

Mach-II 1.1.1 - Stable

Mach-II 1.1.1 - Stable

Table of Contents

.....	1
mach-ii	2
filters	7
EventArgsFilter	8
EventBeanFilter	11
PermissionsFilter	16
RequiredFieldsFilter	22
framework	26
AppFactory	27
AppLoader	31
AppManager	39
BaseComponent	48
Event	58
EventCommand	67
EventContext	71
EventFilter	96
EventHandler	98
EventManager	102
FilterManager	113
Listener	119
ListenerInvoker	122
ListenerManager	124
Plugin	131
PluginManager	138
PropertyManager	152
RequestHandler	160
ViewContext	165
ViewManager	173
framework.commands	178
AnnounceCommand	179
EventArgsCommand	183
EventBeanCommand	190
EventMappingCommand	197
FilterCommand	201

NotifyCommand	205
RedirectCommand	212
ViewPageCommand	219
framework.invokers	226
CFCInvoker_Event	227
CFCInvoker_EventArgs	230
EventArgsInvoker	233
EventInvoker	236
plugins	239
SimplePlugin	240
TracePlugin	245
util	273
BeanUtil	274
Exception	280
Queue	289
SizedQueue	294
XmlValidationException	298

mach-ii

Package: MachII

Base component for Application.cfc integration

Method Summary

public AppManager	getAppManager() Get the Mach-II AppManager. Not available until loadFramework has been called.
public any	getProperty(string propertyName, [any defaultValue=""]) Returns the property value by name. If the property is not defined, and a default value is passed, it will be returned. If the property and a default value are both not defined then an exception is thrown. Not available until loadFramework() has been called.
public void	handleRequest() Handles a Mach-II request. Recommend to call in onRequestStart() event.
public boolean	isPropertyDefined(string propertyName) Checks if property name is defined in the properties. Not available until loadFramework() has been called.
public void	loadFramework() Loads the framework. Only call in onApplicationStart() event.
public void	setProperty(string propertyName, any propertyValue) Sets the property value by name. Not available until loadFramework() has been called.
public boolean	shouldReloadConfig() Returns if the config should be dynamically reloaded.

Method Detail

getManager

```
public AppManager getManager( )
```

Get the Mach-II AppManager. Not available until loadFramework has been called.

Parameters:

Code:

```
<cffunction name="getManager" access="public" returnType="MachII.framework.AppManager" output="false"
    hint="Get the Mach-II AppManager. Not available until loadFramework has been called.">
    <cfreturn application[MACHII_APP_KEY].appLoader.getManager() />
</cffunction>
```

getProperty

```
public any getProperty( string propertyName, [any defaultValue=""] )
```

Returns the property value by name. If the property is not defined, and a default value is passed, it will be returned. If the property and a default value are both not defined then an exception is thrown. Not available until loadFramework() has been called.

Parameters:

```
string propertyName
[any defaultValue=""]
```

Code:

```
<cffunction name="getProperty" access="public" returnType="any" output="false"
    hint="Returns the property value by name. If the property is not defined, and a default value is passed, it will
    <cfargument name="propertyName" type="string" required="true" />
    <cfargument name="defaultValue" type="any" required="false" default="" />
    <cfreturn getManager().getPropertyManager().getProperty(propertyName, arguments.defaultValue) />
</cffunction>
```

handleRequest

public void handleRequest()

Handles a Mach-II request. Recommend to call in onRequestStart() event.

Parameters:

Code:

```
<cffunction name="handleRequest" access="public" returntype="void" output="true"
  hint="Handles a Mach-II request. Recommend to call in onRequestStart() event.">

  <cfif StructKeyExists(request,"MachIIConfigMode")>
    <cfset MACHII_CONFIG_MODE = request.MachIIConfigMode />
  </cfif>

  <cfif NOT StructKeyExists(application, MACHII_APP_KEY)>
    <cflock name="application_#MACHII_APP_KEY#_reload" type="exclusive" timeout="120">
      <cfif NOT StructKeyExists(application, MACHII_APP_KEY)>
        <cfset loadFramework() />
      </cfif>
    </cflock>
  </cfif>

  <cfif MACHII_CONFIG_MODE EQ 1 AND NOT StructKeyExists(request, "MachIIReload")>
    <cflock name="application_#MACHII_APP_KEY#_reload" type="exclusive" timeout="120">
      <cfset application[MACHII_APP_KEY].appLoader.reloadConfig(MACHII_VALIDATE_XML, MACHII_VERSION) />
    </cflock>
  <cfelseif MACHII_CONFIG_MODE EQ 0 AND application[MACHII_APP_KEY].appLoader.shouldReloadConfig(>
    <cflock name="application_#MACHII_APP_KEY#_reload" type="exclusive" timeout="120">
      <cfset application[MACHII_APP_KEY].appLoader.reloadConfig(MACHII_VALIDATE_XML, MACHII_VERSION) />
    </cflock>
  <cfelseif MACHII_CONFIG_MODE EQ -1>

  </cfif>

  <cfset application[MACHII_APP_KEY].appLoader.getAppManager().getRequestHandler().handleRequest() />
</cffunction>
```

isPropertyDefined

public boolean isPropertyDefined(string propertyName)

Checks if property name is defined in the properties. Not available until loadFramework() has been called.

Parameters:

string propertyName

Code:

```
<cffunction name="isPropertyDefined" access="public" returntype="boolean" output="false"
    hint="Checks if property name is defined in the properties. Not available until loadFramework() has been called."
    <cfargument name="propertyName" type="string" required="true"/>
    <cfreturn getAppManager().getPropertyManager().isPropertyDefined(arguments.propertyName) />
</cffunction>
```

loadFramework

public void loadFramework()

Loads the framework. Only call in onApplicationStart() event.

Parameters:

Code:

```
<cffunction name="loadFramework" access="public" returntype="void" output="false"
    hint="Loads the framework. Only call in onApplicationStart() event.">
    <cfset application[MACHII_APP_KEY] = StructNew() />
    <cfset application[MACHII_APP_KEY].appLoader = CreateObject('component', 'MachII.framework.AppLoader').init(MACHII_APP_KEY) />
    <cfset request.MachIIReload = FALSE />
</cffunction>
```

setProperty

public void setProperty(string propertyName, any propertyValue)

Sets the property value by name. Not available until loadFramework() has been called.

Parameters:

string propertyName
any propertyValue

Code:

```
<cffunction name="setProperty" access="public" returntype="void" output="false"
    hint="Sets the property value by name. Not available until loadFramework() has been called.">
    <cfargument name="propertyName" type="string" required="true" />
    <cfargument name="propertyValue" type="any" required="true" />
    <cfset getAppManager().getPropertyManager().setProperty(arguments.propertyName, arguments.propertyValue) /> />
</cffunction>
```

shouldReloadConfig

public boolean shouldReloadConfig()

Returns if the config should be dynamically reloaded.

Parameters:

Code:

```
<cffunction name="shouldReloadConfig" access="public" returntype="boolean" output="false"
    hint="Returns if the config should be dynamically reloaded.">
    <cfreturn application[MACHII_APP_KEY].appLoader.shouldReloadConfig() />
</cffunction>
```

filters

EventArgsFilter

Package: MachII.filters

Inherits from: framework.BaseComponent < framework.EventFilter

An EventFilter for adding args to the current event being handled.

Method Summary

public void	configure() This configure method does nothing.
public boolean	filterEvent(Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"]) Runs the filter event.

Methods inherited from framework.EventFilter: init

Methods inherited from framework.BaseComponent: isParameterDefined , setPropertyManager , setParameters , getPropertyManager , announceEvent , getAppManager , setAppManager , hasParameter , getParameter , getProperty , getParameters , setProperty , setParameter

Method Detail

configure

public void configure()

This configure method does nothing.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void" output="false"
    hint="This configure method does nothing.">

</cffunction>
```

filterEvent

```
public boolean filterEvent( Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"] )
```

Runs the filter event.

Parameters:

Event event
EventContext eventContext
[struct paramArgs="#StructNew()#"]

Code:

```
<cffunction name="filterEvent" access="public" returntype="boolean"
    hint="Runs the filter event.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfargument name="paramArgs" type="struct" required="false" default="#StructNew()#" />

    <cfset var paramArgKeys = StructKeyArray(arguments.paramArgs) />
    <cfset var i = 0 />
    <cfset var argName = 0 />

    <cfloop index="i" from="1" to="#ArrayLen(paramArgKeys)#">
        <cfset argName = paramArgKeys[i] />
        <cfset arguments.event.setArg(argName, paramArgs[argName]) />
    </cfloop>

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


EventBeanFilter

Package: MachII.filters

Inherits from: framework.BaseComponent < framework.EventFilter

A robust EventFilter for creating and populating beans in events.

Method Summary

public void	configure() Configures the filter.
public boolean	filterEvent(Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"]) Runs the filter event.
private BeanUtil	getBeanUtil()
private void	setBeanUtil(BeanUtil beanUtil)
private void	throwUsageException()

Methods inherited from framework.EventFilter: init

Methods inherited from framework.BaseComponent: isParameterDefined , setPropertyManager , setParameters , getPropertyManager , announceEvent , getAppManager , setAppManager , hasParameter , getParameter , getProperty , getParameters , setProperty , setParameter

Method Detail

configure

```
public void configure( )
```

Configures the filter.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void" output="false"
    hint="Configures the filter.">
    <cfset setBeanUtil( CreateObject('component','MachII.util.BeanUtil') ) />
</cffunction>
```

filterEvent

```
public boolean filterEvent( Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"] )
```

Runs the filter event.

Parameters:

Event event

EventContext eventContext

[struct paramArgs="#StructNew()#"]

Code:

```
<cffunction name="filterEvent" access="public" returntype="boolean"
    hint="Runs the filter event.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfargument name="paramArgs" type="struct" required="false" default="#StructNew()#" />

    <cfset var bean = "" />
    <cfset var beanName = "" />
    <cfset var beanType = "" />
    <cfset var beanFields = "" />
    <cfset var isFieldsDefined = false />
```

```
<cfif StructKeyExists(arguments.paramArgs, this.BEAN_NAME_PARAM)>
    <cfset beanName = paramArgs[this.BEAN_NAME_PARAM] />
<cfelseif isParameterDefined(this.BEAN_NAME_PARAM)>
    <cfset beanName = getParameter(this.BEAN_NAME_PARAM) />
</cfif>

<cfif StructKeyExists(arguments.paramArgs, this.BEAN_TYPE_PARAM)>
    <cfset beanType = paramArgs[this.BEAN_TYPE_PARAM] />
<cfelseif isParameterDefined(this.BEAN_TYPE_PARAM)>
    <cfset beanType = getParameter(this.BEAN_TYPE_PARAM) />
</cfif>

<cfif StructKeyExists(arguments.paramArgs, this.BEAN_FIELDS_PARAM)>
    <cfset beanFields = paramArgs[this.BEAN_FIELDS_PARAM] />
    <cfset isFieldsDefined = true />
<cfelseif isParameterDefined(this.BEAN_FIELDS_PARAM)>
    <cfset beanFields = getParameter(this.BEAN_FIELDS_PARAM) />
    <cfset isFieldsDefined = true />
<cfelse>
    <cfset isFieldsDefined = false />
</cfif>

<cfif beanName EQ '' OR beanType EQ ''>
    <cfset throwUsageException() />
</cfif>

<cfif isFieldsDefined>
    <cfset bean = getBeanUtil().createBean(beanType) />
    <cfset getBeanUtil().setBeanFields(bean, beanFields, arguments.event.getArgs()) />
<cfelse>
    <cfset bean = getBeanUtil().createBean(beanType, arguments.event.getArgs()) />
</cfif>

<cfset arguments.event.setArg(beanName, bean, beanType) />

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

getBeanUtil

private BeanUtil getBeanUtil()

Parameters:

Code:

```
<cffunction name="getBeanUtil" access="private" returntype="MachII.util.BeanUtil" output="false">
    <cfreturn variables.beanUtil />
</cffunction>
```

setBeanUtil

private void setBeanUtil(BeanUtil beanUtil)

Parameters:

BeanUtil beanUtil

Code:

```
<cffunction name="setBeanUtil" access="private" returntype="void" output="false">
    <cfargument name="beanUtil" type="MachII.util.BeanUtil" required="true" />
    <cfset variables.beanUtil = arguments.beanUtil />
</cffunction>
```

throwUsageException

private void throwUsageException()

Parameters:

Code:

```
<cffunction name="throwUsageException" access="private" returntype="void" output="false">
    <cfset var throwMsg = "EventBeanFilter requires the following usage parameters: " & this.BEAN_NAME_PARAM & ", " &
    <cfthrow message="#throwMsg#" />
</cffunction>
```

PermissionsFilter

Package: MachII.filters

Inherits from: framework.BaseComponent < framework.EventFilter

A robust EventFilter for testing that a user has the proper permissions to execute and event.

Method Summary

public void	configure() This configure does nothing.
public boolean	filterEvent(Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"]) Runs the filter event.
public any	getUserPermissions() Checks if user permissions is defined.
private void	throwUsageException() Throws an usage exception.
public boolean	validatePermissions(string requiredPermissions, string userPermissions) Validates if required permissions exists in the user's permissions.

Methods inherited from framework.EventFilter: init

Methods inherited from framework.BaseComponent: isParameterDefined , setPropertyManager , setParameters , getPropertyManager , announceEvent , getAppManager , setAppManager , hasParameter , getParameter , getProperty , getParameters , setProperty , setParameter

Method Detail**configure**

```
public void configure( )
```

This configure does nothing.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void" output="false"
    hint="This configure does nothing.">

</cffunction>
```

filterEvent

```
public boolean filterEvent( Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"] )
```

Runs the filter event.

Parameters:

Event event

EventContext eventContext

[struct paramArgs="#StructNew()#"]

Code:

```
<cffunction name="filterEvent" access="public" returntype="boolean" output="true"
    hint="Runs the filter event.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfargument name="paramArgs" type="struct" required="false" default="#StructNew()#" />
```

```
<cfset var isContinue = true />
<cfset var requiredPermissions = '' />
<cfset var invalidEvent = '' />
<cfset var invalidMessage = '' />
<cfset var clearEventQueue = '' />
<cfset var userPermissions = '' />
<cfset var newEventArgs = 0 />

<cfif StructKeyExists(arguments.paramArgs,this.REQUIRED_PERMISSIONS_PARAM)>
    <cfset requiredPermissions = paramArgs[this.REQUIRED_PERMISSIONS_PARAM] />
<cfelse>
    <cfset requiredPermissions = getParameter(this.REQUIRED_PERMISSIONS_PARAM,'') />
</cfif>

<cfif StructKeyExists(arguments.paramArgs,this.INVALID_EVENT_PARAM)>
    <cfset invalidEvent = paramArgs[this.INVALID_EVENT_PARAM] />
<cfelse>
    <cfset invalidEvent = getParameter(this.INVALID_EVENT_PARAM,'') />
</cfif>

<cfif StructKeyExists(arguments.paramArgs,this.INVALID_MESSAGE_PARAM)>
    <cfset invalidMessage = paramArgs[this.INVALID_MESSAGE_PARAM] />
<cfelse>
    <cfset invalidMessage = getParameter(this.INVALID_MESSAGE_PARAM,'') />
</cfif>

<cfif StructKeyExists(arguments.paramArgs,this.CLEAR_EVENT_QUEUE_PARAM)>
    <cfset clearEventQueue = paramArgs[this.CLEAR_EVENT_QUEUE_PARAM] />
<cfelse>
    <cfset clearEventQueue = getParameter(this.CLEAR_EVENT_QUEUE_PARAM,true) />
</cfif>

<cfif NOT (requiredPermissions EQ '' OR invalidEvent EQ '')>
    <cfset userPermissions = getUserPermissions() />
    <cfset isContinue = validatePermissions(requiredPermissions, userPermissions) />
<cfelse>
    <cfset throwUsageException() />
</cfif>

<cfif isContinue>
```

```
        <cfreturn true />
    </cfelse>

    <cfif clearEventQueue>
        <cfset arguments.eventContext.clearEventQueue() />
    </cfif>

    <cfset newEventArgs = arguments.event.getArgs() />
    <cfset newEventArgs[this.INVALID_MESSAGE_PARAM] = invalidMessage />
    <cfset arguments.eventContext.announceEvent(invalidEvent, newEventArgs) />

    <cfreturn false />
</cfif>
</cffunction>
```

getUserPermissions

public any getUserPermissions()

Checks if user permissions is defined.

Parameters:

Code:

```
<cffunction name="getUserPermissions" access="public" returntype="any"
    hint="Checks if user permissions is defined.">

    <cfif IsDefined('session.permissions')>
        <cfreturn session.permissions />
    </cfif>
    <cfreturn '' />
</cffunction>
```

throwUsageException

```
private void throwUsageException( )
```

Throws an usage exception.

Parameters:

Code:

```
<cffunction name="throwUsageException" access="private" returntype="void" output="false"
    hint="Throws an usage exception.">
    <cfset var throwMsg = "PermissionsFilter requires the following usage parameters: " & this.REQUIRED_PERMISSIONS_I
    <cfthrow message="#throwMsg#" />
</cffunction>
```

validatePermissions

```
public boolean validatePermissions( string requiredPermissions, string userPermissions )
```

Validates if required permissions exists in the user's permissions.

Parameters:

string requiredPermissions

string userPermissions

Code:

```
<cffunction name="validatePermissions" access="public" returntype="boolean"
    hint="Validates if required permissions exists in the user's permissions.">
    <cfargument name="requiredPermissions" type="string" required="true" />
    <cfargument name="userPermissions" type="string" required="true" />

    <cfset var isValidated = true />
    <cfset var permission = 0 />

    <cfloop index="permission" list="#requiredPermissions#" delimiters=",">
        <cfif NOT ListContainsNoCase(arguments.userPermissions,permission)>
            <cfset isValidated = false />
        </cfif>
    </cfloop>
</cffunction>
```

```
        <cfreturn isValidated />
    </cffunction>
```

RequiredFieldsFilter

Package: MachII.filters

Inherits from: framework.BaseComponent < framework.EventFilter

An EventFilter for testing that an event's args contain a list of required fields.

Method Summary

public void	configure() This configure does nothing.
public boolean	filterEvent(Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"]) Runs the filter event.
private void	throwUsageException() Throws an usage exception.

Methods inherited from framework.EventFilter: init

Methods inherited from framework.BaseComponent: isParameterDefined , setPropertyManager , setParameters , getPropertyManager , announceEvent , getAppManager , setAppManager , hasParameter , getParameter , getProperty , getParameters , setProperty , setParameter

Method Detail

configure

```
public void configure( )
```

This configure does nothing.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void" output="false"
    hint="This configure does nothing.">

</cffunction>
```

filterEvent

```
public boolean filterEvent( Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"] )
```

Runs the filter event.

Parameters:

Event event

EventContext eventContext

[struct paramArgs="#StructNew()#"]

Code:

```
<cffunction name="filterEvent" access="public" returntype="boolean"
    hint="Runs the filter event.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfargument name="paramArgs" type="struct" required="false" default="#StructNew()#" />

    <cfset var isContinue = true />
    <cfset var missingFields = '' />
    <cfset var requiredFields = '' />
    <cfset var invalidEvent = '' />
    <cfset var field = 0 />
    <cfset var newEventArgs = 0 />
```

```

    <cfif StructKeyExists(arguments.paramArgs,this.REQUIRED_FIELDS_PARAM)
        AND StructKeyExists(arguments.paramArgs,this.INVALID_EVENT_PARAM)>
        <cfset requiredFields = arguments.paramArgs[this.REQUIRED_FIELDS_PARAM] />
        <cfset invalidEvent = arguments.paramArgs[this.INVALID_EVENT_PARAM] />

        <cfloop index="field" list="#requiredFields#" delimiters=",">
            <cfif (NOT event.isArgDefined(field)) OR (event.getArg(field,'') EQ '')>
                <cfset missingFields = ListAppend(missingFields, field, ',') />
                <cfset isContinue = false />
            </cfif>
        </cfloop>
    <cfelse>
        <cfset throwUsageException() />
    </cfif>

    <cfif isContinue>
        <cfreturn true />
    <cfelse>
        <cfset newEventArgs = arguments.event.getArgs() />
        <cfset newEventArgs['message'] = "Please provide all required fields. Missing fields: " & ReplaceNoCase(
        <cfset newEventArgs['missingFields'] = missingFields />
        <cfset arguments.eventContext.announceEvent(invalidEvent, newEventArgs) />

        <cfreturn false />
    </cfif>
</cffunction>

```

throwUsageException

private void throwUsageException()

Throws an usage exception.

Parameters:

Code:

```

<cffunction name="throwUsageException" access="private" returnType="void" output="false"
    hint="Throws an usage exception.">
    <cfset var throwMsg = "RequiredFieldsFilter requires the following usage parameters: " & this.REQUIRED_FIELDS_PA
    <cfthrow message="#throwMsg#" />

```

```
</cffunction>
```

framework

AppFactory

Package: MachII.framework

Factory class for creating instances of AppManager.

Method Summary

public AppFactory	init() Used by the framework for initialization. Do not override.
public AppManager	createAppManager(string configXmlPath, string configDtdPath, [boolean validateXml="false"], [string version="Unknown BER"]) Creates the AppManager and reads (and optionally validates) the XML configuration file.

Method Detail

createAppManager

```
public AppManager createAppManager( string configXmlPath, string configDtdPath, [boolean validateXml="false"], [string version="Unknown BER"] )
```

Creates the AppManager and reads (and optionally validates) the XML configuration file.

Parameters:

```
string configXmlPath
string configDtdPath
[boolean validateXml="false"]
[string version="Unknown BER"]
```

Code:

```
<cffunction name="createAppManager" access="public" returntype="MachII.framework.AppManager" output="false"
```

```
hint="Creates the AppManager and reads (and optionally validates) the XML configuration file.">
<cfargument name="configXmlPath" type="string" required="true"
  hint="The full path to the configuration XML file." />
<cfargument name="configDtdPath" type="string" required="true"
  hint="The full path to the configuration DTD file." />
<cfargument name="validateXml" type="boolean" required="false" default="false"
  hint="Should the XML be validated before parsing." />
<cfargument name="version" type="string" required="false" default="Unknown BER"
  hint="The version number of Mach-II." />

<cfset var appManager = "" />
<cfset var propertyManager = "" />
<cfset var listenerManager = "" />
<cfset var filterManager = "" />
<cfset var eventManager = "" />
<cfset var viewManager = "" />
<cfset var pluginManager = "" />
<cfset var configXML = "" />
<cfset var configXmlFile = "" />
<cfset var validationResult = "" />
<cfset var validationException = "" />

<cftry>

  <cffile
    action="READ"
    file="#arguments.configXmlPath#"
    variable="configXmlFile" />

  <cfif arguments.validateXml AND ListFirst(server.ColdFusion.ProductVersion) GTE 7>
    <cfset validationResult = XmlValidate(arguments.configXmlPath, arguments.configDtdPath)>
    <cfif NOT validationResult.Status>
      <cfset validationException = CreateObject('component','MachII.util.XmlValidationException')
      <cfset validationException.wrapValidationResult(validationResult, arguments.configXmlPath)>
      <cfthrow type="MachII.framework.XmlValidationException"
        message="#validationException.getFormattedMessage()#" />
    </cfif>
  </cfif>

  <cfset configXML = XmlParse(configXmlFile) />

  <cfcatch type="Any">
    <cfrethrow />
  </cfcatch>
</cftry>
```

```
        </cfcatch>
    </cftry>

    <cfset appManager = CreateObject('component', 'MachII.framework.AppManager') />
    <cfset appManager.init() />

    <cfset propertyManager = CreateObject('component', 'MachII.framework.PropertyManager') />
    <cfset propertyManager.init(configXML, appManager, arguments.version) />
    <cfset appManager.setPropertyManager(propertyManager) />

    <cfset listenerManager = CreateObject('component', 'MachII.framework.ListenerManager') />
    <cfset listenerManager.init(configXML, appManager) />
    <cfset appManager.setListenerManager(listenerManager) />

    <cfset filterManager = CreateObject('component', 'MachII.framework.FilterManager') />
    <cfset filterManager.init(configXML, appManager) />
    <cfset appManager.setFilterManager(filterManager) />

    <cfset eventManager = CreateObject('component', 'MachII.framework.EventManager') />
    <cfset eventManager.init(configXML, appManager) />
    <cfset appManager.setEventManager(eventManager) />

    <cfset viewManager = CreateObject('component', 'MachII.framework.ViewManager') />
    <cfset viewManager.init(configXML, appManager) />
    <cfset appManager.setViewManager(viewManager) />

    <cfset pluginManager = CreateObject('component', 'MachII.framework.PluginManager') />
    <cfset pluginManager.init(configXML, appManager) />
    <cfset appManager.setPluginManager(pluginManager) />

    <cfset appManager.configure() />

    <cfreturn appManager />
</cffunction>
```

init

```
public AppFactory init( )
```

Used by the framework for initialization. Do not override.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="AppFactory" output="false"
    hint="Used by the framework for initialization. Do not override.">
    <cfreturn this />
</cffunction>
```

AppLoader

Package: MachII.framework

Responsible for controlling the loading/reloading of the AppManager.

Method Summary

public AppLoader	init(string configPath, string dtdPath, [boolean validateXml="false"], [string version="Unknown BER"])
	Used by the framework for initialization. Do not override.
public AppFactory	getAppFactory()
public AppManager	getAppManager()
public string	getConfigFileReloadHash()
	Get the current reload hash of the config file which is based on dateLastModified and size.
public string	getConfigPath()
public string	getDtdPath()
public string	getLastReloadHash()
public void	reloadConfig([boolean validateXml="false"], [string version="Unknown BER"])
	Reloads the config file and sets the last reload hash.
public void	setAppFactory(AppFactory appFactory)
public void	setAppManager(AppManager appManager)
public void	setConfigPath(string configPath)
public void	setDtdPath(string dtdPath)
public void	setLastReloadHash(string lastReloadHash)
public boolean	shouldReloadConfig()
	Determines if the configuration file should be reloaded.

Method Detail

getAppFactory

```
public AppFactory getAppFactory( )
```

Parameters:

Code:

```
<cffunction name="getAppFactory" access="public" returntype="MachII.framework.AppFactory" output="false">  
    <cfreturn variables.appFactory />  
</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Parameters:

Code:

```
<cffunction name="getAppManager" access="public" returntype="MachII.framework.AppManager" output="false">  
    <cfreturn variables.appManager />  
</cffunction>
```

getConfigFileReloadHash

```
public string getConfigFileReloadHash( )
```

Get the current reload hash of the config file which is based on dateLastModified and size.

Parameters:

Code:

```
<cffunction name="getConfigFileReloadHash" access="public" returntype="string" output="false"
  hint="Get the current reload hash of the config file which is based on dateLastModified and size.">
  <cfset var configFile = "" />

  <cfdirectory action="LIST" directory="#GetDirectoryFromPath(getConfigPath())#"
    name="configFile" filter="#GetFileFromPath(getConfigPath())#" />

  <cfreturn hash(configFile.dateLastModified & configFile.size) />
</cffunction>
```

getConfigPath

public string getConfigPath()

Parameters:

Code:

```
<cffunction name="getConfigPath" access="public" returntype="string" output="false">
  <cfreturn variables.configPath />
</cffunction>
```

getDtdPath

public string getDtdPath()

Parameters:

Code:

```
<cffunction name="getDtdPath" access="public" returntype="string" output="false">
```

```
<cfreturn variables.dtdPath />
</cffunction>
```

getLastReloadHash

```
public string getLastReloadHash( )
```

Parameters:

Code:

```
<cffunction name="getLastReloadHash" access="public" returntype="string" output="false">
    <cfreturn variables.lastReloadHash />
</cffunction>
```

init

```
public AppLoader init( string configPath, string dtdPath, [boolean validateXml="false"], [string version="Unknown BER"] )
```

Used by the framework for initialization. Do not override.

Parameters:

```
string configPath
string dtdPath
[boolean validateXml="false"]
[string version="Unknown BER"]
```

Code:

```
<cffunction name="init" access="public" returntype="MachII.framework.AppLoader" output="true"
    hint="Used by the framework for initialization. Do not override.">
    <cfargument name="configPath" type="string" required="true"
        hint="The full path to the configuration XML file." />
    <cfargument name="dtdPath" type="string" required="true"
        hint="The full path to the Mach-II DTD file." />
```

```

<cfargument name="validateXml" type="boolean" required="false" default="false"
  hint="Should the XML be validated before parsing." />
<cfargument name="version" type="string" required="false" default="Unknown BER"
  hint="The version number of Mach-II." />

<cfset var appFactory = CreateObject('component', 'MachII.framework.AppFactory').init() />
<cfset setAppFactory(appFactory) />

<cfset setConfigPath(arguments.configPath) />
<cfset setDtdPath(arguments.dtdPath) />

<cfset reloadConfig(arguments.validateXml, arguments.version) />

<cfreturn this />
</cffunction>

```

reloadConfig

```
public void reloadConfig( [boolean validateXml="false"], [string version="Unknown BER"] )
```

Reloads the config file and sets the last reload hash.

