

Mach-II 1.5.0 - Stable

Mach-II 1.5.0 - Stable

Table of Contents

.....	1
mach-ii	2
filters	7
EventArgsFilter	8
EventBeanFilter	11
PermissionsFilter	16
RequiredFieldsFilter	22
framework	26
AppFactory	27
AppLoader	39
AppManager	55
BaseComponent	70
Command	84
CommandLoaderBase	88
Event	100
EventContext	112
EventFilter	132
EventFilterManager	135
EventHandler	145
EventManager	149
Listener	162
ListenerInvoker	165
ListenerManager	167
Module	177
ModuleManager	187
Plugin	199
PluginManager	206
Property	225
PropertyManager	227
RequestHandler	241
RequestManager	264
SubroutineHandler	288
SubroutineManager	291
ViewContext	301

ViewManager	310
framework.commands	317
AnnounceCommand	318
EventArgCommand	323
EventBeanCommand	330
EventMappingCommand	338
ExecuteCommand	343
FilterCommand	346
NotifyCommand	350
RedirectCommand	357
ViewPageCommand	371
framework.invokers	378
CFCInvoker_Event	379
CFCInvoker_EventArgs	382
EventArgsInvoker	385
EventInvoker	388
plugins	391
SimplePlugin	392
TracePlugin	397
util	428
BeanUtil	429
Exception	435
Queue	444
SizedQueue	449
Utils	453
XmlValidationException	458

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(arguments.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 IsDefined("application.#MACHII_APP_KEY#.appLoader") OR NOT IsObject(application[MACHII_APP_KEY].appLoader)>
    <cflock name="application_#MACHII_APP_KEY#_reload" type="exclusive" timeout="120">
      <cfif NOT IsDefined("application.#MACHII_APP_KEY#.appLoader") OR NOT IsObject(application[MACHII_APP_KEY].appLoader)>
        <cfset loadFramework() />
      </cfif>
    </cflock>
  </cfif>

  <cfif MACHII_CONFIG_MODE EQ -1>

  <cfelseif 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) />
    </cflock>
  <cfelseif MACHII_CONFIG_MODE EQ 0 AND application[MACHII_APP_KEY].appLoader.shouldReloadBaseConfig(>
    <cflock name="application_#MACHII_APP_KEY#_reload" type="exclusive" timeout="120">
      <cfset application[MACHII_APP_KEY].appLoader.reloadConfig(MACHII_VALIDATE_XML) />
    </cflock>
  <cfelseif MACHII_CONFIG_MODE EQ 0 AND application[MACHII_APP_KEY].appLoader.shouldReloadModuleConfig(>
    <cflock name="application_#MACHII_APP_KEY#_reload" type="exclusive" timeout="120">
      <cfset application[MACHII_APP_KEY].appLoader.reloadModuleConfig(MACHII_VALIDATE_XML) />
    </cflock>
  </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: setParameters , hasParameter , buildUrlToModule , isParameterDefined , buildUrl , announceEventInModule , setAppManager , getParameter , getProperty , getParameters , setProperty , getPropertyManager , bindValue , getAppManager , setParameter , announceEvent

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>
</cffunction>
```

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

EventBeanFilter

Package: MachII.filters

Inherits from: framework.BaseComponent < framework.EventFilter

DEPRECATED. A robust EventFilter for creating and populating beans in events.

Method Summary

public void	configure() DEPRECATED. Configures the filter.
public boolean	filterEvent(Event event, EventContext eventContext, [struct paramArgs="#StructNew()#"]) DEPRECATED. 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: setParameters , hasParameter , buildUriToModule , isParameterDefined , buildUri , announceEventInModule , setAppManager , getParameter , getProperty , getParameters , setProperty , getPropertyManager , bindValue , get-AppManager , setParameter , announceEvent

Method Detail

configure

```
public void configure( )
```

DEPRECATED. Configures the filter.

Parameters:

Code:

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

filterEvent

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

DEPRECATED. Runs the filter event.

Parameters:

Event event

EventContext eventContext

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

Code:

```
<cffunction name="filterEvent" access="public" returntype="boolean"
    hint="DEPRECATED. 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: setParameters , hasParameter , buildUrlToModule , isParameterDefined , buildUrl , announceEventInModule , setAppManager , getParameter , getProperty , getParameters , setProperty , getPropertyManager , bindValue , getAppManager , setParameter , announceEvent

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 />
    <cfelse>
        <cfreturn '' />
    </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 = "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>
    <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: setParameters , hasParameter , buildUrlToModule , isParameterDefined , buildUrl , announceEventInModule , setAppManager , getParameter , getProperty , getParameters , setProperty , getPropertyManager , bindValue , get-AppManager , setParameter , announceEvent

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.
private void	appendConfigFilePath(string configFilePath) Appends a config file path to be used by AppLoader to check if reloading is necessary when in dynamic mode.
private void	checkIfAlreadyIncluded(struct alreadyLoaded, string includeFilePath) Checks if the include has already been processed.
public AppManager	createAppManager(string configXmlPath, string configDtdPath, string appkey, [boolean validateXml="false"], [any parentAppManager="", [any overrideXml=""], [string moduleName=""]]) Creates the AppManager and reads (and optionally validates) the XML configuration file.
public array	getConfigFilePaths() Returns an array of config file paths.
private array	loadIncludes(array configFiles, string configXML, boolean validateXml, string configDtdPath, string parentConfigFilePathDirectory, [boolean overrideIncludeType="false"], [struct alreadyLoaded="#StructNew(##)"]) Loads files to be included into the config xml array.
private void	resetConfigFilePaths() Resets the config file paths to a zero element array.
private void	validateConfigXml(boolean validateXml, any configXml, string configXmlPath, string configDtdPath)

Method Summary

Validates an xml file.

Method Detail**appendConfigFilePath**

```
private void appendConfigFilePath( string configFilePath )
```

Appends a config file path to be used by AppLoader to check if reloading is necessary when in dynamic mode.

Parameters:

string configFilePath

Code:

```
<cffunction name="appendConfigFilePath" access="private" returntype="void" output="false"
    hint="Appends a config file path to be used by AppLoader to check if reloading is necessary when in dynamic mode"
    <cfargument name="configFilePath" type="string" required="true" />
    <cfset ArrayAppend(variables.configFilePaths, arguments.configFilePath) />
</cffunction>
```

checkIfAlreadyIncluded

```
private void checkIfAlreadyIncluded( struct alreadyLoaded, string includeFilePath )
```

Checks if the include has already been processed.

Parameters:

struct alreadyLoaded
string includeFilePath

Code:

```

<cffunction name="checkIfAlreadyIncluded" access="private" returntype="void" output="false"
  hint="Checks if the include has already been processed.">
  <cfargument name="alreadyLoaded" type="struct" required="true" />
  <cfargument name="includeFilePath" type="string" required="true" />

  <cfset var includeFilePathHash = Hash(arguments.includeFilePath) />

  <cfif StructKeyExists(arguments.alreadyLoaded, includeFilePathHash)>
    <cfthrow type="MachII.framework.IncludeAlreadyDefined"
      message="An include located at '#arguments.includeFilePath#' has already been included. You cannot" />
  </cfif>
  <cfset arguments.alreadyLoaded[includeFilePathHash] = true />
</cffunction>

```

createAppManager

```
public AppManager createAppManager( string configXmlPath, string configDtdPath, string appkey, [boolean validateXml="false"], [any parentAppManager=""], [any overrideXml=""], [string moduleName=""] )
```

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

Parameters:

```

string configXmlPath
string configDtdPath
string appkey
[boolean validateXml="false"]
[any parentAppManager=""]
[any overrideXml=""]
[string moduleName=""]

```

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="appkey" type="string" required="true"
    hint="Unqiue key for this application.">
<cfargument name="validateXml" type="boolean" required="false" default="false"
    hint="Should the XML be validated before parsing." />
<cfargument name="parentAppManager" type="any" required="false" default=""
    hint="Optional argument for a parent app manager. Defaults to empty string." />
<cfargument name="overrideXml" type="any" required="false" default=""
    hint="Optional argument for override Xml for a module. Defaults to empty string." />
<cfargument name="moduleName" type="string" required="false" default=""
    hint="Optional argument for the name of a module. Defaults to empty string." />

<cfset var appManager = "" />
<cfset var utils = "" />
<cfset var propertyManager = "" />
<cfset var parentPropertyManager = "" />
<cfset var requestManager = "" />
<cfset var listenerManager = "" />
<cfset var parentListenerManager = "" />
<cfset var filterManager = "" />
<cfset var parentFilterManager = "" />
<cfset var subroutineManager = "" />
<cfset var parentSubroutineManager = "" />
<cfset var eventManager = "" />
<cfset var viewManager = "" />
<cfset var parentViewManager = "" />
<cfset var pluginManager = "" />
<cfset var parentPluginManager = "" />
<cfset var parentEventManager = "" />
<cfset var moduleManager = "" />
<cfset var configXml = "" />
<cfset var configXmlFile = "" />
<cfset var configXmls = ArrayNew(1) />
<cfset var overrideIncludeNodes = "" />
<cfset var overrideIncludes = ArrayNew(1) />
<cfset var temp = StructNew() />
<cfset var i = "" />

<cfset resetConfigFilePaths() />
```

```
<cfif IsObject(arguments.parentAppManager)>
    <cfset utils = arguments.parentAppManager.getUtils() />
<cfelse>
    <cfset utils = CreateObject("component", "MachII.util.Utils").init() />
</cfif>

<cfset variables.utils = utils />

<cftry>
    <cffile
        action="READ"
        file="#arguments.configXmlPath#"
        variable="configXmlFile" />
    <cfcatch type="any">
        <cfthrow type="MachII.framework.CannotFindBaseConfigFile"
            message="Unable to find the base config file for module '#arguments.moduleName#'."
            detail="configPath=#arguments.configXmlPath#" />
    </cfcatch>
</cftry>

<cfset appendConfigFilePath(arguments.configXmlPath) />

<cfset temp.configXml = XmlParse(configXmlFile) />
<cfset temp.override = false />

<cfset validateConfigXml(arguments.validateXml, temp.configXml, arguments.configXmlPath, arguments.configDtdPath) />

<cfset ArrayAppend(configXmIs, temp) />

<cfset configXmIs = loadIncludes(configXmIs, temp.configXml, arguments.validateXml, arguments.configDtdPath, GetT

<cfif Len(arguments.overrideXml)>
    <cfset configXmIs = loadIncludes(configXmIs, arguments.overrideXml, arguments.validateXml, arguments.con
</cfif>
```

```
<cfset appManager = CreateObject("component", "MachII.framework.AppManager").init() />
<cfset appManager.setAppKey(arguments.appkey) />
<cfif Len(arguments.moduleName)>
    <cfset appManager.setModuleName(arguments.moduleName) />
</cfif>

<cfif IsObject(arguments.parentAppManager)>
    <cfset appManager.setParent(arguments.parentAppManager) />
    <cfset parentPropertyManager = appManager.getParent().getPropertyManager() />
    <cfset parentListenerManager = appManager.getParent().getListenerManager() />
    <cfset parentFilterManager = appManager.getParent().getFilterManager() />
    <cfset parentSubroutineManager = appManager.getParent().getSubroutineManager() />
    <cfset parentEventManager = appManager.getParent().getEventManager() />
    <cfset parentViewManager = appManager.getParent().getViewManager() />
    <cfset parentPluginManager = appManager.getParent().getPluginManager() />
</cfif>

<cfset appManager.setUtils(utils) />

<cfset propertyManager = CreateObject("component", "MachII.framework.PropertyManager").init(appManager, parentPr
<cfloop from="1" to="#ArrayLen(configXmIs)#" index="i">
    <cfset propertyManager.loadXml(configXmIs[i].configXml, configXmIs[i].override) />
</cfloop>
<cfif Len(arguments.overrideXml)>
    <cfset propertyManager.loadXml(arguments.overrideXml, true) />
</cfif>
<cfset appManager.setPropertyManager(propertyManager) />

<cfif IsObject(arguments.parentAppManager)>
    <cfset requestManager = arguments.parentAppManager.getRequestManager() />
<cfelse>
    <cfset requestManager = CreateObject("component", "MachII.framework.RequestManager").init(appManager) />
</cfif>
<cfif isObject(appManager.getParent())>
    <cfset appManager.setRequestManager(appManager.getParent().getRequestManager()) />
<cfelse>
    <cfset appManager.setRequestManager(requestManager) />
</cfif>
```

```
<cfset listenerManager = CreateObject("component", "MachII.framework.ListenerManager").init(appManager, parentLi
<cfloop from="1" to="#ArrayLen(configXmles)#" index="i">
  <cfset listenerManager.loadXml(configXmles[i].configXml, configXmles[i].override) />
</cfloop>
<cfif Len(arguments.overrideXml)>
  <cfset listenerManager.loadXml(arguments.overrideXml, true) />
</cfif>
<cfset appManager.setListenerManager(listenerManager) />

<cfset filterManager = CreateObject("component", "MachII.framework.EventFilterManager").init(appManager, parentF
<cfloop from="1" to="#ArrayLen(configXmles)#" index="i">
  <cfset filterManager.loadXml(configXmles[i].configXml, configXmles[i].override) />
</cfloop>
<cfif Len(arguments.overrideXml)>
  <cfset filterManager.loadXml(arguments.overrideXml, true) />
</cfif>
<cfset appManager.setFilterManager(filterManager) />

<cfset subroutineManager = CreateObject("component", "MachII.framework.SubroutineManager").init(appManager, pare
<cfloop from="1" to="#ArrayLen(configXmles)#" index="i">
  <cfset subroutineManager.loadXml(configXmles[i].configXml, configXmles[i].override) />
</cfloop>
<cfif Len(arguments.overrideXml)>
  <cfset subroutineManager.loadXml(arguments.overrideXml, true) />
</cfif>
<cfset appManager.setSubroutineManager(subroutineManager) />

<cfset eventManager = CreateObject("component", "MachII.framework.EventManager").init(appManager, parentEventMan
<cfloop from="1" to="#ArrayLen(configXmles)#" index="i">
  <cfset eventManager.loadXml(configXmles[i].configXml, configXmles[i].override) />
</cfloop>
<cfif Len(arguments.overrideXml)>
  <cfset eventManager.loadXml(arguments.overrideXml, true) />
</cfif>
<cfset appManager.setEventManager(eventManager) />

<cfset viewManager = CreateObject("component", "MachII.framework.ViewManager").init(appManager, parentViewManager
<cfloop from="1" to="#ArrayLen(configXmles)#" index="i">
  <cfset viewManager.loadXml(configXmles[i].configXml, configXmles[i].override) />
</cfloop>
<cfif Len(arguments.overrideXml)>
  <cfset viewManager.loadXml(arguments.overrideXml, true) />
</cfif>
<cfset appManager.setViewManager(viewManager) />
```

```

    <cfset pluginManager = CreateObject("component", "MachII.framework.PluginManager").init(appManager, parentPluginManager) />
    <cfloop from="1" to="#ArrayLen(configXmles)#" index="i">
        <cfset pluginManager.loadXml(configXmles[i].configXml, configXmles[i].override) />
    </cfloop>
    <cfif Len(arguments.overrideXml)>
        <cfset pluginManager.loadXml(arguments.overrideXml, true) />
    </cfif>
    <cfset appManager.setPluginManager(pluginManager) />

    <cfif NOT IsObject(arguments.parentAppManager)>
        <cfset moduleManager = CreateObject("component", "MachII.framework.ModuleManager").init(appManager, GetDefaultModuleManager()) />
        <cfloop from="1" to="#ArrayLen(configXmles)#" index="i">
            <cfset moduleManager.loadXml(configXmles[i].configXml, configXmles[i].override) />
        </cfloop>
    <cfelse>
        <cfset moduleManager = arguments.parentAppManager.getModuleManager() />
    </cfif>
    <cfset appManager.setModuleManager(moduleManager) />

    <cfset appManager.configure() />

    <cfreturn appManager />
</cffunction>

```

getConfigFilePaths

public array getConfigFilePaths()

Returns an array of config file paths.

Parameters:

Code:

```

<cffunction name="getConfigFilePaths" access="public" returntype="array" output="false"
    hint="Returns an array of config file paths.">
    <cfreturn variables.configFilePaths />
</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>
```

loadIncludes

```
private array loadIncludes( array configFiles, string configXML, boolean validateXml, string configDtdPath, string parentConfigFilePathDirectory, [boolean
overrideIncludeType="false"], [struct alreadyLoaded="#StructNew()#"] )
```

Loads files to be included into the config xml array.

Parameters:

```
array configFiles
string configXML
boolean validateXml
string configDtdPath
string parentConfigFilePathDirectory
[boolean overrideIncludeType="false"]
[struct alreadyLoaded="#StructNew()#"]
```

Code:

```
<cffunction name="loadIncludes" access="private" returntype="array" output="false"
    hint="Loads files to be included into the config xml array.">
    <cfargument name="configFiles" type="array" required="true" />
```

```
<cfargument name="configXML" type="string" required="true" />
<cfargument name="validateXml" type="boolean" required="true" />
<cfargument name="configDtdPath" type="string" required="true" />
<cfargument name="parentConfigFilePathDirectory" type="string" required="true" />
<cfargument name="overrideIncludeType" type="boolean" required="false" default="false" />
<cfargument name="alreadyLoaded" type="struct" required="false" default="#StructNew()#" />

<cfset var includeNodes = "" />
<cfset var temp = StructNew() />
<cfset var includeFilePath = "" />
<cfset var includeXmlFile = "" />
<cfset var i = 0 />

<cfset includeNodes = XmlSearch(arguments.configXML, "../includes/include") />
<cfloop from="1" to="#ArrayLen(includeNodes)#" index="i">

    <cfset temp = StructNew() />
    <cfset includeFilePath = includeNodes[i].xmlAttributes["file"] />

    <cfif Left(includeFilePath, 1) IS ".">
        <cfset includeFilePath = variables.utils.expandRelativePath(arguments.parentConfigFilePathDirectory, includeFilePath) />
    <cfelse>
        <cfset includeFilePath = ExpandPath(includeFilePath) />
    </cfif>

    <cfif NOT arguments.overrideIncludeType>
        <cfif StructKeyExists(includeNodes[i].xmlAttributes, "override")>
            <cfset temp.override = includeNodes[i].xmlAttributes["override"] />
        <cfelse>
            <cfset temp.override = false />
        </cfif>
    <cfelse>
        <cfset temp.override = true />
    </cfif>

    <cfset checkIfAlreadyIncluded(arguments.alreadyLoaded, includeFilePath) />

    <cftry>
        <cffile
            action="read"
            file="#includeFilePath#"
            </cffile>
    </cftry>
</cfloop>
```

```
        variable="includeXMLFile" />
    <cfcatch type="any">
        <cfthrow type="MachII.framework.CannotFindIncludeConfigFile"
            message="Unable to find the include config file. This could be due to an incorrec
            detail="includePath=#includeFilePath#" />
    </cfcatch>
</cftry>

<cfset temp.configXml = XmlParse(includeXmlFile) />

<cfset validateConfigXml(arguments.validateXml, temp.configXml, includeFilePath, arguments.configDtdPath

<cfset appendConfigFilePath(includeFilePath) />

<cfset ArrayAppend(arguments.configFiles, temp) />

    <cfset arguments.configFiles = loadIncludes(arguments.configFiles, temp.configXml, arguments.validateXml
</cfloop>

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

resetConfigFilePaths

```
private void resetConfigFilePaths( )
```

Resets the config file paths to a zero element array.

Parameters:

Code:

```
<cffunction name="resetConfigFilePaths" access="private" returntype="void" output="false"
    hint="Resets the config file paths to a zero element array.">
    <cfset ArrayClear(variables.configFilePaths) />
</cffunction>
```

validateConfigXml

```
private void validateConfigXml( boolean validateXml, any configXml, string configXmlPath, string configDtdPath )
```

Validates an xml file.

Parameters:

boolean validateXml
any configXml
string configXmlPath
string configDtdPath

Code:

```
<cffunction name="validateConfigXml" access="private" returntype="void" output="false"
  hint="Validates an xml file.">
  <cfargument name="validateXml" type="boolean" required="true" />
  <cfargument name="configXml" type="any" required="true" />
  <cfargument name="configXmlPath" type="string" required="true" />
  <cfargument name="configDtdPath" type="string" required="true" />

  <cfset var validationResult = "" />
  <cfset var validationException = "" />

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

AppLoader

Package: MachII.framework

Responsible for controlling the loading/reloading of the AppManager.

Method Summary

public AppLoader	init(string configPath, string dtdPath, string appkey, [boolean validateXml="false"], [any parentAppManager=""], [any overrideXml=""], [string moduleName=""])	Used by the framework for initialization. Do not override.
public AppFactory	getAppFactory()	
public any	getAppKey()	
public AppManager	getAppManager()	
private string	getConfigFileReloadHash()	Get the current reload hash of the master config file and any include files which is based on dateLastModified and size.
public string	getConfigPath()	
public string	getDtdPath()	
public any	getLastReloadDatetime()	Gets the last reload datetime for this module or base application.
public string	getLastReloadHash()	
public any	getModuleName()	Gets the module name
public any	getOverrideXml()	Gets the override Xml for this module.
public boolean	getValidateXML()	

Method Summary

public void	reloadConfig([boolean validateXml="false"], [any parentAppManager=""]) Reloads the config file and sets the last reload hash.
public void	reloadModuleConfig([boolean validateXml="false"], [any parentAppManager=""]) Reloads the config file and sets the last reload hash.
public void	setAppFactory(AppFactory appFactory)
public void	setAppKey(string appkey)
public void	setAppManager(AppManager appManager)
public void	setConfigPath(string configPath)
public void	setDtdPath(string dtdPath)
public void	setLastReloadHash(string lastReloadHash)
public void	setModuleName(string moduleName) Sets the name of the module
public void	setOverrideXml(any overrideXml) Sets the override Xml for this module.
public void	setValidateXML(string validateXML)
public boolean	shouldReloadBaseConfig() Determines if any of the base configuration files should be reloaded.
public boolean	shouldReloadConfig() Determines if the configuration file should be reloaded.
public boolean	shouldReloadModuleConfig() Determines if any of the module configuration files should be reloaded.
private void	updateLastReloadDatetime() Updates the last reload datetime for this module or base application.

Method Detail

getAppFactory

```
public AppFactory getAppFactory( )
```

Parameters:

Code:

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

getAppKey

```
public any getAppKey( )
```

Parameters:

Code:

```
<cffunction name="getAppKey" access="public" type="string" output="false">
    <cfreturn variables.appkey />
</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Parameters:

Code:

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

getConfigFileReloadHash

```
private string getConfigFileReloadHash( )
```

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

Parameters:

Code:

```
<cffunction name="getConfigFileReloadHash" access="private" returntype="string" output="false"
    hint="Get the current reload hash of the master config file and any include files which is based on dateLastModified and size" />
    <cfset var configFilePaths = getAppFactory().getConfigFilePaths() />
    <cfset var directoryResults = "" />
    <cfset var hashableString = "" />
    <cfset var i = "" />

    <cfloop from="1" to="#ArrayLen(configFilePaths)#" index="i">
        <cfdirectory action="LIST" directory="#GetDirectoryFromPath(configFilePaths[i])#"
            name="directoryResults" filter="#GetFileFromPath(configFilePaths[i])#" />
        <cfset hashableString = hashableString & directoryResults.dateLastModified & directoryResults.size />
    </cfloop>

    <cfreturn Hash(hashableString) />
</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>
```

getLastReloadDatetime

```
public any getLastReloadDatetime( )
```

Gets the last reload datetime for this module or base application.

Parameters:

Code:

```
<cffunction name="getLastReloadDatetime" access="public" type="date" output="false"
    hint="Gets the last reload datetime for this module or base application.">
    <cfreturn variables.lastReloadDatetime />
</cffunction>
```

getLastReloadHash

public string getLastReloadHash()

Parameters:

Code:

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

getModuleName

public any getModuleName()

Gets the module name

Parameters:

Code:

```
<cffunction name="getModuleName" access="public" type="string" output="false"
    hint="Gets the module name">
    <cfreturn variables.moduleName />
</cffunction>
```

getOverrideXml

public any getOverrideXml()

Gets the override Xml for this module.

Parameters:

Code:

```
<cffunction name="getOverrideXml" access="public" type="any" output="false"
```

```
        hint="Gets the override Xml for this module.">
        <cfreturn variables.overrideXml />
    </cffunction>
```

getValidateXML

```
public boolean getValidateXML( )
```

Parameters:

Code:

```
<cffunction name="getValidateXML" access="public" returntype="boolean" output="false">
    <cfreturn variables.validateXML />
</cffunction>
```

init

```
public AppLoader init( string configPath, string dtdPath, string appkey, [boolean validateXml="false"], [any parentAppManager=""], [any overrideXml=""],
[string moduleName=""] )
```

Used by the framework for initialization. Do not override.

Parameters:

```
string configPath
string dtdPath
string appkey
[boolean validateXml="false"]
[any parentAppManager=""]
[any overrideXml=""]
[string moduleName=""]
```

Code:

```
<cffunction name="init" access="public" returntype="MachII.framework.AppLoader" output="false"
  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="appkey" type="string" required="true" hint="Unqie key for this application.">
  <cfargument name="validateXml" type="boolean" required="false" default="false"
    hint="Should the XML be validated before parsing." />
  <cfargument name="parentAppManager" type="any" required="false" default=""
    hint="Optional argument for a parent app manager. If there isn't one default to empty string." />
  <cfargument name="overrideXml" type="any" required="false" default=""
    hint="Optional argument for override Xml for a module. Default to empty string." />
  <cfargument name="moduleName" type="string" required="false" default=""
    hint="Optional argument for the name of a module. Defaults to empty string." />

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

  <cfset setConfigPath(arguments.configPath) />
  <cfset setDtdPath(arguments.dtdPath) />
  <cfset setValidateXml(arguments.validateXml) />
  <cfset setOverrideXml(arguments.overrideXml) />
  <cfset setModuleName(arguments.moduleName) />
  <cfset setAppKey(arguments.appkey) />

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

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

reloadConfig

```
public void reloadConfig( [boolean validateXml="false"], [any parentAppManager=""] )
```

Reloads the config file and sets the last reload hash.

Parameters:

```
[boolean validateXml="false"]  
[any parentAppManager=""]
```

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="parentAppManager" type="any" required="false" default=""
    hint="Optional argument for a parent app manager. If there isn't one default to empty string." />

  <cfset updateLastReloadDatetime() />
  <cfset setAppManager(getAppFactory().createAppManager(getConfigPath(), getDtdPath(),
    getAppKey(), getValidateXml(), arguments.parentAppManager, getOverrideXml(), getModuleName())) />
  <cfset getAppManager().setAppLoader(this) />
  <cfset setLastReloadHash(getConfigFileReloadHash()) />
</cffunction>

```

reloadModuleConfig

```
public void reloadModuleConfig( [boolean validateXml="false"], [any parentAppManager=""] )
```

Reloads the config file and sets the last reload hash.

