Mach-II Frequently Asked Questions

Mach-II 1.1.1 for ColdFusion

Last Updated on November 15, 2006

Peter J. Farrell
Lead Developer

What is Mach-II?

Mach-II is a framework for developing object oriented Model-View-Controller web-applications. The framework focuses on easing software development and maintenance. Mach-II was the first object-oriented framework for ColdFusion and continues to mature as a strong and viable framework choice for developers.

What is a framework?

"A defined support structure in which another software project can be organized and developed. A framework may include support programs, code libraries, a scripting language, or other software to help develop and glue together the different components of a software project."
– Wikipedia (http://en.wikipedia.org/wiki/Framework)

Boiled down, a framework is a set of core files that you can leverage to help you better and more consistently organize your code and make your development life easier. It provides a pre-defined structure for your application. Since a framework provides a common and universally known structure for your application, any developer that knows the framework will understand how the application is structured. This can save a great deal of time when people who weren't involved with the initial creation of the application need to come in and work on the code.

Despite the reputation that frameworks may have in some circles, frameworks exist to make our lives as developers easier. A good framework provides the means to handle the basic “plumbing” or flow of an application so we don’t have to reinvent that wheel for each application. In addition, a good framework will include a set of core files that can be leveraged during application development to make common tasks easier and less repetitive. Your application uses the framework’s Application Programming Interface (API) to simplify and accelerate common development tasks.

What is the Model-View-Controller pattern?

Model-View-Controller (MVC) is type of software architecture pattern that divides an application into three separate parts – a business logic/data persistence layer (model), user interface (view) and control logic (controller).

The Model handles business logic and data integrity. The Model is sometimes talked about as the “engine” of an application. A View provides a “view” into the state of the application. With web-based software, Views can written in

(X)HTML, Flash and emails as well as others. The Controller (Mach-II) receives user/system events and interacts with the Model and/or View to produce the desire results.

The main purpose of MVC is keep your business logic (model) and user interface (view) separate and independent from each other. For a more in depth look at MVC, you can read about it at Wikipedia – http://en.wikipedia.org/wiki/MVC.

What is framework's mission?

Mach-II's place in the framework landscape is as an application development framework as opposed to an Object-Relational Mapping (ORM) framework like Reactor, or an Inversion of Control (IoC) framework such as ColdSpring. The mission of Mach-II is to do MVC and do that well without sacrificing extensibility. Mach-II is highly extensible through its' filter and plugin architecture. This design allows you to leverage third-party functionality or other frameworks easily. For example, ColdSpring ships with a plugin for Mach-II that enables seamless integration between the two frameworks.

A by-product of this basic credo is that Mach-II is not hampered by the release schedules or bugs of other frameworks because they are not intrinsically integrated into the Mach-II core. This allows Mach-II developers to "virtually customize" Mach-II so your application always have your architecture and specifications without being forced to abide by convention over configuration.

Is Mach-II a full-stack framework?

Mach-II is not a full-stack framework like Ruby On Rails. Mach-II is designed to be singular in purpose and provides extensibility through the framework's architecture instead of providing built-in functionality to code generation or Object Relational Mapping (ORM). See the "What is your mission?" FAQ for more information on why the framework authors decided to be a short-stack framework.

Why should I use Mach-II instead of Fusebox?

Mach-II is object-oriented which lends itself to developing loosely coupled and tightly cohesive models. In general, Fusebox is largely procedural unless you choose to follow the MVC pattern. Choosing between Mach-II and Fusebox depends on the way you or your development team works. If you understand or want to learn more about object-oriented design and development, Mach-II might be the framework of your choice. If prefer to write procedural code or do not have the time to learn about object-oriented design and development, Fusebox might be a better framework choice for you.

Why should I use Mach-II instead of Model-Glue?

Model-Glue is another popular framework for ColdFusion that utilitizes the MVC design pattern. According to Model-Glue's author, the framework has slightly simpler architecture than Mach-II and provide some full-stack features like scaffolding and built-in ORM capabilities with the Unity release.

Every framework has its' own pluses and minuses which may or may not aid your specific project. The choice of framework for a project is up to the developer to use the right tool for the right job and it is out of the scope of this document (and its' author) to make the decision for you. Hopefully these FAQs can help you make a more informed decision. For more information about Model-Glue, please refer to the Model-Glue website at http://www.model-glue.com.

What does the "II" in Mach-II stand for?

Mach-II uses an **implicit invocation** architecture, hence the "II". In an implicit invocation architecture rather than explicitly invoking procedures, the application announces events and this in turn causes objects that have registered an interest in the event to invoke methods. For example, an announcement of the "processLoginAttempt" event might implicitly cause a "checkLoginCredentials" method to be invoked in a login object. This approach keeps the application's flow extremely flexible.

How is Mach-II licensed?

Mach-II is licensed under an Apache 1.1 license until the 1.1.0 release of the framework and owned by the Mach-II Corporation. With the 1.1.1 release, Mach-II is released under the Apache 2.0 license. You can use Mach-II on any commercial application as long as you abide by the license. For more details, please see the license that is shipped with the framework.

Who created and maintains the Mach-II framework?

Mach-II for ColdFusion was originally created by Ben Edwards and Hal Helms. The framework is currently maintained by a dedicated team of contributors. The Mach-II release coordinator is Matt Woodward and the lead developer is Peter J. Farrell.

How do I contact Team Mach-II and how can I help further the framework?

The best way of contacting Team Mach-II or helping the Mach-II project is info@mach-ii.com. All bugs reports can be emailed to Team Mach-II at bugs@mach-ii.com.

You can also join our public email listserv/forums at http://groups.google.com/group/mach-ii-for-coldfusion. This is the best place to get quick help from other developers that use Mach-II.

If you have any questions, comments, bug reports or want to become a contributor, feel free to post a message to the Mach-II email list at the location previously mentioned.

What platforms does Mach-II run on?

The Mach-II framework is available for ColdFusion and PHP (Beta). Mach-II for ColdFusion has officially been QA'ed on Adobe ColdFusion MX6.1 and MX7.

Does Mach-II run on BlueDragon?