Parameters:

```
[boolean validateXml="false"]
[string version="Unknown BER"]
```

Code:

```

<cffunction name="reloadConfig" access="public" returntype="void" output="false"
  hint="Reloads the config file and sets the last reload hash.">
  <cfargument name="validateXml" type="boolean" required="false" default="false"
    hint="Should the XML be validated before parsing." />
  <cfargument name="version" type="string" required="false" default="Unknown BER"
    hint="The version number of Mach-II." />

  <cfset setAppManager(getAppFactory().createAppManager(getConfigPath(), getDtdPath(), arguments.validateXml, argu
  <cfset setLastReloadHash(getConfigFileReloadHash()) />
</cffunction>

```

setAppFactory

```
public void setAppFactory( AppFactory appFactory )
```

Parameters:

AppFactory appFactory

Code:

```
<cffunction name="setAppFactory" access="public" returntype="void" output="false">  
    <cfargument name="appFactory" type="MachII.framework.AppFactory" required="true" />  
    <cfset variables.appFactory = arguments.appFactory />  
</cffunction>
```

setAppManager

```
public void setAppManager( AppManager appManager )
```

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="public" returntype="void" output="false">  
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />  
    <cfset variables.appManager = arguments.appManager />  
</cffunction>
```

setConfigPath

```
public void setConfigPath( string configPath )
```

Parameters:

string configPath

Code:

```
<cffunction name="setConfigPath" access="public" returntype="void" output="false">
  <cfargument name="configPath" type="string" required="true" />
  <cfset variables.configPath = arguments.configPath />
</cffunction>
```

setDtdPath

public void setDtdPath(string dtdPath)

Parameters:

string dtdPath

Code:

```
<cffunction name="setDtdPath" access="public" returntype="void" output="false">
  <cfargument name="dtdPath" type="string" required="true" />
  <cfset variables.dtdPath = arguments.dtdPath />
</cffunction>
```

setLastReloadHash

public void setLastReloadHash(string lastReloadHash)

Parameters:

string lastReloadHash

Code:

```
<cffunction name="setLastReloadHash" access="public" returntype="void" output="false">
  <cfargument name="lastReloadHash" type="string" required="true" />
  <cfset variables.lastReloadHash = arguments.lastReloadHash />
</cffunction>
```

shouldReloadConfig

public boolean shouldReloadConfig()

Determines of the configuration file should be reloaded.

Parameters:

Code:

```
<cffunction name="shouldReloadConfig" access="public" returntype="boolean" output="false"
  hint="Determines of the configuration file should be reloaded.">
  <cfif CompareNoCase(getLastReloadHash(), getConfigFileReloadHash())>
    <cfreturn true />
  <cfelse>
    <cfreturn false />
  </cfif>
</cffunction>
```

AppManager

Package: MachII.framework

The main framework manager.

Method Summary

public void	init() Used by the framework for initialization. Do not override.
public void	configure() Calls configure() on each of the manager instances.
public EventContext	createEventContext(string requestEventName) Creates an EventContext instance.
public RequestHandler	createRequestHandler() Creates a RequestHandler instance.
public EventManager	getEventManager()
public FilterManager	getFilterManager()
public ListenerManager	getListenerManager()
public PluginManager	getPluginManager()
public PropertyManager	getPropertyManager()
public RequestHandler	getRequestHandler([boolean createNew="false"]) Returns a new or cached instance of a RequestHandler.
public ViewManager	getViewManager()
public void	setEventManager(EventManager eventManager)

Method Summary

public void	setFilterManager(FilterManager filterManager)
public void	setListenerManager(ListenerManager listenerManager)
public void	setPluginManager(PluginManager pluginManager)
public void	setPropertyManager(PropertyManager propertyManager)
public void	setViewManager(ViewManager viewManager)

Method Detail**configure**

```
public void configure( )
```

Calls configure() on each of the manager instances.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void"
  hint="Calls configure() on each of the manager instances.">
  <cfset getPropertyManager().configure() />
  <cfset getPluginManager().configure() />
  <cfset getListenerManager().configure() />
  <cfset getFilterManager().configure() />
  <cfset getEventManager().configure() />
  <cfset getViewManager().configure() />
</cffunction>
```

createEventContext

```
public EventContext createEventContext( string requestEventName )
```

Creates an EventContext instance.

Parameters:

string requestEventName

Code:

```
<cffunction name="createEventContext" access="public" returntype="MachII.framework.EventContext" output="false"
  hint="Creates an EventContext instance.">
  <cfargument name="requestEventName" type="string" required="true" />

  <cfset var eventContext = CreateObject('component', 'MachII.framework.EventContext') />
  <cfset eventContext.init(this, arguments.requestEventName) />
  <cfreturn eventContext />
</cffunction>
```

createRequestHandler

```
public RequestHandler createRequestHandler( )
```

Creates a RequestHandler instance.

Parameters:

Code:

```
<cffunction name="createRequestHandler" access="public" returntype="MachII.framework.RequestHandler" output="false"
  hint="Creates a RequestHandler instance.">
  <cfset var requestHandler = CreateObject('component', 'MachII.framework.RequestHandler') />
  <cfset requestHandler.init(this) />
  <cfreturn requestHandler />
</cffunction>
```

getEventManager

```
public EventManager getEventManager( )
```

Parameters:

Code:

```
<cffunction name="getEventManager" access="public" returnType="MachII.framework.EventManager" output="false">
    <cfreturn variables.eventManager />
</cffunction>
```

getFilterManager

public FilterManager getFilterManager()

Parameters:

Code:

```
<cffunction name="getFilterManager" access="public" returnType="MachII.framework.FilterManager" output="false">
    <cfreturn variables.filterManager />
</cffunction>
```

getListenerManager

public ListenerManager getListenerManager()

Parameters:

Code:

```
<cffunction name="getListenerManager" access="public" returnType="MachII.framework.ListenerManager" output="false">
    <cfreturn variables.listenerManager />
</cffunction>
```

getPluginManager

public PluginManager getPluginManager()

Parameters:

Code:

```
<cffunction name="getPluginManager" access="public" returntype="MachII.framework.PluginManager" output="false">  
    <cfreturn variables.pluginManager />  
</cffunction>
```

getPropertyManager

public PropertyManager getPropertyManager()

Parameters:

Code:

```
<cffunction name="getPropertyManager" access="public" returntype="MachII.framework.PropertyManager" output="false">  
    <cfreturn variables.propertyManager />  
</cffunction>
```

getRequestHandler

public RequestHandler getRequestHandler([boolean createNew="false"])

Returns a new or cached instance of a RequestHandler.

Parameters:

[boolean createNew="false"]

Code:

```
<cffunction name="getRequestHandler" access="public" returntype="MachII.framework.RequestHandler" output="false"
```

```
hint="Returns a new or cached instance of a RequestHandler.">
<cfargument name="createNew" type="boolean" required="false" default="false"
    hint="Pass true to return a new instance of a RequestHandler." />

<cfif arguments.createNew>
    <cfreturn createRequestHandler() />
<cfelse>
    <cfreturn variables.requestHandler />
</cfif>
</cffunction>
```

getViewManager

```
public ViewManager getViewManager( )
```

Parameters:

Code:

```
<cffunction name="getViewManager" access="public" returntype="MachII.framework.ViewManager" output="false">
    <cfreturn variables.viewManager />
</cffunction>
```

init

```
public void init( )
```

Used by the framework for initialization. Do not override.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="void" output="false"
    hint="Used by the framework for initialization. Do not override.">
    <cfset variables.requestHandler = CreateObject('component', 'MachII.framework.RequestHandler') />
</cffunction>
```

```
<cfset variables.requestHandler.init(this) />
</cffunction>
```

setEventManager

```
public void setEventManager( EventManager eventManager )
```

Parameters:

EventManager eventManager

Code:

```
<cffunction name="setEventManager" access="public" returntype="void" output="false">
    <cfargument name="eventManager" type="MachII.framework.EventManager" required="true" />
    <cfset variables.eventManager = arguments.eventManager />
</cffunction>
```

setFilterManager

```
public void setFilterManager( FilterManager filterManager )
```

Parameters:

FilterManager filterManager

Code:

```
<cffunction name="setFilterManager" access="public" returntype="void" output="false">
    <cfargument name="filterManager" type="MachII.framework.FilterManager" required="true" />
    <cfset variables.filterManager = arguments.filterManager />
</cffunction>
```

setListenerManager

```
public void setListenerManager( ListenerManager listenerManager )
```

Parameters:

ListenerManager listenerManager

Code:

```
<cffunction name="setListenerManager" access="public" returntype="void" output="false">
    <cfargument name="listenerManager" type="MachII.framework.ListenerManager" required="true" />
    <cfset variables.listenerManager = arguments.listenerManager />
</cffunction>
```

setPluginManager

```
public void setPluginManager( PluginManager pluginManager )
```

Parameters:

PluginManager pluginManager

Code:

```
<cffunction name="setPluginManager" access="public" returntype="void" output="false">
    <cfargument name="pluginManager" type="MachII.framework.PluginManager" required="true" />
    <cfset variables.pluginManager = arguments.pluginManager />
</cffunction>
```

setPropertyManager

```
public void setPropertyManager( PropertyManager propertyManager )
```

Parameters:

PropertyManager propertyManager

Code:

```
<cffunction name="setPropertyManager" access="public" returntype="void" output="false">
    <cfargument name="propertyManager" type="MachII.framework.PropertyManager" required="true" />
    <cfset variables.propertyManager = arguments.propertyManager />
</cffunction>
```

setViewManager

```
public void setViewManager( ViewManager viewManager )
```

Parameters:

ViewManager viewManager

Code:

```
<cffunction name="setViewManager" access="public" returntype="void" output="false">
    <cfargument name="viewManager" type="MachII.framework.ViewManager" required="true" />
    <cfset variables.viewManager = arguments.viewManager />
</cffunction>
```

BaseComponent

Package: MachII.framework

Base Mach-II component.

Method Summary

public void	init(AppManager appManager, [struct parameters="#StructNew()#"])	Used by the framework for initialization. Do not override.
public void	announceEvent(string eventName, [struct eventArgs="#StructNew()#"])	Announces a new event to the framework.
public void	configure()	Override to provide custom configuration logic. Called after init().
package AppManager	getAppManager()	Gets the components AppManager instance.
public any	getParameter(string name, [string defaultValue=""])	Gets a configuration parameter value, or a default value if not defined.
public struct	getParameters()	Gets the full set of configuration parameters for the component.
public any	getProperty(string propertyName)	Gets the specified property - this is just a shortcut for getAppManager().getPropertyManager().getProperty()
package PropertyManager	getPropertyManager()	Gets the components PropertyManager instance.
public boolean	hasParameter(string name)	

Method Summary

	DEPRECATED - use isParameterDefined() instead. Checks to see whether or not a configuration parameter is defined.
public boolean	isParameterDefined(string name) Checks to see whether or not a configuration parameter is defined.
private void	setAppManager(AppManager appManager) Sets the components AppManager instance.
public void	setParameter(string name, any value) Sets a configuration parameter.
public void	setParameters(struct parameters) Sets the full set of configuration parameters for the component.
public any	setProperty(string propertyName, any propertyValue) Sets the specified property - this is just a shortcut for getAppManager().getPropertyManager().setProperty()
private void	setPropertyManager(PropertyManager propertyManager) Sets the components PropertyManager instance.

Method Detail**announceEvent**

```
public void announceEvent( string eventName, [struct eventArgs="#StructNew()#"] )
```

Announces a new event to the framework.

Parameters:

```
string eventName  
[struct eventArgs="#StructNew()#"]
```

Code:

```
<cffunction name="announceEvent" access="public" returntype="void" output="false"
  hint="Announces a new event to the framework.">
  <cfargument name="eventName" type="string" required="true"
    hint="The name of the event to announce." />
  <cfargument name="eventArgs" type="struct" required="false" default="#StructNew()#"
    hint="A struct of arguments to set as the event's args." />

  <cfif StructKeyExists(request, "eventContext")>
    <cfset request.eventContext.announceEvent(arguments.eventName, arguments.eventArgs) />
  <cfelse>
    <cfthrow message="The EventContext necessary to announce events is not set in 'request.eventContext.'" />
  </cfif>
</cffunction>
```

configure

```
public void configure( )
```

Override to provide custom configuration logic. Called after init().

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void" output="false"
  hint="Override to provide custom configuration logic. Called after init().">

</cffunction>
```

getAppManager

```
package AppManager getAppManager( )
```

Gets the components AppManager instance.

Parameters:

Code:

```
<cffunction name="getAppManager" access="package" returntype="MachII.framework.AppManager" output="false"
    hint="Gets the components AppManager instance.">
    <cfreturn variables.appManager />
</cffunction>
```

getParameter

```
public any getParameter( string name, [string defaultValue=""] )
```

Gets a configuration parameter value, or a default value if not defined.

Parameters:

```
string name
[string defaultValue=""]
```

Code:

```
<cffunction name="getParameter" access="public" returntype="any" output="false"
    hint="Gets a configuration parameter value, or a default value if not defined.">
    <cfargument name="name" type="string" required="true"
        hint="The parameter name." />
    <cfargument name="defaultValue" type="string" required="false" default=""
        hint="The default value to return if the parameter is not defined. Defaults to a blank string." />
    <cfif isParameterDefined(arguments.name)>
        <cfreturn variables.parameters[arguments.name] />
    <cfelse>
        <cfreturn arguments.defaultValue />
    </cfif>
</cffunction>
```

getParameters

public struct getParameters()

Gets the full set of configuration parameters for the component.

Parameters:

Code:

```
<cffunction name="getParameters" access="public" returntype="struct" output="false"
    hint="Gets the full set of configuration parameters for the component.">
    <cfreturn variables.parameters />
</cffunction>
```

getProperty

public any getProperty(string propertyName)

Gets the specified property - this is just a shortcut for getAppManager().getPropertyManager().getProperty()

Parameters:

string propertyName

Code:

```
<cffunction name="getProperty" access="public" returntype="any" output="false"
    hint="Gets the specified property - this is just a shortcut for getAppManager().getPropertyManager().getProperty"
    <cfargument name="propertyName" type="string" required="true"
        hint="The name of the property to return."/>
    <cfreturn getAppManager().getPropertyManager().getProperty(propertyName) />
</cffunction>
```

getPropertyManager

package PropertyManager getPropertyManager()

Gets the components PropertyManager instance.

Parameters:

Code:

```
<cffunction name="getPropertyManager" access="package" returntype="MachII.framework.PropertyManager" output="false"
    hint="Gets the components PropertyManager instance.">
    <cfreturn variables.propertyManager />
</cffunction>
```

hasParameter

```
public boolean hasParameter( string name )
```

DEPRECATED - use isParameterDefined() instead. Checks to see whether or not a configuration parameter is defined.

Parameters:

string name

Code:

```
<cffunction name="hasParameter" access="public" returntype="boolean" output="false"
    hint="DEPRECATED - use isParameterDefined() instead. Checks to see whether or not a configuration parameter is defined.">
    <cfargument name="name" type="string" required="true"
        hint="The parameter name." />
    <cfreturn StructKeyExists(variables.parameters, arguments.name) />
</cffunction>
```

init

```
public void init( AppManager appManager, [struct parameters="#StructNew()#"] )
```

Used by the framework for initialization. Do not override.

Parameters:

AppManager appManager

```
[struct parameters="#StructNew()#"]
```

Code:

```
<cffunction name="init" access="public" returntype="void" output="false"
  hint="Used by the framework for initialization. Do not override.">
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true"
    hint="The framework instances' AppManager." />
  <cfargument name="parameters" type="struct" required="false" default="#StructNew()#"
    hint="The initial set of configuration parameters." />

  <cfset setAppManager(arguments.appManager) />
  <cfset setParameters(arguments.parameters) />
  <cfset setPropertyManager(getAppManager().getPropertyManager()) />
</cffunction>
```

isParameterDefined

```
public boolean isParameterDefined( string name )
```

Checks to see whether or not a configuration parameter is defined.

Parameters:

string name

Code:

```
<cffunction name="isParameterDefined" access="public" returntype="boolean" output="false"
  hint="Checks to see whether or not a configuration parameter is defined.">
  <cfargument name="name" type="string" required="true"
    hint="The parameter name." />
  <cfreturn StructKeyExists(variables.parameters, arguments.name) />
</cffunction>
```

setAppManager

```
private void setAppManager( AppManager appManager )
```

Sets the components AppManager instance.

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="private" returntype="void" output="false"
  hint="Sets the components AppManager instance.">
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true"
    hint="The AppManager instance to set." />
  <cfset variables.appManager = arguments.appManager />
</cffunction>
```

setParameter

```
public void setParameter( string name, any value )
```

Sets a configuration parameter.

Parameters:

string name
any value

Code:

```
<cffunction name="setParameter" access="public" returntype="void" output="false"
  hint="Sets a configuration parameter.">
  <cfargument name="name" type="string" required="true"
    hint="The parameter name." />
  <cfargument name="value" required="true"
    hint="The parameter value." />
  <cfset variables.parameters[arguments.name] = arguments.value />
</cffunction>
```

setParameters

```
public void setParameters( struct parameters )
```

Sets the full set of configuration parameters for the component.

Parameters:

struct parameters

Code:

```
<cffunction name="setParameters" access="public" returntype="void" output="false"
  hint="Sets the full set of configuration parameters for the component.">
  <cfargument name="parameters" type="struct" required="true" />
  <cfset var key = "" />
  <cfloop collection="#arguments.parameters#" item="key">
    <cfset setParameter(key, parameters[key]) />
  </cfloop>
</cffunction>
```

setProperty

```
public any setProperty( string propertyName, any propertyValue )
```

Sets the specified property - this is just a shortcut for `getAppManager().getPropertyManager().setProperty()`

Parameters:

string propertyName

any propertyValue

Code:

```
<cffunction name="setProperty" access="public" returntype="any" output="false"
  hint="Sets the specified property - this is just a shortcut for getAppManager().getPropertyManager().setProperty"
  <cfargument name="propertyName" type="string" required="true"
    hint="The name of the property to set." />
  <cfargument name="propertyValue" type="any" required="true"
    hint="The value to store in the property." />
  <cfreturn getPropertyManager().setProperty(arguments.propertyName, arguments.propertyValue) />
</cffunction>
```

setPropertyManager

```
private void setPropertyManager( PropertyManager propertyManager )
```

Sets the components PropertyManager instance.

Parameters:

PropertyManager propertyManager

Code:

```
<cffunction name="setPropertyManager" access="private" returntype="void" output="false"
    hint="Sets the components PropertyManager instance.">
    <cfargument name="propertyManager" type="MachII.framework.PropertyManager" required="true"
        hint="The PropertyManager instance to set." />
    <cfset variables.propertyManager = arguments.propertyManager />
</cffunction>
```

Event

Package: MachII.framework

The Event object encapsulates the event args.

Method Summary

public void	init([string name=""], [struct args="#StructNew()#"], [string requestName=""]) Used by the framework for initialization. Do not override.
public any	getArg(string name, [any defaultValue=""]) Returns the value of an arg or the default value if the arg is not defined.
public struct	getArgs() Returns all args in this event.
public string	getArgType(string argName) Returns the arg type of the arg name.
public string	getName() Returns the name of the event object.
public string	getRequestName() Returns the event name that started the request lifecycle.
public boolean	isArgDefined(string name) Checks if an arg is defined in the event object.
public void	removeArg(string name) Deletes an arg from the even object.
public void	setArg(string name, any value, [string argType])

Method Summary

	Sets an arg in the event object.
public void	setArgs(struct args)
	Sets a structure of args to the event object.
public void	setArgType(string argName, string argType)
	Sets the arg type for the specified arg.
public void	setName(string name)
	Sets the name of the event object.
public void	setRequestName(string requestName)
	Sets the event name that started the request lifecycle.

Method Detail**getArg**

```
public any getArg( string name, [any defaultValue=""] )
```

Returns the value of an arg or the default value if the arg is not defined.

Parameters:

```
string name  
[any defaultValue=""]
```

Code:

```
<cffunction name="getArg" access="public" returntype="any" output="false"  
    hint="Returns the value of an arg or the default value if the arg is not defined.">  
    <cfargument name="name" type="string" required="true"  
        hint="Name of arg to get in the event object." />
```

```
<cfargument name="defaultValue" type="any" required="false" default=""
    hint="Used to return a default value if the arg does not exist in the event object." />

<cfif StructKeyExists(variables.args, arguments.name)>
    <cfreturn variables.args[arguments.name] />
<cfelse>
    <cfreturn arguments.defaultValue />
</cfif>
</cffunction>
```

getArgs

public struct getArgs()

Returns all args in this event.

Parameters:

Code:

```
<cffunction name="getArgs" access="public" returntype="struct" output="false"
    hint="Returns all args in this event.">
    <cfreturn variables.args />
</cffunction>
```

getArgType

public string getArgType(string argName)

Returns the arg type of the arg name.

Parameters:

string argName

Code:

```
<cffunction name="getArgType" access="public" returnType="string" output="false"
  hint="Returns the arg type of the arg name.">
  <cfargument name="argName" type="string" required="true"
    hint="The name of the arg to get the arg type." />
  <cfif StructKeyExists(variables.argTypes, arguments.argName)>
    <cfreturn variables.argTypes[arguments.argName] />
  <cfelse>
    <cfreturn " " />
  </cfif>
</cffunction>
```

getName

public string getName()

Returns the name of the event object.

Parameters:

Code:

```
<cffunction name="getName" access="public" returnType="string" output="false"
  hint="Returns the name of the event object.">
  <cfreturn variables.name />
</cffunction>
```

getRequestName

public string getRequestName()

Returns the event name that started the request lifecycle.

Parameters:

Code:

```
<cffunction name="getRequestName" access="public" returnType="string" output="false"
```

```
hint="Returns the event name that started the request lifecycle.">
  <cfreturn variables.requestName />
</cffunction>
```

init

```
public void init( [string name=""], [struct args="#StructNew()#"], [string requestName=""] )
```

Used by the framework for initialization. Do not override.

Parameters:

```
[string name=""]
[struct args="#StructNew()#"]
[string requestName=""]
```

Code:

```
<cffunction name="init" access="public" returntype="void" output="false"
  hint="Used by the framework for initialization. Do not override.">
  <cfargument name="name" type="string" required="false" default=""
    hint="The name of the event object." />
  <cfargument name="args" type="struct" required="false" default="#StructNew()#"
    hint="Event args to populate this event object." />
  <cfargument name="requestName" type="string" required="false" default=""
    hint="A request name for this request lifecycle." />

  <cfset setName(arguments.name) />
  <cfset setArgs(arguments.args) />
  <cfset setRequestName(arguments.requestName) />
</cffunction>
```

isArgDefined

```
public boolean isArgDefined( string name )
```

Checks if an arg is defined in the event object.

Parameters:

string name

Code:

```
<cffunction name="isArgDefined" access="public" returntype="boolean" output="false"
    hint="Checks if an arg is defined in the event object.">
    <cfargument name="name" type="string" required="true"
        hint="Name of arg to check." />
    <cfreturn StructKeyExists(variables.args, arguments.name) />
</cffunction>
```

removeArg

public void removeArg(string name)

Deletes an arg from the even object.

Parameters:

string name

Code:

```
<cffunction name="removeArg" access="public" returntype="void" output="false"
    hint="Deletes an arg from the even object.">
    <cfargument name="name" type="string" required="true"
        hint="Name of arg to delete from the event object." />
    <cfset StructDelete(variables.args, arguments.name) />
</cffunction>
```

setArg

public void setArg(string name, any value, [string argType])

Sets an arg in the event object.

Parameters:

string name
any value
[string argType]

Code:

```
<cffunction name="setArg" access="public" returntype="void" output="false"
  hint="Sets an arg in the event object.">
  <cfargument name="name" type="string" required="true"
    hint="The name of the arg to set." />
  <cfargument name="value" type="any" required="true"
    hint="The value of the arg to set." />
  <cfargument name="argType" type="string" required="false"
    hint="The type of the arg to set." />

  <cfset variables.args[arguments.name] = arguments.value />
  <cfif StructKeyExists(arguments, 'argType')>
    <cfset setArgType(arguments.name, arguments.argType) />
  </cfif>
</cffunction>
```

setArgs

public void setArgs(struct args)

Sets a structure of args to the event object.

Parameters:

struct args

Code:

```
<cffunction name="setArgs" access="public" returntype="void" output="false"
  hint="Sets a structure of args to the event object.">
  <cfargument name="args" type="struct" required="true"
    hint="A structure of args to set." />
  <cfset var key = "" />
```

```
        <cfloop collection="#arguments.args#" item="key">
            <cfset setArg(key, arguments.args[key]) />
        </cfloop>
    </cffunction>
```

setArgType

```
public void setArgType( string argName, string argType )
```

Sets the arg type for the specified arg.

Parameters:

```
string argName
string argType
```

Code:

```
<cffunction name="setArgType" access="public" returntype="void" output="false"
    hint="Sets the arg type for the specified arg.">
    <cfargument name="argName" type="string" required="true"
        hint="The name of the arg." />
    <cfargument name="argType" type="string" required="true"
        hint="The arg type to set." />
    <cfset variables.argTypes[arguments.argName] = arguments.argType />
</cffunction>
```

setName

```
public void setName( string name )
```

Sets the name of the event object.

Parameters:

```
string name
```

Code:

```
<cffunction name="setName" access="public" returnType="void" output="false"
  hint="Sets the name of the event object.">
  <cfargument name="name" type="string" required="true"
    hint="A name for this event." />
  <cfset variables.name = arguments.name />
</cffunction>
```

setRequestName

```
public void setRequestName( string requestName )
```

Sets the event name that started the request lifecycle.

Parameters:

string requestName

Code:

```
<cffunction name="setRequestName" access="public" returnType="void" output="false"
  hint="Sets the event name that started the request lifecycle.">
  <cfargument name="requestName" type="string" required="true"
    hint="A request name for this event." />
  <cfset variables.requestName = arguments.requestName />
</cffunction>
```

EventCommand

Package: MachII.framework

Base EventCommand component.

Method Summary

public EventCommand	init() Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext) Overridden by the command that extends this component.
public any	getParameter(string name)
public void	setParameter(string name, any value)
package void	setParameters(struct parameters)

Method Detail

execute

```
public boolean execute( Event event, EventContext eventContext )
```

Overridden by the command that extends this component.

Parameters:

Event event
EventContext eventContext

Code:

```
<cffunction name="execute" access="public" returntype="boolean" output="true"
    hint="Overridden by the command that extends this component.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

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

getParameter

public any getParameter(string name)

Parameters:

string name

Code:

```
<cffunction name="getParameter" access="public" returntype="any" output="false">
    <cfargument name="name" type="string" required="true" />
    <cfreturn variables.parameters[arguments.name] />
</cffunction>
```

init

public EventCommand init()

Used by the framework for initialization.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="EventCommand" output="false"
    hint="Used by the framework for initialization.">
    <cfreturn this />
</cffunction>
```

```
</cffunction>
```

setParameter

```
public void setParameter( string name, any value )
```

Parameters:

string name
any value

Code:

```
<cffunction name="setParameter" access="public" returnType="void" output="false">  
  <cfargument name="name" type="string" required="true" />  
  <cfargument name="value" type="any" required="true" />  
  <cfset variables.parameters[arguments.name] = arguments.value />  
</cffunction>
```

setParameters

```
package void setParameters( struct parameters )
```

Parameters:

struct parameters

Code:

```
<cffunction name="setParameters" access="package" returnType="void" output="false">  
  <cfargument name="parameters" type="struct" required="true" />  
  <cfset var key = "" />  
  <cfloop collection="#arguments.parameters#" item="key">  
    <cfset setParameter(key, parameters[key]) />  
  </cfloop>
```

```
</cffunction>
```

EventContext

Package: MachII.framework

The framework workhorse. Handles event-queue functionality and event-command execution. Controls the event queue and event processing mechanism for a request/event lifecycle.

Method Summary

public EventContext	init(AppManager appManager, [string requestEventName=""]) Initializes the event-context.
public void	announceEvent(string eventName, [struct eventArgs="#StructNew()#"]) Queues an event for the framework to handle.
public void	clearEventMappings() Clears the current event mappings.
public void	clearEventQueue() Clears the event queue.
private Exception	createException([string type="", [string message="", [string errorCode="", [string detail="", [string extendedInfo=""], [array tagContext="#ArrayNew(1)#"]) Creates an exception object (with no cfcatch).
public void	displayView(Event event, string viewName, [string contentKey="", [string contentArg=""], [boolean append="false"]) Displays a view.
private any	getAppManager()
public Event	getCurrentEvent() Gets the current event object.
private EventHandler	getCurrentEventHandler()
public numeric	getEventCount()

Method Summary

	Returns the number of events that have been processed for this context.
public string	getEventMapping(string eventName) Gets an event mappiong by the event name.
private SizedQueue	getEventQueue()
public string	getExceptionEventName()
private string	getIsException()
private boolean	getIsProcessing()
public numeric	getMaxEvents()
public Event	getNextEvent() Peeks at the next event in the queue.
public Event	getPreviousEvent() Returns the previous handled event.
private string	getRequestEventName()
private any	getViewContext()
private void	handleEvent(Event event) Handles the current event.
public void	handleException(Exception exception, [boolean clearEventQueue="true"]) Handles an exception.
private void	handleNextEvent() Handles the next event in the queue.
public boolean	hasCurrentEvent() Checks if the current event has an event object.
public boolean	hasMoreEvents() Checks if there are more events in the queue.

Method Summary

public boolean	hasNextEvent() Peeks at the next event in the queue.
public boolean	hasPreviousEvent() Returns whether or not getPreviousEvent() can be called to return an event.
private void	incrementEventCount() Increments the current event count by 1.
public void	processEvents() Begins processing of queued events. Can only be called once.
private void	setAppManager(AppManager appManager)
private void	setCurrentEvent(Event currentEvent)
private void	setCurrentEventHandler(EventHandler currentEventHandler)
public string	setEventMapping(string eventName, string mappingName) Sets an event mapping.
private void	setEventQueue(SizedQueue eventQueue)
public void	setExceptionEventName(string exceptionEventName)
private void	setIsException(boolean isException)
private void	setIsProcessing(boolean isProcessing)
public void	setMaxEvents(numeric maxEvents)
private void	setPreviousEvent(Event previousEvent)
private void	setRequestEventName(string requestEventName)
private void	setViewContext(ViewContext viewContext)
private Exception	wrapException(any caughtException) Creates an exception object (with cfcatch).

Method Detail**announceEvent**

```
public void announceEvent( string eventName, [struct eventArgs="#StructNew()#"] )
```

Queues an event for the framework to handle.

Parameters:

```
string eventName  
[struct eventArgs="#StructNew()#"]
```

Code:

```
<cffunction name="announceEvent" access="public" returntype="void" output="true"  
    hint="Queues an event for the framework to handle.">  
    <cfargument name="eventName" type="string" required="true" />  
    <cfargument name="eventArgs" type="struct" required="false" default="#StructNew()#" />  
  
    <cfset var nextEventName = "" />  
    <cfset var nextEvent = "" />  
    <cfset var exception = "" />  
  
    <cftry>  
        <cfset nextEventName = getEventMapping(arguments.eventName) />  
        <cfset nextEvent = getAppManager().getEventManager().createEvent(nextEventName, arguments.eventArgs) />  
        <cfset nextEvent.setRequestName(getRequestEventName()) />  
        <cfset getEventQueue().put(nextEvent) />  
  
        <cfcatch>  
            <cfset exception = wrapException(cfcatch) />  
            <cfset handleException(exception, true) />  
        </cfcatch>  
    </cftry>  
</cffunction>
```

clearEventMappings

```
public void clearEventMappings( )
```

Clears the current event mappings.

Parameters:

Code:

```
<cffunction name="clearEventMappings" access="public" returntype="void" output="false"
    hint="Clears the current event mappings.">
    <cfset StructClear(variables.mappings) />
</cffunction>
```

clearEventQueue

```
public void clearEventQueue( )
```

Clears the event queue.

Parameters:

Code:

```
<cffunction name="clearEventQueue" access="public" returntype="void" output="false"
    hint="Clears the event queue.">
    <cfset getEventQueue().clear() />
</cffunction>
```

createException

```
private Exception createException( [string type=""], [string message=""], [string errorCode=""], [string detail=""], [string extendedInfo=""], [array tagContext="#ArrayNew(1)#"] )
```

Creates an exception object (with no cfcatch).

Parameters:

```
[string type=""]  
[string message=""]  
[string errorCode=""]  
[string detail=""]  
[string extendedInfo=""]  
[array tagContext="#ArrayNew(1)#"]
```

Code:

```
<cffunction name="createException" access="private" returntype="MachII.util.Exception" output="false"  
  hint="Creates an exception object (with no cfcatch).">  
  <cfargument name="type" type="string" required="false" default="" />  
  <cfargument name="message" type="string" required="false" default="" />  
  <cfargument name="errorCode" type="string" required="false" default="" />  
  <cfargument name="detail" type="string" required="false" default="" />  
  <cfargument name="extendedInfo" type="string" required="false" default="" />  
  <cfargument name="tagContext" type="array" required="false" default="#ArrayNew(1)#" />  
  
  <cfset var exception = CreateObject('component', 'MachII.util.Exception') />  
  <cfset exception.init(arguments.type, arguments.message, arguments.errorCode, arguments.detail, arguments.extendedInfo, arguments.tagContext) />  
  <cfset setIsException(true) />  
  
  <cfreturn exception />  
</cffunction>
```

displayView

```
public void displayView( Event event, string viewName, [string contentKey=""], [string contentArg=""], [boolean append="false"] )
```

Displays a view.

Parameters:

```
Event event  
string viewName
```

```
[string contentKey=""]  
[string contentArg=""]  
[boolean append="false"]
```

Code:

```
<cffunction name="displayView" access="public" returntype="void" output="true"  
    hint="Displays a view.">  
    <cfargument name="event" type="MachII.framework.Event" required="true" />  
    <cfargument name="viewName" type="string" required="true" />  
    <cfargument name="contentKey" type="string" required="false" default="" />  
    <cfargument name="contentArg" type="string" required="false" default="" />  
    <cfargument name="append" type="boolean" required="false" default="false" />  
  
    <cfset getAppManager().getPluginManager().preView(this) />  
  
    <cfset getViewContext().displayView(arguments.event, arguments.viewName, arguments.contentKey, arguments.contentArg, arguments.append) />  
  
    <cfset getAppManager().getPluginManager().postView(this) />  
</cffunction>
```

getAppManager

```
private any getAppManager( )
```

Parameters:

Code:

```
<cffunction name="getAppManager" access="private" type="MachII.framework.AppManager" output="false">  
    <cfreturn variables.appManager />  
</cffunction>
```

getCurrentEvent

```
public Event getCurrentEvent( )
```

Gets the current event object.

Parameters:

Code:

```
<cffunction name="getCurrentEvent" access="public" returntype="MachII.framework.Event" output="false"
    hint="Gets the current event object.">
    <cfreturn variables.currentEvent />
</cffunction>
```

getCurrentEventHandler

private EventHandler getCurrentEventHandler()

Parameters:

Code:

```
<cffunction name="getCurrentEventHandler" access="private" returntype="MachII.framework.EventHandler" output="false">
    <cfreturn variables.currentEventHandler />
</cffunction>
```

getEventCount

public numeric getEventCount()

Returns the number of events that have been processed for this context.

Parameters:

Code:

```
<cffunction name="getEventCount" access="public" returntype="numeric" output="false"
    hint="Returns the number of events that have been processed for this context.">
    <cfreturn variables.eventCount />
</cffunction>
```

```
</cffunction>
```

getEventMapping

```
public string getEventMapping( string eventName )
```

Gets an event mapping by the event name.

Parameters:

```
string eventName
```

Code:

```
<cffunction name="getEventMapping" access="public" returnType="string" output="false"
  hint="Gets an event mapping by the event name.">
  <cfargument name="eventName" type="string" required="true" />
  <cfif StructKeyExists(variables.mappings, arguments.eventName)>
    <cfreturn variables.mappings[arguments.eventName] />
  <cfelse>
    <cfreturn arguments.eventName />
  </cfif>
</cffunction>
```

getEventQueue

```
private SizedQueue getEventQueue( )
```

Parameters:

Code:

```
<cffunction name="getEventQueue" access="private" returnType="MachII.util.SizedQueue" output="false">
  <cfreturn variables.eventQueue />
</cffunction>
```

getExceptionEventName

public string getExceptionEventName()

Parameters:

Code:

```
<cffunction name="getExceptionEventName" access="public" returntype="string" output="false">
    <cfreturn variables.exceptionEventName />
</cffunction>
```

getIsException

private string getIsException()

Parameters:

Code:

```
<cffunction name="getIsException" access="private" returntype="string" output="false">
    <cfreturn variables.isException />
</cffunction>
```

getIsProcessing

private boolean getIsProcessing()

Parameters:

Code:

```
<cffunction name="getIsProcessing" access="private" returntype="boolean" output="false">
    <cfreturn variables.isProcessing />
</cffunction>
```

getMaxEvents

public numeric getMaxEvents()

Parameters:

Code:

```
<cffunction name="getMaxEvents" access="public" returntype="numeric" output="false">
    <cfreturn variables.maxEvents />
</cffunction>
```

getNextEvent

public Event getNextEvent()

Peeks at the next event in the queue.

Parameters:

Code:

```
<cffunction name="getNextEvent" access="public" returntype="MachII.framework.Event" output="false"
    hint="Peeks at the next event in the queue.">
    <cfreturn getEventQueue().peek() />
</cffunction>
```

getPreviousEvent

public Event getPreviousEvent()

Returns the previous handled event.

Parameters:

Code:

```
<cffunction name="getPreviousEvent" access="public" returntype="MachII.framework.Event" output="false"
    hint="Returns the previous handled event.">
    <cfreturn variables.previousEvent />
</cffunction>
```

getRequestEventName

private string getRequestEventName()

Parameters:

Code:

```
<cffunction name="getRequestEventName" access="private" returntype="string" output="false">
    <cfreturn variables.requestEventName />
</cffunction>
```

getViewContext

private any getViewContext()

Parameters:

Code:

```
<cffunction name="getViewContext" access="private" type="MachII.framework.ViewContext" output="false">
    <cfreturn variables.viewContext />
</cffunction>
```

handleEvent

```
private void handleEvent( Event event )
```

Handles the current event.

Parameters:

Event event

Code:

```
<cffunction name="handleEvent" access="private" returnType="void" output="true"
  hint="Handles the current event.">
  <cfargument name="event" type="MachII.framework.Event" required="true" />

  <cfset var eventName = "" />
  <cfset var eventHandler = 0 />

  <cfif hasCurrentEvent()>
    <cfset setPreviousEvent(getCurrentEvent()) />
  </cfif>
  <cfset setCurrentEvent(arguments.event) />
  <cfset request.event = arguments.event />

  <cfset eventName = arguments.event.getName() />

  <cfset eventHandler = getAppManager().getEventManager().getEventHandler(eventName) />
  <cfset setCurrentEventHandler(eventHandler) />

  <cfset getAppManager().getPluginManager().preEvent(this) />

  <cfset eventHandler.handleEvent(arguments.event, this) />

  <cfset getAppManager().getPluginManager().postEvent(this) />

  <cfset clearEventMappings() />
</cffunction>
```

handleException

```
public void handleException( Exception exception, [boolean clearEventQueue="true"] )
```

Handles an exception.

Parameters:

Exception exception
[boolean clearEventQueue="true"]

Code:

```
<cffunction name="handleException" access="public" returntype="void" output="true"
  hint="Handles an exception.">
  <cfargument name="exception" type="MachII.util.Exception" required="true" />
  <cfargument name="clearEventQueue" type="boolean" required="false" default="true" />

  <cfset var nextEventName = "" />
  <cfset var exceptionEvent = "" />

  <cftry>
    <cfset nextEventName = getEventMapping(getExceptionEventName()) />
    <cfset exceptionEvent = getAppManager().getEventManager().createEvent(nextEventName) />

    <cfset exceptionEvent.setRequestName(getRequestEventName()) />

    <cfset exceptionEvent.setArg('exception', arguments.exception) />

    <cfif hasCurrentEvent()>
      <cfset exceptionEvent.setArg('exceptionEvent', getCurrentEvent()) />
    </cfif>

    <cfset getAppManager().getPluginManager().handleException(this, arguments.exception) />

    <cfif arguments.clearEventQueue>
      <cfset variables.clearEventQueue() />
    </cfif>
  </cftry>
```

```
        <cfset getEventQueue().put(exceptionEvent) />
        <cfcatch type="any">
            <cfrethrow />
        </cfcatch>
    </cftry>
</cffunction>
```

handleNextEvent

private void handleNextEvent()

Handles the next event in the queue.

Parameters:

Code:

```
<cffunction name="handleNextEvent" access="private" returntype="void" output="true"
    hint="Handles the next event in the queue.">
    <cfset var exception = 0 />

    <cftry>
        <cfset incrementEventCount() />
        <cfset handleEvent(getEventQueue().get()) />

        <cfcatch type="AbortEventException">

        </cfcatch>
        <cfcatch type="any">
            <cfif getIsException()>
                <cfrethrow />
            <cfelse>
                <cfset exception = wrapException(cfcatch) />
                <cfset handleException(exception, true) />
            </cfif>
        </cfcatch>
    </cftry>
</cffunction>
```

hasCurrentEvent

```
public boolean hasCurrentEvent( )
```

Checks if the current event has an event object.

Parameters:

Code:

```
<cffunction name="hasCurrentEvent" access="public" returntype="boolean" output="false"
    hint="Checks if the current event has an event object.">
    <cfreturn IsObject(variables.currentEvent) />
</cffunction>
```

hasMoreEvents

```
public boolean hasMoreEvents( )
```

Checks if there are more events in the queue.

Parameters:

Code:

```
<cffunction name="hasMoreEvents" access="public" returntype="boolean" output="false"
    hint="Checks if there are more events in the queue.">
    <cfreturn NOT getEventQueue().isEmpty() />
</cffunction>
```

hasNextEvent

```
public boolean hasNextEvent( )
```

Peeks at the next event in the queue.

Parameters:

Code:

```
<cffunction name="hasNextEvent" access="public" returntype="boolean" output="false"
    hint="Peeks at the next event in the queue.">
    <cfreturn hasMoreEvents() />
</cffunction>
```

hasPreviousEvent

```
public boolean hasPreviousEvent( )
```

Returns whether or not getPreviousEvent() can be called to return an event.

Parameters:

Code:

```
<cffunction name="hasPreviousEvent" access="public" returntype="boolean" output="false"
    hint="Returns whether or not getPreviousEvent() can be called to return an event.">
    <cfreturn IsObject(variables.previousEvent) />
</cffunction>
```

incrementEventCount

```
private void incrementEventCount( )
```

Increments the current event count by 1.

Parameters:

Code:

```
<cffunction name="incrementEventCount" access="private" returntype="void" output="false"
    hint="Increments the current event count by 1.">
    <cfset variables.eventCount = variables.eventCount + 1 />
</cffunction>
```

```
</cffunction>
```

init

```
public EventContext init( AppManager appManager, [string requestEventName=""] )
```

Initializes the event-context.

Parameters:

AppManager appManager
[string requestEventName=""]

Code:

```
<cffunction name="init" access="public" returntype="EventContext" output="false"
    hint="Initializes the event-context.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfargument name="requestEventName" type="string" required="false" default="" />

    <cfset var eventQueue = 0 />
    <cfset var viewContext = 0 />

    <cfset setAppManager(arguments.appManager) />
    <cfset setRequestEventName(arguments.requestEventName) />
    <cfset setExceptionEventName(getAppManager().getPropertyManager().getProperty('exceptionEvent')) />
    <cfset setMaxEvents(getAppManager().getPropertyManager().getProperty('maxEvents')) />

    <cfset eventQueue = CreateObject('component', 'MachII.util.SizedQueue') />
    <cfset eventQueue.init(getMaxEvents()) />
    <cfset setEventQueue(eventQueue) />

    <cfset viewContext = CreateObject('component', 'MachII.framework.ViewContext') />
    <cfset viewContext.init(getAppManager()) />
    <cfset setViewContext(viewContext) />

    <cfreturn this />
</cffunction>
```

processEvents

```
public void processEvents()
```

Begins processing of queued events. Can only be called once.

Parameters:

Code:

```
<cffunction name="processEvents" access="public" returntype="void" output="true"
  hint="Begins processing of queued events. Can only be called once.">

  <cfset var pluginManager = "" />
  <cfset var eventManager = "" />
  <cfset var exception = "" />

  <cfif getIsProcessing()>
    <cfthrow message="The EventContext is already processing the events in the queue. The processEvents() method
  </cfif>
  <cfset setIsProcessing(true) />

  <cfset pluginManager = getAppManager().getPluginManager() />
  <cfset eventManager = getAppManager().getEventManager() />

  <cfset pluginManager.preProcess(this) />

  <cfloop condition="hasMoreEvents() AND getEventCount() LT getMaxEvents()>
    <cfset handleNextEvent() />
  </cfloop>

  <cfif NOT getEventQueue().isEmpty()>
    <cfset exception = createException("MachII.framework.MaxEventsExceeded", "The maximum number of events (1
    <cfset handleException(exception, true) />
  </cfif>

  <cfset pluginManager.postProcess(this) />
  <cfset setIsProcessing(false) />
</cffunction>
```

setAppManager

```
private void setAppManager( AppManager appManager )
```

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="private" returntype="void" output="false">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfset variables.appManager = arguments.appManager />
</cffunction>
```

setCurrentEvent

```
private void setCurrentEvent( Event currentEvent )
```

Parameters:

Event currentEvent

Code:

```
<cffunction name="setCurrentEvent" access="private" returntype="void" output="false">
    <cfargument name="currentEvent" type="MachII.framework.Event" required="true" />
    <cfset variables.currentEvent = arguments.currentEvent />
</cffunction>
```

setCurrentEventHandler

```
private void setCurrentEventHandler( EventHandler currentEventHandler )
```

Parameters:

EventHandler currentEventHandler

Code:

```
<cffunction name="setCurrentEventHandler" access="private" returntype="void" output="false">
    <cfargument name="currentEventHandler" type="MachII.framework.EventHandler" required="true" />
    <cfset variables.currentEventHandler = arguments.currentEventHandler />
</cffunction>
```

setEventMapping

```
public string setEventMapping( string eventName, string mappingName )
```

Sets an event mapping.

Parameters:

string eventName
string mappingName

Code:

```
<cffunction name="setEventMapping" access="public" returntype="string" output="false"
    hint="Sets an event mapping.">
    <cfargument name="eventName" type="string" required="true" />
    <cfargument name="mappingName" type="string" required="true" />
    <cfset variables.mappings[arguments.eventName] = arguments.mappingName />
</cffunction>
```

setEventQueue

```
private void setEventQueue( SizedQueue eventQueue )
```

Parameters:

SizedQueue eventQueue

Code:

```
<cffunction name="setEventQueue" access="private" returnType="void" output="false">
    <cfargument name="eventQueue" type="MachII.util.SizedQueue" required="true" />
    <cfset variables.eventQueue = arguments.eventQueue />
</cffunction>
```

setExceptionEventName

```
public void setExceptionEventName( string exceptionEventName )
```

Parameters:

string exceptionEventName

Code:

```
<cffunction name="setExceptionEventName" access="public" returnType="void" output="false">
    <cfargument name="exceptionEventName" type="string" required="true" />
    <cfset variables.exceptionEventName = arguments.exceptionEventName />
</cffunction>
```

setIsException

```
private void setIsException( boolean isException )
```

Parameters:

boolean isException

Code:

```
<cffunction name="setIsException" access="private" returntype="void" output="false">
  <cfargument name="isException" type="boolean" required="true" />
  <cfset variables.isException = arguments.isException />
</cffunction>
```

setIsProcessing

private void setIsProcessing(boolean isProcessing)

Parameters:

boolean isProcessing

Code:

```
<cffunction name="setIsProcessing" access="private" returntype="void" output="false">
  <cfargument name="isProcessing" type="boolean" required="true" />
  <cfset variables.isProcessing = arguments.isProcessing />
</cffunction>
```

setMaxEvents

public void setMaxEvents(numeric maxEvents)

Parameters:

numeric maxEvents

Code:

```
<cffunction name="setMaxEvents" access="public" returntype="void" output="false">
  <cfargument name="maxEvents" required="true" type="numeric" />
  <cfset variables.maxEvents = arguments.maxEvents />
</cffunction>
```

setPreviousEvent

private void setPreviousEvent(Event previousEvent)

Parameters:

Event previousEvent

Code:

```
<cffunction name="setPreviousEvent" access="private" returntype="void" output="false">
    <cfargument name="previousEvent" type="MachII.framework.Event" required="true" />
    <cfset variables.previousEvent = arguments.previousEvent />
</cffunction>
```

setRequestEventName

private void setRequestEventName(string requestEventName)

Parameters:

string requestEventName

Code:

```
<cffunction name="setRequestEventName" access="private" returntype="void" output="false">
    <cfargument name="requestEventName" type="string" required="true" />
    <cfset variables.requestEventName = arguments.requestEventName />
</cffunction>
```

setViewContext

private void setViewContext(ViewContext viewContext)

Parameters:

ViewContext viewContext

Code:

```
<cffunction name="setViewContext" access="private" returntype="void" output="false">
  <cfargument name="viewContext" type="MachII.framework.ViewContext" required="true" />
  <cfset variables.viewContext = arguments.viewContext />
</cffunction>
```

wrapException

private Exception wrapException(any caughtException)

Creates an exception object (with cfcatch).

Parameters:

any caughtException

Code:

```
<cffunction name="wrapException" access="private" returntype="MachII.util.Exception" output="false"
  hint="Creates an exception object (with cfcatch).">
  <cfargument name="caughtException" type="any" required="true" />

  <cfset var exception = CreateObject('component', 'MachII.util.Exception') />
  <cfset exception.wrapException(arguments.caughtException) />
  <cfset setIsException(true) />

  <cfreturn exception />
</cffunction>
```

EventFilter

Package: MachII.framework

Inherits from: framework.BaseComponent

Base EventFilter component.

Method Summary

public EventFilter	init(AppManager appManager, [struct parameters="#StructNew()#"])
	Used by the framework for initialization. Do not override.
public boolean	filterEvent(Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"])
	Override (be sure to keep the same arguments and returntype) to provide event filtering logic.

Methods inherited from framework.BaseComponent: isParameterDefined , setPropertyManager , setParameters , getPropertyManager , announceEvent , getAppManager , setAppManager , configure , hasParameter , getParameter , getProperty , getParameters , setProperty , setParameter

Method Detail

filterEvent

```
public boolean filterEvent( Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"] )
```

Override (be sure to keep the same arguments and returntype) to provide event filtering logic.

Parameters:

Event event
EventContext eventContext
[struct paramArgs="#StructNew()#"]

Code:

```
<cffunction name="filterEvent" access="public" returntype="boolean"
    hint="Override (be sure to keep the same arguments and returntype) to provide event filtering logic.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfargument name="paramArgs" type="struct" required="false" default="#StructNew()#" />

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

init

```
public EventFilter init( AppManager appManager, [struct parameters="#StructNew()#"] )
```

Used by the framework for initialization. Do not override.

Parameters:

AppManager appManager
[struct parameters="#StructNew()#"]

Code:

```
<cffunction name="init" access="public" returntype="EventFilter" output="false"
    hint="Used by the framework for initialization. Do not override.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfargument name="parameters" type="struct" required="false" default="#StructNew()#" />

    <cfset super.init(arguments.appManager, arguments.parameters) />

    <cfreturn this />
</cffunction>
```

EventHandler

Package: MachII.framework

Handles processing of EventCommands for an Event.

Method Summary

public EventHandler	init() Used by the framework for initialization. Do not override.
public void	addCommand(EventCommand command) Adds an EventCommand.
public string	getAccess()
public void	handleEvent(Event event, EventContext eventContext) Handles an Event.
public void	setAccess(string access)

Method Detail

addCommand

```
public void addCommand( EventCommand command )
```

Adds an EventCommand.

Parameters:

EventCommand command

Code:

```
<cffunction name="addCommand" access="public" returntype="void" output="false"
  hint="Adds an EventCommand.">
  <cfargument name="command" type="MachII.framework.EventCommand" required="true" />
  <cfset ArrayAppend(variables.commands, arguments.command) />
</cffunction>
```

getAccess

```
public string getAccess( )
```

Parameters:

Code:

```
<cffunction name="getAccess" access="public" returntype="string" output="false">
  <cfreturn variables.access />
</cffunction>
```

handleEvent

```
public void handleEvent( Event event, EventContext eventContext )
```

Handles an Event.

Parameters:

Event event
EventContext eventContext

Code:

```
<cffunction name="handleEvent" access="public" returntype="void" output="true"
  hint="Handles an Event.">
  <cfargument name="event" type="MachII.framework.Event" required="true" />
```

```
<cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

<cfset var continue = true />
<cfset var command = "" />
<cfset var i = 0 />

<cfloop index="i" from="1" to="#ArrayLen(variables.commands)#">
    <cfset command = variables.commands[i] />
    <cfset continue = command.execute(arguments.event, arguments.eventContext) />
    <cfif continue IS false>
        <cfbreak />
    </cfif>
</cfloop>
</cffunction>
```

init

```
public EventHandler init()
```

Used by the framework for initialization. Do not override.

Parameters:

Code:

```
<cffunction name="init" access="public" returnType="EventHandler" output="false"
    hint="Used by the framework for initialization. Do not override.">
    <cfreturn this />
</cffunction>
```

setAccess

```
public void setAccess( string access )
```

Parameters:

string access

Code:

```
<cffunction name="setAccess" access="public" returntype="void" output="false">  
  <cfargument name="access" type="string" required="true" />  
  <cfset variables.access = arguments.access />  
</cffunction>
```

EventManager

Package: MachII.framework

Manages registered EventHandlers for the framework.

Method Summary

public void	init(string configXML, AppManager appManager)	Initialization function called by the framework.
public void	addEventHandler(string eventName, EventHandler eventHandler)	Registers an EventHandler by name.
public void	configure()	Configures each of the registered EventHandlers/Events.
public Event	createEvent(string eventName, [struct eventArgs="#StructNew()#"], [string eventType="MachII.framework.Event"])	Creates an Event instance.
private EventCommand	createEventCommand(any commandNode)	
public AppManager	getAppManager()	
public EventHandler	getEventHandler(string eventName)	Returns the EventHandler for the named Event.
public boolean	isEventDefined(string eventName)	Returns true if an EventHandler for the named Event is defined; otherwise false.
public boolean	isEventPublic(string eventName)	Returns true if the EventHandler for the named Event is publicly accessible; otherwise false.
public void	setAppManager(AppManager appManager)	

Method Detail**addEventHandler**

```
public void addEventHandler( string eventName, EventHandler eventHandler )
```

Registers an EventHandler by name.

Parameters:

string eventName
EventHandler eventHandler

Code:

```
<cffunction name="addEventHandler" access="public" returntype="void" output="false"
    hint="Registers an EventHandler by name.">
    <cfargument name="eventName" type="string" required="true" />
    <cfargument name="eventHandler" type="MachII.framework.EventHandler" required="true" />

    <cfif isEventDefined(arguments.eventName)>
        <cfthrow type="MachII.framework.EventHandlerAlreadyDefined"
            message="An EventHandler with name '#arguments.eventName#' is already registered." />
    <cfelse>
        <cfset variables.handlers[eventName] = eventHandler />
    </cfif>
</cffunction>
```

configure

```
public void configure( )
```

Configures each of the registered EventHandlers/Events.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void"
    hint="Configures each of the registered EventHandlers/Events.">

</cffunction>
```

createEvent

```
public Event createEvent( string eventName, [struct eventArgs="#StructNew()#"], [string eventType="MachII.framework.Event"] )
```

Creates an Event instance.

Parameters:

```
string eventName
[struct eventArgs="#StructNew()#"]
[string eventType="MachII.framework.Event"]
```

Code:

```
<cffunction name="createEvent" access="public" returntype="MachII.framework.Event" output="true"
    hint="Creates an Event instance.">
    <cfargument name="eventName" type="string" required="true" />
    <cfargument name="eventArgs" type="struct" required="false" default="#StructNew()#" />
    <cfargument name="eventType" type="string" required="false" default="MachII.framework.Event" />

    <cfset var event = "" />

    <cfif isEventDefined(arguments.eventName)>
        <cfset event = CreateObject('component', arguments.eventType) />
        <cfset event.init(arguments.eventName, arguments.eventArgs) />
        <cfreturn event />
    <cfelse>
        <cfthrow type="MachII.framework.EventHandlerNotDefined"
            message="EventHandler for event '#arguments.eventName#' is not defined." />
    </cfif>
</cffunction>
```

createEventCommand

private EventCommand createEventCommand(any commandNode)

Parameters:

any commandNode

Code:

```
<cffunction name="createEventCommand" access="private" returntype="MachII.framework.EventCommand" output="false">
    <cfargument name="commandNode" required="true" />

    <cfset var eventCommand = "" />
    <cfset var filterName = "" />
    <cfset var filterParams = 0 />
    <cfset var paramNodes = 0 />
    <cfset var paramName = "" />
    <cfset var paramValue = "" />
    <cfset var filter = "" />
    <cfset var argName = "" />
    <cfset var argValue = "" />
    <cfset var argVariable = "" />
    <cfset var mappingEventName = "" />
    <cfset var mappingName = "" />
    <cfset var notifyListener = 0 />
    <cfset var notifyMethod = "" />
    <cfset var notifyResultKey = "" />
    <cfset var notifyResultArg = "" />
    <cfset var listener = "" />
    <cfset var eventName = "" />
    <cfset var copyEventArgs = 0 />
    <cfset var viewName = "" />
    <cfset var contentKey = "" />
    <cfset var contentArg = "" />
    <cfset var appendContent = 0 />
    <cfset var beanName = "" />
    <cfset var beanType = "" />
    <cfset var beanFields = "" />
    <cfset var redirectUrl = "" />
    <cfset var k = 0 />
```

```
<cfif commandNode.xmlName EQ "view-page">
  <cfset viewName = commandNode.xmlAttributes['name'] />
  <cfif StructKeyExists(commandNode.xmlAttributes, 'contentKey')>
    <cfset contentKey = commandNode.xmlAttributes['contentKey'] />
  <cfelse>
    <cfset contentKey = '' />
  </cfif>
  <cfif StructKeyExists(commandNode.xmlAttributes, 'contentArg')>
    <cfset contentArg = commandNode.xmlAttributes['contentArg'] />
  <cfelse>
    <cfset contentArg = '' />
  </cfif>
  <cfif StructKeyExists(commandNode.xmlAttributes, 'append')>
    <cfset appendContent = commandNode.xmlAttributes['append'] />
  <cfelse>
    <cfset appendContent = '' />
  </cfif>
  <cfset eventCommand = CreateObject('component', 'MachII.framework.commands.ViewPageCommand') />
  <cfset eventCommand.init(viewName, contentKey, contentArg, appendContent) />

<cfelseif commandNode.xmlName EQ "notify">
  <cfset notifyListener = commandNode.xmlAttributes['listener'] />
  <cfset notifyMethod = commandNode.xmlAttributes['method'] />
  <cfif StructKeyExists(commandNode.xmlAttributes, 'resultKey')>
    <cfset notifyResultKey = commandNode.xmlAttributes['resultKey'] />
  <cfelse>
    <cfset notifyResultKey = '' />
  </cfif>
  <cfif StructKeyExists(commandNode.xmlAttributes, 'resultArg')>
    <cfset notifyResultArg = commandNode.xmlAttributes['resultArg'] />
  <cfelse>
    <cfset notifyResultArg = '' />
  </cfif>
  <cfset listener = variables.listenerMgr.getListener(notifyListener) />
  <cfset eventCommand = CreateObject('component', 'MachII.framework.commands.NotifyCommand') />
  <cfset eventCommand.init(listener, notifyMethod, notifyResultKey, notifyResultArg) />

<cfelseif commandNode.xmlName EQ "announce">
  <cfset eventName = commandNode.xmlAttributes['event'] />
  <cfif StructKeyExists(commandNode.xmlAttributes, 'copyEventArgs')>
    <cfset copyEventArgs = commandNode.xmlAttributes['copyEventArgs'] />
  <cfelse>
    <cfset copyEventArgs = true />
  </cfif>
</cfif>
```

```
<cfset eventCommand = CreateObject('component', 'MachII.framework.commands.AnnounceCommand') />
<cfset eventCommand.init(eventName, copyEventArgs) />

<cfelseif commandNode.xmlName EQ "event-mapping">
  <cfset mappingEventName = commandNode.xmlAttributes['event'] />
  <cfset mappingName = commandNode.xmlAttributes['mapping'] />
  <cfset eventCommand = CreateObject('component', 'MachII.framework.commands.EventMappingCommand') />
  <cfset eventCommand.init(mappingEventName, mappingName) />

<cfelseif commandNode.xmlName EQ "filter">
  <cfset filterName = commandNode.xmlAttributes['name'] />
  <cfset filterParams = StructNew() />
  <cfset paramNodes = commandNode.xmlChildren />
  <cfloop from="1" to="#ArrayLen(paramNodes)#" index="k">
    <cfset paramName = paramNodes[k].xmlAttributes['name'] />
    <cfset paramValue = paramNodes[k].xmlAttributes['value'] />
    <cfset filterParams[paramName] = paramValue />
  </cfloop>
  <cfset filter = variables.filterMgr.getFilter(filterName) />
  <cfset eventCommand = CreateObject('component', 'MachII.framework.commands.FilterCommand') />
  <cfset eventCommand.init(filter, filterParams) />

<cfelseif commandNode.xmlName EQ "event-bean">
  <cfset beanName = commandNode.xmlAttributes['name'] />
  <cfset beanType = commandNode.xmlAttributes['type'] />
  <cfif StructKeyExists(commandNode.xmlAttributes, 'fields')>
    <cfset beanFields = commandNode.xmlAttributes['fields'] />
  <cfelse>
    <cfset beanFields = '' />
  </cfif>
  <cfset eventCommand = CreateObject('component', 'MachII.framework.commands.EventBeanCommand') />
  <cfset eventCommand.init(beanName, beanType, beanFields) />

<cfelseif commandNode.xmlName EQ "redirect">
  <cfset paramName = getAppManager().getPropertyManager().getProperty('eventParameter', 'event') />
  <cfif StructKeyExists(commandNode.xmlAttributes, 'event')>
    <cfset eventName = commandNode.xmlAttributes['event'] />
  </cfif>
  <cfif StructKeyExists(commandNode.xmlAttributes, 'url')>
    <cfset redirectUrl = commandNode.xmlAttributes['url'] />
  </cfif>
  <cfif StructKeyExists(commandNode.xmlAttributes, 'args')>
    <cfset argVariable = commandNode.xmlAttributes['args'] />
  </cfif>
```

```
        <cfset eventCommand = CreateObject('component', 'MachII.framework.commands.RedirectCommand') />
        <cfset eventCommand.init(eventName,paramName,redirectUrl,argVariable) />

    <cfelseif commandNode.xmlName EQ "event-arg">
        <cfset argName = commandNode.xmlAttributes['name'] />
        <cfif StructKeyExists(commandNode.xmlAttributes, 'value')>
            <cfset argValue = commandNode.xmlAttributes['value'] />
        <cfelse>
            <cfset argValue = "" />
        </cfif>
        <cfif StructKeyExists(commandNode.xmlAttributes, 'variable')>
            <cfset argVariable = commandNode.xmlAttributes['variable'] />
        <cfelse>
            <cfset argVariable = "" />
        </cfif>
        <cfset eventCommand = CreateObject('component', 'MachII.framework.commands.EventArgCommand') />
        <cfset eventCommand.init(argName, argValue, argVariable) />

    <cfelse>
        <cfset eventCommand = CreateObject('component', 'MachII.framework.EventCommand') />
        <cfset eventCommand.init() />
    </cfif>

    <cfreturn eventCommand />
</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Parameters:

Code:

```
<cffunction name="getAppManager" access="public" returntype="MachII.framework.AppManager" output="false">
    <cfreturn variables.appManager />
</cffunction>
```

getEventHandler

```
public EventHandler getEventHandler( string eventName )
```

Returns the EventHandler for the named Event.

Parameters:

string eventName

Code:

```
<cffunction name="getEventHandler" access="public" returnType="MachII.framework.EventHandler"
  hint="Returns the EventHandler for the named Event.">
  <cfargument name="eventName" type="string" required="true"
    hint="The name of the Event to handle." />

  <cfif isEventDefined(arguments.eventName)>
    <cfreturn variables.handlers[arguments.eventName] />
  <cfelse>
    <cfthrow type="MachII.framework.EventHandlerNotDefined"
      message="EventHandler for event '#arguments.eventName#' is not defined." />
  </cfif>
</cffunction>
```

init

```
public void init( string configXML, AppManager appManager )
```

Initialization function called by the framework.

Parameters:

string configXML
AppManager appManager

Code:

```
<cffunction name="init" access="public" returnType="void" output="false"
  hint="Initialization function called by the framework.">
  <cfargument name="configXML" type="string" required="true" />
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
```

```
<cfset var commandNodes = "" />
<cfset var commandNode = "" />
<cfset var eventNodes = "" />
<cfset var eventHandler = "" />
<cfset var eventAccess = "" />
<cfset var eventName = "" />
<cfset var eventCommand = "" />
<cfset var i = 0 />
<cfset var j = 0 />

<cfset setAppManager(arguments.appManager) />

<cfset variables.listenerMgr = arguments.appManager.getListenerManager() />
<cfset variables.filterMgr = arguments.appManager.getFilterManager() />

<cfset eventNodes = XMLSearch(configXML,"//event-handlers/event-handler") />
<cfloop from="1" to="#ArrayLen(eventNodes)#" index="i">
  <cfset eventName = eventNodes[i].xmlAttributes['event'] />
  <cfif StructKeyExists(eventNodes[i].xmlAttributes, 'access')>
    <cfset eventAccess = eventNodes[i].xmlAttributes['access'] />
  <cfelse>
    <cfset eventAccess = 'public' />
  </cfif>

  <cfset eventHandler = CreateObject('component', 'MachII.framework.EventHandler') />
  <cfset eventHandler.init() />
  <cfset eventHandler.setAccess(eventAccess) />

  <cfloop from="1" to="#ArrayLen(eventNodes[i].XMLChildren)#" index="j">
    <cfset commandNode = eventNodes[i].XMLChildren[j] />
    <cfset eventCommand = createEventCommand(commandNode) />
    <cfset eventHandler.addCommand(eventCommand) />
  </cfloop>

  <cfset addEventHandler(eventName, eventHandler) />
</cfloop>

<cfset variables.listenerMgr = "" />
<cfset variables.filterMgr = "" />
</cffunction>
```

isEventDefined

```
public boolean isEventDefined( string eventName )
```

Returns true if an EventHandler for the named Event is defined; otherwise false.

Parameters:

string eventName

Code:

```
<cffunction name="isEventDefined" access="public" returntype="boolean" output="false"
    hint="Returns true if an EventHandler for the named Event is defined; otherwise false.">
    <cfargument name="eventName" type="string" required="true"
        hint="The name of the Event to handle." />
    <cfreturn StructKeyExists(variables.handlers, arguments.eventName) />
</cffunction>
```

isEventPublic

```
public boolean isEventPublic( string eventName )
```

Returns true if the EventHandler for the named Event is publicly accessible; otherwise false.

Parameters:

string eventName

Code:

```
<cffunction name="isEventPublic" access="public" returntype="boolean" output="false"
    hint="Returns true if the EventHandler for the named Event is publicly accessible; otherwise false.">
    <cfargument name="eventName" type="string" required="true" />
    <cfset var eventHandler = "" />
    <cfset eventHandler = getEventHandler(arguments.eventName) />
    <cfreturn eventHandler.getAccess() EQ 'public' />
</cffunction>
```

setAppManager

public void setAppManager(AppManager appManager)

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="public" returntype="void" output="false">  
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />  
  <cfset variables.appManager = arguments.appManager />  
</cffunction>
```

FilterManager

Package: MachII.framework

Manages registered EventFilters for the framework.

Method Summary

public void	init(string configXML, AppManager appManager)	Initialization function called by the framework.
public void	addFilter(string filterName, EventFilter filter)	Registers an EventFilter by name.
public void	configure()	Configures each of the registered EventFilters.
public AppManager	getAppManager()	
public EventFilter	getFilter(string filterName)	
public array	getFilterNames()	Returns an array of filter names.
public boolean	isFilterDefined(string filterName)	
public void	setAppManager(AppManager appManager)	

Method Detail

addFilter

```
public void addFilter( string filterName, EventFilter filter )
```

Registers an EventFilter by name.

Parameters:

string filterName
EventFilter filter

Code:

```
<cffunction name="addFilter" access="public" returntype="void" output="false"
  hint="Registers an EventFilter by name.">
  <cfargument name="filterName" type="string" required="true" />
  <cfargument name="filter" type="MachII.framework.EventFilter" required="true" />

  <cfif isFilterDefined(arguments.filterName)>
    <cfthrow type="MachII.framework.FilterAlreadyDefined"
      message="An EventFilter with name '#arguments.filterName#' is already registered." />
  <cfelse>
    <cfset variables.filters[arguments.filterName] = arguments.filter />
  </cfif>
</cffunction>
```

configure

public void configure()

Configures each of the registered EventFilters.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void"
  hint="Configures each of the registered EventFilters.">
  <cfset var key = "" />
  <cfloop collection="#variables.filters#" item="key">
    <cfset getFilter(key).configure() />
  </cfloop>
</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Parameters:

Code:

```
<cffunction name="getAppManager" access="public" returntype="MachII.framework.AppManager" output="false">
    <cfreturn variables.appManager />
</cffunction>
```

getFilter

```
public EventFilter getFilter( string filterName )
```

Parameters:

string filterName

Code:

```
<cffunction name="getFilter" access="public" returntype="MachII.framework.EventFilter" output="false">
    <cfargument name="filterName" type="string" required="true" />

    <cfif isFilterDefined(arguments.filterName)>
        <cfreturn variables.filters[arguments.filterName] />
    <cfelse>
        <cfthrow type="MachII.framework.FilterNotDefined"
            message="Filter with name '#arguments.filterName#' is not defined." />
    </cfif>
</cffunction>
```

getFilterNames

```
public array getFilterNames( )
```

Returns an array of filter names.

Parameters:

Code:

```
<cffunction name="getFilterNames" access="public" returntype="array" output="false"
    hint="Returns an array of filter names.">
    <cfreturn StructKeyArray(variables.filters) />
</cffunction>
```

init

```
public void init( string configXML, AppManager appManager )
```

Initialization function called by the framework.

Parameters:

```
string configXML
AppManager appManager
```

Code:

```
<cffunction name="init" access="public" returntype="void" output="false"
    hint="Initialization function called by the framework.">
    <cfargument name="configXML" type="string" required="true" />
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />

    <cfset var filterNodes = "" />
    <cfset var filterParams = "" />
    <cfset var name = "" />
    <cfset var type = "" />
    <cfset var paramNodes = "" />
    <cfset var paramName = "" />
    <cfset var paramValue = "" />
    <cfset var filter = "" />
    <cfset var i = 0 />
```

```
<cfset var j = 0 />

<cfset setAppManager(arguments.appManager) />

<cfset filterNodes = XMLSearch(configXML,"//event-filters/event-filter") />
<cfloop from="1" to="#ArrayLen(filterNodes)#" index="i">
    <cfset name = filterNodes[i].xmlAttributes['name'] />
    <cfset type = filterNodes[i].xmlAttributes['type'] />

    <cfset filterParams = StructNew() />
    <cfset paramNodes = XMLSearch(filterNodes[i], "./parameters/parameter") />
    <cfloop from="1" to="#ArrayLen(paramNodes)#" index="j">
        <cfset paramName = paramNodes[j].xmlAttributes['name'] />
        <cfset paramValue = paramNodes[j].xmlAttributes['value'] />
        <cfset filterParams[paramName] = paramValue />
    </cfloop>

    <cfset filter = CreateObject('component', type) />
    <cfset filter.init(arguments.appManager, filterParams) />

    <cfset addFilter(name, filter) />
</cfloop>
</cffunction>
```

isFilterDefined

public boolean isFilterDefined(string filterName)

Parameters:

string filterName

Code:

```
<cffunction name="isFilterDefined" access="public" returntype="boolean" output="false">
    <cfargument name="filterName" type="string" required="true" />
    <cfreturn StructKeyExists(variables.filters, arguments.filterName) />
</cffunction>
```

setAppManager

public void setAppManager(AppManager appManager)

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="public" returntype="void" output="false">
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
  <cfset variables.appManager = arguments.appManager />
</cffunction>
```

Listener

Package: MachII.framework

Inherits from: framework.BaseComponent

Base Listener component.

Method Summary

public Listener	init(AppManager appManager, [struct parameters="#StructNew()#"], [ListenerInvoker invoker]) Used by the framework for initialization. Do not override.
public ListenerInvoker	getDefaultInvoker() Returns an instance of the default invoker (EventInvoker) for this Listener.
public any	getInvoker() Gets the ListenerInvoker to use when invoking methods for this Listener.
public void	setInvoker(ListenerInvoker invoker) Sets the ListenerInvoker to use when invoking methods for this Listener.

Methods inherited from framework.BaseComponent: isParameterDefined , setPropertyManager , setParameters , getPropertyManager , announceEvent , getAppManager , setAppManager , configure , hasParameter , getParameter , getProperty , getParameters , setProperty , setParameter

Method Detail

getDefaultInvoker

public ListenerInvoker getDefaultInvoker()

Returns an instance of the default invoker (EventInvoker) for this Listener.

Parameters:

Code:

```
<cffunction name="getDefaultInvoker" access="public" returnType="MachII.framework.ListenerInvoker" output="false"
    hint="Returns an instance of the default invoker (EventInvoker) for this Listener.">
    <cfreturn CreateObject('component', 'MachII.framework.invokers.EventInvoker').init() />
</cffunction>
```

getInvoker

public any getInvoker()

Gets the ListenerInvoker to use when invoking methods for this Listener.

Parameters:

Code:

```
<cffunction name="getInvoker" access="public" type="MachII.framework.ListenerInvoker" output="false"
    hint="Gets the ListenerInvoker to use when invoking methods for this Listener.">
    <cfreturn variables.invoker />
</cffunction>
```

init

public Listener init(AppManager appManager, [struct parameters="#StructNew()#"], [ListenerInvoker invoker])

Used by the framework for initialization. Do not override.

Parameters:

AppManager appManager

```
[struct parameters="#StructNew()#"]  
[ListenerInvoker invoker]
```

Code:

```
<cffunction name="init" access="public" returntype="Listener" output="false"  
    hint="Used by the framework for initialization. Do not override.">  
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />  
    <cfargument name="parameters" type="struct" required="false" default="#StructNew()#" />  
    <cfargument name="invoker" type="MachII.framework.ListenerInvoker" required="false" />  
  
    <cfset super.init(arguments.appManager, arguments.parameters) />  
  
    <cfif StructKeyExists(arguments, 'invoker')>  
        <cfset setInvoker(arguments.invoker) />  
    </cfif>  
  
    <cfreturn this />  
</cffunction>
```

setInvoker

```
public void setInvoker( ListenerInvoker invoker )
```

Sets the ListenerInvoker to use when invoking methods for this Listener.

Parameters:

ListenerInvoker invoker

Code:

```
<cffunction name="setInvoker" access="public" returntype="void" output="false"  
    hint="Sets the ListenerInvoker to use when invoking methods for this Listener.">  
    <cfargument name="invoker" type="MachII.framework.ListenerInvoker" required="true" />  
    <cfset variables.invoker = arguments.invoker />  
</cffunction>
```

ListenerInvoker

Package: MachII.framework

Base Invoker component.

Method Summary

public ListenerInvoker	init() Initialization function called by the framework.
public void	invokeListener(Event event, any listener, string method, [string resultKey=""]) Invokes the target Listener with the Event.

Method Detail

init

```
public ListenerInvoker init( )
```

Initialization function called by the framework.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="ListenerInvoker" output="false"
    hint="Initialization function called by the framework.">
    <cfreturn this />
</cffunction>
```

invokeListener

```
public void invokeListener( Event event, any listener, string method, [string resultKey=""] )
```

Invokes the target Listener with the Event.

Parameters:

Event event
any listener
string method
[string resultKey=""]

Code:

```
<cffunction name="invokeListener" access="public" returntype="void"
  hint="Invokes the target Listener with the Event.">
  <cfargument name="event" type="MachII.framework.Event" required="true"
    hint="The Event triggering the invocation." />
  <cfargument name="listener" required="true"
    hint="The Listener to invoke." />
  <cfargument name="method" type="string" required="true"
    hint="The name of the Listener's method to invoke." />
  <cfargument name="resultKey" type="string" required="false" default=""
    hint="The result key." />

</cffunction>
```

ListenerManager

Package: MachII.framework

Manages registered Listeners for the framework instance.

Method Summary

public ListenerManager	init(string configXML, AppManager appManager) Initialization function called by the framework.
public void	addListener(string listenerName, Listener listener) Registers a Listener with the specified name.
public void	configure() Configures each of the registered Listeners.
public AppManager	getAppManager() Sets the AppManager instance this ListenerManager belongs to.
public Listener	getListener(string listenerName) Gets a Listener with the specified name.
public array	getListenerNames() Returns an array of listener names.
public boolean	isListenerDefined(string listenerName) Returns true if a Listener is registered with the specified name.
public void	setAppManager(AppManager appManager) Returns the AppManager instance this ListenerManager belongs to.

Method Detail**addListener**

```
public void addListener( string listenerName, Listener listener )
```

Registers a Listener with the specified name.

Parameters:

string listenerName

Listener listener

Code:

```
<cffunction name="addListener" access="public" returntype="void" output="false"
  hint="Registers a Listener with the specified name.">
  <cfargument name="listenerName" type="string" required="true" />
  <cfargument name="listener" type="MachII.framework.Listener" required="true" />

  <cfif isListenerDefined(arguments.listenerName)>
    <cfthrow type="MachII.framework.ListenerAlreadyDefined"
      message="A Listener with name '#arguments.listenerName#' is already registered." />
  <cfelse>
    <cfset variables.listeners[arguments.listenerName] = arguments.listener />
  </cfif>
</cffunction>
```

configure

```
public void configure( )
```

Configures each of the registered Listeners.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void"
    hint="Configures each of the registered Listeners.">
    <cfset var key = "" />
    <cfloop collection="#variables.listeners#" item="key">
        <cfset getListener(key).configure() />
    </cfloop>
</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Sets the AppManager instance this ListenerManager belongs to.

Parameters:

Code:

```
<cffunction name="getAppManager" access="public" returntype="MachII.framework.AppManager" output="false"
    hint="Sets the AppManager instance this ListenerManager belongs to.">
    <cfreturn variables.appManager />
</cffunction>
```

getListener

```
public Listener getListener( string listenerName )
```

Gets a Listener with the specified name.

Parameters:

```
string listenerName
```

Code:

```
<cffunction name="getListener" access="public" returntype="MachII.framework.Listener" output="false"
    hint="Gets a Listener with the specified name.">
```

```
<cfargument name="listenerName" type="string" required="true" />
<cfif isListenerDefined(arguments.listenerName)>
    <cfreturn variables.listeners[arguments.listenerName] />
<cfelse>
    <cfthrow type="MachII.framework.ListenerNotDefined"
            message="Listener with name '#arguments.listenerName#' is not defined." />
</cfif>
</cffunction>
```

getListenerNames

public array getListenerNames()

Returns an array of listener names.

Parameters:

Code:

```
<cffunction name="getListenerNames" access="public" returntype="array" output="false"
            hint="Returns an array of listener names.">
    <cfreturn StructKeyArray(variables.listeners) />
</cffunction>
```

init

public ListenerManager init(string configXML, AppManager appManager)

Initialization function called by the framework.

Parameters:

string configXML
AppManager appManager

Code:

```
<cffunction name="init" access="public" returnType="ListenerManager" output="false"
  hint="Initialization function called by the framework.">
  <cfargument name="configXML" type="string" required="true" />
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />

  <cfset var listenerNodes = "" />
  <cfset var listenerParams = "" />
  <cfset var name = "" />
  <cfset var type = "" />
  <cfset var paramNodes = "" />
  <cfset var paramName = "" />
  <cfset var paramValue = "" />
  <cfset var invokerNodes = "" />
  <cfset var invokerType = "" />
  <cfset var invoker = "" />
  <cfset var listener = "" />
  <cfset var i = 0 />
  <cfset var j = 0 />

  <cfset setAppManager(arguments.appManager) />

  <cfset listenerNodes = XMLSearch(configXML,"//listeners/listener") />
  <cfloop from="1" to="#ArrayLen(listenerNodes)#" index="i">
    <cfset name = listenerNodes[i].xmlAttributes['name'] />
    <cfset type = listenerNodes[i].xmlAttributes['type'] />

    <cfset listenerParams = StructNew() />
    <cfset paramNodes = XMLSearch(listenerNodes[i], "./parameters/parameter") />
    <cfloop from="1" to="#ArrayLen(paramNodes)#" index="j">
      <cfset paramName = paramNodes[j].xmlAttributes['name'] />
      <cfset paramValue = paramNodes[j].xmlAttributes['value'] />
      <cfset listenerParams[paramName] = paramValue />
    </cfloop>

    <cfset listener = CreateObject('component', type) />
    <cfset listener.init(arguments.appManager, listenerParams) />

    <cfset invokerNodes = XMLSearch(listenerNodes[i], "./invoker") />
    <cfif arrayLen(invokerNodes)>
```

```
        <cfset invokerType = invokerNodes[1].xmlAttributes['type'] />
        <cfset invoker = CreateObject('component', invokerType) />
        <cfset invoker.init() />

    <cfelse>
        <cfset invoker = listener.getDefaultInvoker() >
    </cfif>

    <cfset listener.setInvoker(invoker) />

    <cfset addListener(name, listener) />
</cfloop>

    <cfreturn this />
</cffunction>
```

isListenerDefined

```
public boolean isListenerDefined( string listenerName )
```

Returns true if a Listener is registered with the specified name.

Parameters:

string listenerName

Code:

```
<cffunction name="isListenerDefined" access="public" returntype="boolean" output="false"
    hint="Returns true if a Listener is registered with the specified name.">
    <cfargument name="listenerName" type="string" required="true" />
    <cfreturn StructKeyExists(variables.listeners, arguments.listenerName) />
</cffunction>
```

setAppManager

public void setAppManager(AppManager appManager)

Returns the AppManager instance this ListenerManager belongs to.

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="public" returntype="void" output="false"
    hint="Returns the AppManager instance this ListenerManager belongs to.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfset variables.appManager = arguments.appManager />
</cffunction>
```

Plugin

Package: MachII.framework

Inherits from: framework.BaseComponent

Base Plugin component.

Method Summary

public Plugin	init(AppManager appManager, [struct parameters]) Used by the framework for initialization. Do not override.
public void	abortEvent([string message=""]) Call this function to abort processing of the current event. When called, an AbortEventException exception is thrown, caught, and handled by the framework.
public void	handleException(EventContext eventContext, Exception exception) Plugin point called when an exception occurs (before exception event is handled). Override to provide custom functionality.
public void	postEvent(EventContext eventContext) Plugin point called after each Event is processed. Override to provide custom functionality.
public void	postProcess(EventContext eventContext) Plugin point called after Event processing finishes. Override to provide custom functionality.
public void	postView(EventContext eventContext) Plugin point called after each View is processed. Override to provide custom functionality.
public void	preEvent(EventContext eventContext) Plugin point called before each Event is processed. Override to provide custom functionality.
public void	preProcess(EventContext eventContext)

Method Summary

	Plugin point called before Event processing begins. Override to provide custom functionality.
public void	preView(EventContext eventContext)
	Plugin point called before each View is processed. Override to provide custom functionality.

Methods inherited from framework.BaseComponent: isParameterDefined , setPropertyManager , setParameters , getPropertyManager , announceEvent , getAppManager , setAppManager , configure , hasParameter , getParameter , getProperty , getParameters , setProperty , setParameter

Method Detail**abortEvent**

```
public void abortEvent( [string message=""] )
```

Call this function to abort processing of the current event. When called, an AbortEventException exception is thrown, caught, and handled by the framework.

Parameters:

```
[string message=""]
```

Code:

```
<cffunction name="abortEvent" access="public" returnType="void" output="false"
  hint="Call this function to abort processing of the current event. When called, an AbortEventException exception
  <cfargument name="message" type="string" required="false" default="" />
  <cfthrow type="AbortEventException" message="#arguments.message#" />
</cffunction>
```

handleException

```
public void handleException( EventContext eventContext, Exception exception )
```

Plugin point called when an exception occurs (before exception event is handled). Override to provide custom functionality.

Parameters:

EventContext eventContext
Exception exception

Code:

```
<cffunction name="handleException" access="public" returntype="void" output="true"
    hint="Plugin point called when an exception occurs (before exception event is handled). Override to provide custom
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
        hint="The EventContext under which the exception was thrown/caught." />
    <cfargument name="exception" type="MachII.util.Exception" required="true"
        hint="The Exception that was thrown/caught by the framework." />
</cffunction>
```

init

```
public Plugin init( AppManager appManager, [struct parameters] )
```

Used by the framework for initialization. Do not override.

Parameters:

AppManager appManager
[struct parameters]

Code:

```
<cffunction name="init" access="public" returntype="Plugin" output="false"
    hint="Used by the framework for initialization. Do not override.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfargument name="parameters" type="struct" required="false" />

    <cfset super.init(arguments.appManager, arguments.parameters) />
```

```
<cfreturn this />
</cffunction>
```

postEvent

```
public void postEvent( EventContext eventContext )
```

Plugin point called after each Event is processed. Override to provide custom functionality.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postEvent" access="public" returntype="void" output="true"
    hint="Plugin point called after each Event is processed. Override to provide custom functionality.">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
        hint="The EventContext the Event occurred in. Call arguments.eventContext.getCurrentEvent() to access th
    </cffunction>
```

postProcess

```
public void postProcess( EventContext eventContext )
```

Plugin point called after Event processing finishes. Override to provide custom functionality.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postProcess" access="public" returntype="void" output="true"
    hint="Plugin point called after Event processing finishes. Override to provide custom functionality.">
```

```
<cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
            hint="The EventContext of the processing." />

</cffunction>
```

postView

```
public void postView( EventContext eventContext )
```

Plugin point called after each View is processed. Override to provide custom functionality.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postView" access="public" returntype="void" output="true"
            hint="Plugin point called after each View is processed. Override to provide custom functionality.">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
                hint="The EventContext of the processing." />

</cffunction>
```

preEvent

```
public void preEvent( EventContext eventContext )
```

Plugin point called before each Event is processed. Override to provide custom functionality.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preEvent" access="public" returntype="void" output="true"
```

```
        hint="Plugin point called before each Event is processed. Override to provide custom functionality.">
        <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
            hint="The EventContext the Event occurred in. Call arguments.eventContext.getCurrentEvent() to access th
    </cfargument>
    </cffunction>
```

preProcess

```
public void preProcess( EventContext eventContext )
```

Plugin point called before Event processing begins. Override to provide custom functionality.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preProcess" access="public" returntype="void" output="true"
    hint="Plugin point called before Event processing begins. Override to provide custom functionality.">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
        hint="The EventContext of the processing." />
</cffunction>
```

preView

```
public void preView( EventContext eventContext )
```

Plugin point called before each View is processed. Override to provide custom functionality.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preView" access="public" returntype="void" output="true"
    hint="Plugin point called before each View is processed. Override to provide custom functionality.">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
        hint="The EventContext of the processing." />

</cffunction>
```

PluginManager

Package: MachII.framework

Manages registered Plugins for the framework instance.

Method Summary

public void	init(string configXML, AppManager appManager)	Initialization function called by the framework.
public void	addPlugin(string pluginName, Plugin plugin)	Registers a plugin with the specified name.
public void	configure()	Configures each of the registered Plugins.
private string	findPluginPoints(Plugin plugin)	Finds the registered plugin points in a plugin.
private array	gatherPluginMetaData(struct metadata, array points)	Gathers meta data about a plugin.
public AppManager	getAppManager()	Returns the AppManager instance this PluginManager belongs to.
public Plugin	getPlugin(string pluginName)	Gets a plugin with the specified name.
public array	getPluginNames()	Returns an array of plugin names.
public void	handleException(EventContext eventContext, Exception exception)	

Method Summary

	handleException() is called for each exception caught by the framework.
public boolean	isPluginDefined(string pluginName) Returns true if a Plugin is registered with the specified name.
public void	postEvent(EventContext eventContext) postEvent() is called for each announced Event after it has been handled.
public void	postProcess(EventContext eventContext) postProcess() is called for each new EventContext once after event processing completes.
public void	postView(EventContext eventContext) postView() is called for each announced Event after it has been handled.
public void	preEvent(EventContext eventContext) preEvent() is called for each announced Event before it is handled.
public void	preProcess(EventContext eventContext) preProcess() is called for each new EventContext once before event processing begins.
public void	preView(EventContext eventContext) preView() is called for each announced Event after it has been handled.
public void	setAppManager(AppManager appManager) Sets the AppManager instance this PluginManager belongs to.

Method Detail**addPlugin**

```
public void addPlugin( string pluginName, Plugin plugin )
```

Registers a plugin with the specified name.

Parameters:

string pluginName
Plugin plugin

Code:

```
<cffunction name="addPlugin" access="public" returntype="void" output="false"
  hint="Registers a plugin with the specified name.">
  <cfargument name="pluginName" type="string" required="true" />
  <cfargument name="plugin" type="MachII.framework.Plugin" required="true" />
  <cfset var i = 0 />
  <cfset var pointName = 0 />
  <cfset var pluginRegisteredPoints = listToArray(findPluginPoints(arguments.plugin)) />

  <cfif isPluginDefined(arguments.pluginName)>
    <cfthrow type="MachII.framework.PluginAlreadyDefined"
      message="A Plugin with name '#arguments.pluginName#' is already registered." />
  <cfelse>
    <cfset variables.plugins[arguments.pluginName] = arguments.plugin />

    <cfset variables.nPlugins = variables.nPlugins + 1 />
    <cfset variables.pluginArray[variables.nPlugins] = arguments.plugin />

    <cfloop index="i" from="1" to="#arraylen(pluginRegisteredPoints)#">
      <cfset pointName = pluginRegisteredPoints[i] />
      <cfif structKeyExists(variables,pointName & "Plugins")>
        <cfset arrayAppend(variables[pointName & "Plugins" ], arguments.plugin) />
      </cfif>
    </cfloop>
  </cfif>
</cffunction>
```

configure

```
public void configure( )
```

Configures each of the registered Plugins.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void"
    hint="Configures each of the registered Plugins.">
    <cfset var aPlugin = 0 />
    <cfset var i = 0 />
    <cfloop index="i" from="1" to="#variables.nPlugins#">
        <cfset aPlugin = variables.pluginArray[i] />
        <cfset aPlugin.configure() />
    </cfloop>
</cffunction>
```

findPluginPoints

```
private string findPluginPoints( Plugin plugin )
```

Finds the registered plugin points in a plugin.

Parameters:

Plugin plugin

Code:

```
<cffunction name="findPluginPoints" access="private" returntype="string" output="false"
    hint="Finds the registered plugin points in a plugin.">
    <cfargument name="plugin" type="MachII.framework.Plugin" required="true" />
    <cfset var md = getMetaData(arguments.plugin) />
    <cfset var pointArray = arraynew(1) />
    <cfset var returnList = "" />
    <cfset var i = 0 />

    <cfset pointArray = gatherPluginMetaData(md,pointArray) />
```

```

    <cfloop index="i" from="1" to="#arraylen(md.functions)#">
        <cfset arrayAppend(pointArray, md.functions[i].name) />
    </cfloop>

    <cfloop index="i" from="1" to="#arraylen(pointArray)#">
        <cfif not listFindNoCase(returnList, pointArray[i])>
            <cfset returnList = listAppend(returnList, pointArray[i]) />
        </cfif>
    </cfloop>

    <cfreturn returnList />
</cffunction>

```

gatherPluginMetaData

private array gatherPluginMetaData(struct metadata, array points)

Gathers meta data about a plugin.