Parameters:

```
[boolean validateXml="false"]
[any parentAppManager=""]
```

Code:

```

<cffunction name="reloadModuleConfig" 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="parentAppManager" type="any" required="false" default=""
    hint="Optional argument for a parent app manager. If there isn't one default to empty string." />

  <cfset var modules = getAppManager().getModuleManager().getModules() />
  <cfset var module = 0 />

```

```
<cfif NOT IsObject(getAppManager().getParent())>
  <cfloop collection="#modules#" item="module">
    <cfif modules[module].shouldReloadConfig()>
      <cfset modules[module].shouldReloadConfig() />
    </cfif>
  </cfloop>
</cfif>
</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>
```

setAppKey

```
public void setAppKey( string appkey )
```

Parameters:

string appkey

Code:

```
<cffunction name="setAppKey" access="public" returnType="void" output="false">
```

```
<cfargument name="appkey" type="string" required="true" />
<cfset variables.appkey = arguments.appkey />
</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>
```

setModuleName

```
public void setModuleName( string moduleName )
```

Sets the name of the module

Parameters:

string moduleName

Code:

```
<cffunction name="setModuleName" access="public" returntype="void" output="false"
    hint="Sets the name of the module">
    <cfargument name="moduleName" type="string" required="true" />
    <cfset variables.moduleName = arguments.moduleName />
</cffunction>
```

setOverrideXml

public void setOverrideXml(any overrideXml)

Sets the override Xml for this module.

Parameters:

any overrideXml

Code:

```
<cffunction name="setOverrideXml" access="public" returntype="void" output="false"
    hint="Sets the override Xml for this module.">
    <cfargument name="overrideXml" type="any" required="true" />
    <cfset variables.overrideXml = arguments.overrideXml />
</cffunction>
```

setValidateXML

public void setValidateXML(string validateXML)

Parameters:

string validateXML

Code:

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

shouldReloadBaseConfig

public boolean shouldReloadBaseConfig()

Determines if any of the base configuration files should be reloaded.

Parameters:

Code:

```
<cffunction name="shouldReloadBaseConfig" access="public" returntype="boolean" output="false"
    hint="Determines if any of the base configuration files should be reloaded.">
    <cfset var result = false />
    <cfif CompareNoCase(getLastReloadHash(), getConfigFileReloadHash()) NEQ 0>
        <cfset result = true />
    </cfif>
    <cfreturn result />
</cffunction>
```

shouldReloadConfig

public boolean shouldReloadConfig()

Determines if the configuration file should be reloaded.

Parameters:

Code:

```
<cffunction name="shouldReloadConfig" access="public" returntype="boolean" output="false"
  hint="Determines if the configuration file should be reloaded.">

  <cfset var result = false />

  <cfif shouldReloadBaseConfig() OR shouldReloadModuleConfig(>
    <cfset result = true />
  </cfif>

  <cfreturn result />
</cffunction>
```

shouldReloadModuleConfig

public boolean shouldReloadModuleConfig()

Determines if any of the module configuration files should be reloaded.

Parameters:

Code:

```
<cffunction name="shouldReloadModuleConfig" access="public" returntype="boolean" output="false"
  hint="Determines if any of the module configuration files should be reloaded.">
  <cfset var modules = getAppManager().getModuleManager().getModules() />
  <cfset var module = 0 />
  <cfset var result = false />

  <cfif NOT IsObject(getAppManager().getParent())>
    <cfloop collection="#modules#" item="module">
      <cfif modules[module].shouldReloadConfig(>
        <cfset result = true />
        <cfbreak />
      </cfif>
    </cfloop>
  </cfif>

  <cfreturn result />
</cffunction>
```

```
                </cfif>
            </cfloop>
        </cfif>
        <cfreturn result />
    </cffunction>
```

updateLastReloadDatetime

```
private void updateLastReloadDatetime( )
```

Updates the last reload datetime for this module or base application.

Parameters:

Code:

```
<cffunction name="updateLastReloadDatetime" access="private" returntype="void" output="false"
    hint="Updates the last reload datetime for this module or base application.">
    <cfset variables.lastReloadDatetime = Now() />
</cffunction>
```

AppManager

Package: MachII.framework

The main framework manager.

Method Summary

public AppManager	init() Used by the framework for initialization. Do not override.
public void	configure() Calls configure() on each of the manager instances.
public any	getAppKey()
public AppLoader	getAppLoader() Returns the AppLoader instance.
public EventManager	getEventManager()
public EventFilter- Manager	getFilterManager()
public ListenerMan- ager	getListenerManager()
public ModuleMan- ager	getModuleManager()
public string	getModuleName()
public any	getParent()
public PluginMan- ager	getPluginManager()
public PropertyMan- ager	getPropertyManager()

Method Summary

public RequestHandler	getRequestHandler([boolean createNew="false"]) Returns a new or cached instance of a RequestHandler.
public RequestManager	getRequestManager()
public SubroutineManager	getSubroutineManager()
public Utils	getUtils()
public ViewManager	getViewManager()
public void	setAppKey(string appkey)
public void	setAppLoader(AppLoader appLoader) Sets the AppLoader instance.
public void	setEventManager(EventManager eventManager)
public void	setFilterManager(EventFilterManager filterManager)
public void	setListenerManager(ListenerManager listenerManager)
public void	setModuleManager(ModuleManager moduleManager)
public void	setModuleName(string moduleName)
public void	setParent(AppManager parentAppManager)
public void	setPluginManager(PluginManager pluginManager)
public void	setPropertyManager(PropertyManager propertyManager)
public void	setRequestManager(RequestManager requestManager)
public void	setSubroutineManager(SubroutineManager subroutineManager)
public void	setUtils(Utils utils)
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 getRequestManager().configure() />
  <cfset getPluginManager().configure() />
  <cfset getListenerManager().configure() />
  <cfset getFilterManager().configure() />
  <cfset getSubroutineManager().configure() />
  <cfset getEventManager().configure() />
  <cfset getViewManager().configure() />

  <cfif NOT IsObject(getParent())>
    <cfset getModuleManager().configure() />
  </cfif>
</cffunction>
```

getAppKey

```
public any getAppKey( )
```

Parameters:

Code:

```
<cffunction name="getAppKey" access="public" type="string" output="false">
  <cfreturn variables.appkey />
</cffunction>
```

```
</cffunction>
```

getAppLoader

```
public AppLoader getAppLoader( )
```

Returns the AppLoader instance.

Parameters:

Code:

```
<cffunction name="getAppLoader" access="public" returntype="MachII.framework.AppLoader" output="false"
    hint="Returns the AppLoader instance.">
    <cfreturn variables.appLoader />
</cffunction>
```

getEventManager

```
public EventManager getEventManager( )
```

Parameters:

Code:

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

getFilterManager

```
public EventFilterManager getFilterManager( )
```

Parameters:

Code:

```
<cffunction name="getFilterManager" access="public" returnType="MachII.framework.EventFilterManager" 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>
```

getModuleManager

```
public ModuleManager getModuleManager( )
```

Parameters:

Code:

```
<cffunction name="getModuleManager" access="public" returnType="MachII.framework.ModuleManager" output="false">
    <cfreturn variables.moduleManager />
</cffunction>
```

getModuleName

public string getModuleName()

Parameters:

Code:

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

getParent

public any getParent()

Parameters:

Code:

```
<cffunction name="getParent" access="public" returntype="any" output="false">
    <cfreturn variables.parentAppManager />
</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." />
    <cfreturn getRequestManager().getRequestHandler(arguments.createNew) />
</cffunction>
```

getRequestManager

```
public RequestManager getRequestManager( )
```

Parameters:

Code:

```
<cffunction name="getRequestManager" access="public" returntype="MachII.framework.RequestManager" output="false">
    <cfreturn variables.requestManager />
</cffunction>
```

getSubroutineManager

```
public SubroutineManager getSubroutineManager( )
```

Parameters:

Code:

```
<cffunction name="getSubroutineManager" access="public" returntype="MachII.framework.SubroutineManager" output="false">
    <cfreturn variables.subroutineManager />
</cffunction>
```

getUtils

```
public Utils getUtils( )
```

Parameters:

Code:

```
<cffunction name="getUtils" access="public" returntype="MachII.util.Utils" output="false">
    <cfreturn variables.utils />
</cffunction>
```

getViewManager

public ViewManager getViewManager()

Parameters:

Code:

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

init

public AppManager init()

Used by the framework for initialization. Do not override.

Parameters:

Code:

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

setAppKey

public void setAppKey(string appkey)

Parameters:

string appkey

Code:

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

setAppLoader

```
public void setAppLoader( AppLoader appLoader )
```

Sets the AppLoader instance.

Parameters:

AppLoader appLoader

Code:

```
<cffunction name="setAppLoader" access="public" returntype="void" output="false"
  hint="Sets the AppLoader instance.">
  <cfargument name="appLoader" type="MachII.framework.AppLoader" required="true" />
  <cfset variables.appLoader = arguments.appLoader />
</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>
```

```
</cffunction>
```

setFilterManager

```
public void setFilterManager( EventFilterManager filterManager )
```

Parameters:

EventFilterManager filterManager

Code:

```
<cffunction name="setFilterManager" access="public" returntype="void" output="false">
    <cfargument name="filterManager" type="MachII.framework.EventFilterManager" 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>
```

setModuleManager

public void setModuleManager(ModuleManager moduleManager)

Parameters:

ModuleManager moduleManager

Code:

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

setModuleName

public void setModuleName(string moduleName)

Parameters:

string moduleName

Code:

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

setParent

public void setParent(AppManager parentAppManager)

Parameters:

AppManager parentAppManager

Code:

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

setRequestManager

```
public void setRequestManager( RequestManager requestManager )
```

Parameters:

RequestManager requestManager

Code:

```
<cffunction name="setRequestManager" access="public" returnType="void" output="false">
  <cfargument name="requestManager" type="MachII.framework.RequestManager" required="true" />
  <cfset variables.requestManager = arguments.requestManager />
</cffunction>
```

setSubroutineManager

```
public void setSubroutineManager( SubroutineManager subroutineManager )
```

Parameters:

SubroutineManager subroutineManager

Code:

```
<cffunction name="setSubroutineManager" access="public" returnType="void" output="false">
  <cfargument name="subroutineManager" type="MachII.framework.SubroutineManager" required="true" />
  <cfset variables.subroutineManager = arguments.subroutineManager />
</cffunction>
```

setUtils

public void setUtils(Utils utils)

Parameters:

Utils utils

Code:

```
<cffunction name="setUtils" access="public" returntype="void" output="false">
    <cfargument name="utils" type="MachII.util.Utils" required="true" />
    <cfset variables.utils = arguments.utils />
</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	announceEventInModule(string moduleName, string eventName, [struct eventArgs="#StructNew()#"])	Announces a new event to the framework.
private any	bindValue(string parameterName)	Binds placeholders values in parameters.
public string	buildUrl(string eventName, [any urlParameters=""], [string urlBase])	Builds a framework specific url with module name.
public string	buildUrlToModule(string moduleName, string eventName, [any urlParameters=""], [string urlBase])	Builds a framework specific url.
public void	configure()	Override to provide custom configuration logic. Called after init().
public AppManager	getAppManager()	Gets the components AppManager instance.

Method Summary

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 getPropertyManager().getProperty()
public PropertyManager	getPropertyManager()	Gets the components PropertyManager instance.
public boolean	hasParameter(string name)	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 getPropertyManager().setProperty()

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()#" />  
  
    <cfset var appKey = getAppManager().getAppLoader().getAppKey() />  
  
    <cfif StructKeyExists(request, "_MachIIRequestHandler_" & appKey)>  
        <cfset request["_MachIIRequestHandler_" & appKey].getEventContext().announceEvent(arguments.eventName, arguments.eventArgs) />  
    <cfelse>  
        <cfthrow message="The required RequestHandler is necessary to announce events is not set in 'request['_MachIIRequestHandler_' & appKey]" />  
    </cfif>  
</cffunction>
```

announceEventInModule

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

Announces a new event to the framework.

Parameters:

```
string moduleName
```

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

Code:

```
<cffunction name="announceEventInModule" access="public" returntype="void" output="false"
    hint="Announces a new event to the framework.">
    <cfargument name="moduleName" type="string" required="true"
        hint="The name of the module in which event exists." />
    <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." />

    <cfset var appKey = getAppManager().getAppLoader().getAppKey() />

    <cfif StructKeyExists(request, "_MachIIRequestHandler_" & appKey)>
        <cfset request["_MachIIRequestHandler_" & appKey].getEventContext().announceEvent(arguments.eventName, arguments.eventArgs) />
    <cfelse>
        <cfthrow message="The required RequestHandler is necessary to announce events is not set in 'request['_MachIIRequestHandler_' & appKey]'" />
    </cfif>
</cffunction>
```

bindValue

```
private any bindValue( string parameterName )
```

Binds placeholders values in parameters.

Parameters:

```
string parameterName
```

Code:

```
<cffunction name="bindValue" access="private" returntype="any" output="false"
    hint="Binds placeholders values in parameters.">
    <cfargument name="parameterName" type="string" required="true"
        hint="The parameter name." />

    <cfset var propertyName = "" />
```

```

    <cfset var value = variables.parameters[arguments.parameterName] />

    <cfif IsSimpleValue(value) AND REFindNoCase("\${(.)*?}", value)>
        <cfset propertyName = Mid(value, 3, Len(value) -3) />
        <cfif getPropertyManager().isPropertyDefined(propertyName)>
            <cfset value = getProperty(propertyName) />
        <cfelse>
            <cfthrow type="MachII.framework.PropertyNotDefinedToBindToParameter"
                message="The required property is not defined to bind to a parameter named '#arguments.p"
        </cfif>
    </cfif>

    <cfreturn value />
</cffunction>

```

buildUrl

```
public string buildUrl( string eventName, [any urlParameters=""], [string urlBase] )
```

Builds a framework specific url with module name.

Parameters:

```
string eventName
[any urlParameters=""]
[string urlBase]
```

Code:

```

<cffunction name="buildUrl" access="public" returntype="string" output="false"
    hint="Builds a framework specific url with module name.">
    <cfargument name="eventName" type="string" required="true"
        hint="Name of the event to build the url with." />
    <cfargument name="urlParameters" type="any" required="false" default=""
        hint="Name/value pairs (urlArg1=value1|urlArg2=value2) to build the url with or a struct of data." />
    <cfargument name="urlBase" type="string" required="false"
        hint="Base of the url. Defaults to the value of the urlBase property." />

    <cfset var appKey = getAppManager().getAppLoader().getAppKey() />

```

```
<cfset arguments.moduleName = request["_MachIIRequestHandler_" & appKey].getEventContext().getAppManager().getMod  
<cfreturn getAppManager().getRequestManager().buildUrl(argumentcollection=arguments) />  
</cffunction>
```

buildUrlToModule

public string buildUrlToModule(string moduleName, string eventName, [any urlParameters=""], [string urlBase])

Builds a framework specific url.

Parameters:

string moduleName
string eventName
[any urlParameters=""]
[string urlBase]

Code:

```
<cffunction name="buildUrlToModule" access="public" returntype="string" output="false"  
    hint="Builds a framework specific url.">  
    <cfargument name="moduleName" type="string" required="true"  
        hint="Name of the module to build the url with. Defaults to base module if empty string." />  
    <cfargument name="eventName" type="string" required="true"  
        hint="Name of the event to build the url with." />  
    <cfargument name="urlParameters" type="any" required="false" default=""  
        hint="Name/value pairs (urlArg1=value1|urlArg2=value2) to build the url with or a struct of data." />  
    <cfargument name="urlBase" type="string" required="false"  
        hint="Base of the url. Defaults to the value of the urlBase property." />  
    <cfreturn getAppManager().getRequestManager().buildUrl(argumentcollection=arguments) />  
</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>
```

getManager

public AppManager getManager()

Gets the components AppManager instance.

Parameters:

Code:

```
<cffunction name="getManager" access="public" 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 bindValue(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.">

  <cfset var key = "" />
  <cfset var resolvedParameters = StructNew() />

  <cfloop collection="#variables.parameters#" item="key">
    <cfset resolvedParameters[key] = bindValue(key) />
  </cfloop>

  <cfreturn resolvedParameters />
</cffunction>
```

getProperty

```
public any getProperty( string propertyName )
```

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

getPropertyManager

```
public PropertyManager getPropertyManager( )
```

Gets the components PropertyManager instance.

Parameters:

Code:

```
<cffunction name="getPropertyManager" access="public" returntype="MachII.framework.PropertyManager" output="false"
    hint="Gets the components PropertyManager instance.">
    <cfreturn getAppManager().getPropertyManager() />
</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 d
    <cfargument name="name" type="string" required="true"
        hint="The parameter name." />

    <cftry>
        <cfthrow type="MachII.framework.deprecatedMethod"
            message="The hasParameter() method has been deprecated. Please use isParameterDefined() instead.
        <cfcatch type="MachII.framework.deprecatedMethod">

            </cfcatch>
        </cftry>

        <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) />
</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 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 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>
```

Command

Package: MachII.framework

Base Command component.

Method Summary

public Command	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)
public void	setParameters(struct parameters) Sets a struct of parameters to this command.

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 Command init()

Used by the framework for initialization.

Parameters:

Code:

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

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

Sets a struct of parameters to this command.

Parameters:

struct parameters

Code:

```
<cffunction name="setParameters" access="public" returntype="void" output="false"
  hint="Sets a struct of parameters to this command.">
  <cfargument name="parameters" type="struct" required="true" />

  <cfset var key = "" />

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

CommandLoaderBase

Package: MachII.framework

Base component to load commands for the framework.

Method Summary

public void	init() Initialization function called by the framework.
private Command	createCommand(any commandNode)
private Announce- Command	setupAnnounce(any commandNode) Sets up an announce command.
private Command	setupDefault() Sets up a default command.
private EventAr- gCommand	setupEventArg(any commandNode) Sets up an event-arg command.
private EventBean- Command	setupEventBean(any commandNode) Sets up a event-bean command.
private EventMap- pingCommand	setupEventMapping(any commandNode) Sets up an event-mapping command.
private ExecuteCom- mand	setupExecute(any commandNode) Sets up an execute command.
private FilterCom- mand	setupFilter(any commandNode)

Method Summary

	Sets up a filter command.
private Notify-Command	setupNotify(any commandNode) Sets up a notify command.
private RedirectCommand	setupRedirect(any commandNode) Sets up a redirect command.
private ViewPage-Command	setupViewPage(any commandNode) Sets up a view-page command.

Method Detail**createCommand**

private Command createCommand(any commandNode)

Parameters:

any commandNode

Code:

```
<cffunction name="createCommand" access="private" returntype="MachII.framework.Command" output="false">
  <cfargument name="commandNode" type="any" required="true" />

  <cfset var command = "" />

  <cfif arguments.commandNode.xmlName EQ "view-page">
    <cfset command = setupViewPage(arguments.commandNode) />
  </cfif>
</cffunction>
```

```
<cfelseif arguments.commandNode.xmlName EQ "notify">
    <cfset command = setupNotify(arguments.commandNode) />

<cfelseif arguments.commandNode.xmlName EQ "announce">
    <cfset command = setupAnnounce(arguments.commandNode) />

<cfelseif arguments.commandNode.xmlName EQ "event-mapping">
    <cfset command = setupEventMapping(arguments.commandNode) />

<cfelseif arguments.commandNode.xmlName EQ "execute">
    <cfset command = setupExecute(arguments.commandNode) />

<cfelseif arguments.commandNode.xmlName EQ "filter">
    <cfset command = setupFilter(arguments.commandNode) />

<cfelseif arguments.commandNode.xmlName EQ "event-bean">
    <cfset command = setupEventBean(arguments.commandNode) />

<cfelseif arguments.commandNode.xmlName EQ "redirect">
    <cfset command = setupRedirect(arguments.commandNode) />

<cfelseif arguments.commandNode.xmlName EQ "event-arg">
    <cfset command = setupEventArg(arguments.commandNode) />

<cfelse>
    <cfset command = setupDefault(arguments.commandNode) />
</cfif>

<cfreturn command />
</cffunction>
```

init

public void init()

Initialization function called by the framework.

Parameters:

Code:

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

</cffunction>
```

setupAnnounce

private AnnounceCommand setupAnnounce(any commandNode)

Sets up an announce command.

Parameters:

any commandNode

Code:

```
<cffunction name="setupAnnounce" access="private" returntype="MachII.framework.commands.AnnounceCommand" output="false"
    hint="Sets up an announce command.">
    <cfargument name="commandNode" type="any" required="true" />

    <cfset var command = "" />
    <cfset var eventName = arguments.commandNode.xmlAttributes["event"] />
    <cfset var copyEventArgs = true />
    <cfset var moduleName = "" />

    <cfif StructKeyExists(arguments.commandNode.xmlAttributes, "copyEventArgs")>
        <cfset copyEventArgs = arguments.commandNode.xmlAttributes["copyEventArgs"] />
    </cfif>
    <cfif StructKeyExists(arguments.commandNode.xmlAttributes, "module")>
        <cfset moduleName = arguments.commandNode.xmlAttributes["module"] />
    <cfelse>
        <cfset moduleName = getAppManager().getModuleName() />
    </cfif>

    <cfset command = CreateObject("component", "MachII.framework.commands.AnnounceCommand").init(eventName, copyEventArgs)

    <cfreturn command />
</cffunction>
```

setupDefault

```
private Command setupDefault( )
```

Sets up a default command.

Parameters:

Code:

```
<cffunction name="setupDefault" access="private" returntype="MachII.framework.Command" output="false"
    hint="Sets up a default command.">
    <cfset var command = CreateObject("component", "MachII.framework.Command").init() />
    <cfreturn command />
</cffunction>
```

setupEventArg

```
private EventArgCommand setupEventArg( any commandNode )
```

Sets up an event-arg command.

Parameters:

any commandNode

Code:

```
<cffunction name="setupEventArg" access="private" returntype="MachII.framework.commands.EventArgCommand" output="false"
    hint="Sets up an event-arg command.">
    <cfargument name="commandNode" type="any" required="true" />
    <cfset var command = "" />
    <cfset var argValue = "" />
    <cfset var argVariable = "" />
    <cfset var argName = arguments.commandNode.xmlAttributes["name"] />
```

```
<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "value")>
    <cfset argValue = arguments.commandNode.xmlAttributes["value"] />
</cfif>
<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "variable")>
    <cfset argVariable = arguments.commandNode.xmlAttributes["variable"] />
</cfif>

<cfset command = CreateObject("component", "MachII.framework.commands.EventArgCommand").init(argName, argValue, argVariable) />
<cfreturn command />
</cffunction>
```

setupEventBean

private EventBeanCommand setupEventBean(any commandNode)

Sets up a event-bean command.

Parameters:

any commandNode

Code:

```
<cffunction name="setupEventBean" access="private" returntype="MachII.framework.commands.EventBeanCommand" output="false"
    hint="Sets up a event-bean command.">
    <cfargument name="commandNode" type="any" required="true" />

    <cfset var command = "" />
    <cfset var beanName = arguments.commandNode.xmlAttributes["name"] />
    <cfset var beanType = arguments.commandNode.xmlAttributes["type"] />
    <cfset var beanFields = "" />
    <cfset var reinit = true />

    <cfif StructKeyExists(arguments.commandNode.xmlAttributes, "fields")>
        <cfset beanFields = arguments.commandNode.xmlAttributes["fields"] />
    </cfif>
    <cfif StructKeyExists(arguments.commandNode.xmlAttributes, "reinit")>
        <cfset reinit = arguments.commandNode.xmlAttributes["reinit"] />
    </cfif>
```

```

        <cfset command = CreateObject("component", "MachII.framework.commands.EventBeanCommand").init(beanName, beanType)
        <cfreturn command />
    </cffunction>

```

setupEventMapping

private EventMappingCommand setupEventMapping(any commandNode)

Sets up an event-mapping command.

Parameters:

any commandNode

Code:

```

<cffunction name="setupEventMapping" access="private" returntype="MachII.framework.commands.EventMappingCommand" output="false"
    hint="Sets up an event-mapping command.">
    <cfargument name="commandNode" type="any" required="true" />

    <cfset var command = "" />
    <cfset var eventName = arguments.commandNode.xmlAttributes["event"] />
    <cfset var mappingName = arguments.commandNode.xmlAttributes["mapping"] />
    <cfset var mappingModule = "" />

    <cfif StructKeyExists(arguments.commandNode.xmlAttributes, "mappingModule")>
        <cfset mappingModule = arguments.commandNode.xmlAttributes["mappingModule"] />
    <cfelse>
        <cfset mappingModule = getAppManager().getModuleName() />
    </cfif>

    <cfset command = CreateObject("component", "MachII.framework.commands.EventMappingCommand").init(eventName, mappingName, mappingModule)

    <cfreturn command />
</cffunction>

```

setupExecute

private ExecuteCommand setupExecute(any commandNode)

Sets up an execute command.

Parameters:

any commandNode

Code:

```
<cffunction name="setupExecute" access="private" returnType="MachII.framework.commands.ExecuteCommand" output="false"
    hint="Sets up an execute command.">
    <cfargument name="commandNode" type="any" required="true" />

    <cfset var command = "" />
    <cfset var subroutine = arguments.commandNode.xmlAttributes["subroutine"] />

    <cfset command = CreateObject("component", "MachII.framework.commands.ExecuteCommand").init(subroutine) />

    <cfreturn command />
</cffunction>
```

setupFilter

private FilterCommand setupFilter(any commandNode)

Sets up a filter command.

Parameters:

any commandNode

Code:

```
<cffunction name="setupFilter" access="private" returnType="MachII.framework.commands.FilterCommand" output="false"
    hint="Sets up a filter command.">
    <cfargument name="commandNode" type="any" required="true" />

    <cfset var command = "" />
    <cfset var filterName = arguments.commandNode.xmlAttributes["name"] />
```

```

<cfset var filterParams = StructNew() />
<cfset var paramNodes = arguments.commandNode.xmlChildren />
<cfset var paramName = "" />
<cfset var paramValue = "" />
<cfset var filter = getAppManager().getFilterManager().getFilter(filterName) />
<cfset var i = "" />

<cfloop from="1" to="#ArrayLen(paramNodes)#" index="i">
  <cfset paramName = paramNodes[i].xmlAttributes["name"] />
  <cfif NOT StructKeyExists(paramNodes[i].xmlAttributes, "value")>
    <cfset paramValue = getAppManager().getUtils().recurseComplexValues(paramNodes[i]) />
  <cfelse>
    <cfset paramValue = paramNodes[i].xmlAttributes["value"] />
  </cfif>
  <cfset filterParams[paramName] = paramValue />
</cfloop>

<cfset command = CreateObject("component", "MachII.framework.commands.FilterCommand").init(filter, filterParams)

<cfreturn command />
</cffunction>

```

setupNotify

private NotifyCommand setupNotify(any commandNode)

Sets up a notify command.

Parameters:

any commandNode

Code:

```

<cffunction name="setupNotify" access="private" returnType="MachII.framework.commands.NotifyCommand" output="false"
  hint="Sets up a notify command.">
  <cfargument name="commandNode" type="any" required="true" />

  <cfset var command = "" />
  <cfset var notifyListener = arguments.commandNode.xmlAttributes["listener"] />
  <cfset var notifyMethod = arguments.commandNode.xmlAttributes["method"] />

```

```

<cfset var notifyResultKey = "" />
<cfset var notifyResultArg = "" />
<cfset var listener = getAppManager().getListenerManager().getListener(notifyListener) />

