invoker type specified determines how event information is passed to a listener when invoked. We'll learn more about invokers a little later, but for now ensure that the invoker type used is 'MachII.framework.invokers.CFCInvoker_Event'.

We'll also want to add more event-handlers to take advantage of our new listener functionality. We'll need event-handlers for a login event, a loginSuccess event, and a loginFailure event.

The login event-handler should notify the LoginListener's attemptLogin() function, and it should be publicly accessible. The loginSuccess event-handler will simply show the loginWelcome view. The loginFailure event-handler will re-show the loginForm by announcing the showLoginForm event. The loginSuccess and loginFailure event-handlers need only be privately accessible.

Add these event-handlers to the <event-handlers> section of the configuration file:

```
<!-- Notify the listener to validate login. -->
<event-handler event="login" access="public">
```

Mach-II How to... Develop Listeners For Mach-II for ColdFusion version 1.0

```
<notify listener="LoginListener" method="attemptLogin" />
</event-handler>

<!-- On login success, show a welcome page. -->
<event-handler event="loginSuccess" access="private">
    <view-page name="loginWelcome" />
</event-handler>

<!-- On login failure, show the login form. -->
<event-handler event="loginFailure" access="private">
    <announce event="showLoginForm" />
</event-handler>
```

At this point we want to go back to the LoginListener's <listener> element and add a few configuration parameters. Configuration parameters are specified in the XML and are accessible in the listener via the `getParameter()` function.

With a real application, the username and password would likely need to be validated against a database or LDAP server. For this example listener, we'll add two parameters: `validUsername` and `validPassword`. We'll use these parameters to specify an acceptable username and password to validate against. The `validUsername` and `validPassword` parameters can be set to any value, but for this example we'll use 'implicit' and 'invocation'.

Add these parameters to the <parameters> section of <listener> in the configuration file:

```
<listener name="LoginListener"
    type="MachII_HowTo_Listeners.model.LoginListener">
    <invoker type="MachII.framework.invokers.CFCInvoker_Event" />
    <parameters>
        <parameter name="validUsername" value="implicit" />
        <parameter name="validPassword" value="invocation" />
    </parameters>
</listener>
```

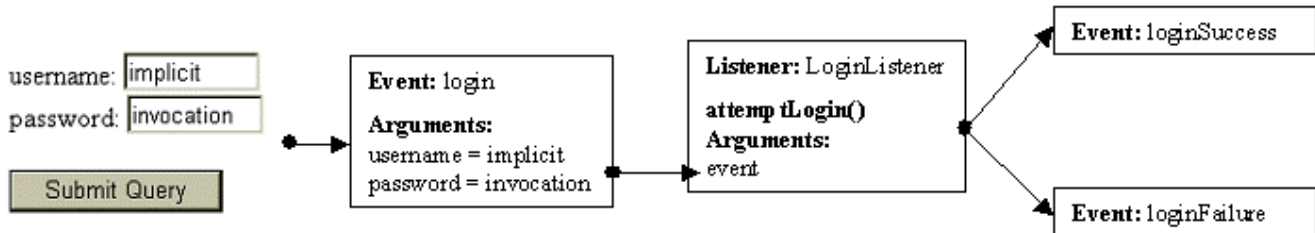
Now we'll finish the `isLoginValid()` function by making it validate the user's username and password against the `validUsername` and `validPassword` parameters set in the configuration file. Use the base Listener's `getParameter()` function we've inherited in our listener to access the configuration parameters:

```
<cffunction name="isLoginValid" access="public" returntype="boolean">
    <cfargument name="username" type="string" required="true" />
    <cfargument name="password" type="string" required="true" />

    <!--- AUTHENTICATE AGAINST PARAMETERS IN CONFIG XML --->
    <cfif arguments.username EQ getParameter('validUsername') AND
        arguments.password EQ getParameter('validPassword')>
        <cfreturn true />
    <cfelse>
        <cfreturn false />
    </cfif>
</cffunction>
```

While we are using the parameters to specify a valid username and password, another possibility could have been to use the parameters to define how to connect to an LDAP server or database.

Now when we go to `http://localhost:8500/MachII_HowTo_Listeners`, we can submit the login form. When we submit the form the request to the Mach-II framework will trigger the creation of an event with name login and arguments username and password. The event-handler for the login event will notify the LoginListener and pass to its `attemptLogin()` method the event. Using information in the event the listener will perform business logic and announce a new event, either `loginSuccess` or `loginFailure`.



Invoking Listeners

When we submit the login form to the Mach-II application, the framework does a lot of work to make sure the login event gets delivered to our LoginListener. One framework component essential to making this happen is called an invoker.