Mach-II is not "officially" supported on New Atlanta's BlueDragon by the Mach-II Team. However, it is possible to run Mach-II on BlueDragon. According to Vinci Bonfanti at New Atlanta, Mach-II 1.1.0 runs without modification on BlueDragon 6.2.1. We plan to "officially" support BlueDragon with the 1.1.1 release.

If you have tested Mach-II on BlueDragon, feel free to share your experience with Team Mach-II (see FAQ regarding how to contact Team Mach-II) so we may update our documentation with more useful information.

Is Mach-II compatible with other frameworks such as Tartan, ColdSpring or Reactor?

Yes. Mach-II has been used by many people who also use Tartan, ColdSpring or Reactor. The Tartan website uses

Mach-II in conjunction with the Tartan framework. ColdSpring has been deployed with Mach-II on several websites. Reactor can easily be integrated through a plugin or by integrating it with ColdSpring. See the respective framework's documentation for more information on how to use Mach-II and your choice of framework.

How do I install the Mach-II framework?

1. Download the core framework code from http://www.mach-ii.com.
2. Unzip the framework to your web root.
    a. For example, on Windows the default web root is `[DRIVE]:\Inetpub\wwwroot`, or if you are using ColdFusion's built-in web server, your web root is likely `[DRIVE]:\CFusionMX\wwwroot`. If you are on a Unix system, your web root will vary.
    b. The end result of this step is that you should have the directory MachII (No hyphen) in your web root. Inside this directory you will find the core framework files.
    c. If you wish to place the core Mach-II framework files in a location that is different from your webroot, just create a mapping called "MachII" in the ColdFusion Administrator that points this mapping to the location where you placed the framework files.
3. If you use sandbox security on your ColdFusion server, you may have to add the framework's directory to your sandbox otherwise ColdFusion may throw an java security exception.

Just three simple steps and you are done. The Mach-II framework is just a ColdFusion application, so it will operate and behave just like any other ColdFusion application.

If your webroot is "wwwroot", your directory structure should look like this:

```
|-+ wwwroot
  |- MachII
  |- ContactManager
  |- Roulette
  |- ShoppingCart
```

If you encounter problems:

1. Please first ensure that your directory structure matches the one shown above.
2. Try restarting your ColdFusion Application Server service.
3. If problems persist, please check out the Mach-II for ColdFusion Google Group: http://groups.google.com/group/mach-ii-for-coldfusion

I just upgraded my installation of Mach-II. Why am I getting this strange exception?

If you are updating your Mach-II installation from an older version, you must restart your ColdFusion Application Server after you replace your older version with a newer version.

If you do not restart your server, you will receive an error similar to this:

```
The value returned from function getAppFactory() is not of type
MachII.framework.AppFactory.

The error occurred in C:\[PathToYourWebRoot]\MachII\framework\AppLoader.cfc: line XX
Called from C:\[PathToYourWebRoot]\MachII\mach-ii.cfm: line XX
```

Where can I find information about a certain method/function in the framework?

In the 1.1.0 release of Mach-II, we improved the internal code documentation of the framework. You can easily find out more information by looking at the hints in the function and argument tags of any framework file. Fire up your CFCExplorer and check it out.

What does the mach-ii.xml file do?

The mach-ii.xml file is the main configuration file for your Mach-II application. The XML configuration file has an accompanying DTD currently called mach-ii_1_1_1.dtd that is included with the core framework files. This file is commonly referred to as the “configuration file” throughout these FAQs.

How is the configuration file structured?

There are six sections that make up a configuration file:

- **properties**
  Contains general application settings, both those requird by Mach-II as well as developer-defined settings
- **listeners**
  Contains the instance name and specific CFC path for the application’s Listener objects
- **event-filters**
  Contains information about the application's filters, which are used to pre-process events and potentially take action (such as short-circuiting the event) at any point during a event
- **plugins**
  Similar to filters, plugins can be used to take action at any of seven “plugin points” (preProcess, preEvent, postEvent, preView, postView, postProcess, and handleException) during the course of an event
- **event-handlers**
  Each event handler contains the details of what occurs during events that you define. Events are built up of commands.
- **page-views**
  The view pages are registered in this section, and each has a name (which serves as an alias called via the view-page command in Mach-II) and the specific location of any view (i.e. CFM, HTML, etc.)

Are there required properties for my application and what can I do with them?

Yes, there are six properties that are required by all Mach-II applications:

- **applicationRoot**
  The root directory of the application relative to your web server’s root directory. This could be a mapped directory if your application requires one.
- **defaultEvent**
  The default event that is announced if an event name is not specified in the URL.
- **eventParameter**
  The URL variable that contains the event name. This defaults to “event” if not specified.
- **parameterPrecedence**

Because Mach-II automatically puts all URL and form variables in the event object, this property determines which scope (URL or form) takes precedence if a URL and form variable are named the same

- **maxEvents**
  The maximum number of events that can be announced in a row. This safeguards against event announcement infinite loops that could accidentally be introduced by developer error.
- **exceptionEvent**
  The event that is announced when Mach-II encounter an exception that is not explicitly handled by your application.

You are not limited to these six properties. You can add your own properties as needed by specifying additional name / value pairs. For example, to store your datasource name as a Mach-II property, simply add this line to the properties section:

```
<property name="dsn" value="MyDatasourceName" />
```

The property you just created is then available throughout your application by calling `getProperty("dsn ")`. Complex variables and CFCs can also be stored as Mach-II properties by calling the Mach-II PropertyManager directly as opposed to adding name/value pairs to the XML configuration file.

An error is thrown when validating the configuration file, how can I fix this?

When I turn on the new configuration file validation feature in Mach-II, ColdFusion MX7 throws an error similar to this one:

```
Error validating XML file: C:\wwwroot\myApp\config\mach-ii.xml: Line 12, Column 24:
cvc-elt.1 - Cannot find the declaration of element 'mach-ii'.
```

The problem lies in Xerces2-J XML parser from Apache which CFMX7 uses to validate the configuration file against the DTD.

You can correct the error by explicitly adding a doctype declaration to configuration file:

```
<!DOCTYPE mach-ii SYSTEM
    "http://localhost:8500/MachII/mach-ii_1_1_1.dtd">
```

or

```
<!DOCTYPE mach-ii PUBLIC
    "-//Mach-II//DTD Mach-II Configuration 1.1.1//EN"
    "http://www.mach-ii.com/dtds/mach-ii_1_1_1.dtd">
```

It appears that as long as you specify a doctype declaration, even if the url to the DTD is not even valid, ColdFusion will validate the configuration file against the default DTD that is specified in your index.cfm or Application.cfc file.