<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "resultKey")>
    <cfset notifyResultKey = arguments.commandNode.xmlAttributes["resultKey"] />
</cfif>
<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "resultArg")>
    <cfset notifyResultArg = arguments.commandNode.xmlAttributes["resultArg"] />
</cfif>

<cfset command = CreateObject("component", "MachII.framework.commands.NotifyCommand").init(listener, notifyMethod) />

<cfreturn command />
</cffunction>

```

setupRedirect

private RedirectCommand setupRedirect(any commandNode)

Sets up a redirect command.

Parameters:

any commandNode

Code:

```

<cffunction name="setupRedirect" access="private" returntype="MachII.framework.commands.RedirectCommand" output="false"
    hint="Sets up a redirect command.">
    <cfargument name="commandNode" type="any" required="true" />

    <cfset var command = "" />
    <cfset var eventName = "" />
    <cfset var redirectUrl = getAppManager().getPropertyManager().getProperty("urlBase", "index.cfm") />
    <cfset var moduleName = "" />
    <cfset var args = "" />
    <cfset var persist = false />
    <cfset var persistArgs = "" />
    <cfset var statusType = "temporary" />
    <cfset var eventParameter = getAppManager().getPropertyManager().getProperty("eventParameter", "event") />

```

```
<cfset var redirectPersistParameter = getAppManager().getPropertyManager().getProperty("redirectPersistParameter") />
<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "event")>
    <cfset eventName = arguments.commandNode.xmlAttributes["event"] />
</cfif>
<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "url")>
    <cfset redirectUrl = arguments.commandNode.xmlAttributes["url"] />
</cfif>
<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "args")>
    <cfset args = arguments.commandNode.xmlAttributes["args"] />
</cfif>
<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "persist")>
    <cfset persist = arguments.commandNode.xmlAttributes["persist"] />
</cfif>
<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "persistArgs")>
    <cfset persistArgs = arguments.commandNode.xmlAttributes["persistArgs"] />
</cfif>
<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "module")>
    <cfset moduleName = arguments.commandNode.xmlAttributes["module"] />
<cfelse>
    <cfset moduleName = getAppManager().getModuleName() />
</cfif>
<cfif StructKeyExists(arguments.commandNode.xmlAttributes, "statusType")>
    <cfset statusType = arguments.commandNode.xmlAttributes["statusType"] />
</cfif>

<cfset command = CreateObject("component", "MachII.framework.commands.RedirectCommand").init(eventName, eventParameters, redirectUrl, args, persist, persistArgs, moduleName, statusType) />
<cfreturn command />
</cffunction>
```

setupViewPage

private ViewPageCommand setupViewPage(any commandNode)

Sets up a view-page command.

Parameters:

any commandNode

Code:

```
<cffunction name="setupViewPage" access="private" returntype="MachII.framework.commands.ViewPageCommand" output="false"
  hint="Sets up a view-page command.">
  <cfargument name="commandNode" type="any" required="true" />

  <cfset var command = "" />
  <cfset var viewName = arguments.commandNode.xmlAttributes["name"] />
  <cfset var contentKey = "" />
  <cfset var contentArg = "" />
  <cfset var appendContent = "" />

  <cfif StructKeyExists(arguments.commandNode.xmlAttributes, "contentKey")>
    <cfset contentKey = commandNode.xmlAttributes["contentKey"] />
  </cfif>
  <cfif StructKeyExists(arguments.commandNode.xmlAttributes, "contentArg")>
    <cfset contentArg = commandNode.xmlAttributes["contentArg"] />
  </cfif>
  <cfif StructKeyExists(arguments.commandNode.xmlAttributes, "append")>
    <cfset appendContent = arguments.commandNode.xmlAttributes["append"] />
  </cfif>

  <cfset command = CreateObject("component", "MachII.framework.commands.ViewPageCommand").init(viewName, contentKey,
  contentArg, appendContent) />
  <cfreturn command />
</cffunction>
```

Event

Package: MachII.framework

The Event object encapsulates the event args.

Method Summary

public Event	init([string name=""], [struct args="#StructNew()#"], [string requestName=""], [string requestModuleName=""], [string moduleName=""]) 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	getModuleName() Returns the module name of the event object.
public string	getName() Returns the name of the event object.
public string	getRequestModuleName() Returns the module name that started the request lifecycle.
public string	getRequestName() Returns the event name that started the request lifecycle.

Method Summary

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]) 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	setModuleName(string moduleName) Sets the module name of the event object. (Not for public use.)
public void	setName(string name) Sets the name of the event object. (Not for public use.)
public void	setRequestModuleName(string requestModuleName) Sets the module name that started the request lifecycle. (Not for public use.)
public void	setRequestName(string requestName) Sets the event name that started the request lifecycle. (Not for public use.)

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>
```

getModuleName

```
public string getModuleName( )
```

Returns the module name of the event object.

Parameters:

Code:

```
<cffunction name="getModuleName" access="public" returntype="string" output="false"
  hint="Returns the module name of the event object.">
  <cfreturn variables.moduleName />
</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>
```

getRequestModuleName

public string getRequestModuleName()

Returns the module name that started the request lifecycle.

Parameters:

Code:

```
<cffunction name="getRequestModuleName" access="public" returntype="string" output="false"
    hint="Returns the module name that started the request lifecycle.">
    <cfreturn variables.requestModuleName />
</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 Event init( [string name=""], [struct args="#StructNew()#"], [string requestName=""], [string requestModuleName=""], [string moduleName=""] )
```

Used by the framework for initialization. Do not override.

Parameters:

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

Code:

```
<cffunction name="init" access="public" returntype="Event" 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="The request event name for this request lifecycle." />
    <cfargument name="requestModuleName" type="string" required="false" default=""
        hint="The request module name for this request lifecycle." />
    <cfargument name="moduleName" type="string" required="false" default=""
        hint="The module name of the event object." />

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

```
<cfset setRequestName(arguments.requestName) />
<cfset setRequestModuleName(arguments.requestModuleName) />
<cfset setModuleName(arguments.moduleName) />

<cfreturn this />
</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>
```

setModuleName

```
public void setModuleName( string moduleName )
```

Sets the module name of the event object. (Not for public use.)

Parameters:

string moduleName

Code:

```
<cffunction name="setModuleName" access="public" returntype="void" output="false"
    hint="Sets the module name of the event object. (Not for public use.)">
    <cfargument name="moduleName" type="string" required="true"
        hint="A module name for this event." />
    <cfset variables.moduleName = arguments.moduleName />
</cffunction>
```

setName

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

Sets the name of the event object. (Not for public use.)

Parameters:

string name

Code:

```
<cffunction name="setName" access="public" returntype="void" output="false"
    hint="Sets the name of the event object. (Not for public use.)">
    <cfargument name="name" type="string" required="true"
```

```
        hint="A name for this event." />
        <cfset variables.name = arguments.name />
    </cffunction>
```

setRequestModuleName

```
public void setRequestModuleName( string requestModuleName )
```

Sets the module name that started the request lifecycle. (Not for public use.)

Parameters:

string requestModuleName

Code:

```
<cffunction name="setRequestModuleName" access="public" returntype="void" output="false"
    hint="Sets the module name that started the request lifecycle. (Not for public use.)">
    <cfargument name="requestModuleName" type="string" required="true"
        hint="A request name for this event." />
    <cfset variables.requestModuleName = arguments.requestModuleName />
</cffunction>
```

setRequestName

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

Sets the event name that started the request lifecycle. (Not for public use.)

Parameters:

string requestName

Code:

```
<cffunction name="setRequestName" access="public" returntype="void" output="false"
    hint="Sets the event name that started the request lifecycle. (Not for public use.)">
```

```
<cfargument name="requestName" type="string" required="true"
            hint="A request name for this event." />
<cfset variables.requestName = arguments.requestName />
</cffunction>
```

EventContext

Package: MachII.framework

Handles event-command execution and event processing mechanism for an event lifecycle.

Method Summary

public EventContext	init(RequestHandler requestHandler, SizedQueue eventQueue)	Initializes the event-context.
public void	announceEvent(string eventName, [struct eventArgs="#StructNew()#"], [string moduleName="#getAppManager().getModuleName()#"])	Queues an event for the framework to handle.
public void	clearEventMappings()	Clears the current event mappings.
public void	clearEventQueue()	Clears the event queue.
public void	displayView(Event event, string viewName, [string contentKey=""], [string contentArg=""], [boolean append="false"])	Displays a view.
public boolean	executeSubroutine(string subroutineName, Event event)	Executes a subroutine.
public AppManager	getAppManager()	Sets the appManager that pertains to context of currently executing event.
public Event	getCurrentEvent()	Gets the current event object.

Method Summary

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

Method Summary

	Checks if an event mapping is defined.
private void	setAppManager(AppManager appManager) Sets the appManager that pertains to context of currently executing event.
private void	setCurrentEvent(Event currentEvent)
public void	setEventMapping(string eventName, string mappingName, [string moduleName="#getAppManager().getModuleName()#"]) Sets an event mapping.
private void	setEventQueue(SizedQueue eventQueue)
public void	setExceptionEventName(string exceptionEventName)
private void	setPreviousEvent(Event previousEvent)
private void	setRequestHandler(RequestHandler requestHandler)
public void	setup(AppManager appManager, [any currentEvent=""]) Sets up the event-context.
private void	setViewContext(ViewContext viewContext)

Method Detail**announceEvent**

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

Queues an event for the framework to handle.

Parameters:

```
string eventName
[struct eventArgs="#StructNew()#"]
[string moduleName="#getAppManager().getModuleName()#"]
```

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()#" />
  <cfargument name="moduleName" type="string" required="false" default="#getAppManager().getModuleName()#" />

  <cfset var mapping = "" />
  <cfset var nextEvent = "" />
  <cfset var nextModuleName = arguments.moduleName />
  <cfset var nextEventName = arguments.eventName />
  <cfset var exception = "" />

  <cftry>

    <cfif isEventMappingDefined(arguments.eventName)>
      <cfset mapping = getEventMapping(arguments.eventName) />
      <cfset nextModuleName = mapping.moduleName />
      <cfset nextEventName = mapping.eventName />
    </cfif>

    <cfset nextEvent = getAppManager().getEventManager().createEvent(nextModuleName, nextEventName, arguments

  <cfset getEventQueue().put(nextEvent) />

  <cfcatch type="any">
    <cfset exception = getRequestHandler().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>
```

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>
```

executeSubroutine

```
public boolean executeSubroutine( string subroutineName, Event event )
```

Executes a subroutine.

Parameters:

string subroutineName

Event event

Code:

```
<cffunction name="executeSubroutine" access="public" returntype="boolean" output="true"
  hint="Executes a subroutine.">
  <cfargument name="subroutineName" type="string" required="true" />
  <cfargument name="event" type="MachII.framework.Event" required="true" />

  <cfset var subroutineHandler = "" />
  <cfset var exception = "" />
  <cfset var continue = true />

  <cftry>
```

```
        <cfset subroutineHandler = getAppManager().getSubroutineManager().getSubroutineHandler(arguments.subroutine) />
        <cfset continue = subroutineHandler.handleSubroutine(arguments.event, this) />
        <cfcatch type="any">
            <cfset exception = getRequestHandler().wrapException(cfcatch) />
            <cfset handleException(exception, true) />
        </cfcatch>
    </cftry>
    <cfreturn continue />
</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Sets the appManager that pertains to context of currently executing event.

Parameters:

Code:

```
<cffunction name="getAppManager" access="public" returntype="MachII.framework.AppManager" output="false"
    hint="Sets the appManager that pertains to context of currently executing event.">
    <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>
```

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 getRequestHandler().getEventCount() />
</cffunction>
```

getEventMapping

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

Gets an event mapping by the event name.

Parameters:

string eventName

Code:

```
<cffunction name="getEventMapping" access="public" returntype="struct" output="false"
    hint="Gets an event mapping by the event name.">
    <cfargument name="eventName" type="string" required="true" />
```

```
<cfset var mapping = StructNew() />

<cfif StructKeyExists(variables.mappings, arguments.eventName)>
    <cfset mapping = variables.mappings[arguments.eventName] />
<cfelse>
    <cfset mapping.eventName = arguments.eventName />
    <cfset mapping.moduleName = getAppManager().getModuleName() />
</cfif>

<cfreturn mapping />
</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>
```

```
</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>
```

getRequestHandler

```
private any getRequestHandler( )
```

Parameters:

Code:

```
<cffunction name="getRequestHandler" access="private" type="MachII.framework.RequestHandler" output="false">
    <cfreturn variables.requestHandler />
</cffunction>
```

getViewContext

private any getViewContext()

Parameters:

Code:

```
<cffunction name="getViewContext" access="private" type="MachII.framework.ViewContext" output="false">
    <cfreturn variables.viewContext />
</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 nextEvent = "" />
<cfset var eventArgs = StructNew() />
<cfset var appManager = getAppManager() />
<cfset var result = StructNew() />

<cfset result.eventName = getExceptionEventName() />

<cftry>

    <cfset eventArgs.exception = arguments.exception />
    <cfif hasCurrentEvent()>
        <cfset eventArgs.exceptionEvent = getCurrentEvent() />
    </cfif>

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

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

    <cfif isEventMappingDefined(result.eventName)>
        <cfset result = getEventMapping(exceptionEventName) />
        <cfif NOT Len(result.moduleName)>
            <cfset appManager = appManager.getModuleManager().getModule(result.moduleName).getModuleManager() />
        </cfif>
        <cfset appManager = appManager.getModuleManager().getAppManager() />
    </cfif>

    <cfelseif appManager.getPropertyManager().isPropertyDefined("exceptionEvent")>
        <cfset result.moduleName = appManager.getModuleName() />
    </cfelseif>
    <cfset result.moduleName = "" />
</cfif>

<cfset nextEvent = appManager.getEventManager().createEvent(result.moduleName, result.eventName, eventArgs) />
<cfset getEventQueue().put(nextEvent) />
```

```
        <cfcatch type="any">
            <cfrethrow />
        </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>
```

init

```
public EventContext init( RequestHandler requestHandler, SizedQueue eventQueue )
```

Initializes the event-context.

Parameters:

RequestHandler requestHandler
SizedQueue eventQueue

Code:

```
<cffunction name="init" access="public" returntype="EventContext" output="false"
    hint="Initializes the event-context.">
    <cfargument name="requestHandler" type="MachII.framework.RequestHandler" required="true" />
    <cfargument name="eventQueue" type="MachII.util.SizedQueue" required="true" />

    <cfset setRequestHandler(arguments.requestHandler) />
    <cfset setEventQueue(arguments.eventQueue) />
    <cfset setViewContext(CreateObject("component", "MachII.framework.ViewContext")) />

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

isEventMappingDefined

```
public boolean isEventMappingDefined( string eventName )
```

Checks if an event mapping is defined.

Parameters:

string eventName

Code:

```
<cffunction name="isEventMappingDefined" type="public" returntype="boolean" output="false"
    hint="Checks if an event mapping is defined.">
    <cfargument name="eventName" type="string" required="true" />

    <cfset var result = false />

    <cfif StructKeyExists(variables.mappings, arguments.eventName)>
        <cfset result = true />
    </cfif>
</cffunction>
```

```
<cfreturn result />
</cffunction>
```

setAppManager

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

Sets the appManager that pertains to context of currently executing event.

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="private" returnType="void" output="false"
    hint="Sets the appManager that pertains to context of currently executing event.">
    <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>
```

setEventMapping

```
public void setEventMapping( string eventName, string mappingName, [string mappingModuleName="#getAppManager().getModuleName()#"] )
```

Sets an event mapping.

Parameters:

```
string eventName  
string mappingName  
[string mappingModuleName="#getAppManager().getModuleName()#"]
```

Code:

```
<cffunction name="setEventMapping" access="public" returntype="void" output="false"  
    hint="Sets an event mapping.">  
    <cfargument name="eventName" type="string" required="true" />  
    <cfargument name="mappingName" type="string" required="true" />  
    <cfargument name="mappingModuleName" type="string" required="false"  
        default="#getAppManager().getModuleName()#" />  
  
    <cfset var mapping = StructNew() />  
  
    <cfif Len(arguments.mappingModuleName)  
        AND NOT getAppManager().getModuleManager().isModuleDefined(arguments.mappingModuleName)>  
        <cfthrow type="MachII.framework.eventMappingModuleNotDefined"  
            message="The module '#arguments.mappingModuleName#' cannot be found for this event-mapping." />  
    </cfif>  
  
    <cfset mapping.eventName = arguments.mappingName />  
    <cfset mapping.moduleName = arguments.mappingModuleName />  
  
    <cfset variables.mappings[arguments.eventName] = mapping />  
</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>
```

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>
```

setRequestHandler

```
private void setRequestHandler( RequestHandler requestHandler )
```

Parameters:

RequestHandler requestHandler

Code:

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

setup

```
public void setup( AppManager appManager, [any currentEvent=""] )
```

Sets up the event-context.

Parameters:

AppManager appManager
[any currentEvent=""]

Code:

```
<cffunction name="setup" access="public" returntype="void" output="false"
```

```
hint="Sets up the event-context.">
<cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
<cfargument name="currentEvent" type="any" required="false" default="" />

<cfset setAppManager(arguments.appManager) />
<cfif hasCurrentEvent()>
    <cfset setPreviousEvent(getCurrentEvent()) />
</cfif>
<cfif IsObject(arguments.currentEvent)>
    <cfset setCurrentEvent(arguments.currentEvent) />
</cfif>

<cfset setExceptionEventName(getAppManager().getPropertyManager().getProperty("exceptionEvent")) />

<cfset getViewContext().init(getAppManager()) />

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

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: setParameters , hasParameter , buildUrlToModule , isParameterDefined , buildUrl , announceEventInModule , setAppManager , getParameter , getProperty , getParameters , setProperty , getPropertyManager , bindValue , configure , getAppManager , setParameter , announceEvent

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" output="false"
    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>
```


EventFilterManager

Package: MachII.framework

Manages registered EventFilters for the framework.

Method Summary

public EventFilter-Manager	init(AppManager appManager, [any parentFilterManager=""])	Initialization function called by the framework.
public void	addFilter(string filterName, EventFilter filter, [boolean overrideCheck="false"])	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 any	getParent()	Sets the parent FilterManager instance this FilterManager belongs to. It will return empty string if no parent is defined.
public boolean	isFilterDefined(string filterName)	Checks if a filter is defined in this event filter manager. Does NOT check the parent.
public void	loadXml(string configXML, [boolean override="false"])	Loads xml for the manager.
public void	removeFilter(string filterName)	

Method Summary

	Removes a filter. Does NOT remove from a parent.
public void	setAppManager(AppManager appManager)
public void	setParent(EventFilterManager parentFilterManager)
	Returns the parent FilterManager instance this FilterManager belongs to.

Method Detail**addFilter**

```
public void addFilter( string filterName, EventFilter filter, [boolean overrideCheck="false"] )
```

Registers an EventFilter by name.

Parameters:

```
string filterName
EventFilter filter
[boolean overrideCheck="false"]
```

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" />
  <cfargument name="overrideCheck" type="boolean" required="false" default="false" />

  <cfif NOT arguments.overrideCheck AND 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" returtype="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" returtype="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] />
  <cfelseif isObject(getParent()) AND getParent().isFilterDefined(arguments.filterName)>
    <cfreturn getParent().getFilter(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>
```

getParent

```
public any getParent( )
```

Sets the parent FilterManager instance this FilterManager belongs to. It will return empty string if no parent is defined.

Parameters:

Code:

```
<cffunction name="getParent" access="public" returnType="any" output="false"
    hint="Sets the parent FilterManager instance this FilterManager belongs to. It will return empty string if no pa
    <cfreturn variables.parentFilterManager />
</cffunction>
```

init

```
public EventFilterManager init( AppManager appManager, [any parentFilterManager=""] )
```

Initialization function called by the framework.

Parameters:

AppManager appManager
[any parentFilterManager=""]

Code:

```
<cffunction name="init" access="public" returnType="EventFilterManager" output="false"
    hint="Initialization function called by the framework.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfargument name="parentFilterManager" type="any" required="false" default=""
        hint="Optional argument for a parent filter manager. If there isn't one default to empty string." />

    <cfset setAppManager(arguments.appManager) />
    <cfset variables.utils = getAppManager().getUtils() />

    <cfif isObject(arguments.parentFilterManager)>
        <cfset setParent(arguments.parentFilterManager) />
    </cfif>

    <cfreturn this />
```

```
</cffunction>
```

isFilterDefined

```
public boolean isFilterDefined( string filterName )
```

Checks if a filter is defined in this event filter manager. Does NOT check the parent.

Parameters:

```
string filterName
```

Code:

```
<cffunction name="isFilterDefined" access="public" returntype="boolean" output="false"
    hint="Checks if a filter is defined in this event filter manager. Does NOT check the parent.">
    <cfargument name="filterName" type="string" required="true" />
    <cfreturn StructKeyExists(variables.filters, arguments.filterName) />
</cffunction>
```

loadXml

```
public void loadXml( string configXML, [boolean override="false"] )
```

Loads xml for the manager.

Parameters:

```
string configXML
```

```
[boolean override="false"]
```

Code:

```
<cffunction name="loadXml" access="public" returntype="void" output="false"
    hint="Loads xml for the manager.">
    <cfargument name="configXML" type="string" required="true" />
    <cfargument name="override" type="boolean" required="false" default="false" />
</cffunction>
```

```
<cfset var filterNodes = "" />
<cfset var filterParams = "" />
<cfset var paramNodes = "" />
<cfset var paramName = "" />
<cfset var paramValue = "" />
<cfset var filter = "" />
<cfset var filterName = "" />
<cfset var filterType = "" />
<cfset var hasParent = isObject(getParent()) />
<cfset var mapping = "" />
<cfset var i = 0 />
<cfset var j = 0 />

<cfif NOT arguments.override>
    <cfset filterNodes = XMLSearch(arguments.configXML, "mach-ii/event-filters/event-filter") />
<cfelse>
    <cfset filterNodes = XMLSearch(arguments.configXML, "../event-filters/event-filter") />
</cfif>

<cfloop from="1" to="#ArrayLen(filterNodes)#" index="i">
    <cfset filterName = filterNodes[i].xmlAttributes["name"] />

    <cfif hasParent AND arguments.override AND StructKeyExists(filterNodes[i].xmlAttributes, "overrideAction")
        <cfif filterNodes[i].xmlAttributes["overrideAction"] EQ "useParent">
            <cfset removeFilter(filterName) />
        <cfelseif filterNodes[i].xmlAttributes["overrideAction"] EQ "addFromParent">
            <cfif StructKeyExists(filterNodes[i].xmlAttributes, "mapping")>
                <cfset mapping = filterNodes[i].xmlAttributes["mapping"] />
            <cfelse>
                <cfset mapping = filterName />
            </cfif>

            <cfif NOT getParent().isFilterDefined(mapping)>
                <cfthrow type="MachII.framework.overrideFilterNotDefined"
                    message="An filter named '#mapping#' cannot be found in the parent event"
            </cfif>

            <cfset addFilter(filterName, getParent().getFilter(mapping), arguments.override) />
        </cfif>
    </cfloop>
```

```

        </cfif>
    <cfelse>
        <cfset filterType = filterNodes[i].xmlAttributes["type"] />

        <cfset filterParams = StructNew() />

        <cfif StructKeyExists(filterNodes[i], "parameters")>
            <cfset paramNodes = filterNodes[i].parameters.xmlChildren />
            <cfloop from="1" to="#ArrayLen(paramNodes)#" index="j">
                <cfset paramName = paramNodes[j].xmlAttributes["name"] />
                <cftry>
                    <cfset paramValue = variables.utils.recurseComplexValues(paramNodes[j]) />
                    <cfcatch type="any">
                        <cfthrow type="MachII.framework.InvalidParameterXml"
                            message="Xml parsing error for the parameter named '#paramName'" />
                    </cfcatch>
                </cftry>
                <cfset filterParams[paramName] = paramValue />
            </cfloop>
        </cfif>

        <cfset filter = CreateObject("component", filterType).init(getAppManager(), filterParams) />
        <cfset addFilter(filterName, filter, arguments.override) />
    </cfif>
</cfloop>
</cffunction>

```

removeFilter

```
public void removeFilter( string filterName )
```

Removes a filter. Does NOT remove from a parent.

Parameters:

string filterName

Code:

```
<cffunction name="removeFilter" access="public" returntype="void" output="false"
  hint="Removes a filter. Does NOT remove from a parent.">
  <cfargument name="filterName" type="string" required="true" />
  <cfset StructDelete(variables.filters, arguments.filterName, false) />
</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>
```

setParent

```
public void setParent( EventFilterManager parentFilterManager )
```

Returns the parent FilterManager instance this FilterManager belongs to.

Parameters:

EventFilterManager parentFilterManager

Code:

```
<cffunction name="setParent" access="public" returntype="void" output="false"
  hint="Returns the parent FilterManager instance this FilterManager belongs to.">
  <cfargument name="parentFilterManager" type="MachII.framework.EventFilterManager" required="true" />
```

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

EventHandler

Package: MachII.framework

Handles processing of EventCommands for an Event.

Method Summary

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

Method Detail

addCommand

```
public void addCommand( Command command )
```

Adds an Command.

Parameters:

Command command

Code:

```
<cffunction name="addCommand" access="public" returntype="void" output="false"
    hint="Adds an Command.">
    <cfargument name="command" type="MachII.framework.Command" 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 from="1" to="#ArrayLen(variables.commands)#" index="i">
  <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( string access )
```

Used by the framework for initialization. Do not override.

Parameters:

string access

Code:

```
<cffunction name="init" access="public" returnType="EventHandler" output="false"
  hint="Used by the framework for initialization. Do not override.">
  <cfargument name="access" type="string" required="true" />

  <cfset setAccess(arguments.access) />

  <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

Inherits from: framework.CommandLoaderBase

Manages registered EventHandlers for the framework.

Method Summary

public EventManager	init(AppManager appManager, [any parentEventManager=""]) Initialization function called by the framework.
public void	addEventHandler(string eventName, EventHandler eventHandler, [boolean overrideCheck="false"]) Registers an EventHandler by name.
public void	configure() Configures the EventManager and checks if default and exception are defined as required.
public Event	createEvent(string moduleName, string eventName, [struct eventArgs="#StructNew()#"], [string requestName=""], [string requestModuleName=""], [string eventType="MachII.framework.Event"]) Creates an Event instance.
public AppManager	getAppManager()
public EventHandler	getEventHandler(string eventName, [string moduleName=""]) Returns the EventHandler for the named Event.
public array	getEventNames() Returns an array of event-handler names.
public any	getParent()

Method Summary

	Sets the parent EventManager instance this EventManager belongs to. It will return empty string if no parent is defined.
public boolean	isEventDefined(string eventName, [boolean checkParent="false"], [string moduleName=""]) Returns true if an EventHandler for the named Event is defined; otherwise false.
public boolean	isEventPublic(string eventName, [boolean checkParent="false"]) Returns true if the EventHandler for the named Event is publicly accessible; otherwise false.
public void	loadXml(string configXML, [boolean override="false"]) Loads xml for the manager.
public void	removeEvent(string eventName) Removes an event-handler. Does NOT remove from parent.
public void	setAppManager(AppManager appManager)
public void	setParent(EventManager parentEventManager) Returns the parent EventManager instance this EventManager belongs to.