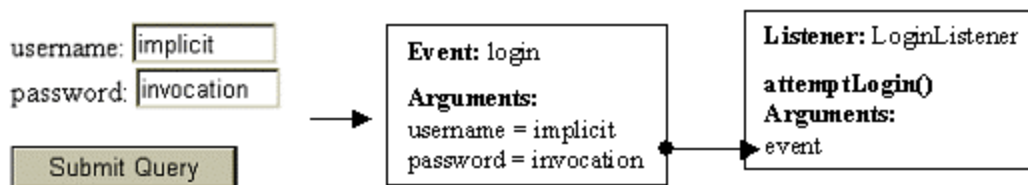
Invokers are used to dynamically call a listener's methods. When we register listeners in the configuration XML we also specify an invoker for the listener. The framework uses the invoker to call methods on the listener to notify the listener to handle an event.

Like listeners, all Mach-II invokers are developed as ColdFusion Components (CFCs). Custom invokers will all extend the base `ListenerInvoker.cfc` (`MachII.framework.ListenerInvoker`) included with the Mach-II framework code. Two invokers come with the Mach-II framework: `CFCInvoker_Event` and `CFCInvoker_EventArgs`. Both are in the `MachII.framework.invokers` package.

The `CFCInvoker_Event` invoker passes the entire event being handled to a listener method. Consequently, a listener method that will be invoked in this way needs to have a single, required argument named 'event' of type `MachII.framework.Event`.

```
<invoker type="MachII.framework.invokers.CFCInvoker_Event" />
```

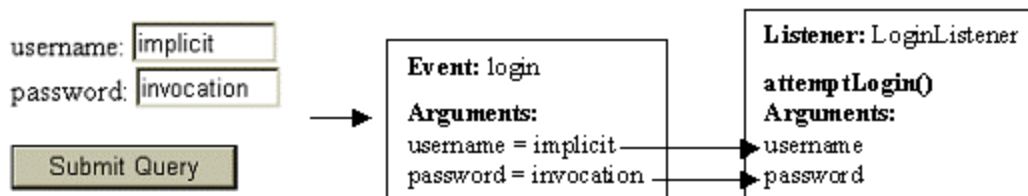
```
<cffunction name="attemptLogin" access="public" returntype="void">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    .
    .
    .
</cffunction>
```



The CFCInvoker_EventArgs invoker will match an event's arguments against the names of arguments of the method on a listener being invoked and pass them individually. For instance, if an event has arguments with names 'username' and 'password' and the listener function being invoked has arguments with names 'username' and 'password' then the event arguments will be passed to the function being invoked as arguments with the same name.

```
<invoker type="MachII.framework.invokers.CFCInvoker_EventArgs" />
```

```
<cffunction name="attemptLogin" access="public" returntype="void">
  <cfargument name="username" type="string" required="true" />
  <cfargument name="password" type="string" required="true" />
  . . .
</cffunction>
```



The <notify> element within an <event-handler> tag tells the framework to use a listener's invoker to call a particular method. The listener attribute's value should be the name of a listener registered under the <listeners> section of the configuration file. The method attribute specifies which method inside the listener to invoke. An optional resultKey attribute can also be specified (not shown here) to store a returned value in a defined scope.

```
<notify listener="LoginListener" method="attemptLogin" />
```

We set the invoker for our LoginListener to CFCInvoker_Event. The attemptLogin() function accepted a single argument named event because the invoker will pass it a full reference to the event when invoked. If we would have used the CFCInvoker_EventArgs invoker the attemptLogin() function would have had two arguments, username and password, that would have been matched up against the events arguments with the same name.

Conclusion and Review

Listeners have three primary roles in Mach-II:

- Listener components contain the business logic of a Mach-II application
- Listeners are notified of events they have registered interest in
- Listeners announce new events

In this Mach-II How To article we've learned how to:

- Create a Mach-II application
- Create a Listener component
- Register and configure a listener in a Mach-II configuration file
- Specify an invoker for a listener
- Create event-handlers that use listeners to perform business logic
- Announce events from listeners

Terms and Definitions

cohesion – a measure of the degree to which a component (or method) has a singular purpose

coupling – the degree of interdependence between components

encapsulation – the idea that a class or component is a cohesive bundle containing methods and properties it is responsible for

inheritance – implementing a specialization relationship with another class and inheriting its functionality in a way that it can be used as is or specialized

supertype – the parent type that another type extends

overriding – keeping the same method signature of a supertype and specializing its functionality

Resources

All the following resources are related, useful, and available at www.mach-ii.com:

- Code for this article (MachII_HowTo_Listeners.zip)
- MachAppSkeleton code (MachAppSkeleton.zip)
- Mach-II Configuration Guide for ColdFusion
- Introduction to Implicit Invocation Architectures, an article by Ben Edwards
- More Mach-II How To articles

About the Author

Ben Edwards is cofounder of the Mach-II organization and the development lead for the Mach-II for ColdFusion project. He is a Sun Certified Java Programmer and holds a degree in Computer Science from the Georgia Institute of Technology with specializations in Software Engineering and Educational Technology.