Why does Mach-II appear to run really slow?

Actually, Mach-II is very fast. The most likely reason Mach-II may appear to be running slow is that debugging has been enabled on your development or production server. ColdFusion debugging creates a significant amount of overhead when debugging complex CFC models.

For more accurate average timings, you can temporarily disable debugging on the server and use the TracePlugin (first included with the 1.1.0 version of the core framework files). Instructions for configuring and using the TracePlugin are in the header comments of the TracePlugin.cfc file. If you do not know how to disable ColdFusion debugging, please see the following FAQ for more information.

Another reason Mach-II may appear slow is that the value of the `MACHII_CONFIG_MODE` variable in the index.cfm file has been set to “1” (always reload). This causes Mach-II to reload your application on every request causing excessive overhead. By setting the config mode to “0" (dynamic reload), Mach-II will only reload the application if it detects a change in your configuration file. For more information about the config mode, please see the relevant FAQ.

How do I disable ColdFusion debugging?

This is not a Mach-II specific FAQ, but relates to others FAQs.
In ColdFusion MX, debugging can be disabled by logging into the ColdFusion Administrator.

1. Click on “Debugging & Logging” in the left navigation rail to expand.
2. Click on “Debugging Settings”underneath.
3. Uncheck the “Enable Debugging” checkbox in the settings page.
4. Click on the “Submit Changes” button.

What does the `MACH_CONFIG_MODE` variable in the index.cfm file of the skeleton do?

This variables tells Mach-II when the configuration file (mach-ii.xml) should be reloaded to reconfigure your application components and model. The `MACH_CONFIG_MODE` variable takes the following values:

- **-1 (never reload)**
  This value directs Mach-II to not reload the configuration file except on the initialization of the application. This is recommended for applications in production.
- **0 (dynamic reload)**
  This value directs Mach-II to reload the configuration file automatically by checking a hash of the file’s date last modified and byte size. This is recommended for applications in development.
- **1 (always reload)**
  This value directs Mach-II to reload the configuration on each request. This option incurs severe overhead penalties and is not recommended for applications in production. This option can be useful while debugging specific bugs during development.

How can I use an Application.cfc instead of the old mach-ii.cfm bootstrapper?

Mach-II 1.1.1 added support for Application.cfc. This feature is only available in Adobe ColdFusion MX 7 or higher.

```
<cfcomponent name="Application.cfc" extends="MachII.mach-ii" output="false">

    <!---
    PROPERTIES - APPLICATION SPECIFIC
    --->
    <cfset this.name = "YourApplication" />
    <cfset this.applicationTimeout = CreateTimeSpan(30,0,0,0) />
    ...More application specific attributes...

    <!---
    PROPERTIES - MACH-II SPECIFIC
    --->
    <!---Set the path to the application's mach-ii.xml file --->
    <cfset MACHII_CONFIG_PATH = ExpandPath("/lightpost/config/mach-ii.xml") />
    <!--- Set the app key for sub-applications within a single cf-application. --->
    <cfset MACHII_APP_KEY =  "YourApplication" />
    <!--- Set the configuration mode (when to reload): -1=never, 0=dynamic, 1=always --->
    <cfset MACHII_CONFIG_MODE = -1 />
    <!--- Validate the configuration XML before parsing. Default to false. --->
    <cfset MACHII_VALIDATE_XML = TRUE />
    <!--- Set the path to the Mach-II's DTD file. --->
    <cfset MACHII_DTD_PATH = ExpandPath("./config/mach-ii_1_1.dtd") />

    <cffunction name="onApplicationStart" access="public" returntype="void"
output="false">
        <--- Add a higher request timeout if you application takes a long time to load
        <cfsetting requesttimeout="60" />
        --->
        <cfset LoadFramework() />
    </cffunction>
```

As you can see, your `Application.cfc` file must extend a the `MachII.mach-ii` CFC which is included with core of the framework. `Application.cfc` support allows you to take advantage of the application events that `Application.cfc` provides within the context of a Mach-II application. In the example code above, `onSessionStart()` calls a method in the a Session Facade (or any other model object that you define) when the session starts.

What methods are available to me if I use the Application.cfc support?

The mach-ii.cfc file give you access to two public methods and four utility methods.

Public Methods:

- `loadFramework()`
  This method loads the framework and is called in the `onApplicationStart()` application event. This method takes no arguments and is required to properly run a Mach-II application that utilizes the Appliction.cfc support.
- `handleRequest()`
  This method handles all Mach-II requests and is typically called in the `onRequestStart()` application event. This method takes no arguments and is required to properly run a Mach-II application that utilizes the Appliction.cfc support.

Utility Methods (these methods can only be called after the `LoadFramework()`method has been invoked):

- `GetProperty()`
  Gets a property from the Mach-II property manager.
- `SetProperty()`
  Sets a property to the Mach-II property manager.
- `IsPropertyDefined()`
  Checks if the desired property is currently defined in the Mach-II property manager.
- `GetAppManager()`
  Gets the Mach-II AppManager. Only useful for advanced and non-standard Application.cfc techniques.

- `shouldReloadConfig()`
  Checks if the framework will reload the configuration file on the next `HandleRequest()` call. Useful for setting higher request timeouts if you have the `MACHII_CONFIG_MODE` set to `0`.

How can I force my application to reload from a URL parameter?

You can easily force your application to reload by modifying the `index.cfm` file or the `application.cfc` file (place the code in the `onRequestStart()` event) in the skeleton of your application. Since Mach-II relies on the `MACH_CONFIG_MODE` variable to determine if the application needs to be reloaded, you can add some logic to temporarily reset the `MACH_CONFIG_MODE` variable.

Example code:

```
<!--- Check if the application needs to be reloaded via url parameter --->
<cfif StructKeyExists(URL,"reloadApp")>
    <!--- Set the configuration mode (when to reload): -1=never, 0=dynamic, 1=always --->
    <cfset MACHII_CONFIG_MODE = 1 />
<cfelse>
    <cfset MACHII_CONFIG_MODE = 0 />
</cfif>
```

The above code basically looks for a specific URL parameter (reloadApp) and sets the `MACH_CONFIG_MODE` variable to `1` (always reload).