Methods inherited from framework.CommandLoaderBase: createCommand , setupEventArg , setupAnnounce , setupEventBean , setupViewPage , setupFilter , setupRedirect , setupDefault , setupEventMapping , setupNotify , setupExecute

Method Detail**addEventHandler**

```
public void addEventHandler( string eventName, EventHandler eventHandler, [boolean overrideCheck="false"] )
```

Registers an EventHandler by name.

Parameters:

string eventName
 EventHandler eventHandler
 [boolean overrideCheck="false"]

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" />
  <cfargument name="overrideCheck" type="boolean" required="false" default="false" />

  <cfif NOT arguments.overrideCheck AND isEventDefined(arguments.eventName)>
    <cfthrow type="MachII.framework.EventHandlerAlreadyDefined"
      message="An EventHandler with name '#arguments.eventName#' is already registered." />
  </cfif>
  <cfset variables.handlers[arguments.eventName] = arguments.eventHandler />
</cffunction>
```

configure

public void configure()

Configures the EventManager and checks if default and exception are defined as required.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void" output="false"
  hint="Configures the EventManager and checks if default and exception are defined as required.">

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

  <cfif NOT IsObject(getAppManager().getParent())>
    <cfset defaultEvent = getAppManager().getPropertyManager().getProperty("defaultEvent") />
  </cfif>
  <cfif NOT isEventDefined(defaultEvent, false)>
```

```

        <cfthrow type="MachII.framework.noDefaultEvent"
            message="A default event named '#defaultEvent#' has been not defined, but is required. P
    </cfif>
    <cfset exceptionEvent = getAppManager().getPropertyManager().getProperty("exceptionEvent") />
    <cfif NOT isEventDefined(exceptionEvent, false)>
        <cfthrow type="MachII.framework.noExceptionEvent"
            message="A exception event named '#exceptionEvent#' has been not defined, but is required
    </cfif>

    <cfelse>
        <cfif getAppManager().getPropertyManager().isPropertyDefined("defaultEvent")>
            <cfset defaultEvent = getAppManager().getPropertyManager().getProperty("defaultEvent") />
            <cfif NOT isEventDefined(defaultEvent, true)>
                <cfthrow type="MachII.framework.noDefaultEvent"
                    message="A default event named '#defaultEvent#' has been defined for this module
            </cfif>
        </cfif>
        <cfif getAppManager().getPropertyManager().isPropertyDefined("exceptionEvent")>
            <cfset exceptionEvent = getAppManager().getPropertyManager().getProperty("exceptionEvent") />
            <cfif NOT isEventDefined(exceptionEvent, true)>
                <cfthrow type="MachII.framework.noExceptionEvent"
                    message="A exception event named '#exceptionEvent#' has been defined for this mo
            </cfif>
        </cfif>
    </cfif>
</cffunction>

```

createEvent

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

Creates an Event instance.

Parameters:

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

```
[string requestModuleName=""]
[string eventType="MachII.framework.Event"]
```

Code:

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

    <cfset var event = "" />

    <cfif isEventDefined(arguments.eventName, true, arguments.moduleName)>
        <cfset event = CreateObject("component", arguments.eventType).init(arguments.eventName, arguments.eventA
    <cfelse>
        <cfthrow type="MachII.framework.EventHandlerNotDefined"
            message="EventHandler for event '#arguments.eventName#' in module '#arguments.moduleName#' is not
    </cfif>

    <cfreturn event />
</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, [string moduleName=""] )
```

Returns the EventHandler for the named Event.

Parameters:

```
string eventName  
[string moduleName=""]
```

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." />
  <cfargument name="moduleName" type="string" required="false" default="" />

  <cfset var moduleEventManager = 0 />
  <cfset var moduleManager = 0 />

  <cfif arguments.moduleName neq "">
    <cfif NOT isObject(getAppManager().getParent())>
      <cfset moduleManager = getAppManager().getModuleManager() />
    <cfelse>
      <cfset moduleManager = getAppManager().getParent().getModuleManager() />
    </cfif>
    <cfset moduleEventManager = moduleManager.getModule(arguments.moduleName).getModuleAppManager().getEventManager() />
  <cfreturn moduleEventManager.getEventHandler(arguments.eventName) />
<cfelseif isEventDefined(arguments.eventName)>
  <cfreturn variables.handlers[arguments.eventName] />
<cfelseif isObject(getParent()) AND getParent().isEventDefined(arguments.eventName)>
  <cfreturn getParent().getEventHandler(arguments.eventName) />
<cfelse>
  <cfthrow type="MachII.framework.EventHandlerNotDefined"
    message="EventHandler for event '#arguments.eventName#' is not defined." />
</cfif>
</cffunction>
```

getEventNames

public array getEventNames()

Returns an array of event-handler names.

Parameters:

Code:

```
<cffunction name="getEventNames" access="public" returntype="array" output="false"
    hint="Returns an array of event-handler names.">
    <cfreturn StructKeyArray(variables.handlers) />
</cffunction>
```

getParent

public any getParent()

Sets the parent EventManager instance this EventManager belongs to. It will return empty string if no parent is defined.

Parameters:

Code:

```
<cffunction name="getParent" access="public" returntype="any" output="false"
    hint="Sets the parent EventManager instance this EventManager belongs to. It will return empty string if no parent"
    <cfreturn variables.parentEventManager />
</cffunction>
```

init

public EventManager init(AppManager appManager, [any parentEventManager=""])

Initialization function called by the framework.

Parameters:

AppManager appManager

[any parentEventManager=""]

Code:

```
<cffunction name="init" access="public" returntype="EventManager" output="false"
  hint="Initialization function called by the framework.">
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
  <cfargument name="parentEventManager" type="any" required="false" default=""
    hint="Optional argument for a parent event manager. If there isn't one default to empty string." />

  <cfset setAppManager(arguments.appManager) />

  <cfif isObject(arguments.parentEventManager)>
    <cfset setParent(arguments.parentEventManager) />
  </cfif>

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

isEventDefined

public boolean isEventDefined(string eventName, [boolean checkParent="false"], [string moduleName=""])

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

Parameters:

string eventName

[boolean checkParent="false"]

[string moduleName=""]

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." />
  <cfargument name="checkParent" type="boolean" required="false" default="false"
    hint="Allows you to check the parent to see if the event is in there" />
  <cfargument name="moduleName" type="string" required="false" default=""
    hint="Allows you to check in a specific module for an event" />
```

```

<cfset var moduleManager = "" />
<cfset var moduleEventManager = "" />

<cfif arguments.moduleName neq "">
  <cfif NOT isObject(getAppManager().getParent())>
    <cfset moduleManager = getAppManager().getModuleManager() />
  <cfelse>
    <cfset moduleManager = getAppManager().getParent().getModuleManager() />
  </cfif>
  <cfif moduleManager.isModuleDefined(arguments.moduleName)>
    <cfset moduleEventManager = moduleManager.getModule(arguments.moduleName).getModuleAppManager().getModuleEventManager() />
    <cfif moduleEventManager.isEventDefined(arguments.eventName, true)>
      <cfreturn true />
    <cfelse>
      <cfreturn false />
    </cfif>
  <cfelse>
    <cfreturn false />
  </cfif>
</cfif>
<cfif StructKeyExists(variables.handlers, arguments.eventName)>
  <cfreturn true />
<cfelseif arguments.checkParent AND isObject(getParent())>
  <cfreturn getParent().isEventDefined(arguments.eventName, false, arguments.moduleName) />
<cfelse>
  <cfreturn false />
</cfif>
</cfif>
</cffunction>

```

isEventPublic

public boolean isEventPublic(string eventName, [boolean checkParent="false"])

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

Parameters:

string eventName
[boolean checkParent="false"]

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" />
    <cfargument name="checkParent" type="boolean" required="false" default="false" />

    <cfset var eventHandler = "" />

    <cfif isEventDefined(arguments.eventName)>
        <cfset eventHandler = getEventHandler(arguments.eventName) />
    <cfelseif arguments.checkParent AND isObject(getParent()) AND getParent().isEventDefined(arguments.eventName)>
        <cfset eventHandler = getParent().getEventHandler(arguments.eventName) />
    <cfelse>
        <cfreturn false />
    </cfif>

    <cfreturn eventHandler.getAccess() EQ "public" />
</cffunction>
```

loadXml

```
public void loadXml( string configXML, [boolean override="false"] )
```

Loads xml for the manager.

Parameters:

```
string configXML
[boolean override="false"]
```

Code:

```
<cffunction name="loadXml" access="public" returntype="void" output="false"
    hint="Loads xml for the manager.">
    <cfargument name="configXML" type="string" required="true" />
    <cfargument name="override" type="boolean" required="false" default="false" />

    <cfset var commandNode = "" />
    <cfset var eventNodes = "" />
```

```
<cfset var eventHandler = "" />
<cfset var eventAccess = "" />
<cfset var eventName = "" />
<cfset var command = "" />
<cfset var hasParent = isObject(getParent()) />
<cfset var mapping = "" />
<cfset var i = 0 />
<cfset var j = 0 />

<cfif NOT arguments.override>
    <cfset eventNodes = XMLSearch(arguments.configXML, "mach-ii/event-handlers/event-handler") />
<cfelse>
    <cfset eventNodes = XMLSearch(arguments.configXML, "../event-handlers/event-handler") />
</cfif>

<cfloop from="1" to="#ArrayLen(eventNodes)#" index="i">
    <cfset eventName = eventNodes[i].xmlAttributes["event"] />

    <cfif hasParent AND arguments.override AND StructKeyExists(eventNodes[i].xmlAttributes, "overrideAction")
        <cfif eventNodes[i].xmlAttributes["overrideAction"] EQ "useParent">
            <cfset removeEvent(eventName) />
        <cfelseif eventNodes[i].xmlAttributes["overrideAction"] EQ "addFromParent">
            <cfif StructKeyExists(eventNodes[i].xmlAttributes, "mapping")>
                <cfset mapping = eventNodes[i].xmlAttributes["mapping"] />
            <cfelse>
                <cfset mapping = eventName />
            </cfif>

            <cfif NOT getParent().isEventDefined(mapping)>
                <cfthrow type="MachII.framework.overrideEventHandlerNotDefined"
                    message="An event-handler named '#mapping#' cannot be found in the parent" />
            </cfif>

            <cfset addEventHandler(eventName, getParent().getEventHandler(mapping), arguments.override) />
        </cfif>

    <cfelse>
        <cfif StructKeyExists(eventNodes[i].xmlAttributes, "access")>
            <cfset eventAccess = eventNodes[i].xmlAttributes["access"] />
        </cfif>
    </cfif>
</cfloop>
```

```

        <cfelse>
            <cfset eventAccess = "public" />
        </cfif>

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

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

        <cfset addEventHandler(eventName, eventHandler, arguments.override) />
    </cfif>
</cfloop>
</cffunction>

```

removeEvent

```
public void removeEvent( string eventName )
```

Removes an event-handler. Does NOT remove from parent.

Parameters:

string eventName

Code:

```

<cffunction name="removeEvent" access="public" returnType="void" output="false"
    hint="Removes an event-handler. Does NOT remove from parent.">
    <cfargument name="eventName" type="string" required="true"
        hint="The name of the Event to handle." />
    <cfset StructDelete(variables.handlers, arguments.eventName, false) />
</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>
```

setParent

public void setParent(EventManager parentEventManager)

Returns the parent EventManager instance this EventManager belongs to.

Parameters:

EventManager parentEventManager

Code:

```
<cffunction name="setParent" access="public" returntype="void" output="false"
    hint="Returns the parent EventManager instance this EventManager belongs to.">
    <cfargument name="parentEventManager" type="MachII.framework.EventManager" required="true" />
    <cfset variables.parentEventManager = arguments.parentEventManager />
</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 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: setParameters , hasParameter , buildUrlToModule , isParameterDefined , buildUrl , announceEventInModule , setAppManager , getParameter , getProperty , getParameters , setProperty , getPropertyManager , bindValue , configure , getAppManager , setParameter , announceEvent

Method Detail

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(AppManager appManager, [any parentListenerManager=""])	Initialization function called by the framework.
public void	addListener(string listenerName, Listener listener, [boolean overrideCheck="false"])	Registers a Listener with the specified name.
public void	configure()	Configures each of the registered Listeners and its' invoker.
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 any	getParent()	Sets the parent ListenerManager instance this ListenerManager belongs to. It will return empty string if no parent is defined.
public boolean	isListenerDefined(string listenerName)	Returns true if a Listener is registered with the specified name. Does NOT check parent.

Method Summary

public void	loadXml(string configXML, [boolean override="false"])	Loads xml into the manager.
public void	removeListener(string listenerName)	Removes a listener. Does NOT remove from a parent.
public void	setAppManager(AppManager appManager)	Returns the AppManager instance this ListenerManager belongs to.
public void	setParent(ListenerManager parentListenerManager)	Returns the parent ListenerManager instance this ListenerManager belongs to.

Method Detail**addListener**

```
public void addListener( string listenerName, Listener listener, [boolean overrideCheck="false"] )
```

Registers a Listener with the specified name.

Parameters:

```
string listenerName
Listener listener
[boolean overrideCheck="false"]
```

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" />
    <cfargument name="overrideCheck" type="boolean" required="false" default="false" />
```

```
<cfif NOT arguments.overrideCheck AND 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 and its' invoker.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void"
  hint="Configures each of the registered Listeners and its' invoker.">
  <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] />
    <cfelseif isObject(getParent()) AND getParent().isListenerDefined(arguments.listenerName)>
        <cfreturn getParent().getListener(arguments.listenerName) />
    <cfelse>
        <cfthrow type="MachII.framework.ListenerNotDefined"
            message="Listener with name '#arguments.listenerName#' is not defined. Listeners: '#ArrayToList('
    </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>
```

getParent

public any getParent()

Sets the parent ListenerManager instance this ListenerManager belongs to. It will return empty string if no parent is defined.

Parameters:

Code:

```
<cffunction name="getParent" access="public" returntype="any" output="false"
    hint="Sets the parent ListenerManager instance this ListenerManager belongs to. It will return empty string if no
    parent is defined.">
    <cfreturn variables.parentListenerManager />
</cffunction>
```

init

public ListenerManager init(AppManager appManager, [any parentListenerManager=""])

Initialization function called by the framework.

Parameters:

AppManager appManager
[any parentListenerManager=""]

Code:

```
<cffunction name="init" access="public" returntype="ListenerManager" output="false"
  hint="Initialization function called by the framework.">
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
  <cfargument name="parentListenerManager" type="any" required="false" default=""
    hint="Optional argument for a parent listener manager. If there isn't one default to empty string." />

  <cfset setAppManager(arguments.appManager) />
  <cfset variables.utils = getAppManager().getUtils() />

  <cfif isObject(arguments.parentListenerManager)>
    <cfset setParent(arguments.parentListenerManager) />
  </cfif>

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

isListenerDefined

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

Returns true if a Listener is registered with the specified name. Does NOT check parent.

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. Does NOT check parent.">
  <cfargument name="listenerName" type="string" required="true" />
  <cfreturn StructKeyExists(variables.listeners, arguments.listenerName) />
</cffunction>
```

loadXml

```
public void loadXml( string configXML, [boolean override="false"] )
```

Loads xml into the manager.

Parameters:

```
string configXML  
[boolean override="false"]
```

Code:

```
<cffunction name="loadXml" access="public" returntype="void" output="false"  
    hint="Loads xml into the manager.">  
    <cfargument name="configXML" type="string" required="true" />  
    <cfargument name="override" type="boolean" required="false" default="false" />  
  
    <cfset var listenerNodes = "" />  
    <cfset var listenerParams = "" />  
    <cfset var listenerName = "" />  
    <cfset var listenerType = "" />  
    <cfset var paramNodes = "" />  
    <cfset var paramName = "" />  
    <cfset var paramValue = "" />  
    <cfset var invokerType = "" />  
    <cfset var invoker = "" />  
    <cfset var listener = "" />  
    <cfset var hasParent = isObject(getParent()) />  
    <cfset var mapping = "" />  
    <cfset var i = 0 />  
    <cfset var j = 0 />  
  
    <cfif NOT arguments.override>  
        <cfset listenerNodes = XMLSearch(arguments.configXML, "mach-ii/listeners/listener") />  
    <cfelse>  
        <cfset listenerNodes = XMLSearch(arguments.configXML, "./listeners/listener") />  
    </cfif>  
  
    <cfloop from="1" to="#ArrayLen(listenerNodes)#" index="i">  
        <cfset listenerName = listenerNodes[i].xmlAttributes["name"] />
```

```

<cfif hasParent AND arguments.override AND StructKeyExists(listenerNodes[i].xmlAttributes, "overrideAction")
  <cfif listenerNodes[i].xmlAttributes["overrideAction"] EQ "useParent">
    <cfset removeListener(listenerName) />
  <cfelseif listenerNodes[i].xmlAttributes["overrideAction"] EQ "addFromParent">
    <cfif StructKeyExists(listenerNodes[i].xmlAttributes, "mapping")>
      <cfset mapping = listenerNodes[i].xmlAttributes["mapping"] />
    <cfelse>
      <cfset mapping = listenerName />
    </cfif>

    <cfif NOT getParent().isListenerDefined(mapping)>
      <cfthrow type="MachII.framework.overrideListenerNotDefined"
        message="An listener named '#mapping#' cannot be found in the parent listener list" />
    </cfif>

    <cfset addListener(listenerName, getParent().getListener(mapping), arguments.override) />
  </cfif>
<cfelse>
  <cfset listenerType = listenerNodes[i].xmlAttributes["type"] />

  <cfset listenerParams = StructNew() />

  <cfif StructKeyExists(listenerNodes[i], "parameters")>
    <cfset paramNodes = listenerNodes[i].parameters.xmlChildren />
    <cfloop from="1" to="#ArrayLen(paramNodes)#" index="j">
      <cfset paramName = paramNodes[j].xmlAttributes["name"] />
      <cftry>
        <cfset paramValue = variables.utils.recurseComplexValues(paramNodes[j]) />
      <cfcatch type="any">
        <cfthrow type="MachII.framework.InvalidParameterXml"
          message="Xml parsing error for the parameter named '#paramName#' with value '#paramValue#'" />
      </cfcatch>
    </cftry>
    <cfset listenerParams[paramName] = paramValue />
  </cfloop>
</cfif>

  <cfset listener = CreateObject("component", listenerType).init(getAppManager(), listenerParams) />

```

```

        <cfif StructKeyExists(listenerNodes[i], "invoker")>
            <cfset invokerType = listenerNodes[i].invoker.xmlAttributes["type"] />
            <cfset invoker = CreateObject("component", invokerType).init() />
        <cfelse>
            <cfset invoker = CreateObject("component", "MachII.framework.invokers.EventInvoker").init() />
        </cfif>

        <cfset listener.setInvoker(invoker) />
        <cfset addListener(listenerName, listener, arguments.override) />
    </cfif>
</cfloop>
</cffunction>

```

removeListener

```
public void removeListener( string listenerName )
```

Removes a listener. Does NOT remove from a parent.

Parameters:

string listenerName

Code:

```

<cffunction name="removeListener" access="public" returnType="void" output="false"
    hint="Removes a listener. Does NOT remove from a parent.">
    <cfargument name="listenerName" type="string" required="true" />
    <cfset StructDelete(variables.listeners, arguments.listenerName, false) />
</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>
```

setParent

```
public void setParent( ListenerManager parentListenerManager )
```

Returns the parent ListenerManager instance this ListenerManager belongs to.

Parameters:

ListenerManager parentListenerManager

Code:

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

Module

Package: MachII.framework

Holds a Module.

Method Summary

public Module	init(AppManager appManager, string moduleName, string file, any overrideXml)	Used by the framework for initialization. Do not override.
public void	configure(string configDtdPath, [boolean validateXml="false"])	
public AppManager	getAppManager()	Sets the AppManager instance this ModuleManager belongs to.
public any	getDtdPath()	
public any	getFile()	Gets the file to use when setting up the module's AppManager
public AppManager	getModuleAppManager()	Sets the ModuleAppManager instance this ModuleManager belongs to.
public any	getModuleName()	Gets the module name
public any	getOverrideXml()	Gets the override Xml for this module.
public void	reloadModuleConfig([boolean validateXml="false"])	
public void	setAppManager(AppManager appManager)	Returns the AppManager instance this Module belongs to.

Method Summary

public void	setDtdPath(string dtdPath)
public void	setFile(string file)
	Sets the path to the module Mach II config file
public void	setModuleAppManager(AppManager moduleAppManager)
	Returns the ModuleAppManager instance this ModuleManager belongs to.
public void	setModuleName(string moduleName)
	Sets the name of the module
public void	setOverrideXml(any overrideXml)
	Sets the override Xml for this module.
public boolean	shouldReloadConfig()

Method Detail**configure**

```
public void configure( string configDtdPath, [boolean validateXml="false"] )
```

Parameters:

```
string configDtdPath
[boolean validateXml="false"]
```

Code:

```
<cffunction name="configure" access="public" returntype="void" output="false">
  <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." />
        <cfset var appLoader = CreateObject("component", "MachII.framework.AppLoader").init(
            getFile(), arguments.configDtdPath, getAppManager().getAppKey(), arguments.validateXML, getAppManager()) />
        <cfset var moduleAppManager = appLoader.getAppManager() />

        <cfset setDtdPath(arguments.configDtdPath) />
        <cfset moduleAppManager.setAppLoader(appLoader) />
        <cfset setModuleAppManager(moduleAppManager) />
    </cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Sets the AppManager instance this ModuleManager belongs to.

Parameters:

Code:

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

getDtdPath

```
public any getDtdPath( )
```

Parameters:

Code:

```
<cffunction name="getDtdPath" access="public" type="string" output="false">
    <cfreturn variables.dtdPath />
</cffunction>
```

```
</cffunction>
```

getFile

```
public any getFile( )
```

Gets the file to use when setting up the module's AppManager

Parameters:

Code:

```
<cffunction name="getFile" access="public" type="string" output="false"
    hint="Gets the file to use when setting up the module's AppManager">
    <cfreturn variables.file />
</cffunction>
```

getModuleAppManager

```
public AppManager getModuleAppManager( )
```

Sets the ModuleAppManager instance this ModuleManager belongs to.

Parameters:

Code:

```
<cffunction name="getModuleAppManager" access="public" returntype="MachII.framework.AppManager" output="false"
    hint="Sets the ModuleAppManager instance this ModuleManager belongs to.">
    <cfreturn variables.moduleAppManager />
</cffunction>
```

getModuleName

```
public any getModuleName( )
```

Gets the module name

Parameters:

Code:

```
<cffunction name="getModuleName" access="public" type="string" output="false"
    hint="Gets the module name">
    <cfreturn variables.moduleName />
</cffunction>
```

getOverrideXml

public any getOverrideXml()

Gets the override Xml for this module.

Parameters:

Code:

```
<cffunction name="getOverrideXml" access="public" type="any" output="false"
    hint="Gets the override Xml for this module.">
    <cfreturn variables.overrideXml />
</cffunction>
```

init

public Module init(AppManager appManager, string moduleName, string file, any overrideXml)

Used by the framework for initialization. Do not override.

Parameters:

AppManager appManager
string moduleName
string file

any overrideXml

Code:

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

  <cfset setFile(arguments.file) />
  <cfset setModuleName(arguments.moduleName) />
  <cfset setAppManager(arguments.appManager) />
  <cfset setOverrideXml(arguments.overrideXml) />

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

reloadModuleConfig

public void reloadModuleConfig([boolean validateXml="false"])

Parameters:

[boolean validateXml="false"]

Code:

```
<cffunction name="reloadModuleConfig" access="public" returntype="void" output="false">
  <cfargument name="validateXml" type="boolean" required="false" default="false"
    hint="Should the XML be validated before parsing." />
  <cfset var appLoader = CreateObject("component", "MachII.framework.AppLoader").init(
    getFile(), getDtdPath(), getAppManager().getAppKey(), arguments.validateXML, getAppManager(), get
  <cfset var moduleAppManager = appLoader.getAppManager() />

  <cfset moduleAppManager.setAppLoader(appLoader) />
  <cfset setModuleAppManager(moduleAppManager) />
</cffunction>
```

setAppManager

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

Returns the AppManager instance this Module belongs to.

Parameters:

AppManager appManager

Code:

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

setFile

```
public void setFile( string file )
```

Sets the path to the module Mach II config file

Parameters:

string file

Code:

```
<cffunction name="setFile" access="public" returntype="void" output="false"
    hint="Sets the path to the module Mach II config file">
    <cfargument name="file" type="string" required="true" />
    <cfset variables.file = arguments.file />
</cffunction>
```

setModuleAppManager

```
public void setModuleAppManager( AppManager moduleAppManager )
```

Returns the ModuleAppManager instance this ModuleManager belongs to.

Parameters:

AppManager moduleAppManager

Code:

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

setModuleName

```
public void setModuleName( string moduleName )
```

Sets the name of the module

Parameters:

string moduleName

Code:

```
<cffunction name="setModuleName" access="public" returntype="void" output="false"
    hint="Sets the name of the module">
    <cfargument name="moduleName" type="string" required="true" />
    <cfset variables.moduleName = arguments.moduleName />
</cffunction>
```

setOverrideXml

public void setOverrideXml(any overrideXml)

Sets the override Xml for this module.

Parameters:

any overrideXml

Code:

```
<cffunction name="setOverrideXml" access="public" returntype="void" output="false"
    hint="Sets the override Xml for this module.">
    <cfargument name="overrideXml" type="any" required="true" />
    <cfset variables.overrideXml = arguments.overrideXml />
</cffunction>
```

shouldReloadConfig

public boolean shouldReloadConfig()

Parameters:

Code:

```
<cffunction name="shouldReloadConfig" access="public" returntype="boolean" output="false">  
    <cfreturn getModuleAppManager().getAppLoader().shouldReloadConfig() />  
</cffunction>
```

ModuleManager

Package: MachII.framework

Manages registered modules for the framework instance.

Method Summary

public ModuleManager	init(AppManager appManager, string baseConfigFileDirectory, string configDtdPath, [boolean validateXml="false"]) Initialization function called by the framework.
public void	addModule(string moduleName, Module module, [boolean override="false"]) Registers a module with the specified name.
public void	configure() Configures each of the registered modules.
public AppManager	getAppManager() Sets the AppManager instance this ModuleManager belongs to.
public string	getBaseConfigFileDirectory()
public string	getBaseName()
public string	getDtdPath()
public Module	getModule(string moduleName) Gets a module with the specified name.
public array	getModuleNames() Returns an array of module names.
public struct	getModules() Returns a struct of all registered modules.

Method Summary

public boolean	getValidateXML()
public boolean	isModuleDefined(string moduleName)
	Returns true if a module is registered with the specified name.
public void	loadXml(string configXml, [boolean override="false"])
	Loads xml for the manager
public void	setAppManager(AppManager appManager)
	Returns the AppManager instance this ModuleManager belongs to.
public void	setBaseConfigFileDirectory(string baseConfigFileDirectory)
public void	setBaseName(string baseName)
public void	setDtdPath(string dtdPath)
public void	setValidateXML(string validateXML)

Method Detail**addModule**

```
public void addModule( string moduleName, Module module, [boolean override="false"] )
```

Registers a module with the specified name.