Parameters:

struct metadata
array points

Code:

```

<cffunction name="gatherPluginMetaData" access="private" returntype="array" output="false"
    hint="Gathers meta data about a plugin.">
    <cfargument name="metadata" type="struct" required="true" />
    <cfargument name="points" type="array" required="true" />
    <cfset var i = 0 />

    <cfif structKeyExists(arguments.metadata, "extends") and arguments.metadata.extends.name neq "MachII.framework.D"
        <cfloop index="i" from="1" to="#arraylen(arguments.metadata.extends.functions)#">
            <cfset arrayAppend(arguments.points, arguments.metadata.extends.functions[i].name) />
        </cfloop>
        <cfset gatherPluginMetaData(arguments.metadata.extends, arguments.points) />
    </cfif>

```

```
<cfreturn arguments.points />
</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Returns the AppManager instance this PluginManager belongs to.

Parameters:

Code:

```
<cffunction name="getAppManager" access="public" returntype="MachII.framework.AppManager" output="false"
    hint="Returns the AppManager instance this PluginManager belongs to.">
    <cfreturn variables.appManager />
</cffunction>
```

getPlugin

```
public Plugin getPlugin( string pluginName )
```

Gets a plugin with the specified name.

Parameters:

string pluginName

Code:

```
<cffunction name="getPlugin" access="public" returntype="MachII.framework.Plugin" output="false"
    hint="Gets a plugin with the specified name.">
    <cfargument name="pluginName" type="string" required="true" />

    <cfif isPluginDefined(arguments.pluginName)>
        <cfreturn variables.plugins[arguments.pluginName] />
    <cfelse>
```

```
<cfthrow type="MachII.framework.PluginNotDefined"
        message="Plugin with name '#arguments.pluginName#' is not defined." />
</cfif>
</cffunction>
```

getPluginNames

```
public array getPluginNames( )
```

Returns an array of plugin names.

Parameters:

Code:

```
<cffunction name="getPluginNames" access="public" returntype="array" output="false"
        hint="Returns an array of plugin names.">
    <cfreturn StructKeyArray(variables.plugins) />
</cffunction>
```

handleException

```
public void handleException( EventContext eventContext, Exception exception )
```

handleException() is called for each exception caught by the framework.

Parameters:

EventContext eventContext

Exception exception

Code:

```
<cffunction name="handleException" access="public" returntype="void" output="true"
        hint="handleException() is called for each exception caught by the framework.">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
        hint="The EventContext under which the exception was thrown/caught." />
```

```
<cfargument name="exception" type="MachII.util.Exception" required="true"
    hint="The Exception object." />
<cfset var i = 0 />

<cfloop index="i" from="1" to="#arrayLen(variables.handleExceptionPlugins)#">
    <cfset variables.handleExceptionPlugins[i].handleException(arguments.eventContext, arguments.exception) />
</cfloop>
</cffunction>
```

init

```
public void init( string configXML, AppManager appManager )
```

Initialization function called by the framework.

Parameters:

```
string configXML
AppManager appManager
```

Code:

```
<cffunction name="init" access="public" returnType="void" output="false"
    hint="Initialization function called by the framework.">
    <cfargument name="configXML" type="string" required="true" />
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />

    <cfset var xnPlugins = 0 />
    <cfset var xnParams = 0 />
    <cfset var i = 0 />
    <cfset var j = 0 />
    <cfset var paramName = 0 />
    <cfset var paramValue = 0 />
    <cfset var plugin = 0 />
    <cfset var pluginName = 0 />
    <cfset var pluginType = 0 />
    <cfset var pluginParams = 0 />

    <cfset setAppManager(arguments.appManager) />
```

```

<cfset xnPlugins = XMLSearch(arguments.configXML, "//plugins/plugin" ) />
<cfloop index="i" from="1" to="#ArrayLen(xnPlugins)#">
  <cfset pluginName = xnPlugins[i].XmlAttributes['name'] />
  <cfset pluginType = xnPlugins[i].XmlAttributes['type'] />

  <cfset pluginParams = StructNew() />
  <cfset xnParams = XMLSearch(xnPlugins[i], "./parameters/parameter") />
  <cfloop index="j" from="1" to="#ArrayLen(xnParams)#">
    <cfset paramName = xnParams[j].XmlAttributes['name'] />
    <cfset paramValue = xnParams[j].XmlAttributes['value'] />

    <cfset StructInsert(pluginParams, paramName, paramValue, true) />
  </cfloop>

  <cfset plugin = CreateObject('component', pluginType) />
  <cfset plugin.init(arguments.appManager, pluginParams) />
  <cfset addPlugin(pluginName, plugin) />
</cfloop>
</cffunction>

```

isPluginDefined

public boolean isPluginDefined(string pluginName)

Returns true if a Plugin is registered with the specified name.

Parameters:

string pluginName

Code:

```

<cffunction name="isPluginDefined" access="public" returntype="boolean" output="false"
  hint="Returns true if a Plugin is registered with the specified name.">
  <cfargument name="pluginName" type="string" required="true" />
  <cfreturn StructKeyExists(variables.plugins, arguments.pluginName) />
</cffunction>

```

postEvent

```
public void postEvent( EventContext eventContext )
```

postEvent() is called for each announced Event after it has been handled.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postEvent" access="public" returntype="void" output="true"
    hint="postEvent() is called for each announced Event after it has been handled.">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
        hint="The EventContext the Event occurred in. Call arguments.eventContext.getCurrentEvent() to access the
    <cfset var i = 0 />

    <cfloop index="i" from="1" to="#arrayLen(variables.postEventPlugins)#">
        <cfset variables.postEventPlugins[i].postEvent(arguments.eventContext) />
    </cfloop>
</cffunction>
```

postProcess

```
public void postProcess( EventContext eventContext )
```

postProcess() is called for each new EventContext once after event processing completes.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postProcess" access="public" returntype="void" output="true"
    hint="postProcess() is called for each new EventContext once after event processing completes.">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
```

```
        hint="The EventContext of the processing." />
    <cfset var i = 0 />

    <cfloop index="i" from="1" to="#arrayLen(variables.postProcessPlugins)#">
        <cfset variables.postProcessPlugins[i].postProcess(arguments.eventContext) />
    </cfloop>
</cffunction>
```

postView

```
public void postView( EventContext eventContext )
```

postView() is called for each announced Event after it has been handled.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postView" access="public" returntype="void" output="true"
    hint="postView() is called for each announced Event after it has been handled.">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
        hint="The EventContext of the processing." />
    <cfset var i = 0 />

    <cfloop index="i" from="1" to="#arrayLen(variables.postViewPlugins)#">
        <cfset variables.postViewPlugins[i].postView(arguments.eventContext) />
    </cfloop>
</cffunction>
```

preEvent

```
public void preEvent( EventContext eventContext )
```

preEvent() is called for each announced Event before it is handled.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preEvent" access="public" returnType="void" output="true"
  hint="preEvent() is called for each announced Event before it is handled.">
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
    hint="The EventContext the Event occurred in. Call arguments.eventContext.getCurrentEvent() to access the
  <cfset var i = 0 />

  <cfloop index="i" from="1" to="#arrayLen(variables.preEventPlugins)#">
    <cfset variables.preEventPlugins[i].preEvent(arguments.eventContext) />
  </cfloop>
</cffunction>
```

preProcess

```
public void preProcess( EventContext eventContext )
```

preProcess() is called for each new EventContext once before event processing begins.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preProcess" access="public" returnType="void" output="true"
  hint="preProcess() is called for each new EventContext once before event processing begins.">
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
    hint="The EventContext of the processing." />
  <cfset var i = 0 />

  <cfloop index="i" from="1" to="#arrayLen(variables.preProcessPlugins)#">
    <cfset variables.preProcessPlugins[i].preProcess(arguments.eventContext) />
  </cfloop>
</cffunction>
```

preView

```
public void preView( EventContext eventContext )
```

preView() is called for each announced Event after it has been handled.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preView" access="public" returntype="void" output="true"
  hint="preView() is called for each announced Event after it has been handled.">
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true"
    hint="The EventContext of the processing." />
  <cfset var i = 0 />

  <cfloop index="i" from="1" to="#arrayLen(variables.preViewPlugins)#">
    <cfset variables.preViewPlugins[i].preView(arguments.eventContext) />
  </cfloop>
</cffunction>
```

setAppManager

```
public void setAppManager( AppManager appManager )
```

Sets the AppManager instance this PluginManager belongs to.

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="public" returntype="void" output="false"
  hint="Sets the AppManager instance this PluginManager belongs to.">
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
```

```
        <cfset variables.appManager = arguments.appManager />  
</cffunction>
```

PropertyManager

Package: MachII.framework

Manages defined properties for the framework.

Method Summary

public void	init(string configXML, AppManager appManager, string version) Initialization function called by the framework.
public void	configure() Prepares the manager for use.
public AppManager	getAppManager()
public struct	getProperties() Returns all properties.
public any	getProperty(string propertyName, [any defaultValue=""]) Returns the property value by name. If the property is not defined, and a default value is passed, it will be returned. If the property and a default value are both not defined then an exception is thrown.
public string	getVersion() Gets the version number of the framework.
public boolean	hasProperty(string propertyName) DEPRECATED - use isPropertyDefined() instead. Checks if property name is defined in the properties.
public boolean	isPropertyDefined(string propertyName) Checks if property name is defined in the properties.
public void	setAppManager(AppManager appManager)
public void	setProperty(string propertyName, any propertyValue)

Method Summary

	Sets the property value by name.
private void	setVersion(string version)

Method Detail**configure**

```
public void configure( )
```

Prepares the manager for use.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void" output="false"
    hint="Prepares the manager for use.">
</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Parameters:

Code:

```
<cffunction name="getAppManager" access="public" returntype="MachII.framework.AppManager" output="false">
    <cfreturn variables.appManager />
</cffunction>
```

getProperties

```
public struct getProperties( )
```

Returns all properties.

Parameters:

Code:

```
<cffunction name="getProperties" access="public" returntype="struct" output="false"
    hint="Returns all properties.">
    <cfreturn variables.properties />
</cffunction>
```

getProperty

```
public any getProperty( string propertyName, [any defaultValue=""] )
```

Returns the property value by name. If the property is not defined, and a default value is passed, it will be returned. If the property and a default value are both not defined then an exception is thrown.

Parameters:

```
string propertyName
[any defaultValue=""]
```

Code:

```
<cffunction name="getProperty" access="public" returntype="any" output="false"
    hint="Returns the property value by name. If the property is not defined, and a default value is passed, it will
    <cfargument name="propertyName" type="string" required="true" />
    <cfargument name="defaultValue" type="any" required="false" default="" />

    <cfif isPropertyDefined(arguments.propertyName)>
        <cfreturn variables.properties[arguments.propertyName] />
    <cfelseif StructKeyExists(arguments, 'defaultValue')>
        <cfreturn arguments.defaultValue />
```

```
<cfelse>
    <cfthrow type="MachII.framework.PropertyNotDefined"
            message="Property with name '#arguments.propertyName#' is not defined." />
</cfif>
</cffunction>
```

getVersion

```
public string getVersion( )
```

Gets the version number of the framework.

Parameters:

Code:

```
<cffunction name="getVersion" access="public" returnType="string" output="false"
            hint="Gets the version number of the framework.">
    <cfreturn variables.version />
</cffunction>
```

hasProperty

```
public boolean hasProperty( string propertyName )
```

DEPRECATED - use isPropertyDefined() instead. Checks if property name is defined in the properties.

Parameters:

string propertyName

Code:

```
<cffunction name="hasProperty" access="public" returnType="boolean" output="false"
            hint="DEPRECATED - use isPropertyDefined() instead. Checks if property name is defined in the properties.">
    <cfargument name="propertyName" type="string" required="true" />
```

```
<cfreturn StructKeyExists(variables.properties, arguments.propertyName) />
</cffunction>
```

init

```
public void init( string configXML, AppManager appManager, string version )
```

Initialization function called by the framework.

Parameters:

```
string configXML
AppManager appManager
string version
```

Code:

```
<cffunction name="init" access="public" returntype="void" output="false"
    hint="Initialization function called by the framework.">
    <cfargument name="configXML" type="string" required="true" />
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfargument name="version" type="string" required="true" />

    <cfset var xnProperties = "" />
    <cfset var i = 0 />

    <cfset setAppManager(arguments.appManager) />
    <cfset setVersion(arguments.version) />

    <cfset xnProperties = XMLSearch(configXML,'//property') />

    <cfloop from="1" to="#ArrayLen(xnProperties)#" index="i">
        <cfset setProperty(xnProperties[i].xmlAttributes.name, xnProperties[i].xmlAttributes.value) />
    </cfloop>

    <cfif NOT isPropertyDefined('defaultEvent')>
        <cfset setProperty('defaultEvent', 'defaultEvent') />
    </cfif>
```

```
<cfif NOT isPropertyDefined('exceptionEvent')>
  <cfset setProperty('exceptionEvent', 'exceptionEvent') />
</cfif>
<cfif NOT isPropertyDefined('applicationRoot')>
  <cfset setProperty('applicationRoot', '') />
</cfif>
<cfif NOT isPropertyDefined('eventParameter')>
  <cfset setProperty('eventParameter', 'event') />
</cfif>
<cfif NOT isPropertyDefined('parameterPrecedence')>
  <cfset setProperty('parameterPrecedence', 'form') />
</cfif>
<cfif NOT isPropertyDefined('maxEvents')>
  <cfset setProperty('maxEvents', 10) />
</cfif>
</cffunction>
```

isPropertyDefined

```
public boolean isPropertyDefined( string propertyName )
```

Checks if property name is defined in the properties.

Parameters:

string propertyName

Code:

```
<cffunction name="isPropertyDefined" access="public" returntype="boolean" output="false"
  hint="Checks if property name is defined in the properties.">
  <cfargument name="propertyName" type="string" required="true" />
  <cfreturn StructKeyExists(variables.properties, arguments.propertyName) />
</cffunction>
```

setAppManager

```
public void setAppManager( AppManager appManager )
```

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="public" returntype="void" output="false">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfset variables.appManager = arguments.appManager />
</cffunction>
```

setProperty

public void setProperty(string propertyName, any propertyValue)

Sets the property value by name.

Parameters:

string propertyName

any propertyValue

Code:

```
<cffunction name="setProperty" access="public" returntype="void" output="false"
    hint="Sets the property value by name.">
    <cfargument name="propertyName" type="string" required="true" />
    <cfargument name="propertyValue" type="any" required="true" />
    <cfset variables.properties[arguments.propertyName] = arguments.propertyValue />
</cffunction>
```

setVersion

private void setVersion(string version)

Parameters:

string version

Code:

```
<cffunction name="setVersion" access="private" returntype="void" output="false">  
  <cfargument name="version" type="string" required="true" />  
  <cfset variables.version = arguments.version />  
</cffunction>
```

RequestHandler

Package: MachII.framework

Handles request to event conversion for the framework.

Method Summary

public void	init(AppManager appManager) Initializes the RequestHandler.
private AppManager	getAppManager()
private string	getEventName(struct eventArgs)
private struct	getRequestEventArgs()
public void	handleEventRequest(string eventName, struct eventArgs) Handles an event request made to the framework.
public void	handleRequest() Handles a request made to the framework.
private void	setAppManager(AppManager appManager)

Method Detail

getAppManager

private AppManager getAppManager()

Parameters:

Code:

```
<cffunction name="getAppManager" access="private" returntype="MachII.framework.AppManager" output="false">
    <cfreturn variables.appManager />
</cffunction>
```

getEventName

private string getEventName(struct eventArgs)

Parameters:

struct eventArgs

Code:

```
<cffunction name="getEventName" access="private" returntype="string" output="false">
    <cfargument name="eventArgs" type="struct" required="true" />
    <cfset var eventParam = getAppManager().getPropertyManager().getProperty('eventParameter') />
    <cfset var eventName = "" />

    <cfif StructKeyExists(arguments.eventArgs, eventParam) AND arguments.eventArgs[eventParam] NEQ ''>
        <cfset eventName = arguments.eventArgs[eventParam] />
    <cfelse>
        <cfset eventName = getAppManager().getPropertyManager().getProperty('defaultEvent') />
    </cfif>

    <cfreturn eventName />
</cffunction>
```

getRequestEventArgs

private struct getRequestEventArgs()

Parameters:

Code:

```

<cffunction name="getRequestEventArgs" access="private" returnType="struct" output="false">
    <cfset var eventArgs = StructNew() />
    <cfset var paramPrecedence = getAppManager().getPropertyManager().getProperty('parameterPrecedence') />
    <cfset var overwriteFormParams = (paramPrecedence EQ 'url') />

    <cfset StructAppend(eventArgs, form) />
    <cfset StructAppend(eventArgs, url, overwriteFormParams) />

    <cfreturn eventArgs />
</cffunction>

```

handleEventRequest

```
public void handleEventRequest( string eventName, struct eventArgs )
```

Handles an event request made to the framework.

Parameters:

```
string eventName
struct eventArgs
```

Code:

```

<cffunction name="handleEventRequest" access="public" returnType="void" output="true"
    hint="Handles an event request made to the framework.">
    <cfargument name="eventName" type="string" required="true"
        hint="The name of the requested event." />
    <cfargument name="eventArgs" type="struct" required="true" default="#StructNew()#"
        hint="The event arguments provided in the request." />
    <cfset var exception = "" />
    <cfset var eventContext = getAppManager().createEventContext(arguments.eventName) />
    <cfset request.eventContext = eventContext />

    <cftry>
        <cfif NOT getAppManager().getEventManager().isEventDefined(arguments.eventName)>
            <cfthrow type="MachII.framework.EventHandlerNotDefined"
                message="Event-handler for event '#arguments.eventName#' is not defined." />
        </cfif>
    </cftry>

```

```
</cfif>

<cfif getAppManager().getEventManager().isEventPublic(arguments.eventName)>
    <cfset eventContext.announceEvent(arguments.eventName, arguments.eventArgs) />
<cfelse>
    <cfthrow type="MachII.framework.EventHandlerNotAccessible"
            message="Event-handler for event '#arguments.eventName#' is not accessible." />
</cfif>

<cfcatch type="any">
    <cfset exception = CreateObject('component', 'MachII.util.Exception') />
    <cfset exception.wrapException(cfcatch) />
    <cfset eventContext.handleException(exception, true) />
</cfcatch>
</cftry>

    <cfset eventContext.processEvents() />
</cffunction>
```

handleRequest

```
public void handleRequest()
```

Handles a request made to the framework.

Parameters:

Code:

```
<cffunction name="handleRequest" access="public" returnType="void" output="true"
            hint="Handles a request made to the framework.">

    <cfset var eventArgs = getRequestEventArgs() />

    <cfset var eventName = getEventName(eventArgs) />

    <cfset handleEventRequest(eventName, eventArgs) />
</cffunction>
```

init

```
public void init( AppManager appManager )
```

Initializes the RequestHandler.

Parameters:

AppManager appManager

Code:

```
<cffunction name="init" access="public" returntype="void" output="false"
    hint="Initializes the RequestHandler.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />

    <cfset setAppManager(arguments.appManager) />
</cffunction>
```

setAppManager

```
private void setAppManager( AppManager appManager )
```

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="private" returntype="void" output="false">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfset variables.appManager = arguments.appManager />
</cffunction>
```

ViewContext

Package: MachII.framework

Handles view display for an EventContext.

Method Summary

public ViewContext	init(AppManager appManager)	Used by the framework for initialization. Do not override.
public void	displayView(Event event, string viewName, [string contentKey=""], [string contentArg=""], [boolean append="false"])	Displays a view by view name and performs contentKey, contentArg and append functions.
public AppManager	getAppManager()	Sets the AppManager instance this ViewContext belongs to.
private string	getAppRoot()	Gets the application root from the Mach-II properties.
private string	getFullPath(string viewName)	Gets the full path of a view by view name from the view manager.
public any	getProperty(string propertyName)	Gets the specified property - this is just a shortcut for getAppManager().getPropertyManager().getProperty()
package Property-Manager	getPropertyManager()	Gets the components PropertyManager instance.
public void	setAppManager(AppManager appManager)	Returns the AppManager instance this ViewContext belongs to.
public any	setProperty(string propertyName, any propertyValue)	

Method Summary

	Sets the specified property - this is just a shortcut for <code>getAppManager().getPropertyManager().setProperty()</code>
private void	<code>setPropertyManager(PropertyManager propertyManager)</code> Sets the components PropertyManager instance.

Method Detail**displayView**

```
public void displayView( Event event, string viewName, [string contentKey=""], [string contentArg=""], [boolean append="false"] )
```

Displays a view by view name and performs contentKey, contentArg and append functions.

Parameters:

Event event
string viewName
[string contentKey=""]
[string contentArg=""]
[boolean append="false"]

Code:

```
<cffunction name="displayView" access="public" returntype="void" output="true"
  hint="Displays a view by view name and performs contentKey, contentArg and append functions.">
  <cfargument name="event" type="MachII.framework.Event" required="true"
    hint="The current Event object." />
  <cfargument name="viewName" type="string" required="true"
    hint="The view name to display." />
  <cfargument name="contentKey" type="string" required="false" default=""
    hint="The contentKey name if defined." />
  <cfargument name="contentArg" type="string" required="false" default=""
    hint="The contentArg name if defined." />
  <cfargument name="append" type="boolean" required="false" default="false"
```

```
        hint="Directive to append event." />

    <cfset var viewPath = getFullPath(arguments.viewName) />
    <cfset var viewContent = "" />
    <cfset request.event = arguments.event />

    <cfif arguments.contentKey NEQ ''>
        <cfsavecontent variable="viewContent">
            <cfinclude template="#viewPath#" />
        </cfsavecontent>
        <cfif arguments.append AND IsDefined(arguments.contentKey)>
            <cfset viewContent = Evaluate(arguments.contentKey) & viewContent />
        </cfif>
        <cfset setVariable(arguments.contentKey, viewContent) />
    </cfif>

    <cfif arguments.contentArg NEQ ''>
        <cfsavecontent variable="viewContent">
            <cfinclude template="#viewPath#" />
        </cfsavecontent>
        <cfif arguments.append>
            <cfset viewContent = arguments.event.getArg(arguments.contentArg, "") & viewContent />
        </cfif>
        <cfset arguments.event.setArg(arguments.contentArg, viewContent) />
    </cfif>

    <cfif arguments.contentKey EQ '' AND arguments.contentArg EQ ''>
        <cfinclude template="#viewPath#" />
    </cfif>
</cffunction>
```

getManager

```
public AppManager getManager()
```

Sets the AppManager instance this ViewContext belongs to.

Parameters:

Code:

```
<cffunction name="getAppManager" access="public" returntype="MachII.framework.AppManager" output="false"
    hint="Sets the AppManager instance this ViewContext belongs to.">
    <cfreturn variables.appManager />
</cffunction>
```

getAppRoot

```
private string getAppRoot( )
```

Gets the application root from the Mach-II properties.

Parameters:

Code:

```
<cffunction name="getAppRoot" access="private" returntype="string" output="false"
    hint="Gets the application root from the Mach-II properties.">
    <cfreturn getAppManager().getPropertyManager().getProperty('applicationRoot') />
</cffunction>
```

getFullPath

```
private string getFullPath( string viewName )
```

Gets the full path of a view by view name from the view manager.

Parameters:

string viewName

Code:

```
<cffunction name="getFullPath" access="private" returntype="string" output="false"
    hint="Gets the full path of a view by view name from the view manager.">
    <cfargument name="viewName" type="string" required="true" />
    <cfset var viewPath = getAppManager().getViewManager().getViewPath(arguments.viewName) />
</cffunction>
```

```
<cfreturn getAppRoot() & viewPath />
</cffunction>
```

getProperty

public any getProperty(string propertyName)

Gets the specified property - this is just a shortcut for `getAppManager().getPropertyManager().getProperty()`

Parameters:

string propertyName

Code:

```
<cffunction name="getProperty" access="public" returntype="any" output="false"
    hint="Gets the specified property - this is just a shortcut for getAppManager().getPropertyManager().getProperty"
    <cfargument name="propertyName" type="string" required="yes"
        hint="The name of the property to return." />
    <cfreturn getPropertyManager().getProperty(arguments.propertyName) />
</cffunction>
```

getPropertyManager

package PropertyManager getPropertyManager()

Gets the components PropertyManager instance.

Parameters:

Code:

```
<cffunction name="getPropertyManager" access="package" returntype="MachII.framework.PropertyManager" output="false"
    hint="Gets the components PropertyManager instance.">
    <cfreturn variables.propertyManager />
</cffunction>
```

init

```
public ViewContext init( AppManager appManager )
```

Used by the framework for initialization. Do not override.

Parameters:

AppManager appManager

Code:

```
<cffunction name="init" access="public" returntype="ViewContext" output="false"
  hint="Used by the framework for initialization. Do not override.">
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />

  <cfset setAppManager(arguments.appManager) />
  <cfset setPropertyManager(getAppManager().getPropertyManager()) />

  <cfreturn this />
</cffunction>
```

setAppManager

```
public void setAppManager( AppManager appManager )
```

Returns the AppManager instance this ViewContext belongs to.

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="public" returntype="void" output="false"
  hint="Returns the AppManager instance this ViewContext belongs to.">
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
  <cfset variables.appManager = arguments.appManager />
</cffunction>
```

```
</cffunction>
```

setProperty

```
public any setProperty( string propertyName, any propertyValue )
```

Sets the specified property - this is just a shortcut for `getAppManager().getPropertyManager().setProperty()`

Parameters:

string propertyName
any propertyValue

Code:

```
<cffunction name="setProperty" access="public" returntype="any" output="false"
    hint="Sets the specified property - this is just a shortcut for getAppManager().getPropertyManager().setProperty"
    <cfargument name="propertyName" type="string" required="yes"
        hint="The name of the property to set." />
    <cfargument name="propertyValue" type="any" required="yes"
        hint="The value to store in the property." />
    <cfreturn getAppManager().setProperty(arguments.propertyName, arguments.propertyValue) />
</cffunction>
```

setPropertyManager

```
private void setPropertyManager( PropertyManager propertyManager )
```

Sets the components PropertyManager instance.

Parameters:

PropertyManager propertyManager

Code:

```
<cffunction name="setPropertyManager" access="private" returntype="void" output="false"
```

```
    hint="Sets the components PropertyManager instance.">
    <cfargument name="propertyManager" type="MachII.framework.PropertyManager" required="true"
        hint="The PropertyManager instance to set." />
    <cfset variables.propertyManager = arguments.propertyManager />
</cffunction>
```

ViewManager

Package: MachII.framework

Manages registered views for the framework.

Method Summary

public void	init(string configXML, AppManager appManager)	Initialization function called by the framework.
public void	configure()	Prepares the manager for use.
public AppManager	getAppManager()	
public string	getViewPath(string viewName)	Gets the view path.
public boolean	isViewDefined(string viewName)	Checks if the view is defined.
public void	setAppManager(AppManager appManager)	

Method Detail

configure

```
public void configure( )
```

Prepares the manager for use.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void"
    hint="Prepares the manager for use.">

</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Parameters:

Code:

```
<cffunction name="getAppManager" access="public" returntype="MachII.framework.AppManager" output="false">
    <cfreturn variables.appManager />
</cffunction>
```

getViewPath

```
public string getViewPath( string viewName )
```

Gets the view path.

Parameters:

string viewName

Code:

```
<cffunction name="getViewPath" access="public" returntype="string" output="false"
    hint="Gets the view path.">
    <cfargument name="viewName" type="string" required="true"
```

```

        hint="Name of the view path to get." />
    <cfif isViewDefined(arguments.viewName)>
        <cfreturn variables.viewPaths[arguments.viewName] />
    <cfelse>
        <cfthrow type="MachII.framework.ViewNotDefined"
            message="View with name '#arguments.viewName#' is not defined." />
    </cfif>
</cffunction>

```

init

```
public void init( string configXML, AppManager appManager )
```

Initialization function called by the framework.

Parameters:

```
string configXML
AppManager appManager
```

Code:

```

<cffunction name="init" access="public" returntype="void" output="false"
    hint="Initialization function called by the framework.">
    <cfargument name="configXML" type="string" required="true" />
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />

    <cfset var viewNodes = "" />
    <cfset var name = "" />
    <cfset var page = "" />
    <cfset var i = 0 />

    <cfset setAppManager(arguments.appManager) />

    <cfset viewNodes = XMLSearch(configXML,"//page-views/page-view") />
    <cfloop from="1" to="#ArrayLen(viewNodes)#" index="i">
        <cfset name = viewNodes[i].xmlAttributes['name'] />
        <cfset page = viewNodes[i].xmlAttributes['page'] />
    </cfloop>

```

```
                <cfset variables.viewPaths[name] = page />
            </cfloop>
        </cffunction>
```

isViewDefined

```
public boolean isViewDefined( string viewName )
```

Checks if the view is defined.

Parameters:

string viewName

Code:

```
<cffunction name="isViewDefined" access="public" returntype="boolean" output="false"
    hint="Checks if the view is defined.">
    <cfargument name="viewName" type="string" required="true"
        hint="Name of the view to check." />
    <cfreturn StructKeyExists(variables.viewPaths, arguments.viewName) />
</cffunction>
```

setAppManager

```
public void setAppManager( AppManager appManager )
```

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="public" returntype="void" output="false">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
```

```
        <cfset variables.appManager = arguments.appManager />  
</cffunction>
```

framework.commands

AnnounceCommand

Package: MachII.framework.commands

Inherits from: framework.EventCommand

An EventCommand for announcing an event.

Method Summary

public Announce- Command	init(string eventName, [boolean copyEventArgs="true"]) Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext) Executes the command.
private string	getEventName()
private boolean	isCopyEventArgs()
private void	setCopyEventArgs([boolean copyEventArgs="true"])
private void	setEventName(string eventName)

Methods inherited from framework.EventCommand: setParameter , getParameter , setParameters

Method Detail

execute

```
public boolean execute( Event event, EventContext eventContext )
```

Executes the command.