We recommend this type of code be removed on a production application as any user could force your application to reload if they know or guess the URL parameter. Other suggestions include adding a password URL parameter (such as a UUID) as security measure.

How do I implement per session cookies in Mach-II?

Although this is not a Mach-II specific question, it is a commonly asked question. Per session cookies is implemented your application.cfm file of your application. This example implementation also works if you have J2EE sessions enabled on your ColdFusion server and allows your application to be independent of the server's session implementation. Since J2EE sessions automatically sets the jsession cookie, this cookie code is only required for servers that use the standard CFID / CFToken.

Example Code:

```
<cfapplication
    name="yourApplicationName"
    applicationTimeout="1"
    sessionmanagement="yes"
    sessionTimeout=".02084"
    setclientcookies="no" />

<!--- Sets per Session Cookies if not using J2EE sessions --->
<cfif StructKeyExists(session, "cfid") AND (NOT StructKeyExists(cookie, "cfid")
    OR NOT StructKeyExists(cookie, "cftoken"))>
    <cfcookie name="cfid" value="#session.cfid#" />
    <cfcookie name="cftoken" value="#session.cftoken#" />
</cfif>
```

What is the Event object and what does it do?

Mach-II is event-driven. The event object is at the heart of the Mach-II request lifecycle. Mach-II automatically puts all the request's form and URL variables into the event object, and your application can also programmatically put data into and pull data from the event object. The ability to access everything from a single event object is an extremely convenient way to deal with the event's data.

The event object encapsulates your Form and URL scope data into a single and unified data access object. When a request begins in a Mach-II application, the framework “copies” all the Form and URL scope data into a object called the Event objection. This makes the Event object like a unified scope. The behavior is similar to how Fusebox uses the attributes scope.

If a variable name conflict occurs when the Event object is getting populated, Mach-II allows you to set scope precedence in the configure file. This configuration setting lets you decide if the Form or URL scope takes precedence.

You can think of the Event object as a glorified encapsulated structure with methods access the data contained inside. Inside of directly referencing Form or URL scope parameters in your application, you access the Event object.

```
<cfset variables.myArg = event.getArg("someArg") />
```

The Event object can be passed around to your listeners, plugins and filters. It is best practice to pass any data needed in your model from the Event object in your listeners instead of passing in the Event object into your model. This eliminates a dependency on the Mach-II framework in your model.

How should I access the Event object in my views?

The event object in Mach-II exists under `request.event.*` and `arguments.event.*`. It is best practice to access the event object by referencing `event.*`. The reference to the event object is unqualified, which means you are not directly referencing either the request or arguments scope. Using `event.*` allows you to re-use as much code as possible - even if you change frameworks.

Explicitly scoping the event object in your views creates a dependency on a specific scope. By avoiding a scope dependency, you can easily switch to a different framework and generate a "fake" Mach-II event object for your views to use without breaking your application.

In Fusebox for example, you might want to reuse your views in Fusebox by having your fusebox.init.cfm create a structure called “event” with some UDFs inside (like getArg) that access Fusebox's way of packaging arguments – the attributes scope.

Why does the event continues to process even though I have cleared the event queue?

This happens because the event that is currently processing is taken out of the event queue when it begins processing. Since it is no longer in the event queue, clearing the event queue has not effect on the currently processing event.

What are filters and what do they do in Mach-II?

Filters in Mach-II are developer-defined CFCs the perform controller specific logic. One of the main purposes of an filter is to perform application flow control in the event-handler they are defined. This happens on an event by event basis whereas plugins execute on every event (thus cross-cutting the entire application).

For example, you may require a user of your application to be logged in before they can access a certain event. By creating a login filter, you can check if the user login status and announce your login event if they are not logged in. Another example would be if a specific event requires specific URL parameters and/or form fields to defined before the event can perform business logic. You might leverage the RequiredFieldsFilter that is bundled with the Mach-II core to check if specific event arguments exist in the event object and announce an error event of they do not exist.

How do I define, configure and use a filter?

All filters that your application required must be defined in the `<event-filters>` sections of your configuration file. Depending on the what actions a filter processes, some filters may require configuration parameters or may set parameters to default. See the documentation of the filter you are trying for general configuration setup information. Here is sample configure setup:

```
<event-filters>
    <event-filter name="Cool" type="dot.path.to.CoolFilter">
        <parameters>
            <parameter name="coolThreshold" value="10"/>
            <parameter name="invalidEventIfUncool" value="someNotSoCoolEvent"/>
        </parameters>
    </event-filter>
</event-filters>
```

Your filter may have runtime parameters that define event specific parameters or override default configuration parameters. Below is an example usage of our factious Cool filter that utilizes both a runtime parameter (`"coolEventArg"`) and overrides a default configuration (`"invalidEventIfUncool"`) parameter:

```
<event name="doSomethingCool" access="public">
    <filter name="Cool">
        <parameter name="nameOfCoolEventArg" value="cool" />
        <parameter name="invalidEventIfUncool" value="someReallyReallyNotSoCoolEvent"/>
    </filter>
</event>
```

Notice that the filter name of `"Cool"` matches up with the name that was defined in the `<event-filters>` section of the configuration file. The `"nameOfCoolEventArg"` parameter name is a required runtime parameter that tells the filter where to find the `"Cool"` in the event object. The `"invalidEventIfUncool"` parameter is going to override the default invalid event that was set as a configuration parameter. Please note that the parameter names are just for illustration and probably would best be named something else if this was a real filter.

What filters are bundled with the framework?

Mach-II 1.1.0 and 1.1.1 comes bundled with four basic filters. More complex filters can be custom written by a developer or downloaded at the Mach-II Exchange.

Available with the framework (under path `MachII.filters.*`):

- **EventArgsFilter**
  A filter for adding args to the current event being handled.
- **EventBeanFilter**
  A filter for creating and populating beans in events. *Note: This filter has been deprecated for the built-in command* `<event-bean>`.
- **PermissionsFilter**
  A filter for testing that a user has the proper permissions to execute and event.
- **RequiredFieldsFilter**
  A filter for testing that an event object contains args of required fields.