Parameters:

string moduleName

Module module

[boolean override="false"]

Code:

```
<cffunction name="addModule" access="public" returntype="void" output="false"
```

```
hint="Registers a module with the specified name.">
<cfargument name="moduleName" type="string" required="true" />
<cfargument name="module" type="MachII.framework.Module" required="true" />
<cfargument name="override" type="boolean" required="false" default="false" />

<cfif NOT arguments.override AND isModuleDefined(arguments.moduleName)>
    <cfthrow type="MachII.framework.ModuleAlreadyDefined"
        message="A Module with name '#arguments.moduleName#' is already registered." />
<cfelse>
    <cfset variables.modules[arguments.moduleName] = arguments.module />
</cfif>
</cffunction>
```

configure

```
public void configure( )
```

Configures each of the registered modules.

Parameters:

Code:

```
<cffunction name="configure" access="public" returntype="void" output="false"
    hint="Configures each of the registered modules.">
    <cfset var key = "" />
    <cfloop collection="#variables.modules#" item="key">
        <cfset getModule(key).configure(getDtdPath(), getValidateXML()) />
    </cfloop>
</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Sets the AppManager instance this ModuleManager belongs to.

Parameters:

Code:

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

getBaseConfigFileDirectory

```
public string getBaseConfigFileDirectory( )
```

Parameters:

Code:

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

getBaseName

```
public string getBaseName( )
```

Parameters:

Code:

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

getDtdPath

```
public string getDtdPath( )
```

Parameters:

Code:

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

getModule

```
public Module getModule( string moduleName )
```

Gets a module with the specified name.

Parameters:

string moduleName

Code:

```
<cffunction name="getModule" access="public" returntype="MachII.framework.Module" output="false"
    hint="Gets a module with the specified name.">
    <cfargument name="moduleName" type="string" required="true" />

    <cfif isModuleDefined(arguments.moduleName)>
        <cfreturn variables.modules[arguments.moduleName] />
    <cfelse>
        <cfthrow type="MachII.framework.ModuleNotDefined"
            message="Module with name '#arguments.moduleName#' is not defined." />
    </cfif>
</cffunction>
```

getModuleNames

public array getModuleNames()

Returns an array of module names.

Parameters:

Code:

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

getModules

public struct getModules()

Returns a struct of all registered modules.

Parameters:

Code:

```
<cffunction name="getModules" access="public" returntype="struct" output="false"
    hint="Returns a struct of all registered modules.">
    <cfreturn variables.modules />
</cffunction>
```

getValidateXML

public boolean getValidateXML()

Parameters:

Code:

```
<cffunction name="getValidateXML" access="public" returntype="boolean" output="false">
    <cfreturn variables.validateXML />
</cffunction>
```

init

```
public ModuleManager init( AppManager appManager, string baseConfigFileDirectory, string configDtdPath, [boolean validateXml="false"] )
```

Initialization function called by the framework.

Parameters:

```
AppManager appManager
string baseConfigFileDirectory
string configDtdPath
[boolean validateXml="false"]
```

Code:

```
<cffunction name="init" access="public" returntype="ModuleManager" output="false"
    hint="Initialization function called by the framework.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfargument name="baseConfigFileDirectory" type="string" required="true"
        hint="The directory of the base config file. Required for relative path support resolution." />
    <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." />

    <cfset setAppManager(arguments.appManager) />
    <cfset setBaseConfigFileDirectory(arguments.baseConfigFileDirectory) />
    <cfset setDtdPath(arguments.configDtdPath) />
    <cfset setValidateXml(arguments.validateXml) />

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

isModuleDefined

```
public boolean isModuleDefined( string moduleName )
```

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

Parameters:

string moduleName

Code:

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

loadXml

```
public void loadXml( string configXml, [boolean override="false"] )
```

Loads xml for the manager

Parameters:

string configXml

[boolean override="false"]

Code:

```
<cffunction name="loadXml" access="public" returntype="void" output="false"
    hint="Loads xml for the manager">
    <cfargument name="configXml" type="string" required="true" />
    <cfargument name="override" type="boolean" required="false" default="false" />

    <cfset var moduleNodes = "" />
    <cfset var modulesNode = "" />
    <cfset var modulesNodes = "" />
    <cfset var name = "" />
    <cfset var file = "" />
    <cfset var module = "" />
```

```
<cfset var overrideXml = "" />
<cfset var baseName = "" />
<cfset var i = 0 />

<cfif NOT arguments.override>
    <cfset modulesNode = XMLSearch(arguments.configXML, "mach-ii/modules") />
<cfelse>
    <cfset modulesNode = XMLSearch(arguments.configXML, "./modules") />
</cfif>
<cfif arrayLen(modulesNode) eq 1>
    <cfset modulesNode = modulesNode[1]>
    <cfif structKeyExists(modulesNode[1].xmlAttributes, "baseName")>
        <cfset baseName = modulesNode[1].xmlAttributes["baseName"] />
    </cfif>
    <cfset setBaseName(baseName) />
</cfif>

<cfif NOT arguments.override>
    <cfset moduleNodes = XMLSearch(arguments.configXML, "mach-ii/modules/module") />
<cfelse>
    <cfset moduleNodes = XMLSearch(arguments.configXML, "./modules/module") />
</cfif>
<cfloop from="1" to="#ArrayLen(moduleNodes)#" index="i">
    <cfset name = moduleNodes[i].xmlAttributes["name"] />
    <cfset file = moduleNodes[i].xmlAttributes["file"] />

    <cfif Left(file, 1) IS ".">
        <cfset file = getAppManager().getUtils().expandRelativePath(getBaseConfigFileDirectory(), file) />
    <cfelse>
        <cfset file = ExpandPath(file) />
    </cfif>

    <cfif StructKeyExists(moduleNodes[i], "mach-ii")>
        <cfset overrideXml = moduleNodes[i]["mach-ii"] />
    <cfelse>
        <cfset overrideXml = "" />
    </cfif>

    <cfset module = CreateObject("component", "MachII.framework.Module").init(getAppManager(), name, file, ov
```

```
                <cfset addModule(name, module, arguments.override) />
            </cfloop>
        </cffunction>
```

setAppManager

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

Returns the AppManager instance this ModuleManager belongs to.

Parameters:

AppManager appManager

Code:

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

setBaseConfigFileDirectory

```
public void setBaseConfigFileDirectory( string baseConfigFileDirectory )
```

Parameters:

string baseConfigFileDirectory

Code:

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

```
</cffunction>
```

setBaseName

```
public void setBaseName( string baseName )
```

Parameters:

string baseName

Code:

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

setValidateXML

public void setValidateXML(string validateXML)

Parameters:

string validateXML

Code:

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

Method Summary

public void	preProcess(EventContext eventContext) 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: setParameters , hasParameter , buildUriToModule , isParameterDefined , buildUrl , announceEventInModule , setAppManager , getParameter , getProperty , getParameters , setProperty , getPropertyManager , bindValue , configure , getAppManager , setParameter , announceEvent

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 functionality." />
    <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
    </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 PluginManager	init(AppManager appManager, [any parentPluginManager=""])	Initialization function called by the framework.
public void	addPlugin(string pluginName, Plugin plugin, [boolean override="false"])	Registers a plugin with the specified name.
public void	configure()	Configures each of the registered Plugins.
private array	findPluginPoints(Plugin plugin)	Finds the registered plugin points in a plugin.
private void	gatherPluginMetaData(struct metadata, struct points)	Gathers meta data about a plugin.
public AppManager	getAppManager()	Returns the AppManager instance this PluginManager belongs to.
public any	getParent()	Sets the parent PluginManager instance this PluginManager belongs to. It will return empty string if no parent is defined.
public Plugin	getPlugin(string pluginName)	Gets a plugin with the specified name.

Method Summary

public array	getPluginNames() Returns an array of plugin names.
public string	getRunParent()
public void	handleException(EventContext eventContext, Exception exception) 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. Does NOT check parent.
public void	loadXml(string configXML, [boolean override="false"]) Loads xml into the manager.
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 Summary

public void	setParent(PluginManager parentPluginManager)
	Returns the parent PluginManager instance this PluginManager belongs to.
public void	setRunParent(string runParent)

Method Detail**addPlugin**

```
public void addPlugin( string pluginName, Plugin plugin, [boolean override="false"] )
```

Registers a plugin with the specified name.

Parameters:

```
string pluginName
Plugin plugin
[boolean override="false"]
```

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" />
    <cfargument name="override" type="boolean" required="false" default="false" />

    <cfset var i = 0 />
    <cfset var pointName = 0 />
    <cfset var temp = "" />
    <cfset var pluginRegisteredPoints = findPluginPoints(arguments.plugin) />

    <cfif NOT arguments.override AND isPluginDefined(arguments.pluginName)>
        <cfthrow type="MachII.framework.PluginAlreadyDefined"
            message="A Plugin with name '#arguments.pluginName#' is already registered." />
    </cfif>
</cffunction>
```

```

<cfelseif arguments.override AND isPluginDefined(arguments.pluginName)>
  <cfset variables.plugins[arguments.pluginName] = arguments.plugin />
  <cfset variables.pluginArray[variables.pluginArrayPosition[arguments.pluginName]] = arguments.plugin />

  <cfloop from="1" to="#ArrayLen(pluginRegisteredPoints)#" index="i">
    <cfset pointName = pluginRegisteredPoints[i] />
    <cfif StructKeyExists(variables, pointName & "Plugins")>
      <cfif ListFindNoCase(variables[pointName & "PluginsPosition"], arguments.pluginName)>
        <cfset variables[pointName & "Plugins"][ListFindNoCase(variables[pointName & "Plu
      <cfelse>
        <cfset ArrayInsertAt(variables[pointName & "Plugins"], variables.pluginArrayPosi
        <cfif ListLen(variables[pointName & "PluginsPosition"]) GT 1>
          <cfset variables[pointName & "PluginsPosition"] = ListInsertAt(variables
        <cfelse>
          <cfset variables[pointName & "PluginsPosition"] = ListAppend(variables[p
        </cfif>
      </cfif>
      <cfset temp = ListAppend(temp, pointName) />
    </cfif>
  </cfloop>

  <cfloop from="1" to="#ArrayLen(variables.pluginPointArray)#" index="i">
    <cfset pointName = variables.pluginPointArray[i] />
    <cfif ListFindNoCase(variables[pointName & "PluginsPosition"], arguments.pluginName) AND NOT List
      <cfset ArrayDeleteAt(variables[pointName & "Plugins"], ListFindNoCase(variables[pointName
      <cfset ListDeleteAt(variables[pointName & "PluginsPosition"], ListFindNoCase(variables[p
    </cfif>
  </cfloop>
<cfelse>
  <cfset variables.plugins[arguments.pluginName] = arguments.plugin />

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

  <cfloop from="1" to="#ArrayLen(pluginRegisteredPoints)#" index="i">
    <cfset pointName = pluginRegisteredPoints[i] />
    <cfif StructKeyExists(variables, pointName & "Plugins")>
      <cfset ArrayAppend(variables[pointName & "Plugins"], arguments.plugin) />
      <cfset variables[pointName & "PluginsPosition"] = ListAppend(variables[pointName & "Plug
    </cfif>
  </cfloop>

```

```
        </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 from="1" to="#variables.nPlugins#" index="i">
        <cfset aPlugin = variables.pluginArray[i] />
        <cfset aPlugin.configure() />
    </cfloop>
</cffunction>
```

findPluginPoints

```
private array findPluginPoints( Plugin plugin )
```

Finds the registered plugin points in a plugin.

Parameters:

Plugin plugin

Code:

```
<cffunction name="findPluginPoints" access="private" returntype="array" 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 points = StructNew() />

    <cfset gatherPluginMetaData(md, points) />

    <cfreturn StructKeyArray(points) />
</cffunction>
```

gatherPluginMetaData

```
private void gatherPluginMetaData( struct metadata, struct points )
```

Gathers meta data about a plugin.

Parameters:

struct metadata
struct points

Code:

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

    <cfset var i = 0 />

    <cfif StructKeyExists(arguments.metadata, "functions")>
        <cfloop from="1" to="#ArrayLen(arguments.metadata.functions)#" index="i">
            <cfset StructInsert(arguments.points, arguments.metadata.functions[i].name, 1, true) />
        </cfloop>
    </cfif>

    <cfif StructKeyExists(arguments.metadata, "extends") and arguments.metadata.extends.name neq "MachII.framework.P
```

```
        <cfset gatherPluginMetaData(arguments.metadata.extends, arguments.points) />
    </cfif>
</cffunction>
```

getManager

```
public AppManager getManager( )
```

Returns the AppManager instance this PluginManager belongs to.

Parameters:

Code:

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

getParent

```
public any getParent( )
```

Sets the parent PluginManager instance this PluginManager belongs to. It will return empty string if no parent is defined.

Parameters:

Code:

```
<cffunction name="getParent" access="public" returnType="any" output="false"
    hint="Sets the parent PluginManager instance this PluginManager belongs to. It will return empty string if no parent is defined.">
    <cfreturn variables.parentPluginManager />
</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] />
  <cfelseif isObject(getParent()) AND getParent().isPluginDefined(arguments.pluginName)>
    <cfreturn getParent().getPlugin(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>
```

getRunParent

```
public string getRunParent( )
```

Parameters:

Code:

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

    <cfif getRunParent() eq "before">
        <cfif isObject(getParent())>
            <cfset getParent().handleException(arguments.eventContext, arguments.exception) />
        </cfif>
    </cfif>
</cffunction>
```

```

        </cfif>
    </cfif>

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

    <cfif getRunParent() eq "after" OR getRunParent() eq "">
        <cfif isObject(getParent())>
            <cfset getParent().handleException(arguments.eventContext, arguments.exception) />
        </cfif>
    </cfif>
</cffunction>

```

init

```
public PluginManager init( AppManager appManager, [any parentPluginManager=""] )
```

Initialization function called by the framework.

Parameters:

AppManager appManager
[any parentPluginManager=""]

Code:

```

<cffunction name="init" access="public" returntype="PluginManager" output="false"
    hint="Initialization function called by the framework.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfargument name="parentPluginManager" type="any" required="false" default=""
        hint="Optional argument for a parent plugin manager. If there isn't one default to empty string." />

    <cfset setAppManager(arguments.appManager) />
    <cfset variables.utils = getAppManager().getUtils() />

    <cfif isObject(arguments.parentPluginManager)>
        <cfset setParent(arguments.parentPluginManager) />
    </cfif>

```

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

isPluginDefined

```
public boolean isPluginDefined( string pluginName )
```

Returns true if a Plugin is registered with the specified name. Does NOT check parent.

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. Does NOT check parent.">
    <cfargument name="pluginName" type="string" required="true" />
    <cfreturn StructKeyExists(variables.plugins, arguments.pluginName) />
</cffunction>
```

loadXml

```
public void loadXml( string configXML, [boolean override="false"] )
```

Lloads xml into the manager.

Parameters:

```
string configXML
[boolean override="false"]
```

Code:

```
<cffunction name="loadXml" access="public" returntype="void" output="false"
    hint="Lloads xml into the manager.">
    <cfargument name="configXML" type="string" required="true" />
```

```
<cfargument name="override" type="boolean" required="false" default="false" />

<cfset var pluginNodes = 0 />
<cfset var paramNodes = 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 var i = 0 />
<cfset var j = 0 />

<cfif NOT arguments.override>
    <cfset pluginNodes = XMLSearch(arguments.configXML, "mach-ii/plugins/plugin") />
<cfelse>
    <cfset pluginNodes = XMLSearch(arguments.configXML, "./plugins") />
    <cfif arrayLen(pluginNodes) gt 0 AND structKeyExists(pluginNodes[1].xmlAttributes, "runParent")>
        <cfset setRunParent(pluginNodes[1].xmlAttributes["runParent"]) />
    </cfif>
    <cfset pluginNodes = XMLSearch(arguments.configXML, "./plugins/plugin") />
</cfif>
<cfloop index="i" from="1" to="#ArrayLen(pluginNodes)#">
    <cfset pluginName = pluginNodes[i].XmlAttributes["name"] />
    <cfset pluginType = pluginNodes[i].XmlAttributes["type"] />

    <cfset pluginParams = StructNew() />

    <cfif StructKeyExists(pluginNodes[i], "parameters")>
        <cfset paramNodes = pluginNodes[i].parameters.xmlChildren />
        <cfloop from="1" to="#ArrayLen(paramNodes)#" index="j">
            <cfset paramName = paramNodes[j].XmlAttributes["name"] />
            <cftry>
                <cfset paramValue = variables.utils.recurseComplexValues(paramNodes[j]) />
            <cfcatch type="any">
                <cfthrow type="MachII.framework.InvalidParameterXml"
                    message="Xml parsing error for the parameter named '#paramName#'
                />
            </cfcatch>
        </cftry>
        <cfset pluginParams[paramName] = paramValue />
    </cfloop>
</cfloop>
```

```

        </cfif>

        <cfset plugin = CreateObject("component", pluginType).init(getAppManager(), pluginParams) />
        <cfset addPlugin(pluginName, plugin, arguments.override) />
    </cfloop>
</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 />

        <cfif getRunParent() eq "before">
            <cfif isObject(getParent())>
                <cfset getParent().postEvent(arguments.eventContext) />
            </cfif>
        </cfif>

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

        <cfif getRunParent() eq "after" OR getRunParent() eq "">
            <cfif isObject(getParent())>
                <cfset getParent().postEvent(arguments.eventContext) />
            </cfif>
        </cfif>
    </cffunction>

```

```
</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 />

  <cfif getRunParent() eq "before">
    <cfif isObject(getParent())>
      <cfset getParent().postProcess(arguments.eventContext) />
    </cfif>
  </cfif>

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

  <cfif getRunParent() eq "after" OR getRunParent() eq "">
    <cfif isObject(getParent())>
      <cfset getParent().postProcess(arguments.eventContext) />
    </cfif>
  </cfif>
</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 />

  <cfif getRunParent() eq "before">
    <cfif isObject(getParent())>
      <cfset getParent().postView(arguments.eventContext) />
    </cfif>
  </cfif>

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

  <cfif getRunParent() eq "after" OR getRunParent() eq "">
    <cfif isObject(getParent())>
      <cfset getParent().postView(arguments.eventContext) />
    </cfif>
  </cfif>
</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
  </cfargument>
  <cfset var i = 0 />
  <cfif getRunParent() eq "before">
    <cfif isObject(getParent())>
      <cfset getParent().preEvent(arguments.eventContext) />
    </cfif>
  </cfif>
  <cfloop from="1" to="#ArrayLen(variables.preEventPlugins)#" index="i">
    <cfset variables.preEventPlugins[i].preEvent(arguments.eventContext) />
  </cfloop>
  <cfif getRunParent() eq "after" OR getRunParent() eq "">
    <cfif isObject(getParent())>
      <cfset getParent().preEvent(arguments.eventContext) />
    </cfif>
  </cfif>
</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 />

  <cfif getRunParent() eq "before">
    <cfif isObject(getParent())>
      <cfset getParent().preProcess(arguments.eventContext) />
    </cfif>
  </cfif>

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

  <cfif getRunParent() eq "after" OR getRunParent() eq "">
    <cfif isObject(getParent())>
      <cfset getParent().preProcess(arguments.eventContext) />
    </cfif>
  </cfif>
</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 />

<cfif getRunParent() eq "before">
    <cfif isObject(getParent())>
        <cfset getParent().preView(arguments.eventContext) />
    </cfif>
</cfif>

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

<cfif getRunParent() eq "after" OR getRunParent() eq "">
    <cfif isObject(getParent())>
        <cfset getParent().preView(arguments.eventContext) />
    </cfif>
</cfif>
</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>
```

setParent

```
public void setParent( PluginManager parentPluginManager )
```

Returns the parent PluginManager instance this PluginManager belongs to.

Parameters:

PluginManager parentPluginManager

Code:

```
<cffunction name="setParent" access="public" returntype="void" output="false"
    hint="Returns the parent PluginManager instance this PluginManager belongs to.">
    <cfargument name="parentPluginManager" type="MachII.framework.PluginManager" required="true" />
    <cfset variables.parentPluginManager = arguments.parentPluginManager />
</cffunction>
```

setRunParent

```
public void setRunParent( string runParent )
```

Parameters:

string runParent

Code:

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

Property

Package: MachII.framework

Inherits from: framework.BaseComponent

Base Property component.

Method Summary

public Property	init(AppManager appManager, [struct parameters="#StructNew()#"])
	Used by the framework for initialization. Do not override.

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

Method Detail

init

```
public Property 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="Property" 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 super.init(arguments.appManager, arguments.parameters) />

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

PropertyManager

Package: MachII.framework

Manages defined properties for the framework.

Method Summary

public PropertyManager	init(AppManager appManager, [any parentPropertyManager=""])
	Initialization function called by the framework.
public void	configure()
	Prepares the configurable properties for use.
public AppManager	getAppManager()
public array	getConfigurablePropertyNames()
	Returns an array of property names that we can call a configure() method on.
public any	getParent()
	Sets the parent PropertyManager instance this PropertyManager belongs to. It will return empty string if no parent is defined.
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)

Method Summary

	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. Does NOT check a parent.
public void	loadXml(string configXML, [boolean override="false"]) Loads xml into the manager.
public void	removeProperty(string propertyName) Removes a property from the current property manager. Does NOT remove from a parent.
public void	setAppManager(AppManager appManager)
public void	setParent(PropertyManager parentPropertyManager) Returns the parent PropertyManager instance this FilterManager belongs to.
public void	setProperty(string propertyName, any propertyValue) Sets the property value by name.

Method Detail**configure**

```
public void configure( )
```

Prepares the configurable properties for use.

Parameters:

Code:

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

```
<cfset var aConfigurableProperty = "" />
<cfset var i = 0 />

<cfloop from="1" to="#ArrayLen(variables.configurableProperties)#" index="i">
  <cfset aConfigurableProperty = getProperty(variables.configurableProperties[i]) />
  <cfset aConfigurableProperty.configure() />
</cfloop>
</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Parameters:

Code:

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

getConfigurablePropertyNames

```
public array getConfigurablePropertyNames( )
```

Returns an array of property names that we can call a configure() method on.

Parameters:

Code:

```
<cffunction name="getConfigurablePropertyNames" access="public" returntype="array" output="false"
  hint="Returns an array of property names that we can call a configure() method on.">
  <cfreturn variables.configurableProperties />
</cffunction>
```

getParent

```
public any getParent( )
```

Sets the parent PropertyManager instance this PropertyManager belongs to. It will return empty string if no parent is defined.

Parameters:

Code:

```
<cffunction name="getParent" access="public" returntype="any" output="false"
    hint="Sets the parent PropertyManager instance this PropertyManager belongs to. It will return empty string if no
    <cfreturn variables.parentPropertyManager />
</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 isObject(getParent()) AND getParent().isPropertyDefined(arguments.propertyName)>
    <cfreturn getParent().getProperty(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.">
```

```
<cfset var minorVersion = 0 />

<cfif NOT variables.minorVersion IS "@" & "minorVersion" & "@">
    <cfset minorVersion = variables.minorVersion />
</cfif>

<cfreturn variables.majorVersion & "." & minorVersion />
</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" />

    <cftry>
        <cfthrow type="MachII.framework.deprecatedMethod"
            message="The hasProperty() method has been deprecated. Please use isPropertyDefined() instead." />
        <cfcatch type="MachII.framework.deprecatedMethod">

        </cfcatch>
    </cftry>

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

init

```
public PropertyManager init( AppManager appManager, [any parentPropertyManager=""] )
```

Initialization function called by the framework.

Parameters:

AppManager appManager
[any parentPropertyManager=""]

Code:

```
<cffunction name="init" access="public" returntype="PropertyManager" output="false"
  hint="Initialization function called by the framework.">
  <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
  <cfargument name="parentPropertyManager" type="any" required="false" default=""
    hint="Optional argument for a parent property manager. If there isn't one default to empty string." />

  <cfset setAppManager(arguments.appManager) />
  <cfset variables.utils = getAppManager().getUtils() />

  <cfif isObject(arguments.parentPropertyManager)>
    <cfset setParent(arguments.parentPropertyManager) />
  </cfif>

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

isPropertyDefined

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

Checks if property name is defined in the properties. Does NOT check a parent.

Parameters:

string propertyName

Code:

```
<cffunction name="isPropertyDefined" access="public" returntype="boolean" output="false"
```

```

        hint="Checks if property name is defined in the properties. Does NOT check a parent.">
        <cfargument name="propertyName" type="string" required="true" />
        <cfreturn StructKeyExists(variables.properties, arguments.propertyName) />
    </cffunction>

```

loadXml

```
public void loadXml( string configXML, [boolean override="false"] )
```

Loads xml into the manager.