Before Mach-II, Ben co-founded Synthis Corporation while at Georgia Tech where he led the technical adoption of new technologies into their flagship product. Ben has also worked as technical lead, architect, and developer for several projects for companies including Coca-Cola and IBM.

Ben currently trains developers on software engineering practices focusing on Java, object-oriented programming, and software architectures. Check out the Training section of www.mach-ii.com for more details and Mach-II training class schedules. You can contact him at ben@ben-edwards.com.

/MachII_HowTo_Listeners/config/mach-ii.xml

```
<mach-ii version="1.0">

  <!-- PROPERTIES -->
  <properties>
    <property name="applicationRoot" value="/MachII_HowTo_Listeners" />
    <property name="defaultEvent" value="showLoginForm" />
    <property name="eventParameter" value="event" />
    <property name="parameterPrecedence" value="form" />
    <property name="maxEvents" value="10" />
    <property name="exceptionEvent" value="exceptionEvent" />
  </properties>

  <!-- LISTENERS -->
  <listeners>
    <listener name="LoginListener"
type="MachII_HowTo_Listeners.model.LoginListener">
      <invoker type="MachII.framework.invokers.CFCInvoker_Event" />
      <parameters>
        <parameter name="validUsername" value="implicit" />
        <parameter name="validPassword" value="invocation" />
      </parameters>
    </listener>
  </listeners>

  <!-- EVENT-FILTERS -->
  <event-filters>
  </event-filters>

  <!-- EVENT-HANDLERS -->
  <event-handlers>
    <event-handler event="showLoginForm">
      <view-page name="loginForm" />
    </event-handler>

    <!-- Notify the listener to validate login. -->
    <event-handler event="login" access="public">
      <notify listener="LoginListener" method="attemptLogin" />
    </event-handler>

    <!-- On login success, show a welcome page. -->
    <event-handler event="loginSuccess" access="private">
      <view-page name="loginWelcome" />
    </event-handler>

    <!-- On login failure, show the login form. -->
    <event-handler event="loginFailure" access="private">
      <announce event="showLoginForm" />
    </event-handler>

    <event-handler event="exceptionEvent" access="private">
      <view-page name="exception" />
    </event-handler>
  </event-handlers>
</mach-ii>
```

**Mach-II How to... Develop Listeners
For Mach-II for ColdFusion version 1.0**

```
</event-handlers>

<!-- PAGE-VIEWS -->
<page-views>
  <page-view name="loginForm" page="/views/loginForm.cfm" />
  <page-view name="loginWelcome" page="/views/loginWelcome.cfm" />
  <page-view name="exception" page="/views/exception.cfm" />
</page-views>

<!-- PLUGINS -->
<plugins>
</plugins>

</mach-ii>
```

/MachII_HowTo_Listeners/model/LoginListener.cfc

```
<cfcomponent displayname="LoginListener"
    extends="MachII.framework.Listener"
    hint="Authenticates a user's login credentials based on username and
password.">

    <cffunction name="configure" access="public" returntype="void">
        <!--- DO NOTHING --->
    </cffunction>

    <cffunction name="attemptLogin" access="public" returntype="void">
        <cfargument name="event" type="MachII.framework.Event" required="true" />
        <!--- GET USER INFO --->
        <cfset var username = arguments.event.getArg('username') />
        <cfset var password = arguments.event.getArg('password') />

        <!--- AUTHENTICATE USER INFO --->
        <!--- ANNOUNCE SUCCESS/FAILURE EVENT --->
        <cfif isLoginValid(username, password)>
            <cfset announceEvent('loginSuccess') />
        <cfelse>
            <cfset announceEvent('loginFailure') />
        </cfif>
    </cffunction>

    <cffunction name="isLoginValid" access="public" returntype="boolean">
        <cfargument name="username" type="string" required="true" />
        <cfargument name="password" type="string" required="true" />

        <!--- AUTHENTICATE AGAINST PARAMETERS IN CONFIG XML --->
        <cfif arguments.username EQ getParameter('validUsername') AND
            arguments.password EQ getParameter('validPassword')>
            <cfreturn true />
        <cfelse>
            <cfreturn false />
        </cfif>
    </cffunction>
</cfcomponent>
```

/MachII_HowTo_Listeners/views/loginForm.cfm

```
<form action="index.cfm?event=login" method="post">
  username: <input type="text" name="username"><br>
  password: <input type="text" name="password"><br>
  <br>
  <input type="submit">
</form>
```

/MachII_HowTo_Listeners/views/loginWelcome.cfm

```
Login was successful. Welcome.
<br>
<br>
<a href="index.cfm?event=showLoginForm">Go back to the Login form.</a>
```