What are plugins and what do they do in Mach-II?

A plugin is a developer-defined CFC that extends the Mach-II framework. The plugin architecture provides robust extensibility for plugin point for cross-cutting event actions. This means each defined plugin point fires on every request and matching point area. Plugins are useful for executing application wide business login such as loading locale resource bundles before views, checking that the request is secured through SSL or loading specific args into the event object on each request.

There are seven versatile plugin points plus the `configure()` method that only runs when the framework loads the plugin into memory. There plugin points are:

- `PreProcess()`
  Fires at the beginning of each request lifecycle.
- `PreEvent()`
  Fires before each event. This will be called each time an event is processed.
- `PostEvent()`
  Fires after each event. This will be called each time an event is processed.

- `PreView()`
  Fires before each view is rendered. This will be called each time a view is rendered.
- `PostView()`
  Fires after each view is rendered. This will be called each time a view is rendered.
- `PostProcess()`
  Fires at the end of each request lifecycle.
- `HandleException()`
  Fires when Mach-II encounters an exception that is not handled by your model.

All plugins required that a `configure()` method be defined – even if the body of the method does no configuration. Plugins that provide interoperability between other frameworks like Reactor or ColdSpring typically only use the `configure()` method for connectivity to Mach-II and do not implement any of the seven plugin points. Mach-II introspects all plugins when they are loaded into the framework and checks which plugin points are utilized. This is save processing time on each request. Only implement plugin points that you wish to use to save processing cycles.

What plugins are bundled with the framework?

The framework ships with two plugins. One plugin is the `SimplePlugin` and demonstrates the seven plugin points. This is an example plugin only and is not intended for use is real applications. The other plugin is the `TracePlugin` which traces the execution of Mach-II events and displays the trace information on screen and/or logs it to a file. See the comment header of the `TracePlugin` for information on configuration and use. Other plugins are available in the plugins section of the Mach-II Exchange as separate downloads and are not bundled with the core framework files.

In what order are plugins called by the framework?

Mach-II calls the plugins in the order they are defined in the configuration file. The bug related to the order in which the plugin `configure()` methods were called was fixed in version 1.1.0 of Mach-II for ColdFusion.

How do I get an event object in the **`preProcess()`** method of my plugins?

An event object is not set before the `preProcess()` plugin point begins processing, however the event is already in the event queue. The following code will get the first event object for you to use in your `preProcess()` plugin point:

Example Code:

```
<cffunction name="preProcess" access="public" returntype="void" output="false">
    <cfargument name="eventContext" type="MachII.framework.EventContext"
        required="true"/>
    <!--- Peek at the first event in the event queue --->
    <cfset var event = arguments.eventContext.getNextEvent() />
    <!--- Continue with processing... --->
</cffunction>
```

Can I announce an event in the **`handleException()`** plugin point?

No, any announced events in `handleException()`plugin point will be cleared from the event queue before the exception event is processed. This is by architectural design for two compelling reasons:

1. Announcing events in this plugin point can easily lead to infinite loops or the event queue reaching the maximum number of events if an announced event throws an exception due to programmatic error.
2. Announcing events in this plugin point can lead to "interesting" event queuing.

The purpose behind this plugin point is if your plugin is design to perform a specific function and an exception occurs – it may need to stop processing whatever it is supposed to perform. For example, your plugin performs some sort of logging functionality. During the normal execution of the request it builds a logging packet and it saves the logging packet in the `postProcess()` plugin point. However, your application requirements specifies that if an exception occurs the logging packet should not be saved. You can accomplish this by setting a boolean flag in the request scope in your `handleException()` and always check if you should logging packet should be saved in your `postProcess()` plugin point. In short, `handleException()` is used as notification to the plugin that exception has occurred which allows the developer to conditionally perform certain operations based on your application requirements.

What are listeners and what do they do in Mach-II?

If you are new to Mach-II, you may be confused about what purpose listeners serve. Listeners are the “connecting tissue” between Mach-II and your model by acting as a “proxy” or “gateway”. They capture events that are defined in your configuration file, interact with your model and pass data back to the framework.

Listeners play three primary functions in Mach-II. They:

- are notified of events they have registered interest in.
- perform Mach-II specific logic that should not be located in your model such as getting and setting data to and from the event object.
- can announce new events if needed.

Listeners should perform Mach-II specific logic such as announcing new events or getting Mach-II properties. If your model contains Mach-II specific logic such as reference to the event object, you should consider refactoring that logic out of your model and into your listener. It may appear that listeners are a good place for model logic, but when you architect your application in this manner you introduce very tight coupling between Mach-II and your model. Your model should have no knowledge about the framework or dependencies on framework related logic. Be aware that reusing your model for Flex or Flash based UI becomes nearly impossible accomplish since they will not make use of the Mach-II specific logic.

All Mach-II listeners must extend the `MachII.framework.Listener` base component which is included with the core framework files.

Do I have to explicitly define an invoker when defining my listener in the configuration file?

Starting in version 1.1.0 and higher, you do not have to explicity define an invoker in the configuration file unless your listener requires a specific invoker. If you do not define an invoker for your listener, Mach-II automatically defaults to the `MachII.framwork.invokers.EventInvoker` that is bundled in the framework core.

Mach-II 1.0.10 (old syntax):

```
<listener name="someListener" type="path.to.someListener>
    <invoker type="MachII.framework.invokers.CFCInvoker_Event"/>
</listener>
```

Mach-II 1.1.0+ (new syntax):

```
<listener name="someListener" type="path.to.someListener"/>
```

Are the EventInvoker/EventArgsInvoker new and why were they added?

Yes, the EventInvoker and EventArgsInvoker are new invokers that were added to the 1.1.0 version of Mach-II. These invokers takes advantage of the new resultArg attribute in the configuration file. See the related FAQ for more information about the `contentArg` and `resultArg` attributes.

Have the **`CFCInvoker_Event`** and **`CFCInvoker_EventArgs`** invokers been depreciated?

Yes, the `CFCInvoker_Event` and `CFCInvoker_EventArgs` listener invokers have been depreciated in the 1.1.0 release of Mach-II as they do not support the new resultArg attribute. These two invokers are scheduled to be removed from the official framework core in the 2.0 release of Mach-II.