Parameters:

```
string configXML
[boolean override="false"]
```

Code:

```

<cffunction name="loadXml" access="public" returntype="void" output="false"
    hint="Loads xml into the manager.">
    <cfargument name="configXML" type="string" required="true" />
    <cfargument name="override" type="boolean" required="false" default="false" />

    <cfset var propertyNodes = "" />
    <cfset var propertyName = "" />
    <cfset var propertyValue = "" />
    <cfset var propertyType = "" />
    <cfset var propertyParams = "" />
    <cfset var paramsNodes = "" />
    <cfset var paramName = "" />
    <cfset var paramValue = "" />
    <cfset var hasParent = isObject(getParent()) />
    <cfset var mapping = "" />
    <cfset var i = 0 />
    <cfset var j = 0 />

    <cfif NOT arguments.override>
        <cfset propertyNodes = XMLSearch(arguments.configXML, "mach-ii/properties/property") />
    <cfelse>

```

```
</cfif> <cfset propertyNodes = XMLSearch(arguments.configXML, ".//properties/property") />
</cfif>

<cfloop from="1" to="#ArrayLen(PropertyNodes)#" index="i">
  <cfset propertyName = propertyNodes[i].xmlAttributes["name"] />

  <cfif hasParent AND arguments.override AND StructKeyExists(propertyNodes[i].xmlAttributes, "overrideAction")>
    <cfif propertyNodes[i].xmlAttributes["overrideAction"] EQ "useParent">
      <cfset removeProperty(propertyName) />
    <cfelseif propertyNodes[i].xmlAttributes["overrideAction"] EQ "addFromParent">
      <cfif StructKeyExists(propertyNodes[i].xmlAttributes, "mapping")>
        <cfset mapping = propertyNodes[i].xmlAttributes["mapping"] />
      <cfelse>
        <cfset mapping = propertyName />
      </cfif>

      <cfif NOT getParent().isPropertyDefined(mapping)>
        <cfthrow type="MachII.framework.overridePropertyNotDefined"
          message="An property named '#mapping#' cannot be found in the parent prop
        </cfif>

      <cfset setProperty(propertyName, getParent().getProperty(mapping), arguments.override) />
    </cfif>
  </cfif>

  <cfelse>
    <cfif StructKeyExists(propertyNodes[i].xmlAttributes, "type")>
      <cfset propertyType = propertyNodes[i].xmlAttributes["type"] />

      <cfset propertyParams = StructNew() />

      <cfif StructKeyExists(propertyNodes[i], "parameters")>
        <cfset paramsNodes = propertyNodes[i].parameters.xmlChildren />
        <cfloop from="1" to="#ArrayLen(paramsNodes)#" index="j">
          <cfset paramName = paramsNodes[j].xmlAttributes["name"] />
          <cftry>
            <cfset paramValue = variables.utils.recurseComplexValues(paramsNodes[j].xmlChildren) />
          </cftry>
          <cfset propertyParams[paramName] = paramValue />
        </cfloop>
      </cfif>
    </cfif>
  </cfelse>
</cfloop>
```

```

                <cfthrow type="MachII.framework.InvalidPropertyXml"
                    message="Xml parsing error for the property name" />
            </cfcatch>
        </cftry>
        <cfset propertyParams[paramName] = paramValue />
    </cfloop>
</cfif>

<cftry>
    <cfset propertyValue = CreateObject("component", propertyType).init(getAppManager) />
    <cfcatch type="any">
        <cfif StructKeyExists(cfcatch, "missingFileName")>
            <cfthrow type="MachII.framework.CannotFindProperty"
                message="Cannot find a CFC with the type of '#propertyType'" />
        </cfif>
        <cfrethrow />
    </cfcatch>
</cftry>
<cfset ArrayAppend(variables.configurableProperties, propertyName) />

<cfelse>
    <cftry>
        <cfset propertyValue = variables.utils.recurseComplexValues(propertyNodes[i]) />
        <cfcatch type="any">
            <cfthrow type="MachII.framework.InvalidPropertyXml"
                message="Xml parsing error for the property named '#propertyName'" />
        </cfcatch>
    </cftry>
</cfif>

    <cfif (hasParent AND NOT listFindNoCase(propsNotAllowInModule, propertyName))
        OR NOT hasParent>
        <cfset setProperty(propertyName, propertyValue) />
    </cfif>
</cfif>
</cfloop>

<cfif NOT hasParent>
    <cfif NOT isPropertyDefined("defaultEvent")>
        <cfset setProperty("defaultEvent", "defaultEvent") />
    </cfif>
</cfif>

```

```
</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") />
<cfelseif NOT ListFindNoCase("form|url", getProperty("parameterPrecedence"), "|")>
    <cfthrow type="MachII.framework.invalidPropertyValue"
        message="The 'parameterPrecedence' property must have a the value of 'form' or 'url'." />
</cfif>
<cfif NOT isPropertyDefined("maxEvents")>
    <cfset setProperty("maxEvents", 10) />
<cfelseif NOT IsNumeric(getProperty("maxEvents"))>
    <cfthrow type="MachII.framework.invalidPropertyValue"
        message="The 'maxEvents' property must be an integer." />
</cfif>
<cfif NOT isPropertyDefined("redirectPersistParameter")>
    <cfset setProperty("redirectPersistParameter", "persistId") />
</cfif>
<cfif NOT isPropertyDefined("redirectPersistScope")>
    <cfset setProperty("redirectPersistScope", "session") />
</cfif>
<cfif NOT isPropertyDefined("urlBase")>
    <cfset setProperty("urlBase", "index.cfm") />
</cfif>
<cfif NOT isPropertyDefined("urlDelimiters")>
    <cfset setProperty("urlDelimiters", "?|&|=") />
<cfelseif ListLen(getProperty("urlDelimiters"), "|") NEQ 3>
    <cfthrow type="MachII.framework.invalidPropertyValue"
        message="The 'urlDelimiters' property must have a list length of 3 with a delimiter of a
</cfif>
<cfif NOT isPropertyDefined("urlParseSES")>
    <cfset setProperty("urlParseSES", false) />
<cfelseif NOT IsBoolean(getProperty("urlParseSES"))>
    <cfthrow type="MachII.framework.invalidPropertyValue"
        message="The 'urlParseSES' property must be a boolean." />
</cfif>
<cfif NOT isPropertyDefined("moduleDelimiter")>
```

```
        <cfset setProperty("moduleDelimiter", ":") />
    </cfif>
</cfif>
</cffunction>
```

removeProperty

```
public void removeProperty( string propertyName )
```

Removes a property from the current property manager. Does NOT remove from a parent.

Parameters:

string propertyName

Code:

```
<cffunction name="removeProperty" access="public" returnType="void" output="false"
    hint="Removes a property from the current property manager. Does NOT remove from a parent.">
    <cfargument name="propertyName" type="string" required="true" />
    <cfset StructDelete(variables.properties, arguments.propertyName, false) />
</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>
```

```
</cffunction>
```

setParent

```
public void setParent( PropertyManager parentPropertyManager )
```

Returns the parent PropertyManager instance this FilterManager belongs to.

Parameters:

PropertyManager parentPropertyManager

Code:

```
<cffunction name="setParent" access="public" returntype="void" output="false"
    hint="Returns the parent PropertyManager instance this FilterManager belongs to.">
    <cfargument name="parentPropertyManager" type="MachII.framework.PropertyManager" required="true" />
    <cfset variables.parentPropertyManager = arguments.parentPropertyManager />
</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" />
</cffunction>
```

```
<cfif isObject(getParent()) AND listFindNoCase(propsNotAllowInModule, propertyName)>
    <cfthrow type="MachII.framework.propertyNotAllowed"
            message="The '#arguments.propertyName#' property cannot be set inside of a module." />
<cfelse>
    <cfset variables.properties[arguments.propertyName] = arguments.propertyValue />
</cfif>
</cffunction>
```

RequestHandler

Package: MachII.framework

Handles request to event conversion for the framework. The framework workhorse and controls the event-queue functionality.

Method Summary

public RequestHandler	init(AppManager appManager, string eventParameter, string parameterPrecedence, string moduleDelimiter, numeric maxEvents) Initializes the RequestHandler.
public Exception	createException([string type=""], [string message=""], [string errorCode=""], [string detail=""], [string extendedInfo=""], [array tagContext="#ArrayNew(1)#"]) Creates an exception object (with no cfcatch).
private AppManager	getAppManager()
public EventContext	getEventContext()
public numeric	getEventCount() Returns the number of events that have been processed for this context.
private string	getEventParameter()
private SizedQueue	getEventQueue()
private string	getIsException()
private boolean	getIsProcessing()
private numeric	getMaxEvents()
private string	getModuleDelimiter()
private string	getParameterPrecedence()
private struct	getRequestEventArgs() Builds a struct of incoming event args.

Method Summary

public string	getRequestEventName()
public string	getRequestModuleName()
private void	handleEvent(Event event) Handles the current event.
private void	handleNextEvent() Handles the next event in the queue.
public void	handleRequest() Handles a request made to the framework.
private boolean	hasMoreEvents() Checks if there are more events in the queue.
private void	incrementEventCount() Increments the current event count by 1.
private struct	parseEventParameter(struct eventArgs) Gets the module and event name from the incoming event arg struct.
private void	processEvents() Begins processing of queued events. Can only be called once.
private void	resetEventCount() Reset the current event count.
private void	setAppManager(AppManager appManager)
private void	setEventContext(EventContext eventContext)
private void	setEventParameter(string eventParameter)
private void	setEventQueue(SizedQueue eventQueue)
private void	setIsException(boolean isException)
private void	setIsProcessing(boolean isProcessing)

Method Summary

private void	setMaxEvents(numeric maxEvents)
private void	setModuleDelimiter(string moduleDelimiter)
private void	setParameterPrecedence(string parameterPrecedence)
private void	setRequestEventName(string requestEventName)
private void	setRequestModuleName(string requestModuleName)
private void	setupEventContext(AppManager appManager, [any currentEvent=""]) Setup an EventContext instance.
public Exception	wrapException(any caughtException) Creates an exception object (with cfcatch).

Method Detail**createException**

```
public 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="public" 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").init(arguments.type, arguments.message)
  <cfif NOT getIsException()>
    <cfset resetEventCount() />
  </cfif>
  <cfset setIsException(true) />

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

getAppManager

```
private AppManager getAppManager()
```

Parameters:

Code:

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

getEventContext

```
public EventContext getEventContext()
```

Parameters:

Code:

```
<cffunction name="getEventContext" access="public" returntype="MachII.framework.EventContext" output="false">
    <cfreturn variables.eventContext />
</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>
```

getEventParameter

```
private string getEventParameter( )
```

Parameters:

Code:

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

getEventQueue

private SizedQueue getEventQueue()

Parameters:

Code:

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

private numeric getMaxEvents()

Parameters:

Code:

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

getModuleDelimiter

private string getModuleDelimiter()

Parameters:

Code:

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

getParameterPrecedence

private string getParameterPrecedence()

Parameters:

Code:

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

getRequestEventArgs

```
private struct getRequestEventArgs( )
```

Builds a struct of incoming event args.

Parameters:

Code:

```
<cffunction name="getRequestEventArgs" access="private" returntype="struct" output="false"
    hint="Builds a struct of incoming event args.">
    <cfset var eventArgs = StructNew() />
    <cfset var overwriteFormParams = (getParameterPrecedence() EQ "url") />

    <cfset StructAppend(eventArgs, form) />
    <cfset StructAppend(eventArgs, url, overwriteFormParams) />
    <cfset StructAppend(eventArgs, getAppManager().getRequestManager().parseSesParameters(cgi.PATH_INFO), overwriteFormParams) />

    <cfset StructAppend(eventArgs, getAppManager().getRequestManager().readPersistEventData(eventArgs), true) />

    <cfreturn eventArgs />
</cffunction>
```

getRequestEventName

```
public string getRequestEventName( )
```

Parameters:

Code:

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

getRequestModuleName

```
public string getRequestModuleName( )
```

Parameters:

Code:

```
<cffunction name="getRequestModuleName" access="public" returntype="string" output="false">
    <cfreturn variables.requestModuleName />
</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 eventHandler = 0 />
    <cfset var topAppManager = 0 />
    <cfset var thisEventAppManager = 0 />
```

```
<cfif IsObject(getAppManager().getParent())>
    <cfset topAppManager = getAppManager().getParent() />
<cfelse>
    <cfset topAppManager = getAppManager() />
</cfif>

<cfif Len(arguments.event.getModuleName())>
    <cfset thisEventAppManager = topAppManager.getModuleManager().getModule(arguments.event.getModuleName())
<cfelse>
    <cfset thisEventAppManager = topAppManager />
</cfif>

<cfset setupEventContext(thisEventAppManager, arguments.event) />
<cfset request.event = arguments.event />

<cfset thisEventAppManager.getPluginManager().preEvent(getEventContext()) />

<cfset eventHandler = thisEventAppManager.getEventManager().getEventHandler(arguments.event.getName(), arguments
<cfset eventHandler.handleEvent(arguments.event, getEventContext()) />

<cfset thisEventAppManager.getPluginManager().postEvent(getEventContext()) />
</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 getEventContext().handleException(exception, true) />
  </cfif>
</cfcatch>
</cftry>
</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 result = parseEventParameter(eventArgs) />
  <cfset var appManager = getAppManager() />
  <cfset var moduleManager = getAppManager().getModuleManager() />
  <cfset var nextEvent = "" />
  <cfset var exception = "" />

  <cfset setRequestEventName(result.eventName) />
  <cfset setRequestModuleName(result.moduleName) />
  <cfset setupEventContext(appManager) />
```

```

    <cftry>
        <cfif Len(result.moduleName)>
            <cfif NOT moduleManager.isModuleDefined(result.moduleName)>
                <cfthrow type="MachII.framework.ModuleNotDefined"
                    message="The module '#result.moduleName#' for event '#result.eventName#' is not o
            <cfelse>
                <cfset appManager = appManager.getModuleManager().getModule(result.moduleName).getModuleE
            </cfif>
        </cfif>

        <cfif NOT appManager.getEventManager().isEventDefined(result.eventName, false)>
            <cfthrow type="MachII.framework.EventHandlerNotDefined"
                message="Event-handler for event '#result.eventName#', module '#result.moduleName#' is no
        <cfelseif NOT appManager.getEventManager().isEventPublic(result.eventName, false)>
            <cfthrow type="MachII.framework.EventHandlerNotAccessible"
                message="Event-handler for event '#result.eventName#', module '#result.moduleName#' is no
        </cfif>

        <cfset nextEvent = appManager.getEventManager().createEvent(result.moduleName, result.eventName, eventArg
        <cfset getEventQueue().put(nextEvent) />
        <cfset setupEventContext(appManager, nextEvent) />

        <cfcatch type="any">
            <cfset setupEventContext(appManager) />
            <cfset exception = wrapException(cfcatch) />
            <cfset getEventContext().handleException(exception, true) />
        </cfcatch>
    </cftry>

    <cfset processEvents() />
</cffunction>

```

hasMoreEvents

```
private boolean hasMoreEvents()
```

Checks if there are more events in the queue.

Parameters:

Code:

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

init

```
public RequestHandler init( AppManager appManager, string eventParameter, string parameterPrecedence, string moduleDelimiter, numeric maxEvents )
```

Initializes the RequestHandler.

Parameters:

```
AppManager appManager
string eventParameter
string parameterPrecedence
```

string moduleDelimiter
numeric maxEvents

Code:

```
<cffunction name="init" access="public" returntype="RequestHandler" output="false"
    hint="Initializes the RequestHandler.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfargument name="eventParameter" type="string" required="true" />
    <cfargument name="parameterPrecedence" type="string" required="true" />
    <cfargument name="moduleDelimiter" type="string" required="true" />
    <cfargument name="maxEvents" type="numeric" required="true" />

    <cfset setAppManager(arguments.appManager) />
    <cfset setEventParameter(arguments.eventParameter) />
    <cfset setModuleDelimiter(arguments.moduleDelimiter) />
    <cfset setMaxEvents(arguments.maxEvents) />

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

    <cfset setEventContext(CreateObject("component", "MachII.framework.EventContext").init(this, getEventQueue())) />

    <cfset request.eventContext = getEventContext() />

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

parseEventParameter

private struct parseEventParameter(struct eventArgs)

Gets the module and event name from the incoming event arg struct.

Parameters:

struct eventArgs

Code:

```

<cffunction name="parseEventParameter" access="private" returnType="struct" output="false"
  hint="Gets the module and event name from the incoming event arg struct.">
  <cfargument name="eventArgs" type="struct" required="true" />

  <cfset var rawEvent = "" />
  <cfset var eventParameter = getEventParameter() />
  <cfset var moduleDelimiter = getModuleDelimiter() />
  <cfset var result = StructNew() />

  <cfif StructKeyExists(arguments.eventArgs, eventParameter) AND Len(arguments.eventArgs[eventParameter])>

    <cfset rawEvent = arguments.eventArgs[eventParameter] />

    <cfif listLen(rawEvent, moduleDelimiter) eq 2>
      <cfset result.moduleName = listGetAt(rawEvent, 1, moduleDelimiter) />
      <cfset result.eventName = listGetAt(rawEvent, 2, moduleDelimiter) />
    <cfelseif listLen(rawEvent, moduleDelimiter) eq 1 AND Right(rawEvent, 1) eq moduleDelimiter>
      <cfset result.moduleName = listGetAt(rawEvent, 1, moduleDelimiter) />
      <cfset result.eventName = getAppManager().getModuleManager().getModule(result.moduleName).getModu

    <cfelse>
      <cfset result.moduleName = "" />
      <cfset result.eventName = rawEvent />
    </cfif>

  <cfelse>
    <cfset result.moduleName = "" />
    <cfset result.eventName = getAppManager().getPropertyManager().getProperty("defaultEvent") />
  </cfif>

  <cfreturn result />
</cffunction>

```

processEvents

private void processEvents()

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

Parameters:

Code:

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

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

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

  <cfset pluginManager = getEventContext().getAppManager().getPluginManager() />

  <cfset pluginManager.preProcess(getEventContext()) />

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

  <cfif NOT getIsException() AND hasMoreEvents(>
    <cfset exception = createException("MachII.framework.MaxEventsExceeded", "The maximum number of events (#
    <cfset getEventContext().handleException(exception, true) />

    <cfset resetEventCount() />

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

    <cfif hasMoreEvents(>
      <cfthrow
        type="MachII.framework.MaxEventsExceededDuringException"
        message="The maximum number of events (#getMaxEvents()#) has been exceeded. An exception
        detail="Please check your exception handling since it initiated an infinite loop." />
    </cfif>

  <cfelseif getIsException() AND hasMoreEvents(>
```

```
        <cfset exception = getEventContext().getCurrentEvent().getArg("exception").getCaughtException() />
        <cfthrow
            type="MachII.framework.MaxEventsExceededDuringException"
            message="The maximum number of events (#getMaxEvents()) has been exceeded. An exception was generated."
            detail="The last exception was '#exception.detail#' which occurred on line #exception.tagContextLineNumber."
        />
    </cfif>

    <cfset pluginManager = getEventContext().getAppManager().getPluginManager() />
    <cfset pluginManager.postProcess(getEventContext()) />

    <cfset setIsProcessing(false) />
</cffunction>
```

resetEventCount

```
private void resetEventCount( )
```

Reset the current event count.

Parameters:

Code:

```
<cffunction name="resetEventCount" access="private" returntype="void" output="false"
    hint="Reset the current event count.">
    <cfset variables.eventCount = 0 />
</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>
```

setEventContext

```
private void setEventContext( EventContext eventContext )
```

Parameters:

EventContext eventContext

Code:

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

setEventParameter

```
private void setEventParameter( string eventParameter )
```

Parameters:

string eventParameter

Code:

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

```
</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>
```

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

private void setMaxEvents(numeric maxEvents)

Parameters:

numeric maxEvents

Code:

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

setModuleDelimiter

private void setModuleDelimiter(string moduleDelimiter)

Parameters:

string moduleDelimiter

Code:

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

setParameterPrecedence

private void setParameterPrecedence(string parameterPrecedence)

Parameters:

string parameterPrecedence

Code:

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

setRequestModuleName

```
private void setRequestModuleName( string requestModuleName )
```

Parameters:

string requestModuleName

Code:

```
<cffunction name="setRequestModuleName" access="private" returnType="void" output="false">
    <cfargument name="requestModuleName" type="string" required="true" />
    <cfset variables.requestModuleName = arguments.requestModuleName />
</cffunction>
```

setupEventContext

```
private void setupEventContext( AppManager appManager, [any currentEvent=""] )
```

Setup an EventContext instance.

Parameters:

AppManager appManager
[any currentEvent=""]

Code:

```
<cffunction name="setupEventContext" access="private" returnType="void" output="false"
    hint="Setup an EventContext instance.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfargument name="currentEvent" type="any" required="false" default="" />
```

```
<cfset getEventContext().setup(arguments.appManager, arguments.currentEvent) />
</cffunction>
```

wrapException

public Exception wrapException(any caughtException)

Creates an exception object (with cfcatch).

Parameters:

any caughtException

Code:

```
<cffunction name="wrapException" access="public" 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").wrapException(arguments.caughtException) />
  <cfif NOT getIsException()>
    <cfset resetEventCount() />
  </cfif>
  <cfset setIsException(true) />

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

RequestManager

Package: MachII.framework

Manages request functionality for the framework.

Method Summary

public RequestManager	init(AppManager appManager)	Initializes the manager.
public string	buildUrl(string moduleName, string eventName, [any urlParameters=""], [string urlBase="#getDefaultUrlBase()#"])	Builds a framework specific url.
private void	cleanupPersistEventStorage()	Cleanups the persist event data storage.
public void	configure()	Configures nothing.
private string	createPersistId()	Creates a persistId for use.
private string	createTimestamp()	Creates a timestamp for use.
private AppManager	getAppManager()	
private string	getDefaultUrlBase()	
private string	getEventParameter()	
private numeric	getMaxEvents()	
private string	getModuleDelimiter()	

Method Summary

private string	getPairDelimiter()
private string	getParameterPrecedence()
private string	getParseSes()
private struct	getPersistEventStorage() Helper function to get the event data store for persists.
private PropertyManager	getPropertyManager()
private string	getQueryStringDelimiter()
private string	getRedirectPersistParameter()
private string	getRedirectPersistScope()
public RequestHandler	getRequestHandler() Returns a new or cached instance of a RequestHandler.
private string	getSeriesDelimiter()
private Utils	getUtils()
private struct	parseBuildUrlParameters(any urlParameters) Parses the build url parameters into a useable form.
public struct	parseSesParameters(string pathInfo) Parse SES parameters.
public struct	readPersistEventData(struct eventArgs) Gets a persisted event by id if found in event args.
public string	savePersistEventData(struct eventArgs) Saves persisted event data and returns the persistId.
private void	setAppManager(AppManager appManager)
private void	setDefaultUrlBase(string defaultUrlBase)
private void	setEventParameter(string eventParameter)

Method Summary

private void	setMaxEvents(numeric maxEvents)
private void	setModuleDelimiter(string moduleDelimiter)
private void	setPairDelimiter(string pairDelimiter)
private void	setParameterPrecedence(string parameterPrecedence)
private void	setParseSes(string parseSes)
private void	setQueryStringDelimiter(string queryStringDelimiter)
private void	setRedirectPersistParameter(string redirectPersistParameter)
private void	setRedirectPersistScope(string redirectPersistScope)
private void	setSeriesDelimiter(string seriesDelimiter)

Method Detail**buildUrl**

```
public string buildUrl( string moduleName, string eventName, [any urlParameters=""], [string urlBase="#getDefaultUrlBase()#"] )
```

Builds a framework specific url.

Parameters:

```
string moduleName
string eventName
[any urlParameters=""]
[string urlBase="#getDefaultUrlBase()#"]
```

Code:

```
<cffunction name="buildUrl" access="public" returntype="string" output="false"
  hint="Builds a framework specific url.">
  <cfargument name="moduleName" type="string" required="true"
    hint="Name of the module to build the url with." />
  <cfargument name="eventName" type="string" required="true"
```

```
        hint="Name of the event to build the url with." />
<cfargument name="urlParameters" type="any" required="false" default=""
    hint="Name/value pairs (urlArg1=value1|urlArg2=value2) to build the url with or a struct of data." />
<cfargument name="urlBase" type="string" required="false" default="#getDefaultUrlBase()#"
    hint="Base of the url. Defaults to the value of the urlBase property." />

<cfset var builtUrl = "" />
<cfset var queryString = "" />
<cfset var params = parseBuildUrlParameters(arguments.urlParameters) />
<cfset var value = "" />
<cfset var i = "" />

<cfif Len(arguments.moduleName) AND Len(arguments.eventName)>
    <cfset queryString = queryString & getEventParameter() & getPairDelimiter() & arguments.moduleName & get
<cfelseif NOT Len(arguments.moduleName) AND Len(arguments.eventName)>
    <cfset queryString = queryString & getEventParameter() & getPairDelimiter() & arguments.eventName />
</cfif>

<cfloop collection="#params#" item="i">
    <cfif IsSimpleValue(params[i])>
        <cfif getParseSes()>
            <cfset params[i] = Replace(params[i], ";", "U_03B", "all") />
        </cfif>
        <cfset queryString = queryString & getSeriesDelimiter() & i & getPairDelimiter() & URLEncodedForm
    </cfif>
</cfloop>

<cfif Len(queryString)>
    <cfset builtUrl = arguments.urlBase & getQueryStringDelimiter() & queryString />
    <cfif getSeriesDelimiter() NEQ "&">
        <cfset builtUrl = builtUrl & getSeriesDelimiter() />
    </cfif>
<cfelse>
    <cfset builtUrl = arguments.urlBase />
</cfif>

<cfreturn builtUrl />
</cffunction>
```

cleanupPersistEventStorage

```
private void cleanupPersistEventStorage( )
```

Cleanups the persist event data storage.

Parameters:

Code:

```
<cffunction name="cleanupPersistEventStorage" access="private" returntype="void" output="false"
    hint="Cleanups the persist event data storage.">

    <cfset var timestamp = createTimestamp() />
    <cfset var diffTimestamp = DateAdd("n", variables.cleanupDifference, timestamp) />
    <cfset var dataStorage = getPersistEventStorage() />
    <cfset var dataTimestampArray = "" />
    <cfset var i = "" />

    <cfif DateCompare(dataStorage.lastCleanup, diffTimestamp) EQ 1>
        <cflock name="_MachIIPersistEventStorageCleanup" type="exclusive" timeout="5" throwontimeout="false">
            <cfif DateCompare(dataStorage.lastCleanup, diffTimestamp) EQ 1>
                <cfset dataStorage.lastCleanup = timestamp />

                <cfset dataTimestampArray = StructKeyArray(dataStorage.timestamps) />
                <cfset ArraySort(dataTimestampArray, "numeric", "asc") />

                <cfloop from="1" to="#ArrayLen(dataTimestampArray)#" index="i">
                    <cftry>
                        <cfif DateCompare(dataTimestampArray[i], diffTimestamp) EQ 1>
                            <cfset StructDelete(dataStorage.data, dataStorage.timestamps[dataTimestampArray[i]]) />
                            <cfset StructDelete(dataStorage.timestamps, dataTimestampArray[i]) />
                        <cfelse>
                            <cfbreak />
                        </cfif>
                    </cftry>
                    <cfcatch type="any">
                        </cfcatch>
                </cfloop>
            </cflock>
        </cfif>
    </cfif>
</cffunction>
```

```
                </cfloop>
            </cfif>
        </cflock>
    </cfif>
</cffunction>
```

configure

```
public void configure( )
```

Configures nothing.

