Mach-II Application Configuration

For Mach-II for PHP version 1.0

Configuring a Mach-II application involves editing a few files: mach-ii.xml and, possibly, index.php and/or mach-ii.php.

For each sample application provided on php.mach-ii.com the mach-ii.xml file is in a sub-directory called config (i.e. ContactManager/config/mach-ii.xml).

The Mach-II framework code is a toolkit (code library) that can be installed once and used from the same location for each Mach-II application. Currently the framework files (the MachII directory and everything in it) need to be installed in the document root of the web server, or equivalently mapped via PHP's `include_path` directive. Absolute paths (i.e. MachII.framework.AppManager) are used for referring to objects.

index.php and mach-ii.php

The mach-ii.php file is the entry point for a request into the Mach-II framework. It is usually called from index.php. There are three important parameters for the application and framework that should be set in index.php. They are `$MACHII_CONFIG_PATH`, `$MACHII_CONFIG_MODE`, and `$MACHII_APP_KEY`.

`$MACHII_CONFIG_PATH` needs to be set to the relative or absolute path to the mach-ii.xml configuration file. In environments where the current directory (".") is not in the `include_path`, the absolute path may be obtained in PHP by using the built-in `realpath()` function passing a relative path to the mach-ii.xml file.

`$MACHII_CONFIG_MODE` needs to be set to a value of `MACHII_CONFIG_RELOAD_NEVER`, `MACHII_CONFIG_RELOAD_DYNAMIC`, or `MACHII_CONFIG_RELOAD_ALWAYS`. These constants represent when the configuration file (mach-ii.xml) will be reloaded to reconfigure the application's framework components. Prior to inclusion of MachII.php these values should be set as strings, afterward they may be used as constants.

- `MACHII_CONFIG_RELOAD_NEVER`: do not reload the configuration file (recommended for production deployments)
- `MACHII_CONFIG_RELOAD_DYNAMIC`: reload the configuration file whenever the mach-ii.xml file is updated by automatically checking the file's date last modified (recommended for development)
- `MACHII_CONFIG_RELOAD_ALWAYS`: reload the configuration for each request (will result in significant performance penalties and is not recommended for production)

Each time the configuration is reloaded the application framework is reinstantiated. This means the mach-ii.xml file is reread from file, reparsed and the framework and application object instances are created. This also means any state information held in framework components (such as Listeners) will be reset if not persisted.

`$MACHII_APP_KEY` needs to be set to a value that is unique amongst any Mach-II sub-applications that are defined for the same application (site). This property enables multiple Mach-II applications, each independent of each other, to operate under one application. By

default the app-key value is set to the absolute path to the folder in which the application is stored.

Example configuration code from a *index.php* file (example values represent the defaults that will be used by mach-ii.php if they are not set in index.php):

```
/**
 * Set the app key for the application.
 */
$MACHII_APP_KEY = realpath('.');

/**
 * Set the path to the application's config file.
 */
$MACHII_CONFIG_PATH = realpath('./config/mach-ii.xml');

/**
 * Set the configuration mode.
 * MACHII_CONFIG_RELOAD_NEVER   = never reload the config file
 * MACHII_CONFIG_RELOAD_DYNAMIC = reload if it has changed since last
load
 * MACHII_CONFIG_RELOAD_ALWAYS  = reload on each request
 */
$MACHII_CONFIG_MODE = 'MACHII_CONFIG_RELOAD_DYNAMIC';

/**
 * Set the default cache and cache options.
 */
$MACHII_CACHE = 'MachII.cache.PhpSession';
$MACHII_CACHE_OPTIONS = array();
```

mach-ii.xml

The mach-ii.xml file is the main configuration file for a Mach-II application. The XML configuration file has an accompanying DTD currently called mach-ii_1_0.dtd that is included with the framework code files when downloaded.

<mach-ii> : [version]
The root element of a Mach-II configuration file is a <mach-ii> element with an optional version attribute. The version attribute should specify the version of Mach-II the configuration file is supposed to use.

The Mach-II XML configuration file (config file) has six main elements for defining a Mach-II application and its components.