We recommend that you use the new `EventInvoker`, `EventArgsInvoker` or any custom invoker you have written so you may leverage the new `resultArg` attribute in Mach-II.

How do I use the **`resultArg`** or **`contentArg`** attributes?

The `resultArg` and `contentArg` attributes puts your result or content into the event object instead of using the request scope as a data bus.

Mach-II 1.0.10 (old syntax):

```
<notify listener="someListener" method="someMethod" resultKey="request.someData" />
```

Mach-II 1.1.0+ (new syntax):

```
<notify listener="someListener" method="someMethod" resultArg="someData" />
```

The new way put an argument called `someData` into the event object instead of `request.someData`.

Mach-II 1.0.10 (old syntax):

```
<view-page name="someView" contentKey="request.viewData"/>
```

Mach-II 1.1.0+ (new syntax):

```
<view-page name="someView" contentArg="viewData"/>
```

The new way put an argument called `viewData` into the event object instead of `request.viewData`.

Does the **`append`** attribute work when defined with the **`contentArg`** attribute?

No. This is bug that made it through our Q&A testing during the development of the 1.1.0 version of Mach-II since none of our test applications made use of `contentArg` with the `append` attribute. This bug has been fixed in the 1.1.1 alpha version of Mach-II.

The workaround for this bug is to use contentKey instead and referencing the variable inside of your view.

Is the value of the extend attribute case-sensitive when extending the framework?

Yes, however this is not a Mach-II specific problem per se. ColdFusion case-sensitivity depends on what platform you are using, file names are case-sensitive on *nix platforms while case-insensitive on Windows platforms. This can cause migration issues if you move your application from a Windows platform to a *nix platform and you are not careful about the case when specifying a framework component in the extend attribute of the cfcomponent tag. This also applies when specifying the dot delimitated path to a component you are trying to invoke.

Mach-II listeners, filters and plugins all extend the framework. When writing a listener, plugin or filter we recommend that you specify the component in the same case as they appear in your file system. Using the correct case reduces migration issues that may occur if you move your application to a different platform. Listed below are the Mach-II components in the correct case that are used to extend your application.

All listeners are extended by:
`MachII.framework.Listener`

All filters are extended by:
`MachII.framework.EventFilter`

All plugins are extended by:
`MachII.framework.Plugin`

What does the access modifier do in my event-handler?

The access modifier in a event-handler controls how your event can be accessed.

- Public events can be accessed via a form/URL or announced by the framework.
- Private events can only be announce internally by the framework using the `<announce...>` command in your configuration file or the `announceEvent(...)` method in your listeners. Private events are NOT accessible via a form/URL or redirected events initiated via the new `<redirect..>` command.

What is the default access modifier type when defining an event-handler?

If you forget to define an access type for your event-handler, it will default to `public`. Mach-II allows you to not define an access type, however it is not best practice to rely on the framework to use the default. The access attribute may take the value of `public` or `private`.

We recommend that you explicitly define the access attribute when defining your event-handler. Defining an access attribute also improves the readability of your configuration file by explicitly showing the access type. By not defining an access type, you assume other developers know what the default value the Mach-II assigns.

I call several event-handlers during a request, how can I get the original event name?

Before the 1.1.1 release, there is no method of accomplishing this task that is built into the framework. However, there is an easy way of getting the event name that kicked of the request lifecycle by calling `getRequestName()` from the event object.

Example Code (1.1.1 and higher) in a Filter, Plugin or Listener methods:

```
<cfset var requestName = arguments.event.getRequestName() />
```

Example Code (1.1.1 and higher) in a view:

```
<cfset requestName = event.getRequestName() />
```

If you are running a version of Mach-II before 1.1.1, you can get the request named in few lines of code in the `preProcess()` plugin point. Typically, many applications will have an application plugin that perform tasks like this. Just grab the initial event name before it processes and set it into event object arg for later use.

Example Code (pre-1.1.1):

```
<cffunction name="preProcess" access="public" returntype="void" output="false">
    <cfargument name="eventContext"
type="MachII.framework.EventContext"required="true"/>
    <cfset var firstEvent = arguments.eventContext.getNextEvent() />
    <cfset firstEvent.setArg("mainEvent", firstEvent.getName()) />
</cffunction>
```

What are the built-in event-handlers commands available to me in the configuration file?

As of the 1.1.1 release of Mach-II, there are eight built-in event-handler commands.

- `<announce ...>`
  Announces an event and sets it to the queue.
- `<event-arg ...>`
  Sets an arg to the current event object.
- `<event-bean ...>`
  Creates and populates a bean with data from the current event.
- `<Event-Mapping ...>`
  Setups an event mapping for an event handler.
- `<Filter ...>`
  A command to invoke a developer defined event-filter.
- `<Notify ...>`
  Notifies a listener of an event.
- `<Redirect ...>`
  Redirects a request to a new event (via cflocation). Not the same of announce.
- `<View-Page ...>`
  A command that processes a standard view.

Why does Mach-II sometimes ignore my **`<event-mapping...>`** commands?

If Mach-II has thrown an exception to an event-handler has not been defined that was mapped, it is because the mapping has yet to set in the framework. `<event-mapping...>` commands need to be defined before the listener method that uses them. Some developers adhere to a convention that all `<event-mapping...>` commands should be defined at the top of an event-handler for uniformity instead of mixing commands inside the event-handler.

Example of properly defined `<event-mapping...>` commands:

```
<event-handler event="doSomethingProcess" access="public">
    <event-mapping event="fail" mapping="doSomethingFail"/>
    <event-mapping event="pass" mapping="doSomethingPass"/>
    <notify listener="somethingListener" method="processSomething"/>
</event-handler>
```

Can I create custom views that can do process logic before rendering the view?

Yes, you can do this by creating an filter that utilizes the DisplayView() method in the EventContext object. The built-in view-page command is merely a wrapper for the DisplayView() method. This allows you to still use pre-defined views from your configuration file as well as do some sort of pre-defined logic before rendering your output. An example of a filter could be building a simple display table from a passed in query object or a filter that waits for specific event args and pulls in pre-determined javascript packages to be consumed by a layout event.

How can I get the version of Mach-II is my application running?