Parameters:

Code:

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

</cffunction>
```

createPersistId

```
private string createPersistId( )
```

Creates a persistId for use.

Parameters:

Code:

```
<cffunction name="createPersistId" access="private" returntype="string" output="false"
    hint="Creates a persistId for use.">
    <cfreturn REReplace(CreateUUID(), "[[:punct:]]", "", "ALL") />
</cffunction>
```

createTimestamp

private string createTimestamp()

Creates a timestamp for use.

Parameters:

Code:

```
<cffunction name="createTimestamp" access="private" returntype="string" output="false"
    hint="Creates a timestamp for use.">
    <cfreturn REReplace(Now(), "[ts[:punct:][:space:]]", "", "ALL") />
</cffunction>
```

getAppManager

private AppManager getAppManager()

Parameters:

Code:

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

getDefaultUrlBase

private string getDefaultUrlBase()

Parameters:

Code:

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

getEventParameter

private string getEventParameter()

Parameters:

Code:

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

getMaxEvents

private numeric getMaxEvents()

Parameters:

Code:

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

getModuleDelimiter

private string getModuleDelimiter()

Parameters:

Code:

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

getPairDelimiter

private string getPairDelimiter()

Parameters:

Code:

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

getParameterPrecedence

private string getParameterPrecedence()

Parameters:

Code:

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

getParseSes

```
private string getParseSes( )
```

Parameters:

Code:

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

getPersistEventStorage

```
private struct getPersistEventStorage( )
```

Helper function to get the event data store for persists.

Parameters:

Code:

```
<cffunction name="getPersistEventStorage" access="private" returntype="struct" output="false"
    hint="Helper function to get the event data store for persists.">

    <cfset var scope = "" />

    <cfif getRedirectPersistScope() EQ "application">
        <cfset scope = StructGet("application") />
    <cfelseif getRedirectPersistScope() EQ "session">
        <cfset scope = StructGet("session") />
    <cfelseif getRedirectPersistScope() EQ "server">
        <cfset scope = StructGet("server") />
    <cfelse>
        <cfthrow type="MachII.framework.UnsupportedRedirectPersistScope"
            message="You can only use session, application or server scopes." />
    </cfif>

    <cfif NOT StructKeyExists(scope, "_MachIIPersistEventStorage")>
```

```
<cflock name="_MachIIPersistEventStorageCreate" type="exclusive" timeout="5" throwontimeout="false">
  <cfif NOT StructKeyExists(scope, "_MachIIPersistEventStorage")>
    <cfset scope._MachIIPersistEventStorage = StructNew() />
    <cfset scope._MachIIPersistEventStorage.data = StructNew() />
    <cfset scope._MachIIPersistEventStorage.timestamps = StructNew() />
    <cfset scope._MachIIPersistEventStorage.lastCleanup = createTimestamp() />
  </cfif>
</cflock>
</cfif>
<cfreturn scope._MachIIPersistEventStorage />
</cffunction>
```

getPropertyManager

private PropertyManager getPropertyManager()

Parameters:

Code:

```
<cffunction name="getPropertyManager" access="private" returnType="MachII.framework.PropertyManager" output="false">
  <cfreturn getAppManager().getPropertyManager() />
</cffunction>
```

getQueryStringDelimiter

private string getQueryStringDelimiter()

Parameters:

Code:

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

getRedirectPersistParameter

```
private string getRedirectPersistParameter( )
```

Parameters:

Code:

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

getRedirectPersistScope

```
private string getRedirectPersistScope( )
```

Parameters:

Code:

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

getRequestHandler

```
public RequestHandler getRequestHandler( )
```

Returns a new or cached instance of a RequestHandler.

Parameters:

Code:

```
<cffunction name="getRequestHandler" access="public" returnType="MachII.framework.RequestHandler" output="false"
  hint="Returns a new or cached instance of a RequestHandler.">
  <cfset var appKey = getAppManager().getAppLoader().getAppKey() />
  <cfif NOT StructKeyExists(request, "_MachIIRequestHandler_" & appKey)>
    <cfset request["_MachIIRequestHandler_" & appKey] =
      CreateObject("component", "MachII.framework.RequestHandler").init(getAppManager(), getEv
  </cfif>
  <cfreturn request["_MachIIRequestHandler_" & appKey] />
</cffunction>
```

getSeriesDelimiter

```
private string getSeriesDelimiter( )
```

Parameters:

Code:

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

getUtils

```
private Utils getUtils( )
```

Parameters:

Code:

```
<cffunction name="getUtils" access="private" returnType="MachII.util.Utils" output="false">
  <cfreturn getAppManager().getUtils() />
</cffunction>
```

init

```
public RequestManager init( AppManager appManager )
```

Initializes the manager.

Parameters:

AppManager appManager

Code:

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

    <cfset var urlDelimiters = "" />

    <cfset setAppManager(arguments.appManager) />

    <cfset urlDelimiters = getPropertyManager().getProperty("urlDelimiters") />
    <cfset setRedirectPersistParameter(getPropertyManager().getProperty("redirectPersistParameter")) />
    <cfset setRedirectPersistScope(getPropertyManager().getProperty("redirectPersistScope")) />
    <cfset setDefaultUrlBase(getPropertyManager().getProperty("urlBase")) />
    <cfset setEventParameter(getPropertyManager().getProperty("eventParameter")) />
    <cfset setParameterPrecedence(getPropertyManager().getProperty("parameterPrecedence")) />
    <cfset setParseSES(getPropertyManager().getProperty("urlParseSES")) />
    <cfset setModuleDelimiter(getPropertyManager().getProperty("moduleDelimiter")) />
    <cfset setMaxEvents(getPropertyManager().getProperty("maxEvents")) />

    <cfset setQueryStringDelimiter(ListGetAt(urlDelimiters, 1, "|")) />
    <cfset setSeriesDelimiter(ListGetAt(urlDelimiters, 2, "|")) />
    <cfset setPairDelimiter(ListGetAt(urlDelimiters, 3, "|")) />

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

parseBuildUrlParameters

```
private struct parseBuildUrlParameters( any urlParameters )
```

Parses the build url parameters into a useable form.

Parameters:

any urlParameters

Code:

```
<cffunction name="parseBuildUrlParameters" access="private" returnType="struct" output="false"
  hint="Parses the build url parameters into a useable form.">
  <cfargument name="urlParameters" type="any" required="true"
    hint="Takes string of name/value pairs or a struct.">

  <cfset var params = StructNew() />
  <cfset var temp = "" />
  <cfset var i = "" />

  <cfif IsSimpleValue(arguments.urlParameters)>
    <cfloop list="#arguments.urlParameters#" index="i" delimiters="|">
      <cfif ListLen(i, "=") EQ 2>
        <cfset temp = ListLast(i, "=") />
      <cfelse>
        <cfset temp = "" />
      </cfif>
      <cfset params[ListFirst(i, "=")] = temp />
    </cfloop>
  <cfelseif IsStruct(arguments.urlParameters)>
    <cfset params = arguments.urlParameters />
  <cfelse>
    <cfthrow
      type="MachII.framework.urlParametersInvalidType"
      message="BuildUrl()'s urlParameters attribute takes a list or struct." />
  </cfif>

  <cfreturn params />
</cffunction>
```

parseSesParameters

public struct parseSesParameters(string pathInfo)

Parse SES parameters.

Parameters:

string pathInfo

Code:

```
<cffunction name="parseSesParameters" access="public" returntype="struct" output="false"
    hint="Parse SES parameters.">
    <cfargument name="pathInfo" type="string" required="true" />

    <cfset var names = "" />
    <cfset var value = "" />
    <cfset var params = StructNew() />
    <cfset var i = "" />

    <cfif getParseSes() AND Len(arguments.pathInfo) GT 1>

        <cfset arguments.pathInfo = Mid(arguments.pathInfo, 2, Len(arguments.pathInfo)) />

        <cfif Right(arguments.pathInfo, 1) IS getSeriesDelimiter()
            <cfset arguments.pathInfo = Mid(arguments.pathInfo, 1, Len(arguments.pathInfo) - 1) />
        </cfif>

        <cfset arguments.pathInfo = Replace(arguments.pathInfo, "U_03B", ";", "all") />

        <cfif getSeriesDelimiter() EQ getPairDelimiter()>

            <cfset names = ListToArray(getUtils().listFix(arguments.pathInfo, getSeriesDelimiter(), "--_NULL_"))
            <cfloop from="1" to="#ArrayLen(names)#" index="i" step="2">
                <cfif i + 1 LTE ArrayLen(names) AND names[i+1] NEQ "--_NULL_--">
                    <cfset value = names[i+1] />
                <cfelse>
                    <cfset value = "" />
                </cfif>
                <cfset params[names[i]] = value />
            </cfloop>
        </cfif>
    </cfif>
</cffunction>
```

```

        </cfloop>
    <cfelse>
        <cfset names = ListToArray(arguments.pathInfo, getSeriesDelimiter()) />
        <cfloop from="1" to="#ArrayLen(names)#" index="i">
            <cfif ListLen(names[i], getPairDelimiter()) EQ 2>
                <cfset value = ListGetAt(names[i], 2, getPairDelimiter()) />
            <cfelse>
                <cfset value = "" />
            </cfif>
            <cfset params[ListGetAt(names[i], 1, getPairDelimiter())] = value />
        </cfloop>
    </cfif>
</cfif>
    <cfreturn params />
</cffunction>

```

readPersistEventData

public struct readPersistEventData(struct eventArgs)

Gets a persisted event by id if found in event args.

Parameters:

struct eventArgs

Code:

```

<cffunction name="readPersistEventData" access="public" returntype="struct" output="false"
    hint="Gets a persisted event by id if found in event args.">
    <cfargument name="eventArgs" type="struct" required="true" />

    <cfset var persistId = "" />
    <cfset var persistedData = StructNew() />
    <cfset var dataStorage = "" />

    <cfif StructKeyExists(arguments.eventArgs, getRedirectPersistParameter())>

```

```
<cfset persistId = arguments.eventArgs[getRedirectPersistParameter()] />
<cfset dataStorage = getPersistEventStorage() />

<cfif StructKeyExists(dataStorage.data, persistId)>
    <cftry>
        <cfset persistedData = dataStorage.data[persistId]>
        <cfset StructDelete(dataStorage.data, persistId, false) />
        <cfcatch type="any">
            </cfcatch>
        </cftry>
    </cfif>
</cfif>
<cfreturn persistedData />
</cffunction>
```

savePersistEventData

```
public string savePersistEventData( struct eventArgs )
```

Saves persisted event data and returns the persistId.

Parameters:

```
struct eventArgs
```

Code:

```
<cffunction name="savePersistEventData" access="public" returntype="string" output="false"
    hint="Saves persisted event data and returns the persistId.">
    <cfargument name="eventArgs" type="struct" required="true" />

    <cfset var persistId = createPersistId() />
    <cfset var timestamp = createTimestamp() />
    <cfset var dataStorage = getPersistEventStorage() />

    <cfset cleanupPersistEventStorage() />
```

```
<cfset dataStorage.data[persistId] = arguments.eventArgs />
<cfset dataStorage.timestamps[timestamp] = persistId />

<cfreturn persistId />
</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>
```

setDefaultUrlBase

```
private void setDefaultUrlBase( string defaultUrlBase )
```

Parameters:

string defaultUrlBase

Code:

```
<cffunction name="setDefaultUrlBase" access="private" returnType="void" output="false">
    <cfargument name="defaultUrlBase" type="string" required="true" />
```

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

setEventParameter

```
private void setEventParameter( string eventParameter )
```

Parameters:

string eventParameter

Code:

```
<cffunction name="setEventParameter" access="private" returnType="void" output="false">  
    <cfargument name="eventParameter" type="string" required="true" />  
    <cfset variables.eventParameter = arguments.eventParameter />  
</cffunction>
```

setMaxEvents

```
private void setMaxEvents( numeric maxEvents )
```

Parameters:

numeric maxEvents

Code:

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

setModuleDelimiter

private void setModuleDelimiter(string moduleDelimiter)

Parameters:

string moduleDelimiter

Code:

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

setPairDelimiter

private void setPairDelimiter(string pairDelimiter)

Parameters:

string pairDelimiter

Code:

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

setParameterPrecedence

private void setParameterPrecedence(string parameterPrecedence)

Parameters:

string parameterPrecedence

Code:

```
<cffunction name="setParameterPrecedence" access="private" returnType="void" output="false">
    <cfargument name="parameterPrecedence" type="string" required="true" />
    <cfset variables.parameterPrecedence = arguments.parameterPrecedence />
</cffunction>
```

setParameterSes

private void setParseSes(string parseSes)

Parameters:

string parseSes

Code:

```
<cffunction name="setParseSes" access="private" returnType="void" output="false">
    <cfargument name="parseSes" type="string" required="true" />
    <cfset variables.parseSes = arguments.parseSes />
</cffunction>
```

setQueryStringDelimiter

private void setQueryStringDelimiter(string queryStringDelimiter)

Parameters:

string queryStringDelimiter

Code:

```
<cffunction name="setQueryStringDelimiter" access="private" returnType="void" output="false">
```

```
<cfargument name="queryStringDelimiter" type="string" required="true" />
<cfset variables.queryStringDelimiter = arguments.queryStringDelimiter />
</cffunction>
```

setRedirectPersistParameter

```
private void setRedirectPersistParameter( string redirectPersistParameter )
```

Parameters:

string redirectPersistParameter

Code:

```
<cffunction name="setRedirectPersistParameter" access="private" returnType="void" output="false">
  <cfargument name="redirectPersistParameter" type="string" required="true" />
  <cfset variables.redirectPersistParameter = arguments.redirectPersistParameter />
</cffunction>
```

setRedirectPersistScope

```
private void setRedirectPersistScope( string redirectPersistScope )
```

Parameters:

string redirectPersistScope

Code:

```
<cffunction name="setRedirectPersistScope" access="private" returnType="void" output="false">
  <cfargument name="redirectPersistScope" type="string" required="true" />
  <cfset variables.redirectPersistScope = arguments.redirectPersistScope />
</cffunction>
```

setSeriesDelimiter

private void setSeriesDelimiter(string seriesDelimiter)

Parameters:

string seriesDelimiter

Code:

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

SubroutineHandler

Package: MachII.framework

Handles processing of Commands for a Subroutine.

Method Summary

public Subroutine-Handler	init() Used by the framework for initialization. Do not override.
public void	addCommand(Command command) Adds a Command.
public boolean	handleSubroutine(Event event, EventContext eventContext) Handles a Subroutine.

Method Detail

addCommand

```
public void addCommand( Command command )
```

Adds a Command.

Parameters:

Command command

Code:

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

handleSubroutine

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

Handles a Subroutine.

Parameters:

Event event

EventContext eventContext

Code:

```
<cffunction name="handleSubroutine" access="public" returntype="boolean" output="true"
  hint="Handles a Subroutine.">
  <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 from="1" to="#ArrayLen(variables.commands)#" index="i">
    <cfset command = variables.commands[i] />
    <cfset continue = command.execute(arguments.event, arguments.eventContext) />
    <cfif continue IS false>
      <cfbreak />
    </cfif>
  </cfloop>

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

init

public SubroutineHandler init()

Used by the framework for initialization. Do not override.

Parameters:

Code:

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

SubroutineManager

Package: MachII.framework

Inherits from: framework.CommandLoaderBase

Manages registered SubroutineHandlers for the framework.

Method Summary

public SubroutineManager	init(AppManager appManager, [any parentSubroutineManager=""]) Initialization function called by the framework.
public void	addSubroutineHandler(string subroutineName, SubroutineHandler subroutineHandler, [boolean overrideCheck="false"]) Registers a SubroutineHandler by name.
public void	configure() Configures each of the registered SubroutineHandlers.
public AppManager	getAppManager()
public any	getParent() Sets the parent SubroutineManager instance this SubroutineManager belongs to. Return empty string if no parent is defined.
public SubroutineHandler	getSubroutineHandler(string subroutineName) Returns the SubroutineHandler for the named Subroutine.
public array	getSubroutineNames() Returns an array of subroutine names.
public boolean	isSubroutineDefined(string subroutineName) Returns true if a SubroutineHandler for the named Subroutine is defined; otherwise false.

Method Summary

public void	loadXml(string configXML, [boolean override="false"]) Loads xml for the manager.
public void	removeSubroutine(string subroutineName) Removes a subroutine. Does NOT remove from a parent.
public void	setAppManager(AppManager appManager)
public void	setParent(SubroutineManager parentSubroutineManager) Returns the parent SubroutineManager instance this SubroutineManager belongs to.

Methods inherited from framework.CommandLoaderBase: createCommand , setupEventArg , setupAnnounce , setupEventBean , setupViewPage , setupFilter , setupRedirect , setupDefault , setupEventMapping , setupNotify , setupExecute

Method Detail**addSubroutineHandler**

```
public void addSubroutineHandler( string subroutineName, SubroutineHandler subroutineHandler, [boolean overrideCheck="false"] )
```

Registers a SubroutineHandler by name.

Parameters:

```
string subroutineName
SubroutineHandler subroutineHandler
[boolean overrideCheck="false"]
```

Code:

```
<cffunction name="addSubroutineHandler" access="public" returntype="void" output="false"
    hint="Registers a SubroutineHandler by name.">
```

```
<cfargument name="subroutineName" type="string" required="true" />
<cfargument name="subroutineHandler" type="MachII.framework.SubroutineHandler" required="true" />
<cfargument name="overrideCheck" type="boolean" required="false" default="false" />

<cfif NOT arguments.overrideCheck AND isSubroutineDefined(arguments.subroutineName)>
    <cfthrow type="MachII.framework.SubroutineHandlerAlreadyDefined"
        message="A SubroutineHandler with name '#arguments.subroutineName#' is already registered." />
<cfelse>
    <cfset variables.handlers[arguments.subroutineName] = arguments.subroutineHandler />
</cfif>
</cffunction>
```

configure

```
public void configure( )
```

Configures each of the registered SubroutineHandlers.

Parameters:

Code:

```
<cffunction name="configure" access="public" returnType="void" output="false"
    hint="Configures each of the registered SubroutineHandlers.">

</cffunction>
```

getAppManager

```
public AppManager getAppManager( )
```

Parameters:

Code:

```
<cffunction name="getAppManager" access="public" returnType="MachII.framework.AppManager" output="false">
```

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

getParent

public any getParent()

Sets the parent SubroutineManager instance this SubroutineManager belongs to. Return empty string if no parent is defined.

Parameters:

Code:

```
<cffunction name="getParent" access="public" returntype="any" output="false"
    hint="Sets the parent SubroutineManager instance this SubroutineManager belongs to. Return empty string if no pa
    <cfreturn variables.parentSubroutineManager />
</cffunction>
```

getSubroutineHandler

public SubroutineHandler getSubroutineHandler(string subroutineName)

Returns the SubroutineHandler for the named Subroutine.

Parameters:

string subroutineName

Code:

```
<cffunction name="getSubroutineHandler" access="public" returntype="MachII.framework.SubroutineHandler"
    hint="Returns the SubroutineHandler for the named Subroutine.">
    <cfargument name="subroutineName" type="string" required="true"
        hint="The name of the Subroutine to handle." />

    <cfif isSubroutineDefined(arguments.subroutineName)>
        <cfreturn variables.handlers[arguments.subroutineName] />
    </cfif>
</cffunction>
```

```
<cfelseif isObject(getParent()) AND getParent().isSubroutineDefined(arguments.subroutineName)>
  <cfreturn getParent().getSubroutineHandler(arguments.subroutineName) />
<cfelse>
  <cfthrow type="MachII.framework.SubroutineHandlerNotDefined"
    message="SubroutineHandler for subroutine '#arguments.subroutineName#' is not defined." />
</cfif>
</cffunction>
```

getSubroutineNames

```
public array getSubroutineNames( )
```

Returns an array of subroutine names.

Parameters:

Code:

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

init

```
public SubroutineManager init( AppManager appManager, [any parentSubroutineManager=""] )
```

Initialization function called by the framework.

Parameters:

AppManager appManager
[any parentSubroutineManager=""]

Code:

```
<cffunction name="init" access="public" returntype="SubroutineManager" output="false"
```

```

hint="Initialization function called by the framework.">
<cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
<cfargument name="parentSubroutineManager" type="any" required="false" default=""
    hint="Optional argument for a parent subroutine manager. If there isn't one default to empty string." />

<cfset setAppManager(arguments.appManager) />

<cfif isObject(arguments.parentSubroutineManager)>
    <cfset setParent(arguments.parentSubroutineManager) />
</cfif>

<cfreturn this />
</cffunction>

```

isSubroutineDefined

public boolean isSubroutineDefined(string subroutineName)

Returns true if a SubroutineHandler for the named Subroutine is defined; otherwise false.

Parameters:

string subroutineName

Code:

```

<cffunction name="isSubroutineDefined" access="public" returntype="boolean" output="false"
    hint="Returns true if a SubroutineHandler for the named Subroutine is defined; otherwise false.">
    <cfargument name="subroutineName" type="string" required="true"
        hint="The name of the Subroutine to handle." />
    <cfreturn StructKeyExists(variables.handlers, arguments.subroutineName) />
</cffunction>

```

loadXml

public void loadXml(string configXML, [boolean override="false"])

Loads xml for the manager.

Parameters:

string configXML
 [boolean override="false"]

Code:

```

<cffunction name="loadXml" access="public" returntype="void" output="false"
  hint="Loads xml for the manager.">
  <cfargument name="configXML" type="string" required="true" />
  <cfargument name="override" type="boolean" required="false" default="false" />

  <cfset var subroutineNodes = "" />
  <cfset var subroutineHandler = "" />
  <cfset var subroutineName = "" />
  <cfset var commandNode = "" />
  <cfset var command = "" />
  <cfset var hasParent = isObject(getParent()) />
  <cfset var mapping = "" />
  <cfset var i = 0 />
  <cfset var j = 0 />

  <cfif NOT arguments.override>
    <cfset subroutineNodes = XMLSearch(arguments.configXML, "mach-ii/subroutines/subroutine") />
  <cfelse>
    <cfset subroutineNodes = XMLSearch(arguments.configXML, "../subroutines/subroutine") />
  </cfif>

  <cfloop from="1" to="#ArrayLen(subroutineNodes)#" index="i">
    <cfset subroutineName = subroutineNodes[i].xmlAttributes["name"] />

    <cfif hasParent AND arguments.override AND StructKeyExists(subroutineNodes[i].xmlAttributes, "overrideAction")
      <cfif subroutineNodes[i].xmlAttributes["overrideAction"] EQ "useParent">
        <cfset removeSubroutine(subroutineName) />
      <cfelseif subroutineNodes[i].xmlAttributes["overrideAction"] EQ "addFromParent">

        <cfif StructKeyExists(subroutineNodes[i].xmlAttributes, "mapping">
          <cfset mapping = subroutineNodes[i].xmlAttributes["mapping"] />
        <cfelse>
          <cfset mapping = subroutineName />

```

```

        </cfif>

        <cfif NOT getParent().isSubroutineDefined(mapping)>
            <cfthrow type="MachII.framework.overrideSubroutineNotDefined"
                message="An subroutine named '#mapping#' cannot be found in the parent s
        </cfif>

        <cfset addSubroutineHandler(subroutineName, getParent().getSubroutineHandler(mapping), ar
    </cfif>

    <cfelse>
        <cfset subroutineHandler = CreateObject("component", "MachII.framework.SubroutineHandler").init(
        <cfloop from="1" to="#ArrayLen(subroutineNodes[i].XMLChildren)#" index="j">
            <cfset commandNode = subroutineNodes[i].XMLChildren[j] />
            <cfset command = createCommand(commandNode) />
            <cfset subroutineHandler.addCommand(command) />
        </cfloop>

        <cfset addSubroutineHandler(subroutineName, subroutineHandler, arguments.override) />
    </cfif>
</cfloop>
</cffunction>

```

removeSubroutine

```
public void removeSubroutine( string subroutineName )
```

Removes a subroutine. Does NOT remove from a parent.

Parameters:

string subroutineName

Code:

```

<cffunction name="removeSubroutine" access="public" returntype="void" output="false"
    hint="Removes a subroutine. Does NOT remove from a parent.">
    <cfargument name="subroutineName" type="string" required="true"
        hint="The name of the Subroutine to handle." />

```

```
<cfset StructDelete(variables.handlers, arguments.subroutineName, false) />
</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>
```

setParent

```
public void setParent( SubroutineManager parentSubroutineManager )
```

Returns the parent SubroutineManager instance this SubroutineManager belongs to.

Parameters:

SubroutineManager parentSubroutineManager

Code:

```
<cffunction name="setParent" access="public" returntype="void" output="false"
    hint="Returns the parent SubroutineManager instance this SubroutineManager belongs to.">
    <cfargument name="parentSubroutineManager" type="MachII.framework.SubroutineManager" required="true" />
    <cfset variables.parentSubroutineManager = arguments.parentSubroutineManager />
</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 string	buildUrl(string eventName, [any urlParameters=""], [string urlBase])	Builds a framework specific url and automatically escapes entities for html display.
public string	buildUrlToModule(string moduleName, string eventName, [any urlParameters=""], [string urlBase])	Builds a framework specific url with module name and automatically escapes entities for html display.
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	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()
public PropertyManager	getPropertyManager()	Gets the components PropertyManager instance.

Method Summary

private void	setAppManager(AppManager appManager) Returns the AppManager instance this ViewContext belongs to.
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**buildUrl**

public string buildUrl(string eventName, [any urlParameters=""], [string urlBase])

Builds a framework specific url and automatically escapes entities for html display.

Parameters:

string eventName
[any urlParameters=""]
[string urlBase]

Code:

```
<cffunction name="buildUrl" access="public" returnType="string" output="false"
  hint="Builds a framework specific url and automatically escapes entities for html display.">
  <cfargument name="eventName" type="string" required="true"
    hint="Name of the event to build the url with." />
  <cfargument name="urlParameters" type="any" required="false" default=""
    hint="Name/value pairs (urlArg1=value1|urlArg2=value2) to build the url with or a struct of data." />
  <cfargument name="urlBase" type="string" required="false"
    hint="Base of the url. Defaults to the value of the urlBase property." />
```

```

        <cfset arguments.moduleName = getAppManager().getModuleName() />
        <cfreturn HtmlEditFormat(getAppManager().getRequestManager().buildUrl(argumentcollection=arguments)) />
    </cffunction>

```

buildUrlToModule

```
public string buildUrlToModule( string moduleName, string eventName, [any urlParameters=""], [string urlBase] )
```

Builds a framework specific url with module name and automatically escapes entities for html display.

Parameters:

```

string moduleName
string eventName
[any urlParameters=""]
[string urlBase]

```

Code:

```

<cffunction name="buildUrlToModule" access="public" returntype="string" output="false"
    hint="Builds a framework specific url with module name and automatically escapes entities for html display.">
    <cfargument name="moduleName" type="string" required="true"
        hint="Name of the module to build the url with. Defaults to current module if empty string." />
    <cfargument name="eventName" type="string" required="true"
        hint="Name of the event to build the url with." />
    <cfargument name="urlParameters" type="any" required="false" default=""
        hint="Name/value pairs (urlArg1=value1|urlArg2=value2) to build the url with or a struct of data." />
    <cfargument name="urlBase" type="string" required="false"
        hint="Base of the url. Defaults to the value of the urlBase property." />
    <cfreturn HtmlEditFormat(getAppManager().getRequestManager().buildUrl(argumentcollection=arguments)) />
</cffunction>

```

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="getManager" access="public" returnType="MachII.framework.AppManager" output="false"
    hint="Sets the AppManager instance this ViewContext belongs to.">
    <cfreturn variables.appManager />
</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" />
    <cfreturn getAppManager().getViewManager().getViewPath(arguments.viewName) />
</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

```
public PropertyManager getPropertyManager( )
```

Gets the components PropertyManager instance.

Parameters:

Code:

```
<cffunction name="getPropertyManager" access="public" 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

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

Returns the AppManager instance this ViewContext belongs to.

Parameters:

AppManager appManager

Code:

```
<cffunction name="setAppManager" access="private" 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>
```

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 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>
```

ViewManager

Package: MachII.framework

Manages registered views for the framework.

Method Summary

public ViewManager	init(AppManager appManager, [any parentViewManager=""])	Initialization function called by the framework.
public void	configure()	Prepares the manager for use.
public AppManager	getAppManager()	
public any	getParent()	Sets the parent ViewManager instance this ViewManager belongs to. It will return empty string if no parent is defined.
public string	getViewPath(string viewName)	Gets the view path by view name.
public boolean	isViewDefined(string viewName)	Checks if the view is defined.
public void	loadXml(string configXML, [boolean override="false"])	Loads xml for the manager.
public void	setAppManager(AppManager appManager)	
public void	setParent(ViewManager parentViewManager)	Returns the parent ViewManager instance this ViewManager belongs to.

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>
```

getParent

public any getParent()

Sets the parent ViewManager instance this ViewManager belongs to. It will return empty string if no parent is defined.

Parameters:

Code:

```
<cffunction name="getParent" access="public" returntype="any" output="false"
    hint="Sets the parent ViewManager instance this ViewManager belongs to. It will return empty string if no parent
    <cfreturn variables.parentViewManager />
</cffunction>
```

getViewPath

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

Gets the view path by view name.