Parameters:

Event event
EventContext eventContext

Code:

```
<cffunction name="execute" access="public" returntype="boolean" output="false"
  hint="Executes the command.">
  <cfargument name="event" type="MachII.framework.Event" required="true" />
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var eventArgs = "" />
  <cfif isCopyEventArgs()>
    <cfset eventArgs = event.getArgs() />
  <cfelse>
    <cfset eventArgs = StructNew() />
  </cfif>

  <cfset arguments.eventContext.announceEvent(getEventName(), eventArgs) />

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

getEventName

```
private string getEventName( )
```

Parameters:

Code:

```
<cffunction name="getEventName" access="private" returntype="string" output="false">
  <cfreturn variables.eventName />
</cffunction>
```

init

```
public AnnounceCommand init( string eventName, [boolean copyEventArgs="true"] )
```

Used by the framework for initialization.

Parameters:

```
string eventName  
[boolean copyEventArgs="true"]
```

Code:

```
<cffunction name="init" access="public" returntype="AnnounceCommand" output="false"  
    hint="Used by the framework for initialization.">  
    <cfargument name="eventName" type="string" required="true" />  
    <cfargument name="copyEventArgs" type="boolean" required="false" default="true" />  
  
    <cfset setName(arguments.eventName) />  
    <cfset setCopyEventArgs(arguments.copyEventArgs) />  
  
    <cfreturn this />  
</cffunction>
```

isCopyEventArgs

```
private boolean isCopyEventArgs( )
```

Parameters:

Code:

```
<cffunction name="isCopyEventArgs" access="private" returntype="boolean" output="false">  
    <cfreturn variables.copyEventArgs />  
</cffunction>
```

setCopyEventArgs

```
private void setCopyEventArgs( [boolean copyEventArgs="true"] )
```

Parameters:

[boolean copyEventArgs="true"]

Code:

```
<cffunction name="setCopyEventArgs" access="private" returntype="void" output="false">
    <cfargument name="copyEventArgs" type="boolean" required="false" default="true" />
    <cfset variables.copyEventArgs = arguments.copyEventArgs />
</cffunction>
```

setEventName

```
private void setEventName( string eventName )
```

Parameters:

string eventName

Code:

```
<cffunction name="setEventName" access="private" returntype="void" output="false">
    <cfargument name="eventName" type="string" required="true" />
    <cfset variables.eventName = arguments.eventName />
</cffunction>
```

EventArgCommand

Package: MachII.framework.commands

Inherits from: framework.EventCommand

An EventCommand for putting an event arg into the current event.

Method Summary

public EventArgCommand	init(string argName, [string argValue=""], [string argVariable=""]) Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext) Executes the command.
private string	getArgName()
private string	getArgValue()
private string	getArgVariable()
private any	getArgVariableValue()
private boolean	isArgValueDefined()
private boolean	isArgVariableDefined()
private void	setArgName(string argName)
private void	setArgValue(string argValue)
private void	setArgVariable(string argVariable)

Methods inherited from framework.EventCommand: setParameter , getParameter , setParameters

Method Detail

execute

```
public boolean execute( Event event, EventContext eventContext )
```

Executes the command.

Parameters:

Event event

EventContext eventContext

Code:

```
<cffunction name="execute" access="public" returntype="boolean"
  hint="Executes the command.">
  <cfargument name="event" type="MachII.framework.Event" required="true" />
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var value = "" />

  <cfif isArgVariableDefined()>
    <cfset value = getArgVariableValue() />
  <cfelseif isArgValueDefined()>
    <cfset value = getArgValue() />
  <cfelse>
    <cfset value = "" />
  </cfif>

  <cfset arguments.event.setArg(getArgName(), value) />

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

getArgName

```
private string getArgName( )
```

Parameters:

Code:

```
<cffunction name="getArgName" access="private" returntype="string" output="false">
    <cfreturn variables.argName />
</cffunction>
```

getArgValue

private string getArgValue()

Parameters:

Code:

```
<cffunction name="getArgValue" access="private" returntype="string" output="false">
    <cfreturn variables.argValue />
</cffunction>
```

getArgVariable

private string getArgVariable()

Parameters:

Code:

```
<cffunction name="getArgVariable" access="private" returntype="string" output="false">
    <cfreturn variables.argVariable />
</cffunction>
```

getArgVariableValue

private any getArgVariableValue()

Parameters:

Code:

```
<cffunction name="getArgVariableValue" access="private" returntype="any" output="false">
    <cfset var value = "" />
    <cfif IsDefined(getArgVariable())>
        <cfset value = Evaluate(getArgVariable()) />
    </cfif>
    <cfreturn value />
</cffunction>
```

init

```
public EventArgCommand init( string argName, [string argValue=""], [string argVariable=""] )
```

Used by the framework for initialization.

Parameters:

```
string argName
[string argValue=""]
[string argVariable=""]
```

Code:

```
<cffunction name="init" access="public" returntype="EventArgCommand" output="false"
    hint="Used by the framework for initialization.">
    <cfargument name="argName" type="string" required="true" />
    <cfargument name="argValue" type="string" required="false" default="" />
    <cfargument name="argVariable" type="string" required="false" default="" />

    <cfset setArgName(arguments.argName) />
    <cfset setArgValue(arguments.argValue) />
    <cfset setArgVariable(arguments.argVariable) />

    <cfreturn this />
</cffunction>
```

isArgValueDefined

private boolean isArgValueDefined()

Parameters:

Code:

```
<cffunction name="isArgValueDefined" access="private" returntype="boolean" output="false">  
    <cfreturn NOT getArgValue() EQ '' />  
</cffunction>
```

isArgVariableDefined

private boolean isArgVariableDefined()

Parameters:

Code:

```
<cffunction name="isArgVariableDefined" access="private" returntype="boolean" output="false">  
    <cfreturn NOT getArgVariable() EQ '' />  
</cffunction>
```

setArgName

private void setArgName(string argName)

Parameters:

string argName

Code:

```
<cffunction name="setArgName" access="private" returntype="void" output="false">
  <cfargument name="argName" type="string" required="true" />
  <cfset variables.argName = arguments.argName />
</cffunction>
```

setArgValue

```
private void setArgValue( string argValue )
```

Parameters:

string argValue

Code:

```
<cffunction name="setArgValue" access="private" returntype="void" output="false">
  <cfargument name="argValue" type="string" required="true" />
  <cfset variables.argValue = arguments.argValue />
</cffunction>
```

setArgVariable

```
private void setArgVariable( string argVariable )
```

Parameters:

string argVariable

Code:

```
<cffunction name="setArgVariable" access="private" returntype="void" output="false">
  <cfargument name="argVariable" type="string" required="true" />
  <cfset variables.argVariable = arguments.argVariable />
</cffunction>
```

```
</cffunction>
```

EventBeanCommand

Package: MachII.framework.commands

Inherits from: framework.EventCommand

An EventCommand for creating and populating a bean in the current event.

Method Summary

public EventBean- Command	init(string beanName, string beanType, string beanFields) Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext) Executes the command.
private string	getBeanFields()
private string	getBeanName()
private string	getBeanType()
private BeanUtil	getBeanUtil()
public boolean	isBeanFieldsDefined()
private void	setBeanFields(string beanFields)
private void	setBeanName(string beanName)
private void	setBeanType(string beanType)
private void	setBeanUtil(BeanUtil beanUtil)

Methods inherited from framework.EventCommand: setParameter , getParameter , setParameters

Method Detail

execute

```
public boolean execute( Event event, EventContext eventContext )
```

Executes the command.

Parameters:

Event event

EventContext eventContext

Code:

```
<cffunction name="execute" access="public" returntype="boolean"
  hint="Executes the command.">
  <cfargument name="event" type="MachII.framework.Event" required="true" />
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var bean = "" />

  <cfif isBeanFieldsDefined(>
    <cfset bean = getBeanUtil().createBean(getBeanType()) />
    <cfset getBeanUtil().setBeanFields(bean, getBeanFields(), arguments.event.getArgs()) />
  <cfelse>
    <cfset bean = getBeanUtil().createBean(getBeanType(), arguments.event.getArgs()) />
  </cfif>

  <cfset arguments.event.setArg(getBeanName(), bean, getBeanType()) />

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

getBeanFields

```
private string getBeanFields( )
```

Parameters:

Code:

```
<cffunction name="getBeanFields" access="private" returntype="string" output="false">
    <cfreturn variables.beanFields />
</cffunction>
```

getBeanName

private string getBeanName()

Parameters:

Code:

```
<cffunction name="getBeanName" access="private" returntype="string" output="false">
    <cfreturn variables.beanName />
</cffunction>
```

getBeanType

private string getBeanType()

Parameters:

Code:

```
<cffunction name="getBeanType" access="private" returntype="string" output="false">
    <cfreturn variables.beanType />
</cffunction>
```

getBeanUtil

private BeanUtil getBeanUtil()

Parameters:

Code:

```
<cffunction name="getBeanUtil" access="private" returntype="MachII.util.BeanUtil" output="false">
    <cfreturn variables.beanUtil />
</cffunction>
```

init

```
public EventBeanCommand init( string beanName, string beanType, string beanFields )
```

Used by the framework for initialization.

Parameters:

```
string beanName
string beanType
string beanFields
```

Code:

```
<cffunction name="init" access="public" returntype="EventBeanCommand" output="false"
    hint="Used by the framework for initialization.">
    <cfargument name="beanName" type="string" required="true" />
    <cfargument name="beanType" type="string" required="true" />
    <cfargument name="beanFields" type="string" required="true" />

    <cfset setBeanName(arguments.beanName) />
    <cfset setBeanType(arguments.beanType) />
    <cfset setBeanFields(arguments.beanFields) />

    <cfset setBeanUtil( CreateObject('component','MachII.util.BeanUtil').init() ) />

    <cfreturn this />
</cffunction>
```

isBeanFieldsDefined

public boolean isBeanFieldsDefined()

Parameters:

Code:

```
<cffunction name="isBeanFieldsDefined" access="public" returntype="boolean" output="false">
    <cfreturn NOT getBeanFields() EQ '' />
</cffunction>
```

setBeanFields

private void setBeanFields(string beanFields)

Parameters:

string beanFields

Code:

```
<cffunction name="setBeanFields" access="private" returntype="void" output="false">
    <cfargument name="beanFields" type="string" required="true" />
    <cfset variables.beanFields = arguments.beanFields />
</cffunction>
```

setBeanName

private void setBeanName(string beanName)

Parameters:

string beanName

Code:

```
<cffunction name="setBeanName" access="private" returntype="void" output="false">
  <cfargument name="beanName" type="string" required="true" />
  <cfset variables.beanName = arguments.beanName />
</cffunction>
```

setBeanType

```
private void setBeanType( string beanType )
```

Parameters:

string beanType

Code:

```
<cffunction name="setBeanType" access="private" returntype="void" output="false">
  <cfargument name="beanType" type="string" required="true" />
  <cfset variables.beanType = arguments.beanType />
</cffunction>
```

setBeanUtil

```
private void setBeanUtil( BeanUtil beanUtil )
```

Parameters:

BeanUtil beanUtil

Code:

```
<cffunction name="setBeanUtil" access="private" returntype="void" output="false">
  <cfargument name="beanUtil" type="MachII.util.BeanUtil" required="true" />
  <cfset variables.beanUtil = arguments.beanUtil />
</cffunction>
```


EventMappingCommand

Package: MachII.framework.commands

Inherits from: framework.EventCommand

An EventCommand for setting up an event mapping for an event handler.

Method Summary

public EventMappingCommand	init(string eventName, string mappingName) Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext) Executes the command.
private string	getEventName()
private string	getMappingName()
private void	setEventName(string eventName)
private void	setMappingName(string mappingName)

Methods inherited from framework.EventCommand: setParameter , getParameter , setParameters

Method Detail

execute

public boolean execute(Event event, EventContext eventContext)

Executes the command.

Parameters:

Event event
EventContext eventContext

Code:

```
<cffunction name="execute" access="public" returntype="boolean"
  hint="Executes the command.">
  <cfargument name="event" type="MachII.framework.Event" required="true" />
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset arguments.eventContext.setEventMapping(getEventName(), getMappingName()) />

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

getEventName

private string getEventName()

Parameters:

Code:

```
<cffunction name="getEventName" access="private" returntype="string" output="false">
  <cfreturn variables.eventName />
</cffunction>
```

getMappingName

private string getMappingName()

Parameters:

Code:

```
<cffunction name="getMappingName" access="private" returnType="string" output="false">
    <cfreturn variables.mappingName />
</cffunction>
```

init

```
public EventMappingCommand init( string eventName, string mappingName )
```

Used by the framework for initialization.

Parameters:

string eventName
string mappingName

Code:

```
<cffunction name="init" access="public" returnType="EventMappingCommand" output="false"
    hint="Used by the framework for initialization.">
    <cfargument name="eventName" type="string" required="true" />
    <cfargument name="mappingName" type="string" required="true" />

    <cfset setEventName(arguments.eventName) />
    <cfset setMappingName(arguments.mappingName) />

    <cfreturn this />
</cffunction>
```

setEventName

```
private void setEventName( string eventName )
```

Parameters:

string eventName

Code:

```
<cffunction name="setEventName" access="private" returntype="void" output="false">
    <cfargument name="eventName" type="string" required="true" />
    <cfset variables.eventName = arguments.eventName />
</cffunction>
```

setMappingName

private void setMappingName(string mappingName)

Parameters:

string mappingName

Code:

```
<cffunction name="setMappingName" access="private" returntype="void" output="false">
    <cfargument name="mappingName" type="string" required="true" />
    <cfset variables.mappingName = arguments.mappingName />
</cffunction>
```

FilterCommand

Package: MachII.framework.commands

Inherits from: framework.EventCommand

An EventCommand for processing an EventFilter.

Method Summary

public FilterCommand	init(EventFilter filter, [struct paramArgs="#StructNew()#"])
	Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext)
	Executes the command.
private EventFilter	getFilter()
private struct	getParamArgs()
private void	setFilter(EventFilter filter)
private void	setParamArgs(struct paramArgs)

Methods inherited from framework.EventCommand: setParameter , getParameter , setParameters

Method Detail

execute

```
public boolean execute( Event event, EventContext eventContext )
```

Executes the command.

Parameters:

Event event
EventContext eventContext

Code:

```
<cffunction name="execute" access="public" returntype="boolean"
  hint="Executes the command.">
  <cfargument name="event" type="MachII.framework.Event" required="true" />
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var continue = false />

  <cfinvoke component="#getFilter()#" method="filterEvent" returnVariable="continue">
    <cfinvokeargument name="event" value="#arguments.event#" />
    <cfinvokeargument name="eventContext" value="#arguments.eventContext#" />
    <cfinvokeargument name="paramArgs" value="#getParamArgs()#" />
  </cfinvoke>

  <cfreturn continue />
</cffunction>
```

getFilter

private EventFilter getFilter()

Parameters:

Code:

```
<cffunction name="getFilter" access="private" returntype="MachII.framework.EventFilter" output="false">
  <cfreturn variables.filter />
</cffunction>
```

getParamArgs

private struct getParamArgs()

Parameters:

Code:

```
<cffunction name="getParamArgs" access="private" returntype="struct" output="false">
    <cfreturn variables.paramArgs />
</cffunction>
```

init

public FilterCommand init(EventFilter filter, [struct paramArgs="#StructNew()#"])

Used by the framework for initialization.

Parameters:

EventFilter filter

[struct paramArgs="#StructNew()#"]

Code:

```
<cffunction name="init" access="public" returntype="FilterCommand" output="false"
    hint="Used by the framework for initialization.">
    <cfargument name="filter" type="MachII.framework.EventFilter" required="true" />
    <cfargument name="paramArgs" type="struct" required="false" default="#StructNew()#" />

    <cfset setFilter(arguments.filter) />
    <cfset setParamArgs(arguments.paramArgs) />

    <cfreturn this />
</cffunction>
```

setFilter

private void setFilter(EventFilter filter)

Parameters:

EventFilter filter

Code:

```
<cffunction name="setFilter" access="private" returntype="void" output="false">
    <cfargument name="filter" type="MachII.framework.EventFilter" required="true" />
    <cfset variables.filter = arguments.filter />
</cffunction>
```

setParamArgs

private void setParamArgs(struct paramArgs)

Parameters:

struct paramArgs

Code:

```
<cffunction name="setParamArgs" access="private" returntype="void" output="false">
    <cfargument name="paramArgs" type="struct" required="true" />
    <cfset variables.paramArgs = arguments.paramArgs />
</cffunction>
```

NotifyCommand

Package: MachII.framework.commands

Inherits from: framework.EventCommand

An EventCommand for notifying a Listener.

Method Summary

public Notify- Command	init(Listener listener, string method, string resultKey, string resultArg) Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext) Executes the command.
private Listener	getListener()
private string	getMethod()
private string	getResultArg()
private string	getResultKey()
private boolean	hasResultArg()
private boolean	hasResultKey()
private void	setListener(Listener listener)
private void	setMethod(string method)
private void	setResultArg(string resultArg)
private void	setResultKey(string resultKey)

Methods inherited from framework.EventCommand: setParameter , getParameter , setParameters

Method Detail**execute**

```
public boolean execute( Event event, EventContext eventContext )
```

Executes the command.

Parameters:

Event event

EventContext eventContext

Code:

```
<cffunction name="execute" access="public" returntype="boolean"
  hint="Executes the command.">
  <cfargument name="event" type="MachII.framework.Event" required="true" />
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var listener = getListener() />
  <cfset var invoker = listener.getInvoker() />
  <cfset invoker.invokeListener(arguments.event, listener, getMethod(), getResultKey(), getResultArg()) />

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

getListener

```
private Listener getListener( )
```

Parameters:

Code:

```
<cffunction name="getListener" access="private" returntype="MachII.framework.Listener" output="false">
```

```
<cfreturn variables.listener />
</cffunction>
```

getMethod

private string getMethod()

Parameters:

Code:

```
<cffunction name="getMethod" access="private" returntype="string" output="false">
    <cfreturn variables.method />
</cffunction>
```

getResultArg

private string getResultArg()

Parameters:

Code:

```
<cffunction name="getResultArg" access="private" returntype="string" output="false">
    <cfreturn variables.resultArg />
</cffunction>
```

getResultKey

private string getResultKey()

Parameters:

Code:

```
<cffunction name="getResultKey" access="private" returntype="string" output="false">
    <cfreturn variables.resultKey />
</cffunction>
```

hasResultArg

private boolean hasResultArg()

Parameters:

Code:

```
<cffunction name="hasResultArg" access="private" returntype="boolean" output="false">
    <cfreturn variables.resultArg NEQ '' />
</cffunction>
```

hasResultKey

private boolean hasResultKey()

Parameters:

Code:

```
<cffunction name="hasResultKey" access="private" returntype="boolean" output="false">
    <cfreturn getResultKey() NEQ '' />
</cffunction>
```

init

public NotifyCommand init(Listener listener, string method, string resultKey, string resultArg)

Used by the framework for initialization.

Parameters:

Listener listener
string method
string resultKey
string resultArg

Code:

```
<cffunction name="init" access="public" returntype="NotifyCommand" output="false"
  hint="Used by the framework for initialization.">
  <cfargument name="listener" type="MachII.framework.Listener" required="true" />
  <cfargument name="method" type="string" required="true" />
  <cfargument name="resultKey" type="string" required="true" />
  <cfargument name="resultArg" type="string" required="true" />

  <cfset setListener(arguments.listener) />
  <cfset setMethod(arguments.method) />
  <cfset setResultKey(arguments.resultKey) />
  <cfset setResultArg(arguments.resultArg) />

  <cfreturn this />
</cffunction>
```

setListener

```
private void setListener( Listener listener )
```

Parameters:

Listener listener

Code:

```
<cffunction name="setListener" access="private" returntype="void" output="false">
  <cfargument name="listener" type="MachII.framework.Listener" required="true" />
  <cfset variables.listener = arguments.listener />
```

```
</cffunction>
```

setMethod

```
private void setMethod( string method )
```

Parameters:

string method

Code:

```
<cffunction name="setMethod" access="private" returntype="void" output="false">  
    <cfargument name="method" type="string" required="true" />  
    <cfset variables.method = arguments.method />  
</cffunction>
```

setResultArg

```
private void setResultArg( string resultArg )
```

Parameters:

string resultArg

Code:

```
<cffunction name="setResultArg" access="private" returntype="void" output="false">  
    <cfargument name="resultArg" type="string" required="true" />  
    <cfset variables.resultArg = arguments.resultArg />  
</cffunction>
```

setResultKey

private void setResultKey(string resultKey)

Parameters:

string resultKey

Code:

```
<cffunction name="setResultKey" access="private" returntype="void" output="false">
    <cfargument name="resultKey" type="string" required="true" />
    <cfset variables.resultKey = arguments.resultKey />
</cffunction>
```

RedirectCommand

Package: MachII.framework.commands

Inherits from: framework.EventCommand

An EventCommand for redirecting.

Method Summary

public RedirectCommand	init(string eventName, string eventParam, [string url=""], [string args=""])	Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext)	Executes the command.
private string	getArgs()	
private string	getEventName()	
private string	getEventParam()	
private string	getUrl()	
private string	makeRedirectUrl(Event event, EventContext eventContext)	Assembles the redirect url.
private void	setArgs(string args)	
private void	setEventName(string eventName)	
private void	setEventParam(string eventParam)	
private void	setUrl(string url)	

Methods inherited from framework.EventCommand: setParameter , getParameter , setParameters

Method Detail**execute**

```
public boolean execute( Event event, EventContext eventContext )
```

Executes the command.

Parameters:

Event event

EventContext eventContext

Code:

```
<cffunction name="execute" access="public" returntype="boolean"
    hint="Executes the command.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

    <cfset var redirectUrl = makeRedirectUrl(arguments.event, arguments.eventContext) />
    <cflocation url="#redirectUrl#" addtoken="no" />

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

getArgs

```
private string getArgs( )
```

Parameters:

Code:

```
<cffunction name="getArgs" access="private" returntype="string" output="false">
    <cfreturn variables.args />
</cffunction>
```

```
</cffunction>
```

getEventName

```
private string getEventName( )
```

Parameters:

Code:

```
<cffunction name="getEventName" access="private" returntype="string" output="false">  
    <cfreturn variables.eventName />  
</cffunction>
```

getEventParam

```
private string getEventParam( )
```

Parameters:

Code:

```
<cffunction name="getEventParam" access="private" returntype="string" output="false">  
    <cfreturn variables.eventParam />  
</cffunction>
```

getUrl

```
private string getUrl( )
```

Parameters:

Code:

```
<cffunction name="getUrl" access="private" returntype="string" output="false">
    <cfreturn variables.url />
</cffunction>
```

init

```
public RedirectCommand init( string eventName, string eventParam, [string url=""], [string args=""] )
```

Used by the framework for initialization.

Parameters:

```
string eventName
string eventParam
[string url=""]
[string args=""]
```

Code:

```
<cffunction name="init" access="public" returntype="RedirectCommand" output="false"
    hint="Used by the framework for initialization.">
    <cfargument name="eventName" type="string" required="true" />
    <cfargument name="eventParam" type="string" required="true" />
    <cfargument name="url" type="string" required="false" default="" />
    <cfargument name="args" type="string" required="false" default="" />

    <cfset setEventName(arguments.eventName) />
    <cfset setEventParam(arguments.eventParam) />
    <cfset setUrl(arguments.url) />
    <cfset setArgs(arguments.args) />

    <cfreturn this />
</cffunction>
```

makeRedirectUrl

```
private string makeRedirectUrl( Event event, EventContext eventContext )
```

Assembles the redirect url.

Parameters:

Event event

EventContext eventContext

Code:

```
<cffunction name="makeRedirectUrl" access="private" returntype="string" output="false"
    hint="Assembles the redirect url.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

    <cfset var redirectUrl = getUrl() />
    <cfset var redirectQueryStringParam = "" />
    <cfset var redirectQueryString = "" />
    <cfset var argNames = getArgs() />
    <cfset var argName = "" />

    <cfif redirectUrl EQ ''>
        <cfset redirectUrl = "index.cfm" />
    </cfif>

    <cfif Find('?', redirectUrl) GT 0>
        <cfset redirectQueryStringParam = '&' />
    <cfelse>
        <cfset redirectQueryStringParam = '?' />
    </cfif>

    <cfif getEventName() NEQ ''>
        <cfset redirectQueryString = getEventParam() & '=' & getEventName() />
    </cfif>

    <cfloop index="argName" list="#argNames#" delimiters=",">
        <cfif arguments.event.isArgDefined(argName) AND IsSimpleValue(arguments.event.getArg(argName, ''))>
            <cfset redirectQueryString = redirectQueryString & '&' & argName & '=' & URLEncodedFormat(arguments.event.getArg(argName, '')) />
        </cfif>
    </cfloop>
</cffunction>
```

```
        </cfloop>
        <cfif Len(redirectQueryString)>
            <cfreturn redirectUrl & redirectQueryStringParam & redirectQueryString />
        <cfelse>
            <cfreturn redirectUrl />
        </cfif>
    </cffunction>
```

setArgs

```
private void setArgs( string args )
```

Parameters:

string args

Code:

```
<cffunction name="setArgs" access="private" returnType="void" output="false">
    <cfargument name="args" type="string" required="true" />
    <cfset variables.args = arguments.args />
</cffunction>
```

setEventName

```
private void setEventName( string eventName )
```

Parameters:

string eventName

Code:

```
<cffunction name="setEventName" access="private" returnType="void" output="false">
```

```
<cfargument name="eventName" type="string" required="true" />
<cfset variables.eventName = arguments.eventName />
</cffunction>
```

setEventParam

private void setEventParam(string eventParam)

Parameters:

string eventParam

Code:

```
<cffunction name="setEventParam" access="private" returnType="void" output="false">
    <cfargument name="eventParam" type="string" required="true" />
    <cfset variables.eventParam = arguments.eventParam />
</cffunction>
```

setUrl

private void setUrl(string url)

Parameters:

string url

Code:

```
<cffunction name="setUrl" access="private" returnType="void" output="false">
    <cfargument name="url" type="string" required="true" />
    <cfset variables.url = arguments.url />
</cffunction>
```

ViewPageCommand

Package: MachII.framework.commands

Inherits from: framework.EventCommand

An EventCommand for processing a view.

Method Summary

public ViewPage-Command	init(string viewName, [string contentKey=""], [string contentArg=""], [string append="false"]) Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext) Executes the command.
private boolean	getAppend()
private string	getContentArg()
private string	getContentKey()
private string	getViewName()
private boolean	hasContentArg()
private boolean	hasContentKey()
private void	setAppend(string append)
private void	setContentArg(string contentArg)
private void	setContentKey(string contentKey)
private void	setViewName(string viewName)

Methods inherited from framework.EventCommand: setParameter , getParameter , setParameters

Method Detail**execute**

```
public boolean execute( Event event, EventContext eventContext )
```

Executes the command.

Parameters:

Event event

EventContext eventContext

Code:

```
<cffunction name="execute" access="public" returntype="boolean" output="true"
    hint="Executes the command.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

    <cfset arguments.eventContext.displayView(arguments.event, getViewName(), getContentKey(), getContentArg(), getAp

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

getAppend

```
private boolean getAppend( )
```

Parameters:

Code:

```
<cffunction name="getAppend" access="private" returntype="boolean" output="false">
    <cfreturn variables.append />
</cffunction>
```

getContentArg

```
private string getContentArg( )
```

Parameters:

Code:

```
<cffunction name="getContentArg" access="private" returntype="string" output="false">
    <cfreturn variables.contentArg />
</cffunction>
```

getContentKey

```
private string getContentKey( )
```

Parameters:

Code:

```
<cffunction name="getContentKey" access="private" returntype="string" output="false">
    <cfreturn variables.contentKey />
</cffunction>
```

getViewName

```
private string getViewName( )
```

Parameters:

Code:

```
<cffunction name="getViewName" access="private" returnType="string" output="false">
    <cfreturn variables.viewName />
</cffunction>
```

hasContentArg

private boolean hasContentArg()

Parameters:

Code:

```
<cffunction name="hasContentArg" access="private" returnType="boolean" output="false">
    <cfreturn variables.contentArg NEQ '' />
</cffunction>
```

hasContentKey

private boolean hasContentKey()

Parameters:

Code:

```
<cffunction name="hasContentKey" access="private" returnType="boolean" output="false">
    <cfreturn variables.contentKey NEQ '' />
</cffunction>
```

init

public ViewPageCommand init(string viewName, [string contentKey=""], [string contentArg=""], [string append="false"])

Used by the framework for initialization.

Parameters:

string viewName
[string contentKey=""]
[string contentArg=""]
[string append="false"]

Code:

```
<cffunction name="init" access="public" returnType="ViewPageCommand" output="false"
    hint="Used by the framework for initialization.">
    <cfargument name="viewName" type="string" required="true" />
    <cfargument name="contentKey" type="string" required="false" default="" />
    <cfargument name="contentArg" type="string" required="false" default="" />
    <cfargument name="append" type="string" required="false" default="false" />

    <cfset setViewName(arguments.viewName) />
    <cfset setContentKey(arguments.contentKey) />
    <cfset setContentArg(arguments.contentArg) />
    <cfset setAppend(arguments.append) />

    <cfreturn this />
</cffunction>
```

setAppend

```
private void setAppend( string append )
```

Parameters:

string append

Code:

```
<cffunction name="setAppend" access="private" returnType="void" output="false">
    <cfargument name="append" type="string" required="true" />
    <cfset variables.append = (arguments.append is "true") />
</cffunction>
```

setContentArg

private void setContentArg(string contentArg)

Parameters:

string contentArg

Code:

```
<cffunction name="setContentArg" access="private" returnType="void" output="false">
    <cfargument name="contentArg" type="string" required="true" />
    <cfset variables.contentArg = arguments.contentArg />
</cffunction>
```

setContentKey

private void setContentKey(string contentKey)

Parameters:

string contentKey

Code:

```
<cffunction name="setContentKey" access="private" returnType="void" output="false">
    <cfargument name="contentKey" type="string" required="true" />
    <cfset variables.contentKey = arguments.contentKey />
</cffunction>
```

setViewName

private void setViewName(string viewName)

Parameters:

string viewName

Code:

```
<cffunction name="setViewName" access="private" returnType="void" output="false">
  <cfargument name="viewName" type="string" required="true" />
  <cfset variables.viewName = arguments.viewName />
</cffunction>
```

framework.invokers

CFCInvoker_Event

Package: MachII.framework.invokers

Inherits from: framework.ListenerInvoker

DEPRECATED. ListenerInvoker that invokes a Listener's method passing the Event as the sole argument.

Method Summary

public CFCInvoker_Event	init() Used by the framework for initialization. Do not override.
public void	invokeListener(Event event, Listener listener, string method, [string resultKey=""], [string resultArg=""]) Invokes the Listener.

Method Detail

init

```
public CFCInvoker_Event init( )
```

Used by the framework for initialization. Do not override.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="CFCInvoker_Event" output="false"
  hint="Used by the framework for initialization. Do not override.">
  <cfreturn this />
```

```
</cffunction>
```

invokeListener

```
public void invokeListener( Event event, Listener listener, string method, [string resultKey=""], [string resultArg=""] )
```

Invokes the Listener.

Parameters:

Event event
Listener listener
string method
[string resultKey=""]
[string resultArg=""]

Code:

```
<cffunction name="invokeListener" access="public" returntype="void"
  hint="Invokes the Listener.">
  <cfargument name="event" type="MachII.framework.Event" required="true"
    hint="The Event triggering the invocation." />
  <cfargument name="listener" type="MachII.framework.Listener" required="true"
    hint="The Listener to invoke." />
  <cfargument name="method" type="string" required="true"
    hint="The name of the Listener's method to invoke." />
  <cfargument name="resultKey" type="string" required="false" default=""
    hint="The variable to set the result in." />
  <cfargument name="resultArg" type="string" required="false" default=""
    hint="Not supported." />

  <cfset var resultValue = "" />
  <cftry>
    <cfinvoke
      component="#arguments.listener#"
      method="#arguments.method#"
      event="#arguments.event#"
      returnvariable="resultValue" />
  </cftry>
</cffunction>
```

```
        <cfif arguments.resultKey NEQ ''>
            <cfset "#arguments.resultKey#" = resultValue />
        </cfif>

        <cfcatch type="Any">
            <cfrethrow />
        </cfcatch>
    </cftry>
</cffunction>
```

CFCInvoker_EventArgs

Package: MachII.framework.invokers

Inherits from: framework.ListenerInvoker

DEPRECATED. ListenerInvoker that invokes a Listener's method passing the Event's args as an argument collection.