In the 1.1.1 release of Mach-II, you can get the version of Mach-II by calling the following code in a listener, plugin or filter:

```
getPropertyManager().getVersion()
```

If the code errors, you are running a version before 1.1.1. Also, the `TracePlugin` that ships with the 1.1.1 release and above also reports the current framework version number in the trace display.

How do the version numbers work and how can I use them?

Starting with the 1.1.1 release of Mach-II, the framework reports a four digit version number. The first three digits refer to the version of the framework and the last digit refers to the development stage level. Mach-II uses all numeric version numbers to make it simpler for developers to programmatically check versions.

The fourth digit refers to these development levels:

- 0 = Pre-alpha / Bleeding Edge Release (BER)
- 1 = Alpha
- 2 = Beta
- 3-7 = RC1-RC5
- 8 = Development & Production Stable / General Available Release (standard non-duck-typed core)
- 9 = Production-Only Stable / General Available Release (duck-typed core for better performance)

For example if the version number is 1.1.1.0, it would refer to the 1.1.1 BER. If it was 1.1.1.9, it would refer to the 1.1.1 Production-Only Stable release. You can use the version number to create a pre-install or install script that programmatically checks the version of the framework which allows you to make sure that it meets the minimum requirements for your application.

Mach-II threw an exception. What is it and what can I do?

As with any application development, bugs in your application will occur. When Mach-II encounters an exception that your application does not handle, it generates an exception object that holds pertinent information regarding the error and announces your defined exception view to display the error. Exceptions occurring in business logic should typically be caught and handled before reaching the framework level. However, the framework itself will throw some exceptions directly.

The Mach-II skeleton ships with a default exception view. However, Mach-II lets you leverage the exception object as you need. For instance, you may need access to the original cfcatch information to accurately debug your exception. Starting with the 1.1.0 version of Mach-II for ColdFusion, the original cfcatch information is available in the exception object. If you require more information than is provided in the standard Mach-II exception, you can dump the cfcatch information in your exception view by calling the `getCaughtException()` method that exists in the exception object:

Example Code:

```
<cfdump var="#event.getArg("exception").getCaughtException()#" />
```

How can I display custom exception views when an exception occurs?

Mach-II automatically announces the default exception event when it encounters an un-handled exception generated by your application. The default view is typically very useful for a developer, but very unfriendly to user of the application. You can easily customize the default exception event by calling an exception listener that announces the desired event-handler.

The most common Mach-II exceptions are `MachII.framework.EventHandlerNotDefined` (when the requested event-handler is not defined) and `MachII.framework.EventHandlerNotAccessible` (when the requested event-handler is has the access of private).

In the example below, the exception listener also checks if the application is in development mode. This is useful to have a development mode property defined in the Mach-II properties as your `exceptionListener` will need it to determine which mode it should run in.

In your configuration file, the `<event-mapping...>`'s would map to other events you have defined:

```
<event-handler event="exception" access="private">
    <event-mapping event="development" mapping="exception.development"/>
    <event-mapping event="prod.default" mapping="exception.productionDefault"/>
    <event-mapping event="prod.noEvent" mapping="exception.productionNoEventHandler"/>
    <notify listener="exceptionListener" method="doException"/>
</event-handler>
```

In your exceptionListener:

```
<cffunction name="doException" access="public" returntype="void" output="false"
    hint="Writes to log and sends email of exception.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />

    <cfset var exception = arguments.event.getArg("exception", "") />
    <cfset var type = exception.getCaughtException().type />

    <cfif getProperty("DevMode")>
        <cfset announceEvent("development", arguments.event.getArgs()) />
    <cfelse>
        <cfswitch expression="#type#">
            <!--- Announce an event-handler that says page not found --->
            <cfcase value="MachII.framework.EventHandlerNotDefined|
                        MachII.framework.EventHandlerNotAccessible" delimiters="|">
                <cfset announceEvent("prod.noEvent", arguments.event.getArgs()) />
            </cfcase>
            <cfdefaultcase>
                <!--- You might have functionality that sends you an email or
                logs the exception to a database somewhere in your model --->
                <cfset announceEvent("prod.default", arguments.event.getArgs()) />
            </cfdefaultcase>
        </cfswitch>
    </cfif>
</cffunction>
```

How can I reduce or eliminate any whitespace the framework produces?

Mach-II (as well as other frameworks) produces some white space before and after your source when viewed in a browser. This technique is especially effective when your application uses layout views as templates for displaying content. At the top of your layout view, the following line of code on the same line of your doctype:

```
<cfcontent reset="true" /><! DOCTYPE html ... />
```

At the end of your layout view put:

```
<cfsetting enablecfoutputonly="true" />
```

Viola and you got super clean output!

Was the bug known as the “White Screen of Death” fixed?

Yes. When Mach-II failed to load a view file that was specified in the configuration file, the “White Screen of Death” or blank screen occurred. Starting with the 1.1.0 release of Mach-II, the framework now throws the appropriate exception event instead of generating a “White Screen of Death” when it cannot load a view.

One facet of this bug was discovered during the QA process. It remains unfixed in the 1.1.0 version of the framework since it has been classified as non-critical and is scheduled to be fixed in the next version. If an exception occurs and the exception event announces another event which is either undefined or inaccurately defined, Mach-II announces another exception event. This causes an infinite loop to occur until the framework reaches the value defined in the required `maxEvents` framework property. This bug is easily diagnosed by viewing the trace information provided by the new Trace Plugin.

Was the **`LSDatetimeParse()`** bug fixed for Non-EN locales?

Yes. In earlier releases of the framework, the framework’s AppLoader used `parseDateTime()` to process the date modified field when determining if the configuration file should be reloaded. This failed if the ColdFusion server was set to certain non-EN locales. Later, this was changed to use `LSParseDatetimeParse()`, however this failed on certain platforms when the locale set in ColdFusion did not match the underlying locale of the server on which ColdFusion was running. This bug was fixed in the 1.1.0 release of Mach-II by implementing a solution that relies on comparing a hash of the date modified (instead of date strings) and total byte size of the configuration file.

Where do the **`AnnounceEvent()`, `SetProperty()`** and **`GetProperty()`** methods come from?

Each Listener, Filter or Plugin extends a base component specific to that component. These extended components in turn extend the `MachII.framework.BaseComponent`. The `AnnounceEvent()`, `SetProperty()` and `GetProperty()` methods exist in the BaseComponent.