Example code from a *mach-ii.xml* file:

```
<mach-ii version="1.0">
   <!-- PROPERTIES -->
   <properties>
      ...
   </properties>

   <!-- LISTENERS -->
   <listeners>
      ...
   </listeners>
```

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

   <!-- EVENT-HANDLERS -->
   <event-handlers>
      ...
   </event-handlers>

   <!-- PAGE-VIEWS -->
   <page-views>
      ...
   </page-views>

   <!-- PLUGINS -->
   <plugins>
      ...
   </plugins>
</mach-ii>
```

<properties>

<property> : name, value

The <properties> element allows you to configure the framework instance for an application. Properties need to be defined in each config file for the framework to operate properly. These properties are each defined as a <property> element under the <properties> of the config file.

The properties to be defined:

- defaultEvent – the event for the framework to handle if one is not specified in a new request
- exceptionEvent – the event for the framework to handle if there's an unhandled exception in the application
- maxEvents – the maximum number of events for the framework to process on a single request
- applicationRoot – the path to the application's folder absolute or relative to the application's index.php (Prepended to view paths.)
- eventParameter – the name of the request parameter (form or url) that will define the event for the framework to handle
- parameterPrecedence – form | url (or _POST | _GET), which to favor for conflicting request parameters
- locale – ISO locale code, default is “en”.

Example code from a *mach-ii.xml* file:

```
<mach-ii>

   <!-- PROPERTIES -->
   <properties>
      <property name="defaultEvent" value="showHome" />
      <property name="exceptionEvent" value="exception" />
      <property name="applicationRoot" value="." />
      <property name="eventParameter" value="event" />
      <property name="parameterPrecedence" value="form" />
```

```
      <property name="maxEvents" value="10" />
   </properties>
   . . .
</mach-ii>
```

<listeners>

<listener> : name, type

The <listeners> element allows you to register listeners for an application. The name attribute specified for a <listener> element should be unique amongst listeners. The type attribute specified should be a path to an object.

<invoker> : type

Also for each listener an invoker must be specified with a child <invoker> element. The type attribute for the <invoker> element should be a path to an object.

Listeners can also have a <parameters> and <parameter> elements specified to define additional configuration information.

Example code from a *mach-ii.xml* file:

```
<mach-ii>
   . . .
   <!-- LISTENERS -->
   <listeners>
      <listener name="ContactManager"
                type="ContactManager.model.ContactManager">
         <invoker type="MachII.framework.invokers.ObjectEvent" />
         <parameters>
            <parameter name="param1" value="value1" />
         </parameters>
      </listener>
   </listeners>
   . . .
</mach-ii>
```

<event-filters>

<event-filter> : name, type

The <event-filters> element allows you to register event filters for an application. The name attribute specified for an <event-filter> element should be unique amongst event filters. The type attribute specified should be a path to an object.

Event-Filters can also have a <parameters> and <parameter> elements specified to define additional configuration information.

Example code from a *mach-ii.xml* file:

```
<mach-ii>
   . . .
   <!-- EVENT-FILTERS -->
   <event-filters>
      <event-filter name="simpleFilter" type="Test.filters.SimpleFilter ">
         <parameters>
            <parameter name="param1" value="value1" />
         </parameters>
```

```
      </event-filter>
   </event-filters>
   . . .
</mach-ii>
```

<event-handlers>
<event-handler> : event, [access]

The <event-handlers> element allows you to define how to handle an application's events. The event attribute specified for an <event-handler> element should be a unique event name. The access attribute specified for an <event-handler> element is optional, and if defined, must have a value of public or private. Event-handlers are assumed to be public by default.

The access attribute specified for an <event-handler> element determines when an event can be announced. Events specified in a request parameter must have access specified as public. Events announced via listeners in the same application can be public or private.

Example code from a *mach-ii.xml* file:

```
<mach-ii>
   . . .
   <!-- EVENT-HANDLERS -->
   <event-handlers>
      <event-handler event="showHome" access="public">
         . . .
      </event-handler>
   </event-handlers>
   . . .
</mach-ii>
```

Each <event-handler> can specify any of the following elements in its body.

<announce> : event, [copyEventArgs]
An <announce> element will announce a new event. The copyEventArgs attribute specifies whether or not to copy the event-args of the current event being handled.

Example code from a *mach-ii.xml* file:

```
<event-handler event="editUser" access="public">
   <announce event="showEditUserForm" copyEventArgs="true" />
</event-handler>
```

<event-arg> : name, [value], [variable]
An <event-arg> element will set an argument in the current event being handled. The name attribute is the name of the arg to create in the event. The optional value attribute may specify a literal value for the arg. The optional variable attribute may specify an in scope variable to set as the value of the arg. Either the value or variable attribute should be set in an <event-arg> element, not both. If the variable specified is not defined a blank string is set as the value.

Example code from a *mach-ii.xml* file:

```
<event-handler event="editUser" access="public">
   <event-arg name="submitFormEvent" value="editUser" />
   <event-arg name="user" variable="_SESSION['user']" />
   ...
</event-handler>
```

<event-mapping> : event, mapping
An <event-mapping> element will set a temporary mapping in the framework that will map one event (if announced) to another. The mapping exists only for the life of the current event being handled.

Example code from a *mach-ii.xml* file:
```
<event-handler event="createContact" access="public">
   <event-mapping event="contactAdded" mapping="showContact" />
   ...
</event-handler>
```

<event-bean> : name, type, [fields]
An <event-bean> element will create a bean inside the current event. The name of the bean is specified with the name attribute. The type of the bean to create is a fully qualified CFC path specified by the type attribute. The optional fields attribute specifies which fields of the event to use to call setters on the bean.

Example code from a *mach-ii.xml* file:
```
<event-handler event="createContact" access="public">
   <event-bean name="address" type="model.Address"
fields="street,city,state,zip" />
   ...
</event-handler>
```

<filter> : name
A <filter> element allows you apply a filter to the event being handled. The name attribute needs to reference a specified <event-filter> name. One or more <parameter> elements may also be specified for a filter.

Example code from a *mach-ii.xml* file:
```
<event-handler event="editContact" access="public">
   <filter name="loggedInFilter">
      <parameter name="invalidEvent" value="showLoginForm" />
   </filter>
</event-handler>
```

<notify> : listener, method, [resultKey]
A <notify> element will invoke a listener's method using the current event and its arguments. The listener attribute is the name of a registered listener. The method attributes is a method of the listener to invoke. The optional resultKey argument is a variable name to store the returned result of the method call in.

Example code from a *mach-ii.xml* file:
```
<event-handler event="createContact" access="public">
   <notify listener="loginListener" method="login"
      resultKey="_SESSION['isLoggedIn']" />
</event-handler>
```

<view-page> : name, [contentKey], [append]
A <view-page> element will invoke a page-view and optionally store the generated content instead of outputting it. The name attribute is that of a registered page-view. The optional contentKey argument, if present, stores the generated page-view content in the specified variable instead of outputting. The optional append argument defaults

to false and determines whether the contentKey value is appended to (or overwrites the old value.

Example code from a *mach-ii.xml* file:

```
<event-handler event="showContactForm" access="public"
      append=”true”>
   <view-page name="contactForm" />
</event-handler>
```

<page-views>

<page-view> : name, page

The <page-views> element allows you to register page-views for an application. The name attribute specified for a <page-view> element should be unique amongst page-views. The page attribute specified should be a path to a page relative to the applicationRoot defined in <properties>. The <property> applicationRoot will be prepended to this value.

Example code from a *mach-ii.xml* file:

```
<mach-ii>
   . . .
   <!-- PAGE-VIEWS -->
   <page-views>
      <page-view name="mainTemplate" page="/views/mainTemplate.php" />
   </page-views>
   . . .
</mach-ii>
```

<plugins>

<plugin> : name, type

The <plugin> element allows you to register plugins for an application. Plugins will be executed in the order they are declared. The name attribute specified for a <plugin> element should be unique amongst plugins. The type attribute specified should be a full path to a PHP class file.

Plugins can also have a <parameters> and <parameter> elements specified to define additional configuration information.

Example code from a *mach-ii.xml* file:

```
<mach-ii>
   . . .
   <!-- PLUGINS -->
   <plugins>
      <plugin name="simplePlugin" type="Test.plugins.SimplePlugin">
         <parameters>
            <parameter name="param1" value="value1" />
         </parameters>
      </plugin>
   </plugins>
   . . .
</mach-ii>
```

Appendix A

Fully Qualified Object Paths

Unlike some other object-oriented languages PHP does not directly associate paths, filenames and class names. This means that instantiating an object is a two step process: First, include a class definition file, then create an object instance of that class. This scheme is very flexible but can also result in name-space conflicts.

Class names in mach-ii.xml are specified via a Java-style notation and translated by MachII_util_ClassLoader at include-time into a path and filename and a class name. This notation is familiar to many, allows the XML to be a little more concise and is consistent with the ColdFusion Mach-II reference implementation.

Mach-II for PHP has borrowed the filename and class naming scheme from the PEAR project. (http://pear.php.net) Class names are directly tied to the file path and filename.

For example:

```
<plugin name="simplePlugin" type="Test.plugins.SimplePlugin">
```

The `type` will be translated into the path and filename “Test/plugins/SimplePlugin.php” and make available the class “Test_plugins_SimplePlugin”.

A simple mechanism to override this default behavior and allow disassociated paths and class names is also provided. The syntax is documented in the API docs for MachII_util_ClassLoader.