Method Summary

public CFCInvoker_EventArgs	init() Used by the framework for initialization. Do not override.
public void	invokeListener(Event event, Listener listener, string method, [string resultKey=""], [string resultArg=""]) Invokes the Listener.

Method Detail

init

```
public CFCInvoker_EventArgs init( )
```

Used by the framework for initialization. Do not override.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="CFCInvoker_EventArgs" output="false"
  hint="Used by the framework for initialization. Do not override.">
  <cfreturn this />
```

```
</cffunction>
```

invokeListener

```
public void invokeListener( Event event, Listener listener, string method, [string resultKey=""], [string resultArg=""] )
```

Invokes the Listener.

Parameters:

Event event
Listener listener
string method
[string resultKey=""]
[string resultArg=""]

Code:

```
<cffunction name="invokeListener" access="public" returntype="void"
  hint="Invokes the Listener.">
  <cfargument name="event" type="MachII.framework.Event" required="true"
    hint="The Event triggering the invocation." />
  <cfargument name="listener" type="MachII.framework.Listener" required="true"
    hint="The Listener to invoke." />
  <cfargument name="method" type="string" required="true"
    hint="The name of the Listener's method to invoke." />
  <cfargument name="resultKey" type="string" required="false" default=""
    hint="The variable to set the result in." />
  <cfargument name="resultArg" type="string" required="false" default=""
    hint="Not supported." />

  <cfset var resultValue = "" />
  <cftry>
    <cfinvoke
      component="#arguments.listener#"
      method="#arguments.method#"
      argumentcollection="#arguments.event.getArgs()#"
      returnvariable="resultValue" />
  </cftry>
</cffunction>
```

```
        <cfif arguments.resultKey NEQ ''>
            <cfset "#arguments.resultKey#" = resultValue />
        </cfif>

        <cfcatch type="Any">
            <cfrethrow />
        </cfcatch>
    </cftry>
</cffunction>
```

EventArgsInvoker

Package: MachII.framework.invokers

Inherits from: framework.ListenerInvoker

ListenerInvoker that invokes a Listener's method passing the Event's args as an argument collection.

Method Summary

public EventArgsInvoker	init() Used by the framework for initialization. Do not override.
public void	invokeListener(Event event, Listener listener, string method, [string resultKey=""], [string resultArg=""]) Invokes the Listener.

Method Detail

init

```
public EventArgsInvoker init( )
```

Used by the framework for initialization. Do not override.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="EventArgsInvoker" output="false"
  hint="Used by the framework for initialization. Do not override.">
  <cfreturn this />
```

```
</cffunction>
```

invokeListener

```
public void invokeListener( Event event, Listener listener, string method, [string resultKey=""], [string resultArg=""] )
```

Invokes the Listener.

Parameters:

Event event
Listener listener
string method
[string resultKey=""]
[string resultArg=""]

Code:

```
<cffunction name="invokeListener" access="public" returntype="void"
  hint="Invokes the Listener.">
  <cfargument name="event" type="MachII.framework.Event" required="true"
    hint="The Event triggering the invocation." />
  <cfargument name="listener" type="MachII.framework.Listener" required="true"
    hint="The Listener to invoke." />
  <cfargument name="method" type="string" required="true"
    hint="The name of the Listener's method to invoke." />
  <cfargument name="resultKey" type="string" required="false" default=""
    hint="The variable to set the result in." />
  <cfargument name="resultArg" type="string" required="false" default=""
    hint="The eventArg to set the result in." />

  <cfset var resultValue = "" />
  <cftry>
    <cfinvoke
      component="#arguments.listener#"
      method="#arguments.method#"
      argumentcollection="#arguments.event.getArgs()#"
      returnvariable="resultValue" />
  </cftry>
</cffunction>
```

```
<cfif arguments.resultKey NEQ ''>
    <cfset "#arguments.resultKey#" = resultValue />
</cfif>

<cfif arguments.resultArg NEQ ''>
    <cfset arguments.event.setArg(arguments.resultArg, resultValue) />
</cfif>

<cfcatch type="expression">
    <cfif FindNoCase("RESULTVALUE", cfcatch.Message)>
        <cfthrow type="MachII.framework.VoidReturnType"
            message="A ResultArg/Key has been specified on a notify command method th
            detail="Notify method name: '#arguments.method#'" />
    <cfelse>
        <cfrethrow />
    </cfif>
</cfcatch>
<cfcatch type="Any">
    <cfrethrow />
</cfcatch>
</cftry>
</cffunction>
```

EventInvoker

Package: MachII.framework.invokers

Inherits from: framework.ListenerInvoker

ListenerInvoker that invokes a Listener's method passing the Event as the sole argument.

Method Summary

public EventInvoker	init() Used by the framework for initialization. Do not override.
public void	invokeListener(Event event, Listener listener, string method, [string resultKey=""], [string resultArg=""]) Invokes the Listener.

Method Detail

init

```
public EventInvoker init( )
```

Used by the framework for initialization. Do not override.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="EventInvoker" output="false"
  hint="Used by the framework for initialization. Do not override.">
  <cfreturn this />
```

```
</cffunction>
```

invokeListener

```
public void invokeListener( Event event, Listener listener, string method, [string resultKey=""], [string resultArg=""] )
```

Invokes the Listener.

Parameters:

Event event
Listener listener
string method
[string resultKey=""]
[string resultArg=""]

Code:

```
<cffunction name="invokeListener" access="public" returntype="void"
  hint="Invokes the Listener.">
  <cfargument name="event" type="MachII.framework.Event" required="true"
    hint="The Event triggering the invocation." />
  <cfargument name="listener" type="MachII.framework.Listener" required="true"
    hint="The Listener to invoke." />
  <cfargument name="method" type="string" required="true"
    hint="The name of the Listener's method to invoke." />
  <cfargument name="resultKey" type="string" required="false" default=""
    hint="The variable to set the result in." />
  <cfargument name="resultArg" type="string" required="false" default=""
    hint="The eventArg to set the result in." />

  <cfset var resultValue = "" />
  <cftry>
    <cfinvoke
      component="#arguments.listener#"
      method="#arguments.method#"
      event="#arguments.event#"
      returnvariable="resultValue" />
  </cftry>
</cffunction>
```

```
<cfif arguments.resultKey NEQ ''>
    <cfset "#arguments.resultKey#" = resultValue />
</cfif>

<cfif arguments.resultArg NEQ ''>
    <cfset arguments.event.setArg(arguments.resultArg, resultValue) />
</cfif>

<cfcatch type="expression">
    <cfif FindNoCase("RESULTVALUE", cfcatch.Message)>
        <cfthrow type="MachII.framework.VoidReturnType"
            message="A ResultArg/Key has been specified on a notify command method th
            detail="Notify method name: '#arguments.method#'" />
    <cfelse>
        <cfrethrow />
    </cfif>
</cfcatch>
<cfcatch type="Any">
    <cfrethrow />
</cfcatch>
</cftry>
</cffunction>
```

plugins

SimplePlugin

Package: MachII.plugins

Inherits from: framework.BaseComponent < framework.Plugin

A simple Plugin example.

Method Summary

public void	configure() Configures the plugin.
public void	handleException(EventContext eventContext, Exception exception)
public void	postEvent(EventContext eventContext)
public void	postProcess(EventContext eventContext)
public void	postView(EventContext eventContext)
public void	preEvent(EventContext eventContext)
public void	preProcess(EventContext eventContext)
public void	preView(EventContext eventContext)

Methods inherited from framework.Plugin: init , abortEvent

Methods inherited from framework.BaseComponent: isParameterDefined , setPropertyManager , setParameters , getPropertyManager , announceEvent , getAppManager , setAppManager , hasParameter , getParameter , getProperty , getParameters , setProperty , setParameter

Method Detail

configure

```
public void configure( )
```

Configures the plugin.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void" output="false"
    hint="Configures the plugin.">
</cffunction>
```

handleException

```
public void handleException( EventContext eventContext, Exception exception )
```

Parameters:

EventContext eventContext

Exception exception

Code:

```
<cffunction name="handleException" access="public" returntype="void" output="true">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfargument name="exception" type="MachII.util.Exception" required="true" />
    <cfoutput>&nbsp;SimplePlugin.handleException()<br /></cfoutput>
</cffunction>
```

postEvent

```
public void postEvent( EventContext eventContext )
```

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postEvent" access="public" returntype="void" output="true">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfoutput>&nbsp;SimplePlugin.postEvent()<br /></cfoutput>
</cffunction>
```

postProcess

```
public void postProcess( EventContext eventContext )
```

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postProcess" access="public" returntype="void" output="true">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfoutput>&nbsp;SimplePlugin.postProcess()<br /></cfoutput>
</cffunction>
```

postView

```
public void postView( EventContext eventContext )
```

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postView" access="public" returntype="void" output="true">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfoutput>&nbsp;SimplePlugin.postView()<br /></cfoutput>
</cffunction>
```

preEvent

```
public void preEvent( EventContext eventContext )
```

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preEvent" access="public" returntype="void" output="true">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfoutput>&nbsp;SimplePlugin.preEvent()<br /></cfoutput>
</cffunction>
```

preProcess

```
public void preProcess( EventContext eventContext )
```

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preProcess" access="public" returntype="void" output="true">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfoutput>&nbsp;SimplePlugin.preProcess()<br /></cfoutput>
```

```
</cffunction>
```

preView

```
public void preView( EventContext eventContext )
```

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preView" access="public" returntype="void" output="true">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfoutput>&nbsp;SimplePlugin.preView()<br /></cfoutput>
</cffunction>
```

TracePlugin

Package: MachII.plugins

Inherits from: framework.BaseComponent < framework.Plugin

Traces the execution of Mach-II events and displays the trace information on screen and/or logs it to a file.

Method Summary

private void	appendTrace(string event, string point, string timing)	Appends a trace to the trace information array or to the log file.
private string	computeEventName(EventContext eventContext, string point)	Computes the event name for this trace.
private string	computeTraceTime()	Computes the trace time from the last trace until now.
public void	configure()	Configures the plugin.
private string	displayTraceInfo(array traceInfo, string requestEventName)	Gets the trace information and formats for on-screen or HTML commented display.
private boolean	getDisplayCommented()	
private string	getFileName()	
private numeric	getHighlightLongTimings()	
private boolean	getIsInitialTrace()	Gets the initial trace flag from the request.tracePluginScope.
private string	getMachIIVersion()	

Method Summary

	Gets a nice version number instead of just numbers.
private string	getSuppressTraceArg()
private numeric	getTick() Gets the current tick from the request.tracePluginScope.
private numeric	getTickStart() Gets the tick start from the request.tracePluginScope.
private array	getTraceInfo() Gets the trace info array from the request.tracePluginScope.
private string	getTraceMode()
private boolean	getTraceRequest() Gets the trace request from the request.tracePluginScope.
public void	handleException(EventContext eventContext, Exception exception) Runs a trace when an exception occurs (before exception event is handled).
private boolean	hasInitialTrace() Checks if the initial trace flag exists in the request.tracePluginScope.
private boolean	isTrueNumeric(string str) Returns true if all characters in a string are numeric.
public void	postEvent(EventContext eventContext) Runs the trace for the postEvent plugin point.
public void	postProcess(EventContext eventContext) Ends the trace if the trace mode is not none and displays trace on screen if applicable.
public void	postView(EventContext eventContext) Runs the trace for the postView plugin point.

Method Summary

public void	preEvent(EventContext eventContext) Runs the trace for the preEvent plugin point.
public void	preProcess(EventContext eventContext) Starts the trace if mode is not none.
public void	preView(EventContext eventContext) Runs the trace for the preView plugin point.
private void	setDisplayCommented(boolean displayCommented)
private void	setFileName(string fileName)
private void	setHighlightLongTimings(numeric highlightLongTimings)
private void	setIsInitialTrace(boolean isInitialTrace) Sets the initial trace flag in the request.tracePluginScope.
private void	setSuppressTraceArg(string suppressTraceArg)
private void	setTick(numeric tick) Sets the current tick in the request.tracePluginScope.
private void	setTickStart(numeric tickStart) Sets the tick start in the request.tracePluginScope.
private void	setTraceInfo([array traceInfo]) Sets the trace info array in the request.tracePluginScope.
private void	setTraceMode(string traceMode)
private void	setTraceRequest([boolean traceRequest]) Sets the trace request request.tracePluginScope.
private boolean	shouldTrace(boolean suppressTrace) Checks if we should trace
private void	throwUsageException(string message, [string detail="No details."])

Method Summary

	Throws an usage exception.
private void	trace(string point, EventContext eventContext)
	Runs a trace for the passed point and eventContext.

Methods inherited from framework.Plugin: init , abortEvent

Methods inherited from framework.BaseComponent: isParameterDefined , setPropertyManager , setParameters , getPropertyManager , announceEvent , getAppManager , setAppManager , hasParameter , getParameter , getProperty , getParameters , setProperty , setParameter

Method Detail

appendTrace

private void appendTrace(string event, string point, string timing)

Appends a trace to the trace information array or to the log file.

Parameters:

string event
 string point
 string timing

Code:

```
<cffunction name="appendTrace" access="private" returntype="void" output="false"
    hint="Appends a trace to the trace information array or to the log file.">
    <cfargument name="event" type="string" required="true"
        hint="Name of event for this trace." />
    <cfargument name="point" type="string" required="true"
```

```

        hint="Name of plugin method for this trace." />
<cfargument name="timing" type="string" required="true"
    hint="Timing for this trace." />
<cfset var trace = structNew() />

<cfset trace.event = arguments.event />
<cfset trace.point = arguments.point />
<cfset trace.timing = arguments.timing />

<cfif ListFindNoCase("display,both", getTraceMode())>
    <cfset arrayAppend(getTraceInfo(), trace) />
</cfif>
<cfif ListFindNoCase("file,both", getTraceMode())>
    <cflog file="#getFileName()#" text="( #trace.timing# ) - #trace.event# :: #trace.point#" />
</cfif>
</cffunction>

```

computeEventName

```
private string computeEventName( EventContext eventContext, string point )
```

Computes the event name for this trace.

Parameters:

EventContext eventContext
string point

Code:

```

<cffunction name="computeEventName" access="private" returntype="string" output="false"
    hint="Computes the event name for this trace.">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfargument name="point" type="string" required="true" />

    <cfif NOT ListFindNoCase("postProcess,preProcess", arguments.point) AND arguments.eventContext.hasCurrentEvent()>
        <cfreturn arguments.eventContext.getCurrentEvent().getName() />
    <cfelse>
        <cfreturn "Core Process" />
    </cfif>
</cffunction>

```

```
</cfif>  
</cffunction>
```

computeTraceTime

```
private string computeTraceTime( )
```

Computes the trace time from the last trace until now.

Parameters:

Code:

```
<cffunction name="computeTraceTime" access="private" returntype="string" output="false"  
    hint="Computes the trace time from the last trace until now.">  
    <cfset var currentTick = getTickCount() />  
    <cfset var timing = "" />  
  
    <cfif NOT getIsInitialTrace(>  
        <cfset timing = currentTick - getTick()/>  
    <cfelse>  
        <cfset timing = "-" />  
        <cfset setIsInitialTrace(FALSE) />  
        <cfset setTickStart(currentTick) />  
    </cfif>  
  
    <cfset setTick(currentTick) />  
  
    <cfreturn timing />  
</cffunction>
```

configure

```
public void configure( )
```

Configures the plugin.

Parameters:

Code:

```

<cffunction name="configure" access="public" returntype="void" output="false"
  hint="Configures the plugin.">
  <cfset var params = getParameters() />
  <cfset var tempTraceMode = "" />
  <cfset var tempDisplayCommented = "" />

  <cfif StructKeyExists(params, "traceMode")>
    <cfset tempTraceMode = params.traceMode />

    <cfif REFindNoCase("\${(.)*?}", tempTraceMode)>

      <cfset tempTraceMode = Mid(tempTraceMode, 3, Len(tempTraceMode) -3) />

      <cfif NOT getPropertyManager().isPropertyDefined(tempTraceMode)>
        <cfset throwUsageException("The {traceMode} parameter dynamic property cannot be found in
          "Please check that the '#tempTraceMode#' property is available." />
      <cfelse>
        <cfset tempTraceMode = getProperty(tempTraceMode) />
      </cfif>
    </cfif>

    <cfif NOT ListFindNoCase("display,file,both,none", tempTraceMode)>
      <cfset throwUsageException("The TracePlugin {traceMode} parameter must be display, file, both or
    <cfelse>
      <cfset setTraceMode(tempTraceMode) />
    </cfif>
  </cfif>
  <cfif StructKeyExists(params, "displayCommented")>
    <cfset tempDisplayCommented = params.displayCommented />

    <cfif REFindNoCase("\${(.)*?}", tempDisplayCommented)>

      <cfset tempDisplayCommented = Mid(tempDisplayCommented, 3, Len(tempDisplayCommented) -3) />

      <cfif NOT getPropertyManager().isPropertyDefined(tempDisplayCommented)>
        <cfset throwUsageException("The {displayCommented} parameter dynamic property cannot be f
          "Please check that the '#tempDisplayCommented#' property is avail
      <cfelse>

```

```

        <cfset tempDisplayCommented = getProperty(tempDisplayCommented) />
    </cfif>
</cfif>

<cfif NOT isBoolean(tempDisplayCommented)>
    <cfset throwUsageException("The TracePlugin {displayCommented} parameter must be a boolean value") />
<cfelse>
    <cfset setDisplayCommented(tempDisplayCommented) />
</cfif>
</cfif>
<cfif StructKeyExists(params, "highlightLongTimings")>
    <cfif NOT len(params.highlightLongTimings) OR NOT isTrueNumeric(params.highlightLongTimings)>
        <cfset throwUsageException("The TracePlugin {highlightLongTimings} parameter must be a numeric value") />
    <cfelse>
        <cfset setHighlightLongTimings(params.highlightLongTimings) />
    </cfif>
</cfif>
<cfif StructKeyExists(params, "fileName")>
    <cfif NOT len(params.fileName)>
        <cfset throwUsageException("The TracePlugin {fileName} parameter must not be blank. Please set a filename.") />
    <cfelse>
        <cfset setFilename(params.fileName) />
    </cfif>
</cfif>
<cfif StructKeyExists(params, "suppressTraceArg")>
    <cfif NOT len(params.suppressTraceArg)>
        <cfset throwUsageException("The TracePlugin {suppressTraceArg} parameter must not be blank. Please set a suppressTraceArg.") />
    <cfelse>
        <cfset setSuppressTraceArg(params.suppressTraceArg) />
    </cfif>
</cfif>
</cffunction>

```

displayTraceInfo

private string displayTraceInfo(array traceInfo, string requestEventName)

Gets the trace information and formats for on-screen or HTML commented display.

Parameters:

array traceInfo
string requestEventName

Code:

```

<cffunction name="displayTraceInfo" access="private" returntype="string" output="false"
  hint="Gets the trace information and formats for on-screen or HTML commented display.">
  <cfargument name="traceInfo" type="array" required="true"
    hint="Pass in the array from the getTraceInfo() method." />
  <cfargument name="requestEventName" type="string" required="true"
    hint="The event name that started the request lifecycle.">

  <cfset var sc = "" />
  <cfset var traceInfoArrLen = ArrayLen(arguments.traceInfo) />
  <cfset var i = "" />

  <cfif getDisplayCommented(>
    <!-- Leave this code block as-is for proper HTML formatting --->
    <cfsavecontent variable="sc">
      <cfoutput><!--
        Mach-II Trace Information
        *****
        Event Name :: Point Name :: Average Time
        *****
        <cfloop from="1" to="#ArrayLen(arguments.traceInfo)-1#" index="i">#arguments.traceInfo[i].event#
        </cfloop>#arguments.traceInfo[traceInfoArrLen].event# - #arguments.traceInfo[traceInfoArrLen].ti
        *****
        Request Event Name: #arguments.requestEventName#
        Mach-Version: #getPropertyManager().getVersion()#
        Timestamp: #DateFormat(Now())# #TimeFormat(Now())#
        --></cfoutput>
      </cfsavecontent>
    <cfelse>
      <cfsavecontent variable="sc">
        <cfoutput>
          <div id="MachIITraceDisplay">
            <h3>Mach-II Trace Information</h3>
            <table style="border: 1px solid #D0D0D0; padding: 0.5em; width:100%;">
              <tr>
                <td style="border-bottom: 1px solid #000; width:65%;"><strong>Event Name</strong>
                <td style="border-bottom: 1px solid #000; width:20%;"><strong>Trace Point</strong>
                <td style="border-bottom: 1px solid #000; width:15%;"><strong>* Average Time</strong>
              </tr>
            <cfloop from="1" to="#ArrayLen(traceInfo)-1#" index="i">

```

```

        <tr <cfif i MOD 2>style="background-color:##F5F5F5" class="shade"</cfif>>
            <td>#arguments.traceInfo[i].event#</td>
            <td>#arguments.traceInfo[i].point#</td>
        <cfif getHighlightLongTimings() NEQ 0 AND arguments.traceInfo[i].timing GTE getHighlightLongTimings()>
            <td style="text-align: right;"><strong>#arguments.traceInfo[i].timing#</strong> r
        </td>
        </tr>
    </cfloop>
    <tr>
        <td colspan="2" style="border-top: 1px solid ##000;"><em>#arguments.traceInfo[tra
        <td style="border-top: 1px solid ##000; text-align: right;"><em>#arguments.trace
    </td>
    </tr>
    <cfif getHighlightLongTimings()>
        <tr>
            <td colspan="3" style="text-align:right;"><strong>* Timings over #getHighlightLon
        </td>
        </tr>
    </cfif>
</table>
<h3>General Information</h3>
<table style="border: 1px solid ##D0D0D0; padding: 0.5em; width:100%;">
    <tr style="background-color:##F5F5F5" class="shade">
        <td style="border-top: 1px solid ##000;"><strong>Request Event Name</strong></td>
        <td style="border-top: 1px solid ##000;">#arguments.requestEventName#</td>
    </tr>
    <tr>
        <td><strong>Mach-II Version</strong></td>
        <td>#getMachIIVersion()#</td>
    </tr>
    <tr style="background-color:##F5F5F5" class="shade">
        <td><strong>Timestamp</strong></td>
        <td>#DateFormat(Now())# #TimeFormat(Now())#</td>
    </tr>
</table>
</div>
</cfoutput>
</cfsavecontent>
</cfif>

    <cfreturn replace(sc, chr(9) & chr(9) & chr(9) & chr(9), "", "ALL") />
</cffunction>

```

getDisplayCommented

private boolean getDisplayCommented()

Parameters:

Code:

```
<cffunction name="getDisplayCommented" access="private" returntype="boolean" output="false">
    <cfreturn variables.instance.displayCommented />
</cffunction>
```

getFileName

private string getFileName()

Parameters:

Code:

```
<cffunction name="getFileName" access="private" returntype="string" output="false">
    <cfreturn variables.instance.fileName />
</cffunction>
```

getHighlightLongTimings

private numeric getHighlightLongTimings()

Parameters:

Code:

```
<cffunction name="getHighlightLongTimings" access="private" returntype="numeric" output="false">
    <cfreturn variables.instance.highlightLongTimings />
</cffunction>
```

getIsInitialTrace

```
private boolean getIsInitialTrace( )
```

Gets the initial trace flag from the request.tracePluginScope.

Parameters:

Code:

```
<cffunction name="getIsInitialTrace" access="private" returnType="boolean" output="false"
  hint="Gets the initial trace flag from the request.tracePluginScope.">
  <cftry>
    <cfreturn request.tracePluginScope.isInitialTrace />
  <cfcatch type="expression">
    <cfset throwUsageException("Required request scope variable missing.", "Do not delete request.tracePluginScope") />
  </cfcatch>
</cftry>
</cffunction>
```

getMachIIVersion

```
private string getMachIIVersion( )
```

Gets a nice version number instead of just numbers.

Parameters:

Code:

```
<cffunction name="getMachIIVersion" access="private" returnType="string" output="false"
  hint="Gets a nice version number instead of just numbers.">
  <cfset var version = getPropertyManager().getVersion() />
  <cfset var release = "" />

  <cfswitch expression="#Right(version, 1)#">
    <cfcase value="0">
      <cfset release = "Pre-Alpha / Bleeding Edge Release" />
    </cfcase>
  </cfswitch>
</cffunction>
```

```
</cfcase>
<cfcase value="1">
    <cfset release = "Alpha" />
</cfcase>
<cfcase value="2">
    <cfset release = "Beta" />
</cfcase>
<cfcase value="3">
    <cfset release = "RC1" />
</cfcase>
<cfcase value="4">
    <cfset release = "RC2" />
</cfcase>
<cfcase value="5">
    <cfset release = "RC3" />
</cfcase>
<cfcase value="6">
    <cfset release = "RC4" />
</cfcase>
<cfcase value="7">
    <cfset release = "RC5" />
</cfcase>
<cfcase value="8">
    <cfset release = "Development and Production Stable (non-duck typed core)" />
</cfcase>
<cfcase value="9">
    <cfset release = "Production-Only Stable (duck-typed core for performance)" />
</cfcase>
<cfdefaultcase>
    <cfset release = "Unknown Release" />
</cfdefaultcase>
</cfswitch>

    <cfreturn Left(version, Len(version) - 2) & " " & release />
</cffunction>
```

getSuppressTraceArg

```
private string getSuppressTraceArg( )
```

Parameters:

Code:

```
<cffunction name="getSuppressTraceArg" access="private" returntype="string" output="false">
  <cfreturn variables.instance.suppressTraceArg />
</cffunction>
```

getTick

private numeric getTick()

Gets the current tick from the request.tracePluginScope.

Parameters:

Code:

```
<cffunction name="getTick" access="private" returntype="numeric" output="false"
  hint="Gets the current tick from the request.tracePluginScope.">
  <cftry>
    <cfreturn request.tracePluginScope.tick />
    <cfcatch type="expression">
      <cfset throwUsageException("Required request scope variable missing.", "Do not delete request.tracePluginScope") />
    </cfcatch>
  </cftry>
</cffunction>
```

getTickStart

private numeric getTickStart()

Gets the tick start from the request.tracePluginScope.

Parameters:

Code:

```
<cffunction name="getTickStart" access="private" returntype="numeric" output="false"
  hint="Gets the tick start from the request.tracePluginScope.">
  <cftry>
    <cfreturn request.tracePluginScope.tickStart />
  <cfcatch type="expression">
    <cfset throwUsageException("Required request scope variable missing.", "Do not delete request.tracePluginScope.tickStart") />
  </cfcatch>
</cftry>
</cffunction>
```

getTraceInfo

private array getTraceInfo()

Gets the trace info array from the request.tracePluginScope.

Parameters:

Code:

```
<cffunction name="getTraceInfo" access="private" returntype="array" output="false"
  hint="Gets the trace info array from the request.tracePluginScope.">
  <cftry>
    <cfreturn request.tracePluginScope.traceInfo />
  <cfcatch type="expression">
    <cfset throwUsageException("Required request scope variable missing.", "Do not delete request.tracePluginScope.traceInfo") />
  </cfcatch>
</cftry>
</cffunction>
```

getTraceMode

private string getTraceMode()

Parameters:

Code:

```
<cffunction name="getTraceMode" access="private" returnType="string" output="false">
    <cfreturn variables.instance.traceMode />
</cffunction>
```

getTraceRequest

private boolean getTraceRequest()

Gets the trace request from the request.tracePluginScope.

Parameters:

Code:

```
<cffunction name="getTraceRequest" access="private" returnType="boolean" output="false"
    hint="Gets the trace request from the request.tracePluginScope.">
    <cftry>
        <cfreturn request.tracePluginScope.traceRequest />
        <cfcatch type="expression">
            <cfset throwUsageException("Required request scope variable missing.", "Do not delete request.tracePluginScope") />
        </cfcatch>
    </cftry>
</cffunction>
```

handleException

public void handleException(EventContext eventContext, Exception exception)

Runs a trace when an exception occurs (before exception event is handled).

Parameters:

EventContext eventContext
Exception exception

Code:

```

<cffunction name="handleException" access="public" returntype="void" output="false"
    hint="Runs a trace when an exception occurs (before exception event is handled).">
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
    <cfargument name="exception" type="MachII.util.Exception" required="true" />
    <cfset var methodTraceInfo = structNew() />

    <cfset var event = "" />

    <cfif hasIsInitialTrace()>

        <cfset event = arguments.eventContext.getCurrentEvent() />

        <cfif shouldTrace(event.isArgDefined(getSuppresTraceArg()))>
            <cfset trace("handleException", arguments.eventContext) />
            <cfset appendTrace("Message: " & arguments.exception.getMessage(), "exception", "-") />
        </cfif>
    </cfif>
</cffunction>

```

hasIsInitialTrace

private boolean hasIsInitialTrace()

Checks if the initial trace flag exists in the request.tracePluginScope.

Parameters:

Code:

```

<cffunction name="hasIsInitialTrace" access="private" returntype="boolean" output="false"
    hint="Checks if the initial trace flag exists in the request.tracePluginScope.">
    <cfreturn IsDefined("request.tracePluginScope.isInitialTrace") />
</cffunction>

```

isTrueNumeric

```
private boolean isTrueNumeric( string str )
```

Returns true if all characters in a string are numeric.