How do I announce a new event from my listener, filter or plugin?

All user-defined listeners, filters and plugins extend a specific base component. The base components for Listeners, Filters and Plugins in turn extend the `MachII.framework.BaseComponent`. The BaseComponent provides basic Mach-II functionality such as `AnnounceEvent()`, `set/getParameter()` and `set/getProperty()` to name a few.

Since your listener, filter or plugin is extended, all the methods in the BaseComponent are available. To announce a new event, just call:

```
<cfset announceEvent("yourNewEventName", arguments.event.getArgs()) />
```

Can I clear the event queue inside my listener?

No, you cannot clear the event queue inside a listener. This is by design as a listener should not be able to stop the processing of the current event and should focus on interaction between the controller (Mach-II) and your model. If you need to clear the event queue inside a listener, you probably need to use a filter. Filters should focus on the interaction between the controller (Mach-II) and your event-handler.

Does Mach-II have any “reserved” variable names when I create my views?

Yes. When Mach-II displays your views, it does it in the `ViewContext.displayView()` method. Specific variables already exist in the method when your view file is included. This can cause inconsistent behavior in your views if variable name conflicts occur.

The following is the list of variable conflicts that currently exist in the 1.1.0 version of the framework:

- **append**
  `arguments.append` or *unqualified `append`
- **contentKey**
  `arguments.contentKey` or unqualified `contentKey`
- **contentArg**
  `arguments.contentArg` or unqualified `contentArg`
- **resultArg**
  `arguments.resultArg` or unqualified `resultArg`
- **event**
  `arguments.event`, `request.event` or unqualified `event`
- **viewContent**
  unqualified `viewContent` (it is a ^local variable in the method)
- **viewName**
  `arguments.viewName` or unqualified `viewName`
- **viewPath**
  unqualified `viewPath` (it is a local variable in the method)

We recommend that you explicitly scope all variables inside your views as this reduces the possibility of "reserved" variable name conflicts. Therefore, if you specify `variables.append` in your view, it would not conflict with the already defined `arguments.append` variable.

* Unqualified means you are not directly referencing a specific scope when working with a variable.
^ Local means the variables was defined with `var` inside a CFC.

How do I use the new `<redirect...>` command?

The 1.1.0 release of Mach-II introduced a new command to use in your configuration file. This command is designed to replace previous solutions such as filters that perform a redirect or using `<cflocation...>` in a listener for URL redirection (commonly known as post/redirect routines). Below is a set of example event-handlers:

Example Code:

```
<event-handler event="step1" access="public">
    <view-page name="step1" contentArg="layout.content"/>
    <announce event="doLayout" copyEventArgs="true"/>
</event-handler>
<event-handler event="step1_process" access="public">
    <event-mapping event="fail" mapping="step1"/>
    <event-mapping event="pass" mapping="step2_redirect"/>
    <notify listener="myListener" method="processStep"/>
</event-handler>
<event-handler event="step2_redirect" access="private">
    <redirect event="step2" args="list,of,simple,args" />
</event-handler>
<event-handler event="step2" access="public">
    <view-page name="step2" contentArg="layout.content"/>
    <announce event="doLayout" copyEventArgs="true"/>
</event-handler>
```

Description of the chain of events:

1. The user is presented with a form on `step1` which is filled out and submitted to the `step1_process` event-handler.
2. This `step1_process` event-handler performs logic via the `myListener.processStep`. Based on the results, the listner method announces a `pass` or `fail` event (which are commonly mapped for code reuse of a listener method).

If the `pass` event is announced:

1. The developer wants the user's URL in their browser to reflect the new page s/he is announcing, so the developer announces the `pass` event. This `private` access type event redirects to the new event. The `<redirect...>` command will automatically append any simple-type event arguments to the URL. Just specify a list of event arguments to append in the `args` attribute. Please remember that the event arguments you want to append must

currently be in the even object and be of simple data type (no structs, arrays, etc.).
2. The redirect occurs, the URL changes and the user is presented with a form/view specified in `step2`.

If the `fail` event is announced:

1. Since the data given to listener failed, the listener announces the `fail` event and `step1` is announced.
2. Since submitting the form to `step1_process` event-handler does not update the URL, the URL remains at `step1_process`. Depending on your application architecture, a `<redirect...>` may be needed for the `fail` event.

Does the `<redirect...>` command support SES URLs?

No, the `<redirect...>` command does not support SES URLs. This issue has been noted for discussion when Team Mach-II plans the next release of Mach-II.

If your application architecture requires SES URLs, the best option is write a filter that matches the format of your SES URLs.

How to I "persist"complex-type event arguments across/during a `<redirect...>` command?

Mach-II can only append simple-type event arguments (strings) in a URL. To persist complex-type (structures, arrays, components, etc.) event arguments across a `<redirect...>` command, a special filter/plugin combination is used.

The RedirectPersistFilter/Plugin package will "save" and "inject" complex event data during a redirect (cflocation). The package includes debugging and cleanup for orphaned persist event data that does not get picked up.
Refer to Peter J. Farrell's RedirectPersistFilter/Plugin combination at http://blog.maestropublishing.com for more information or to download the Beta package.

About The Author:

Hailing from the frigid tundra of Minnesota, Peter J. Farrell has a Bachelor of Music degree from the Johns Hopkins University in Baltimore, Maryland. While studying music, Peter took his life-long interest with computers to a new level and started learning about web development technologies. He has been working with ColdFusion for since 2001 and is the lead developer for the Mach-II framework.

Peter owns Maestro Publishing, LLC, a small development firm specializing in web solutions for non-profits and small businesses. He currently is the lead developer for GreatBizTools, LLC, a human resources consulting firm. He and his wife, Allyson, live together in Saint Paul, Minnesota.

You can contact Peter at peter@mach-ii.com.

Acknowledgments:

Thank you to Matt Woodward for contributing to this FAQs document.

Copyright & License:

Future Questions:

How do I redirect a user to the original requested event after logging in to a login required event?

How do you integrate ColdSpring with Mach-II?

How do you integrate Reactor integrate well with Mach-II?

How does the Mach-II event queue work?

Can I dynamically add event-mappings in filters and plugins?