Parameters:

string str

Code:

```
<cffunction name="isTrueNumeric" access="private" returntype="boolean" output="false"
  hint="Returns true if all characters in a string are numeric.">
  <cfargument name="str" type="string" required="true"
    hint="String to check.">
    <cfreturn REFind("[^0-9]", arguments.str) IS 0 />
  </cfargument>
</cffunction>
```

postEvent

```
public void postEvent( EventContext eventContext )
```

Runs the trace for the postEvent plugin point.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postEvent" access="public" returntype="void" output="false"
  hint="Runs the trace for the postEvent plugin point.">
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var event = arguments.eventContext.getCurrentEvent() />

  <cfif shouldTrace(event.isArgDefined(getSuppressTraceArg()))>
    <cfset trace("postEvent", arguments.eventContext) />
  </cfif>
</cffunction>
```

postProcess

```
public void postProcess( EventContext eventContext )
```

Ends the trace if the trace mode is not none and displays trace on screen if applicable.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postProcess" access="public" returntype="void" output="true"
  hint="Ends the trace if the trace mode is not none and displays trace on screen if applicable.">
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var event = arguments.eventContext.getCurrentEvent() />

  <cfif shouldTrace(event.isArgDefined(getSuppressTraceArg()))>
    <cfset trace("postProcess", arguments.eventContext) />

    <cfset appendTrace("Total time", "", getTick() - getTickStart()) />

    <cfif ListFindNoCase("display,both", getTraceMode())>
      <cfoutput>#displayTraceInfo(getTraceInfo(), arguments.eventContext.getCurrentEvent().getRequestName()
    </cfif>
  </cfif>
</cffunction>
```

postView

```
public void postView( EventContext eventContext )
```

Runs the trace for the postView plugin point.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="postView" access="public" returntype="void" output="false"
  hint="Runs the trace for the postView plugin point.">
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var event = arguments.eventContext.getCurrentEvent() />

  <cfif shouldTrace(event.isArgDefined(getSuppressTraceArg()))>
    <cfset trace("postView", arguments.eventContext) />
  </cfif>
</cffunction>
```

preEvent

```
public void preEvent( EventContext eventContext )
```

Runs the trace for the preEvent plugin point.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preEvent" access="public" returntype="void" output="false"
  hint="Runs the trace for the preEvent plugin point.">
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var event = arguments.eventContext.getCurrentEvent() />

  <cfif shouldTrace(event.isArgDefined(getSuppressTraceArg()))>
    <cfset trace("preEvent", arguments.eventContext) />
  </cfif>
</cffunction>
```

preProcess

```
public void preProcess( EventContext eventContext )
```

Starts the trace if mode is not none.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preProcess" access="public" returntype="void" output="false"
  hint="Starts the trace if mode is not none.">
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var event = arguments.eventContext.getNextEvent() />

  <cfif NOT getTraceMode() IS "none">
    <cfset setTraceRequest(TRUE) />
  <cfelse>
    <cfset setTraceRequest(FALSE) />
  </cfif>

  <cfif shouldTrace(event.isArgDefined(getSuppressTraceArg()))>
    <cfset setIsInitialTrace(TRUE) />
    <cfset setTraceInfo(arrayNew(1)) />
    <cfset trace("preProcess", arguments.eventContext) />
  </cfif>
</cffunction>
```

preView

```
public void preView( EventContext eventContext )
```

Runs the trace for the preView plugin point.

Parameters:

EventContext eventContext

Code:

```
<cffunction name="preView" access="public" returntype="void" output="false"
  hint="Runs the trace for the preView plugin point.">
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

  <cfset var event = arguments.eventContext.getCurrentEvent() />

  <cfif shouldTrace(event.isArgDefined(getSuppressTraceArg()))>
    <cfset trace("preView", arguments.eventContext) />
  </cfif>
</cffunction>
```

setDisplayCommented

```
private void setDisplayCommented( boolean displayCommented )
```

Parameters:

boolean displayCommented

Code:

```
<cffunction name="setDisplayCommented" access="private" returntype="void" output="false">
  <cfargument name="displayCommented" type="boolean" required="true" />
  <cfset variables.instance.displayCommented = arguments.displayCommented />
</cffunction>
```

setFileName

```
private void setFileName( string fileName )
```

Parameters:

string fileName

Code:

```
<cffunction name="setFileName" access="private" returnType="void" output="false">
  <cfargument name="fileName" type="string" required="true" />
  <cfset variables.instance.fileName = arguments.fileName />
</cffunction>
```

setHighlightLongTimings

```
private void setHighlightLongTimings( numeric highlightLongTimings )
```

Parameters:

numeric highlightLongTimings

Code:

```
<cffunction name="setHighlightLongTimings" access="private" returnType="void" output="false">
  <cfargument name="highlightLongTimings" type="numeric" required="true" />
  <cfset variables.instance.highlightLongTimings = arguments.highlightLongTimings />
</cffunction>
```

setIsInitialTrace

```
private void setIsInitialTrace( boolean isInitialTrace )
```

Sets the initial trace flag in the request.tracePluginScope.

Parameters:

boolean isInitialTrace

Code:

```
<cffunction name="setIsInitialTrace" access="private" returnType="void" output="false"
  hint="Sets the initial trace flag in the request.tracePluginScope.">
```

```
<cfargument name="isInitialTrace" type="boolean" required="true" />
<cfset request.tracePluginScope.isInitialTrace = arguments.isInitialTrace />
</cffunction>
```

setSuppressTraceArg

private void setSuppressTraceArg(string suppressTraceArg)

Parameters:

string suppressTraceArg

Code:

```
<cffunction name="setSuppressTraceArg" access="private" returnType="void" output="false">
  <cfargument name="suppressTraceArg" type="string" required="true" />
  <cfset variables.instance.suppressTraceArg = arguments.suppressTraceArg />
</cffunction>
```

setTick

private void setTick(numeric tick)

Sets the current tick in the request.tracePluginScope.

Parameters:

numeric tick

Code:

```
<cffunction name="setTick" access="private" returnType="void" output="false"
  hint="Sets the current tick in the request.tracePluginScope.">
  <cfargument name="tick" type="numeric" required="true" />
  <cfset request.tracePluginScope.tick = arguments.tick />
</cffunction>
```

setTickStart

```
private void setTickStart( numeric tickStart )
```

Sets the tick start in the request.tracePluginScope.

Parameters:

numeric tickStart

Code:

```
<cffunction name="setTickStart" access="private" returntype="void" output="false"
  hint="Sets the tick start in the request.tracePluginScope.">
  <cfargument name="tickStart" type="numeric" required="true" />
  <cfset request.tracePluginScope.tickStart = arguments.tickStart />
</cffunction>
```

setTraceInfo

```
private void setTraceInfo( [array traceInfo] )
```

Sets the trace info array in the request.tracePluginScope.

Parameters:

[array traceInfo]

Code:

```
<cffunction name="setTraceInfo" access="private" returntype="void" output="false"
  hint="Sets the trace info array in the request.tracePluginScope.">
  <cfargument name="traceInfo" type="array" required="false" />
  <cfset request.tracePluginScope.traceInfo = arguments.traceInfo />
</cffunction>
```

setTraceMode

```
private void setTraceMode( string traceMode )
```

Parameters:

string traceMode

Code:

```
<cffunction name="setTraceMode" access="private" returnType="void" output="false">
    <cfargument name="traceMode" type="string" required="true" />
    <cfset variables.instance.traceMode = arguments.traceMode />
</cffunction>
```

setTraceRequest

```
private void setTraceRequest( [boolean traceRequest] )
```

Sets the trace request request.tracePluginScope.

Parameters:

[boolean traceRequest]

Code:

```
<cffunction name="setTraceRequest" access="private" returnType="void" output="false"
    hint="Sets the trace request request.tracePluginScope.">
    <cfargument name="traceRequest" type="boolean" required="false" />
    <cfset request.tracePluginScope.traceRequest = arguments.traceRequest />
</cffunction>
```

shouldTrace

```
private boolean shouldTrace( boolean suppressTrace )
```

Checks if we should trace

Parameters:

boolean suppressTrace

Code:

```
<cffunction name="shouldTrace" access="private" returnType="boolean" output="false"
  hint="Checks if we should trace">
  <cfargument name="suppressTrace" type="boolean" required="true" />

  <cfif getTraceRequest() AND arguments.suppressTrace>
    <cfsetting showdebugoutput="false" />
    <cfset setTraceRequest(FALSE) />
  </cfif>

  <cfreturn getTraceRequest() />
</cffunction>
```

throwUsageException

private void throwUsageException(string message, [string detail="No details."])

Throws an usage exception.

Parameters:

string message
[string detail="No details."]

Code:

```
<cffunction name="throwUsageException" access="private" returnType="void" output="false"
  hint="Throws an usage exception.">
  <cfargument name="message" type="string" required="true" />
  <cfargument name="detail" type="string" required="false" default="No details." />
  <cfthrow type="TracePlugin.usageException"
    message="#arguments.message#"
    detail="#arguments.detail#" />
</cffunction>
```

trace

```
private void trace( string point, EventContext eventContext )
```

Runs a trace for the passed point and eventContext.

Parameters:

string point
EventContext eventContext

Code:

```
<cffunction name="trace" access="private" returntype="void" output="false"
  hint="Runs a trace for the passed point and eventContext.">
  <cfargument name="point" type="string" required="true" />
  <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />
  <cfset appendTrace(computeEventName(arguments.eventContext, arguments.point), arguments.point, computeTraceTime(
</cffunction>
```

util

BeanUtil

Package: MachII.util

A utility class for working with bean components.

Method Summary

public BeanUtil	init() Used by the framework for initialization.
public any	createBean(string beanType, [struct initArgs]) Creates a bean and calls its init() function.
public struct	describeBean(any bean) Returns a struct of bean properties/values based on getters.
public any	getBeanField(any bean, string field) Returns the value of a field in a bean using method call getBeanField().
public void	setBeanField(any bean, string field, any value) alue).
public void	setBeanFields(any bean, string fields, struct fieldCollection) alue).

Method Detail

createBean

```
public any createBean( string beanType, [struct initArgs] )
```

Creates a bean and calls its init() function.

Parameters:

string beanType
[struct initArgs]

Code:

```
<cffunction name="createBean" access="public" returntype="any" output="false"
  hint="Creates a bean and calls its init() function.">
  <cfargument name="beanType" type="string" required="true"
    hint="A fully qualified path to the bean CFC." />
  <cfargument name="initArgs" type="struct" required="false"
    hint="Optional. The set of arguments to pass to the init() function as an argument collection." />

  <cfset var bean = CreateObject('component', arguments.beanType) />

  <cfif IsDefined('arguments.initArgs') EQ true>
    <cfinvoke component="#bean#" method="init" argumentCollection="#arguments.initArgs#" />
  <cfelse>
    <cfset bean.init() />
  </cfif>

  <cfreturn bean />
</cffunction>
```

describeBean

```
public struct describeBean( any bean )
```

Returns a struct of bean properties/values based on getters.

Parameters:

any bean

Code:

```
<cffunction name="describeBean" access="public" returnType="struct" output="false"
  hint="Returns a struct of bean properties/values based on getters.">
  <cfargument name="bean" type="any" required="true" />

  <cfset var map = StructNew() />
  <cfset var meta = GetMetaData(arguments.bean) />
  <cfset var metaFunctions = meta.functions />
  <cfset var metaFunction = '' />
  <cfset var fieldName = '' />
  <cfset var fieldValue = '' />
  <cfset var i = 0 />

  <cfloop index="i" from="1" to="#ArrayLen(metaFunctions)#">
    <cfset metaFunction = metaFunctions[i] />
    <cfif metaFunction.name.toLowerCase().startsWith('get')
      AND metaFunction.access.equalsIgnoreCase("public")
      AND ArrayLen(metaFunction.parameters) EQ 0>
      <cfset fieldName = Right(metaFunction.name, Len(metaFunction.name)-3) />
      <cfset fieldName = LCase(Left(fieldName,1)) & Right(fieldName, Len(fieldName)-1) />
      <cfinvoke component="#arguments.bean#" method="#metaFunction.name#"
        returnVariable="fieldValue" />
      <cfset map[fieldName] = fieldValue />
    </cfif>
  </cfloop>

  <cfreturn map />
</cffunction>
```

getBeanField

public any getBeanField(any bean, string field)

Returns the value of a field in a bean using method call getBeanField().

Parameters:

any bean
string field

Code:

```
<cffunction name="getBeanField" access="public" returntype="any" output="false"
  hint="Returns the value of a field in a bean using method call getBeanField().">
  <cfargument name="bean" type="any" required="true" />
  <cfargument name="field" type="string" required="true" />

  <cfset var fieldValue = "" />
  <cfinvoke component="#arguments.bean#" method="get#arguments.field#"
    returnvariable="fieldValue" />
  <cfreturn fieldValue />
</cffunction>
```

init

```
public BeanUtil init( )
```

Used by the framework for initialization.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="BeanUtil" output="false"
  hint="Used by the framework for initialization.">
  <cfreturn this />
</cffunction>
```

setBeanField

```
public void setBeanField( any bean, string field, any value )
```

alue).

Parameters:

any bean
string field
any value

Code:

```
<cffunction name="setBeanField" access="public" returntype="void" output="false"
  hint="Sets the value of a field in a bean using method call setBeanField(beanField=value).">
  <cfargument name="bean" type="any" required="true" />
  <cfargument name="field" type="string" required="true" />
  <cfargument name="value" type="any" required="true" />

  <cfinvoke component="#arguments.bean#" method="set#arguments.field#">
    <cfinvokeargument name="#arguments.field#" value="#arguments.value#" />
  </cfinvoke>
</cffunction>
```

setBeanFields

public void setBeanFields(any bean, string fields, struct fieldCollection)

alue).

Parameters:

any bean
string fields
struct fieldCollection

Code:

```
<cffunction name="setBeanFields" access="public" returntype="void" output="false"
  hint="Sets the value of a fields in a bean using method calls setBeanField(beanField=value).">
  <cfargument name="bean" type="any" required="true"
    hint="The bean to populate." />
  <cfargument name="fields" type="string" required="true"
    hint="A comma-delimited list of fields to set in the bean." />
  <cfargument name="fieldCollection" type="struct" required="true"
    hint="A struct of field names mapped to values." />

  <cfset var field = 0 />

  <cfloop index="field" list="#arguments.fields#" delimiters=",">
    <cfif StructKeyExists(arguments.fieldCollection, field)>
```

```
                <cfset setBeanField(arguments.bean, field, arguments.fieldCollection[field]) />
            </cfif>
        </cfloop>
    </cffunction>
```

Exception

Package: MachII.util

Encapsulates exception information.

Method Summary

public Exception	init([string type="", [string message="", [string errorCode="", [string detail="", [string extendedInfo=""], [array tagContext="#ArrayNew(1)#"]])]) Used by the framework for initialization. Do not override.
public any	getCaughtException() Gets caughtException (cfcatch) that was collected at the point of the exception.
public string	getDetail()
public string	getErrorCode()
public string	getExtendedInfo()
public string	getMessage()
public array	getTagContext()
public string	getType()
public void	setCaughtException(any caughtException)
public void	setDetail([string detail])
public void	setErrorCode([string errorCode])
public void	setExtendedInfo([string extendedInfo])
public void	setMessage([string message])
public void	setTagContext([array extendedInfo])
public void	setType([string type])
public Exception	wrapException(any caughtException) Wraps and sets caughtException (cfcatch).

Method Summary

Method Detail**getCaughtException**

public any getCaughtException()

Gets caughtException (cfcatch) that was collected at the point of the exception.

Parameters:

Code:

```
<cffunction name="getCaughtException" access="public" returntype="any" output="false"
    hint="Gets caughtException (cfcatch) that was collected at the point of the exception.">
    <cfreturn variables.caughtException />
</cffunction>
```

getDetail

public string getDetail()

Parameters:

Code:

```
<cffunction name="getDetail" access="public" returntype="string" output="false">
    <cfreturn variables.detail />
</cffunction>
```

getErrorCode

public string getErrorCode()

Parameters:

Code:

```
<cffunction name="getErrorCode" access="public" returntype="string" output="false">
    <cfreturn variables.errorCode />
</cffunction>
```

getExtendedInfo

public string getExtendedInfo()

Parameters:

Code:

```
<cffunction name="getExtendedInfo" access="public" returntype="string" output="false">
    <cfreturn variables.extendedInfo />
</cffunction>
```

getMessage

public string getMessage()

Parameters:

Code:

```
<cffunction name="getMessage" access="public" returntype="string" output="false">
    <cfreturn variables.message />
</cffunction>
```

getTagContext

```
public array getTagContext( )
```

Parameters:

Code:

```
<cffunction name="getTagContext" access="public" returntype="array" output="false">
    <cfreturn variables.tagContext />
</cffunction>
```

getType

```
public string getType( )
```

Parameters:

Code:

```
<cffunction name="getType" access="public" returntype="string" output="false">
    <cfreturn variables.type />
</cffunction>
```

init

```
public Exception init( [string type=""], [string message=""], [string errorCode=""], [string detail=""], [string extendedInfo=""], [array tagContext="#ArrayNew(1)#"]
)
```

Used by the framework for initialization. Do not override.

Parameters:

```
[string type=""]  
[string message=""]  
[string errorCode=""]  
[string detail=""]  
[string extendedInfo=""]  
[array tagContext="#ArrayNew(1)#"]
```

Code:

```
<cffunction name="init" access="public" returnType="Exception" output="false"  
    hint="Used by the framework for initialization. Do not override.">  
    <cfargument name="type" type="string" required="false" default="" />  
    <cfargument name="message" type="string" required="false" default="" />  
    <cfargument name="errorCode" type="string" required="false" default="" />  
    <cfargument name="detail" type="string" required="false" default="" />  
    <cfargument name="extendedInfo" type="string" required="false" default="" />  
    <cfargument name="tagContext" type="array" required="false" default="#ArrayNew(1)#" />  
  
    <cfset setType(arguments.type) />  
    <cfset setMessage(arguments.message) />  
    <cfset setErrorCode(arguments.errorCode) />  
    <cfset setDetail(arguments.detail) />  
    <cfset setExtendedInfo(arguments.extendedInfo) />  
    <cfset setTagContext(arguments.tagContext) />  
  
    <cfreturn this />  
</cffunction>
```

setCaughtException

```
public void setCaughtException( any caughtException )
```

Parameters:

any caughtException

Code:

```
<cffunction name="setCaughtException" access="public" returnType="void" output="false">
```

```
<cfargument name="caughtException" type="any" required="true" />
<cfset variables.caughtException = arguments.caughtException />
</cffunction>
```

setDetail

public void setDetail([string detail])

Parameters:

[string detail]

Code:

```
<cffunction name="setDetail" access="public" returnType="void" output="false">
  <cfargument name="detail" type="string" required="false" />
  <cfset variables.detail = arguments.detail />
</cffunction>
```

setErrorCode

public void setErrorCode([string errorCode])

Parameters:

[string errorCode]

Code:

```
<cffunction name="setErrorCode" access="public" returnType="void" output="false">
  <cfargument name="errorCode" type="string" required="false" />
  <cfset variables.errorCode = arguments.errorCode />
</cffunction>
```

setExtendedInfo

public void setExtendedInfo([string extendedInfo])

Parameters:

[string extendedInfo]

Code:

```
<cffunction name="setExtendedInfo" access="public" returntype="void" output="false">
    <cfargument name="extendedInfo" type="string" required="false" />
    <cfset variables.extendedInfo = arguments.extendedInfo />
</cffunction>
```

setMessage

public void setMessage([string message])

Parameters:

[string message]

Code:

```
<cffunction name="setMessage" access="public" returntype="void" output="false">
    <cfargument name="message" type="string" required="false" />
    <cfset variables.message = arguments.message />
</cffunction>
```

setTagContext

public void setTagContext([array extendedInfo])

Parameters:

[array extendedInfo]

Code:

```
<cffunction name="setTagContext" access="public" returntype="void" output="false">
    <cfargument name="extendedInfo" type="array" required="false" />
    <cfset variables.tagContext = arguments.extendedInfo />
</cffunction>
```

setType

public void setType([string type])

Parameters:

[string type]

Code:

```
<cffunction name="setType" access="public" returntype="void" output="false">
    <cfargument name="type" type="string" required="false" />
    <cfset variables.type = arguments.type />
</cffunction>
```

wrapException

public Exception wrapException(any caughtException)

Wraps and sets caughtException (cfcatch).

Parameters:

any caughtException

Code:

```
<cffunction name="wrapException" access="public" returntype="Exception" output="false"
  hint="Wraps and sets caughtException (cfcatch).">
  <cfargument name="caughtException" type="any" required="true"
    hint="The cfcatch." />

  <cfset setType(arguments.caughtException.type) />
  <cfset setMessage(arguments.caughtException.message) />
  <cfset setErrorCode(arguments.caughtException.errorCode) />
  <cfset setDetail(arguments.caughtException.detail) />
  <cfset setExtendedInfo(arguments.caughtException.extendedInfo) />
  <cfset setTagContext(arguments.caughtException.TagContext) />
  <cfset setCaughtException(arguments.caughtException) />

  <cfreturn this />
</cffunction>
```

Queue

Package: MachII.util

A simple Queue component.

Method Summary

public Queue	init() Initializes the queue.
public void	clear() Clears the queue.
public any	get() Dequeues and returns the next item in the queue.
public numeric	getSize() Returns the size of the queue (number of elements).
public boolean	isEmpty() Returns whether or not the queue is empty.
public any	peek() Peeks the next item in the queue without removing it.
public void	put(any item) Queues the item.

Method Detail

clear

```
public void clear( )
```

Clears the queue.

Parameters:

Code:

```
<cffunction name="clear" access="public" returntype="void" output="false"
    hint="Clears the queue.">
    <cfset ArrayClear(variables.queueArray) />
</cffunction>
```

get

```
public any get( )
```

Dequeues and returns the next item in the queue.

Parameters:

Code:

```
<cffunction name="get" access="public" returntype="any" output="false"
    hint="Dequeues and returns the next item in the queue.">
    <cfset var nextItem = variables.queueArray[1] />
    <cfset ArrayDeleteAt(variables.queueArray, 1) />
    <cfreturn nextItem />
</cffunction>
```

getSize

```
public numeric getSize( )
```

Returns the size of the queue (number of elements).

Parameters:

Code:

```
<cffunction name="getSize" access="public" returntype="numeric" output="false"
    hint="Returns the size of the queue (number of elements).">
    <cfreturn ArrayLen(variables.queueArray) />
</cffunction>
```

init

```
public Queue init( )
```

Initializes the queue.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="Queue" output="false"
    hint="Initializes the queue.">
    <cfreturn this />
</cffunction>
```

isEmpty

```
public boolean isEmpty( )
```

Returns whether or not the queue is empty.

Parameters:

Code:

```
<cffunction name="isEmpty" access="public" returntype="boolean" output="false"
```

```
        hint="Returns whether or not the queue is empty.">
        <cfreturn getSize() EQ 0 />
    </cffunction>
```

peek

```
public any peek( )
```

Peeks the next item in the queue without removing it.

Parameters:

Code:

```
<cffunction name="peek" access="public" returntype="any" output="false"
    hint="Peeks the next item in the queue without removing it.">
    <cfreturn variables.queueArray[1] />
</cffunction>
```

put

```
public void put( any item )
```

Queues the item.

Parameters:

any item

Code:

```
<cffunction name="put" access="public" returntype="void" output="false"
    hint="Queues the item.">
    <cfargument name="item" type="any" required="true"
        hint="Item to append to queue." />
    <cfset ArrayAppend(variables.queueArray, arguments.item) />
</cffunction>
```

SizedQueue

Package: MachII.util

Inherits from: util.Queue

A specialization of Queue to limit size.

Method Summary

public SizedQueue	init([numeric maxSize="100"])
	Initializes the queue.
public numeric	getMaxSize()
	Returns the maximum size of the queue.
public boolean	isFull()
	Returns whether or not the queue is full.
public void	put(any item)
	Queues the item.
public void	setMaxSize(numeric maxSize)
	Sets the maximum size of the queue.

Methods inherited from util.Queue: peek , isEmpty , getSize , get , clear

Method Detail

getMaxSize

public numeric getMaxSize()

Returns the maximum size of the queue.

Parameters:

Code:

```
<cffunction name="getMaxSize" access="public" returnType="numeric" output="false"
    hint="Returns the maximum size of the queue.">
    <cfreturn variables.maxSize />
</cffunction>
```

init

public SizedQueue init([numeric maxSize="100"])

Initializes the queue.

Parameters:

[numeric maxSize="100"]

Code:

```
<cffunction name="init" access="public" returnType="SizedQueue" output="false"
    hint="Initializes the queue.">
    <cfargument name="maxSize" type="numeric" required="false" default="100" />

    <cfset super.init() />
    <cfset setMaxSize(arguments.maxSize) />

    <cfreturn this />
</cffunction>
```

isFull

```
public boolean isFull( )
```

Returns whether or not the queue is full.

Parameters:

Code:

```
<cffunction name="isFull" access="public" returntype="boolean" output="false"
    hint="Returns whether or not the queue is full.">
    <cfreturn getSize() EQ getMaxSize() />
</cffunction>
```

put

```
public void put( any item )
```

Queues the item.

Parameters:

any item

Code:

```
<cffunction name="put" access="public" returntype="void" output="false"
    hint="Queues the item.">
    <cfargument name="item" type="any" required="true" />

    <cfif NOT isFull()>
        <cfset super.put(arguments.item) />
    <cfelse>
        <cfthrow message="Max size of SizedQueue is #getMaxSize()# and has been exceeded." />
    </cfif>
</cffunction>
```

setMaxSize

public void setMaxSize(numeric maxSize)

Sets the maximum size of the queue.

Parameters:

numeric maxSize

Code:

```
<cffunction name="setMaxSize" access="public" returntype="void" output="false"
    hint="Sets the maximum size of the queue.">
    <cfargument name="maxSize" type="numeric" required="true" />
    <cfset variables.maxSize = arguments.maxSize />
</cffunction>
```

XmlValidationException

Package: MachII.util

Inherits from: util.Exception

Encapsulates XML validation exception information.

Method Summary

public string	getDtdPath()
public array	getErrors()
public array	getFatalErrors()
public string	getFormattedMessage() Gets a message from the errors/warnings for display.
public array	getWarnings()
public string	getXmlPath()
public void	setDtdPath([string dtdPath])
public void	setErrors([array errors])
public void	setFatalErrors([array fatalErrors])
public void	setWarnings([array warnings])
public void	setXmlPath([string xmlPath])
public XmlValidationException	wrapValidationResult(struct validationResult, [string xmlPath=""], [string dtdPath=""]) Wraps the result of a failed XML validation.

Methods inherited from util.Exception: `init` , `wrapException` , `setDetail` , `getDetail` , `setTagContext` , `getCaughtException` , `getType` , `getExtendedInfo` , `getErrorCode` , `setErrorCode` , `setType` , `setCaughtException` , `setMessage` , `getMessage` , `getTagContext` , `setExtendedInfo`

Method Detail

getDtdPath

public string getDtdPath()

Parameters:

Code:

```
<cffunction name="getDtdPath" access="public" returntype="string" output="false">
    <cfreturn variables.dtdPath />
</cffunction>
```

getErrors

public array getErrors()

Parameters:

Code:

```
<cffunction name="getErrors" access="public" returntype="array" output="false">
    <cfreturn variables.errors />
</cffunction>
```

getFatalErrors

public array getFatalErrors()

Parameters:

Code:

```
<cffunction name="getFatalErrors" access="public" returntype="array" output="false">
    <cfreturn variables.fatalErrors />
</cffunction>
```

getFormattedMessage

public string getFormattedMessage()

Gets a message from the errors/warnings for display.

Parameters:

Code:

```
<cffunction name="getFormattedMessage" access="public" returntype="string" output="false"
    hint="Gets a message from the errors/warnings for display.">

    <cfset var rawMessage = "" />
    <cfset var formattedMessage = "" />

    <cfif ArrayLen(variables.fatalErrors) GT 0>
        <cfset rawMessage = variables.fatalErrors[1] />
    <cfelseif ArrayLen(variables.errors) GT 0>
        <cfset rawMessage = variables.errors[1] />
    <cfelseif ArrayLen(variables.warnings) GT 0>
        <cfset rawMessage = variables.warnings[1] />
    <cfelse>
        <cfthrow type="MachII.framework.NoMessagesDefined"
            message="There are no XML validation error messages defined. Cannot display a formatted message."
        />
    </cfif>

    <cfset formattedMessage = "Error validating XML file: " />
    <cfif getXmlPath() NEQ ''>
        <cfset formattedMessage = formattedMessage & getXmlPath() & ": " />
    </cfif>
    <cfset formattedMessage = formattedMessage & "Line " & ListGetAt(rawMessage,2,':') & ", " />
    <cfset formattedMessage = formattedMessage & "Column " & ListGetAt(rawMessage,3,':') & ": " />
    <cfset formattedMessage = formattedMessage & ListGetAt(rawMessage,4,':') />
```

```
<cfif ListLen(rawMessage, ":") GTE 5>
    <cfset formattedMessage = formattedMessage & " - " & ListGetAt(rawMessage,5,':') />
</cfif>

<cfreturn formattedMessage />
</cffunction>
```

getWarnings

public array getWarnings()

Parameters:

Code:

```
<cffunction name="getWarnings" access="public" returntype="array" output="false">
    <cfreturn variables.warnings />
</cffunction>
```

getXmlPath

public string getXmlPath()

Parameters:

Code:

```
<cffunction name="getXmlPath" access="public" returntype="string" output="false">
    <cfreturn variables.xmlPath />
</cffunction>
```

setDtdPath

public void setDtdPath([string dtdPath])

Parameters:

[string dtdPath]

Code:

```
<cffunction name="setDtdPath" access="public" returntype="void" output="false">
    <cfargument name="dtdPath" type="string" required="false" />
    <cfset variables.dtdPath = arguments.dtdPath />
</cffunction>
```

setErrors

public void setErrors([array errors])

Parameters:

[array errors]

Code:

```
<cffunction name="setErrors" access="public" returntype="void" output="false">
    <cfargument name="errors" type="array" required="false" />
    <cfset variables.errors = arguments.errors />
</cffunction>
```

setFatalErrors

public void setFatalErrors([array fatalErrors])

Parameters:

[array fatalErrors]

Code:

```
<cffunction name="setFatalErrors" access="public" returntype="void" output="false">
  <cfargument name="fatalErrors" type="array" required="false" />
  <cfset variables.fatalErrors = arguments.fatalErrors />
</cffunction>
```

setWarnings

public void setWarnings([array warnings])

Parameters:

[array warnings]

Code:

```
<cffunction name="setWarnings" access="public" returntype="void" output="false">
  <cfargument name="warnings" type="array" required="false" />
  <cfset variables.warnings = arguments.warnings />
</cffunction>
```

setXmlPath

public void setXmlPath([string xmlPath])

Parameters:

[string xmlPath]

Code:

```
<cffunction name="setXmlPath" access="public" returntype="void" output="false">
```

```
<cfargument name="xmlPath" type="string" required="false" />
<cfset variables.xmlPath = arguments.xmlPath />
</cffunction>
```

wrapValidationResult

```
public XmlValidationException wrapValidationResult( struct validationResult, [string xmlPath=""], [string dtdPath=""] )
```

Wraps the result of a failed XML validation.

Parameters:

```
struct validationResult
[string xmlPath=""]
[string dtdPath=""]
```

Code:

```
<cffunction name="wrapValidationResult" access="public" returntype="XmlValidationException" output="false"
  hint="Wraps the result of a failed XML validation.">
  <cfargument name="validationResult" type="struct" required="true"
    hint="A struct in the format returned by XmlValidate()." />
  <cfargument name="xmlPath" type="string" required="false" default=""
    hint="The full path the XML file that was validated." />
  <cfargument name="dtdPath" type="string" required="false" default=""
    hint="The full path the DTD file used for validation." />

  <cfset setFatalErrors(arguments.validationResult.fatalErrors) />
  <cfset setErrors(arguments.validationResult.errors) />
  <cfset setWarnings(arguments.validationResult.warnings) />
  <cfset setXmlPath(arguments.xmlPath) />
  <cfset setDtdPath(arguments.dtdPath) />

  <cfreturn this />
</cffunction>
```