Parameters:

string viewName

Code:

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

    <cfif isViewDefined(arguments.viewName)>
        <cfreturn variables.viewPaths[arguments.viewName] />
    <cfelseif isObject(getParent()) AND getParent().isViewDefined(arguments.viewName)>
        <cfreturn getParent().getViewPath(arguments.viewName) />
    <cfelse>
        <cfthrow type="MachII.framework.ViewNotDefined"
            message="View with name '#arguments.viewName#' is not defined." />
    </cfif>
</cffunction>
```

init

```
public ViewManager init( AppManager appManager, [any parentViewManager=""] )
```

Initialization function called by the framework.

Parameters:

AppManager appManager
[any parentViewManager=""]

Code:

```
<cffunction name="init" access="public" returntype="ViewManager" output="false"
    hint="Initialization function called by the framework.">
    <cfargument name="appManager" type="MachII.framework.AppManager" required="true" />
    <cfargument name="parentViewManager" type="any" required="false" default=""
        hint="Optional argument for a parent view manager. If there isn't one default to empty string." />

    <cfset setAppManager(arguments.appManager) />

    <cfif isObject(arguments.parentViewManager)>
        <cfset setParent(arguments.parentViewManager) />
    </cfif>

    <cfreturn this />
</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. Does not check parent ViewManager." />
    <cfreturn StructKeyExists(variables.viewPaths, arguments.viewName) />
</cffunction>

```

loadXml

```
public void loadXml( string configXML, [boolean override="false"] )
```

Loads xml for the manager.

Parameters:

```
string configXML
[boolean override="false"]
```

Code:

```

<cffunction name="loadXml" access="public" returntype="void" output="false"
    hint="Loads xml for the manager.">
    <cfargument name="configXML" type="string" required="true" />
    <cfargument name="override" type="boolean" required="false" default="false" />

    <cfset var viewNodes = "" />
    <cfset var name = "" />
    <cfset var page = "" />
    <cfset var hasParent = isObject(getParent()) />
    <cfset var mapping = "" />
    <cfset var i = 0 />

    <cfif NOT arguments.override>
        <cfset viewNodes = XMLSearch(arguments.configXML, "mach-ii/page-views/page-view") />
    <cfelse>
        <cfset viewNodes = XMLSearch(arguments.configXML, "./page-views/page-view") />
    </cfif>

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

```

```

    <cfif hasParent AND arguments.override AND StructKeyExists(viewNodes[i].xmlAttributes, "overrideAction")>
        <cfif viewNodes[i].xmlAttributes["overrideAction"] EQ "useParent">
            <cfset StructDelete(variables.viewPaths, name, false) />
        <cfelseif viewNodes[i].xmlAttributes["overrideAction"] EQ "addFromParent">
            <cfif StructKeyExists(viewNodes[i].xmlAttributes, "mapping")>
                <cfset mapping = viewNodes[i].xmlAttributes["mapping"] />
            <cfelse>
                <cfset mapping = name />
            </cfif>
            <cfif NOT getParent().isViewDefined(mapping)>
                <cfthrow type="MachII.framework.overrideViewNotDefined"
                    message="An view named '#mapping#' cannot be found in the parent view ma
            </cfif>
            <cfset variables.viewPaths[name] = mapping />
        </cfif>
    <cfelse>
        <cfif StructKeyExists(viewNodes[i].xmlAttributes, "useParentAppRoot") AND viewNodes[i].xmlAttribu
            <cfset page = getAppManager().getParent().getPropertyManager().getProperty("applicationR
        <cfelse>
            <cfset page = getAppManager().getPropertyManager().getProperty("applicationRoot") & view
        </cfif>
        <cfset variables.viewPaths[name] = page />
    </cfif>
</cfloop>
</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>
```

setParent

```
public void setParent( ViewManager parentViewManager )
```

Returns the parent ViewManager instance this ViewManager belongs to.

Parameters:

ViewManager parentViewManager

Code:

```
<cffunction name="setParent" access="public" returntype="void" output="false"
    hint="Returns the parent ViewManager instance this ViewManager belongs to.">
    <cfargument name="parentViewManager" type="MachII.framework.ViewManager" required="true" />
    <cfset variables.parentViewManager = arguments.parentViewManager />
</cffunction>
```

framework.commands

AnnounceCommand

Package: MachII.framework.commands

Inherits from: framework.Command

An Command for announcing an event.

Method Summary

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

Methods inherited from framework.Command: 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 = StructNew() />

  <cfif isCopyEventArgs()>
    <cfset eventArgs = arguments.event.getArgs() />
  </cfif>

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

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

getEventName

```
private string getEventName( )
```

Parameters:

Code:

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

getModuleName

```
private string getModuleName( )
```

Parameters:

Code:

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

init

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

Used by the framework for initialization.

Parameters:

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

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" />
    <cfargument name="moduleName" type="string" required="false" default="" />
```

```
<cfset setName(arguments.eventName) />
<cfset setCopyEventArgs(arguments.copyEventArgs) />
<cfset setModuleName(arguments.moduleName) />

<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>
```

setModuleName

private void setModuleName(string moduleName)

Parameters:

string moduleName

Code:

```
<cffunction name="setModuleName" access="private" returnType="void" output="false">
    <cfargument name="moduleName" type="string" required="true" />
    <cfset variables.moduleName = arguments.moduleName />
</cffunction>
```

EventArgCommand

Package: MachII.framework.commands

Inherits from: framework.Command

An Command 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.Command: 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>
```

EventBeanCommand

Package: MachII.framework.commands

Inherits from: framework.Command

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

Method Summary

public EventBean- Command	init(string beanName, string beanType, string beanFields, boolean reinit) 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()
private boolean	getReinit()
public boolean	isBeanFieldsDefined()
private void	setBeanFields(string beanFields)
private void	setBeanName(string beanName)
private void	setBeanType(string beanType)
private void	setBeanUtil(BeanUtil beanUtil)
private void	setReinit(boolean reinit)

Methods inherited from framework.Command: 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 = "" />
  <cfset var reinit = TRUE />

  <cfif NOT getReinit() AND arguments.event.isArgDefined(getBeanName())>
    <cfset bean = arguments.event.getArg(getBeanName()) />

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

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

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

    <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>
```

getReinit

```
private boolean getReinit( )
```

Parameters:

Code:

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

init

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

Used by the framework for initialization.

Parameters:

```
string beanName  
string beanType  
string beanFields  
boolean reinit
```

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" />  
    <cfargument name="reinit" type="boolean" required="true" />  
  
    <cfset setBeanName(arguments.beanName) />  
    <cfset setBeanType(arguments.beanType) />  
    <cfset setBeanFields(arguments.beanFields) />  
    <cfset setReinit(arguments.reinit) />  
  
    <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>
```

setReinit

```
private void setReinit( boolean reinit )
```

Parameters:

boolean reinit

Code:

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

EventMappingCommand

Package: MachII.framework.commands

Inherits from: framework.Command

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

Method Summary

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

Methods inherited from framework.Command: 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(), getMappingModule()) />

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

getEventName

```
private string getEventName( )
```

Parameters:

Code:

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

getMappingModule

private string getMappingModule()

Parameters:

Code:

```
<cffunction name="getMappingModule" access="private" returntype="string" output="false">
    <cfreturn variables.mappingModule />
</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, string mappingModule)

Used by the framework for initialization.

Parameters:

string eventName
string mappingName
string mappingModule

Code:

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

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

  <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>
```

setMappingModule

```
private void setMappingModule( string mappingModule )
```

Parameters:

string mappingModule

Code:

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

ExecuteCommand

Package: MachII.framework.commands

Inherits from: framework.Command

An Command for executing a subroutine.

Method Summary

public ExecuteCommand	init(string subroutineName) Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext) Executes the command.
private string	getSubroutineName()
private void	setSubroutineName(string subroutineName)

Methods inherited from framework.Command: 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" />
    <cfreturn arguments.eventContext.executeSubroutine(getSubroutineName(), arguments.event) />
</cffunction>
```

getSubroutineName

```
private string getSubroutineName( )
```

Parameters:**Code:**

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

init

```
public ExecuteCommand init( string subroutineName )
```

Used by the framework for initialization.

Parameters:

string subroutineName

Code:

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

  <cfset setSubroutineName(arguments.subroutineName) />

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

setSubroutineName

```
private void setSubroutineName( string subroutineName )
```

Parameters:

string subroutineName

Code:

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

FilterCommand

Package: MachII.framework.commands

Inherits from: framework.Command

An Command 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.Command: 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.Command

An Command 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.Command: 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.Command

An Command for redirecting.

Method Summary

public RedirectCommand	init(string eventName, string eventParameter, string redirectPersistParameter, [string moduleName=""], [string url=""], [string args=""], [boolean persist="false"], [string persistArgs=""], [string statusType="temporary"])	Used by the framework for initialization.
public boolean	execute(Event event, EventContext eventContext)	Executes the command.
private string	getArgs()	
private string	getEventName()	
private string	getEventParameter()	
private string	getModuleName()	
private boolean	getPersist()	
private string	getPersistArgs()	
private string	getRedirectPersistParameter()	
private string	getStatusType()	
private string	getUrl()	
private string	makeRedirectUrl(Event event, EventContext eventContext)	Assembles the redirect url.

Method Summary

private void	savePersistEventData(Event event, EventContext eventContext) Saves persisted event data and returns the persistId.
private void	setArgs(string args)
private void	setEventName(string eventName)
private void	setEventParameter(string eventParameter)
private void	setModuleName(string moduleName)
private void	setPersist(boolean persist)
private void	setPersistArgs(string persistArgs)
private void	setRedirectPersistParameter(string redirectPersistParameter)
private void	setStatusType(string statusType)
private void	setUrl(string url)

Methods inherited from framework.Command: 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 = "" />
  <cfset var statusType = getStatusType() />

  <cfif getPersist()>
    <cfset savePersistEventData(arguments.event, arguments.eventContext) />
  </cfif>

  <cfset redirectUrl = makeRedirectUrl(arguments.event, arguments.eventContext) />

  <cfset arguments.eventContext.clearEventQueue() />

  <cfif statusType EQ "permanent">
    <cfheader statuscode="301" statustext="Moved Permanently" />
    <cfheader name="Location" value="#redirectUrl#" />
    <cfexit />
  <cfelseif statusType EQ "prg">
    <cfheader statuscode="303" statustext="See Other" />
    <cfheader name="Location" value="#redirectUrl#" />
    <cfexit />
  <cfelse>

    <cflocation url="#redirectUrl#" addtoken="no" />
  </cfif>

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

getArgs

```
private string getArgs( )
```

Parameters:

Code:

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

getEventName

private string getEventName()

Parameters:

Code:

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

getEventParameter

private string getEventParameter()

Parameters:

Code:

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

getModuleName

private string getModuleName()

Parameters:

Code:

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

getPersist

private boolean getPersist()

Parameters:

Code:

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

getPersistArgs

private string getPersistArgs()

Parameters:

Code:

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

getRedirectPersistParameter

```
private string getRedirectPersistParameter( )
```

Parameters:

Code:

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

getStatusType

```
private string getStatusType( )
```

Parameters:

Code:

```
<cffunction name="getStatusType" access="private" returntype="string" output="false">  
    <cfreturn variables.statusType />  
</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 eventParameter, string redirectPersistParameter, [string moduleName=""], [string url=""], [string
args=""], [boolean persist="false"], [string persistArgs=""], [string statusType="temporary"] )
```

Used by the framework for initialization.

Parameters:

```
string eventName
string eventParameter
string redirectPersistParameter
[string moduleName=""]
[string url=""]
[string args=""]
[boolean persist="false"]
[string persistArgs=""]
[string statusType="temporary"]
```

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="eventParameter" type="string" required="true" />
  <cfargument name="redirectPersistParameter" type="string" required="true" />
  <cfargument name="moduleName" type="string" required="false" default="" />
  <cfargument name="url" type="string" required="false" default="" />
  <cfargument name="args" type="string" required="false" default="" />
  <cfargument name="persist" type="boolean" required="false" default="false" />
  <cfargument name="persistArgs" type="string" required="false" default="" />
  <cfargument name="statusType" type="string" required="false" default="temporary" />

  <cfset setEventName(arguments.eventName) />
  <cfset setEventParameter(arguments.eventParameter) />
```

```
<cfset setRedirectPersistParameter(arguments.redirectPersistParameter) />
<cfset setModuleName(arguments.moduleName) />
<cfset setUrl(arguments.url) />
<cfset setArgs(arguments.args) />
<cfset setPersist(arguments.persist) />
<cfset setPersistArgs(arguments.persistArgs) />
<cfset setStatusType(arguments.statusType) />

<cfif getPersist()>
    <cfset setArgs(ListAppend(getArgs(), getRedirectPersistParameter())) />
</cfif>

<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 = "" />
    <cfset var params = StructNew() />
    <cfset var args = getArgs() />
    <cfset var i = "" />
```

```

    <cfloop list="#args#" index="i" delimiters=", ">
        <cfif arguments.event.isArgDefined(i) AND IsSimpleValue(arguments.event.getArg(i))>
            <cfset params[i] = arguments.event.getArg(i) />
        </cfif>
    </cfloop>

    <cfset redirectUrl = arguments.eventContext.getAppManager().getRequestManager().buildUrl(getModuleName(), getEventName()) />

    <cfreturn redirectUrl />
</cffunction>

```

savePersistEventData

```
private void savePersistEventData( Event event, EventContext eventContext )
```

Saves persisted event data and returns the persistId.

Parameters:

Event event
EventContext eventContext

Code:

```

<cffunction name="savePersistEventData" access="private" returntype="void" output="false"
    hint="Saves persisted event data and returns the persistId.">
    <cfargument name="event" type="MachII.framework.Event" required="true" />
    <cfargument name="eventContext" type="MachII.framework.EventContext" required="true" />

    <cfset var args = StructNew() />
    <cfset var persistArgs = getPersistArgs() />
    <cfset var persistId = "" />
    <cfset var i = "" />

    <cfif NOT ListLen(persistArgs)>
        <cfset args = arguments.event.getArgs() />

        <cfif StructKeyExists(args, getEventParameter())>
            <cfset StructDelete(args, getEventParameter(), FALSE) />
        </cfif>
    </cfif>

```

```
        </cfif>
    <cfelse>
        <cfloop list="#persistArgs#" index="i" delimiters=",">
            <cfif arguments.event.isArgDefined(i)>
                <cfset args[i] = arguments.event.getArg(i) />
            </cfif>
        </cfloop>
    </cfif>

    <cfset persistId = arguments.eventContext.getAppManager().getRequestManager().savePersistEventData(args) />
    <cfset arguments.event.setArg(getRedirectPersistParameter(), persistId) />
</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>
```

setEventParameter

private void setEventParameter(string eventParameter)

Parameters:

string eventParameter

Code:

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

setModuleName

private void setModuleName(string moduleName)

Parameters:

string moduleName

Code:

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

```
</cffunction>
```

setPersist

```
private void setPersist( boolean persist )
```

Parameters:

boolean persist

Code:

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

setPersistArgs

```
private void setPersistArgs( string persistArgs )
```

Parameters:

string persistArgs

Code:

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

setRedirectPersistParameter

private void setRedirectPersistParameter(string redirectPersistParameter)

Parameters:

string redirectPersistParameter

Code:

```
<cffunction name="setRedirectPersistParameter" access="private" returnType="void" output="false">
    <cfargument name="redirectPersistParameter" type="string" required="true" />
    <cfset variables.redirectPersistParameter = arguments.redirectPersistParameter />
</cffunction>
```

setStatusType

private void setStatusType(string statusType)

Parameters:

string statusType

Code:

```
<cffunction name="setStatusType" access="private" returnType="void" output="false">
    <cfargument name="statusType" type="string" required="true" />
    <cfset variables.statusType = arguments.statusType />
</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.Command

An Command 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.Command: 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() DEPRECATED. Used by the framework for initialization. Do not override.
public void	invokeListener(Event event, Listener listener, string method, [string resultKey=""], [string resultArg=""]) DEPRECATED. Invokes the Listener.

Method Detail

init

```
public CFCInvoker_Event init( )
```

DEPRECATED. Used by the framework for initialization. Do not override.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="CFCInvoker_Event" output="false"
```

```

        hint="DEPRECATED. 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=""] )
```

DEPRECATED. Invokes the Listener.

Parameters:

Event event

Listener listener

string method

[string resultKey=""]

[string resultArg=""]

Code:

```

<cffunction name="invokeListener" access="public" returntype="void"
    hint="DEPRECATED. 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" />

    <cfif arguments.resultKey NEQ ''>
        <cfset "#arguments.resultKey#" = resultValue />
    </cfif>

    <cfthrow type="MachII.framework.deprecatedInvoker"
        message="The CFCInvoker_Event has been deprecated. Please use the EventInvoker." />

    <cfcatch type="MachII.framework.deprecatedInvoker">

    </cfcatch>
    <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() DEPRECATED. Used by the framework for initialization. Do not override.
public void	invokeListener(Event event, Listener listener, string method, [string resultKey=""], [string resultArg=""]) DEPRECATED. Invokes the Listener.

Method Detail

init

```
public CFCInvoker_EventArgs init( )
```

DEPRECATED. Used by the framework for initialization. Do not override.

Parameters:

Code:

```
<cffunction name="init" access="public" returntype="CFCInvoker_EventArgs" output="false"
```

```

        hint="DEPRECATED. 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=""] )
```

DEPRECATED. Invokes the Listener.

Parameters:

Event event

Listener listener

string method

[string resultKey=""]

[string resultArg=""]

Code:

```

<cffunction name="invokeListener" access="public" returntype="void"
    hint="DEPRECATED. 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" />

    <cfif arguments.resultKey NEQ ''>
        <cfset "#arguments.resultKey#" = resultValue />
    </cfif>

    <cfthrow type="MachII.framework.deprecatedInvoker"
            message="The CFCInvoker_EventArgs has been deprecated. Please use the EventArgsInvoker." />

    <cfcatch type="MachII.framework.deprecatedInvoker">

    </cfcatch>
    <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" />

    <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="Listener: '#getMetadata(listener).name#' Method: '#arguments.metl
        </cfif>
        <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#"
      resultKey="#arguments.resultKey#"
      resultArg="#arguments.resultArg#"
    />
  </cftry>
</cffunction>
```

```
        returnvariable="resultValue" />

    <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="Listener: '#getMetadata(listener).name#' Method: '#arguments.metl
        </cfif>
        <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: setParameters , hasParameter , buildUrlToModule , isParameterDefined , buildUrl , announceEventInModule , setAppManager , getParameter , getProperty , getParameters , setProperty , getPropertyManager , bindValue , get-AppManager , setParameter , announceEvent

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 module, 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	computeModuleName(EventContext eventContext, string point) Computes the module 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, string requestModuleName) Gets the trace information and formats for on-screen or HTML commented display.
private string	getDebugModeOnly()
private boolean	getDisplayCommented()
private string	getFileName()

Method Summary

private numeric	getHighlightLongTimings()
private boolean	getIsInitialTrace() Gets the initial trace flag from the request._MachIITracePlugin.
private string	getMachIIVersion() Gets a nice version number instead of just numbers.
private string	getSuppressTraceArg()
private numeric	getTick() Gets the current tick from the request._MachIITracePlugin.
private numeric	getTickStart() Gets the tick start from the request._MachIITracePlugin.
private array	getTraceInfo() Gets the trace info array from the request._MachIITracePlugin.
private string	getTraceMode()
private boolean	getTraceRequest() Gets the trace request from the request._MachIITracePlugin.
public void	handleException(EventContext eventContext, Exception exception) Runs a trace when an exception occurs (before exception event is handled).
private boolean	hasIsInitialTrace() Checks if the initial trace flag exists in the request._MachIITracePlugin.
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.

Method Summary

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.
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	setDebugModeOnly(string debugModeOnly)	
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._MachIITracePlugin.
private void	setSuppressTraceArg(string suppressTraceArg)	
private void	setTick(numeric tick)	Sets the current tick in the request._MachIITracePlugin.
private void	setTickStart(numeric tickStart)	Sets the tick start in the request._MachIITracePlugin.
private void	setTraceInfo([array traceInfo])	Sets the trace info array in the request._MachIITracePlugin.
private void	setTraceMode(string traceMode)	

Method Summary

private void	setTraceRequest([boolean traceRequest]) Sets the trace request request._MachIITracePlugin.
private boolean	shouldTrace(boolean suppressTrace) Checks if we should trace
private void	throwUsageException(string message, [string detail="No details."]) 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: setParameters , hasParameter , buildUrlToModule , isParameterDefined , buildUrl , announceEventInModule , setAppManager , getParameter , getProperty , getParameters , setProperty , getPropertyManager , bindValue , get-AppManager , setParameter , announceEvent

Method Detail

appendTrace

private void appendTrace(string event, string module, string point, string timing)

Appends a trace to the trace information array or to the log file.

Parameters:

string event
string module

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="module" type="string" required="true"
    hint="Name of module 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.module = arguments.module />
  <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}) - <cfif Len(trace.module)>#{trace.module}#:</cfif>#tr
  </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>
```

computeModuleName

```
private string computeModuleName( EventContext eventContext, string point )
```

Computes the module name for this trace.

Parameters:

EventContext eventContext
string point

Code:

```
<cffunction name="computeModuleName" access="private" returntype="string" output="false"
    hint="Computes the module 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().getModuleName() />
    <cfelse>
        <cfreturn "" />
    </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.">

  <cfif isParameterDefined("traceMode")>

    <cfif NOT ListFindNoCase("display,file,both,none", getParameter("traceMode"))>
      <cfset throwUsageException("The TracePlugin {traceMode} parameter must be display, file, both or
    </cfif>
    <cfelse>
      <cfset setTraceMode(getParameter("traceMode")) />
    </cfelse>
  </cfif>
  <cfif isParameterDefined("displayCommented")>
    <cfif NOT IsBoolean(getParameter("displayCommented"))>
      <cfset throwUsageException("The TracePlugin {displayCommented} parameter must be a boolean value
    </cfif>
    <cfelse>
      <cfset setDisplayCommented(getParameter("displayCommented")) />
    </cfelse>
  </cfif>
  <cfif isParameterDefined("highlightLongTimings")>
    <cfif NOT Len(getParameter("highlightLongTimings")) OR NOT IsTrueNumeric(getParameter("highlightLongTimi
    </cfif>
    <cfelse>
      <cfset setHighlightLongTimings(getParameter("highlightLongTimings")) />
    </cfelse>
  </cfif>
  <cfif isParameterDefined("fileName")>
    <cfif NOT Len(getParameter("fileName"))>
      <cfset throwUsageException("The TracePlugin {fileName} parameter must not be blank. Please set a
    </cfif>
    <cfelse>
      <cfset setFilename(getParameter("fileName")) />
    </cfelse>
  </cfif>
  <cfif isParameterDefined("suppressTraceArg")>
    <cfif NOT Len(getParameter("suppressTraceArg"))>
      <cfset throwUsageException("The TracePlugin {suppressTraceArg} parameter must not be blank. Pleas
    </cfif>
    <cfelse>
      <cfset setSuppressTraceArg(getParameter("suppressTraceArg")) />
    </cfelse>
  </cfif>
  <cfif isParameterDefined("debugModeOnly")>
    <cfif NOT Len(getParameter("debugModeOnly"))>
      <cfset throwUsageException("The TracePlugin {debugOnlyMode} parameter must not be blank. Please s
    </cfif>
    <cfelse>
```

```

        <cfset setDebugModeOnly(getParameter("debugModeOnly"))>
    </cfif>
</cfif>
</cffunction>

```

displayTraceInfo

private string displayTraceInfo(array traceInfo, string requestEventName, string requestModuleName)

Gets the trace information and formats for on-screen or HTML commented display.

Parameters:

array traceInfo
string requestEventName
string requestModuleName

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.">
    <cfargument name="requestModuleName" 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 = "" />
    <cfset var timing = "" />

    <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
            -->
        </cfsavecontent>
    </cfif>

```

```
*****
<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">
<style type="text/css"><!--
##MachIITraceDisplay {
    color: #000;
    background-color: #FFF;
}
##MachIITraceDisplay h3 {
    color: #000;
}
##MachIITraceDisplay table {
    border: 1px solid #D0D0D0;
    padding: 0.5em;
    width:100%;
}
##MachIITraceDisplay td {
    vertical-align: top;
}
##MachIITraceDisplay td.lineBottom {
    border-bottom: 1px solid #000;
}
##MachIITraceDisplay td.lineTop {
    border-top: 1px solid #000;
}
##MachIITraceDisplay .shade {
    background-color: #F5F5F5;
}
##MachIITraceDisplay ul li {
    margin-left:15px;
}
##MachIITraceDisplay .small {
    font-size: 0.9em;
}

```

```

        ##MachIITraceDisplay .red {
            color: ##FF0000;
        }
        ##MachIITraceDisplay .green {
            color: ##6BB300;
        }
        ##MachIITraceDisplay .strong {
            font-weight: bold;
        }
-->
</style>
<h3>Mach-II Trace Information</h3>
<table>
    <tr>
        <td class="lineBottom strong" style="width:65%;">Event Name</td>
        <td class="lineBottom strong" style="width:20%;">Trace Point</td>
        <td class="lineBottom strong" style="width:15%;">* Average Time</td>
    </tr>
<cfloop from="1" to="#ArrayLen(traceInfo)-1#" index="i">
    <cfif arguments.traceInfo[i].point NEQ "preView">
    <tr <cfif i MOD 2> class="shade"</cfif>>
        <td<cfif ListFindNoCase("preEvent,postProcess", arguments.traceInfo[i].point)> c
        <cfif NOT ListFindNoCase("preView,postView,postEvent", arguments.traceInfo[i].po
            <cfif Len(arguments.traceInfo[i].module)>#arguments.traceInfo[i].module#
        <cfelse>
            &nbsp;
        </cfif>
        </td>
        <td class="small<cfif ListFindNoCase("preEvent,postProcess", arguments.traceInfo
        <cfif arguments.traceInfo[i].point EQ "postView">
            view
        <cfelse>
            #arguments.traceInfo[i].point#
        </cfif>
        </td>
    <cfif arguments.traceInfo[i].point EQ "postView">
        <cfset timing = arguments.traceInfo[i].timing + arguments.traceInfo[i-1].timing>
    <cfelse>
        <cfset timing = arguments.traceInfo[i].timing />
    </cfif>
    <cfif getHighlightLongTimings() NEQ 0 AND timing GTE getHighlightLongTimings()>
        <td class="small red strong<cfif ListFindNoCase("preEvent,postProcess", argument
    <cfelse>
        <td class="small<cfif ListFindNoCase("preEvent,postProcess", arguments.traceInfo

```

```

        </cfif>
        </tr>
        </cfif>
    </cfloop>
    <tr>
        <td colspan="2" class="lineTop"><em>#arguments.traceInfo[traceInfoArrLen].event#</em>
        <td class="lineTop" style="text-align: right;"><em>#arguments.traceInfo[traceIn
    </tr>
    <cfif getHighlightLongTimings()>
    <tr>
        <td colspan="3" class="strong" style="text-align:right;">* Timings over #getHigh
    </tr>
    </cfif>
    </table>
    <h3>General Information</h3>
    <table>
        <tr class="shade">
            <td class="lineTop strong">Request Event Name</td>
            <td class="lineTop">#arguments.requestEventName#</td>
        </tr>
        <tr>
            <td class="strong">Request Module Name</td>
            <td>#arguments.requestModuleName#</td>
        </tr>
        <tr class="shade">
            <td class="strong">Mach-II Version</td>
            <td>#getMachIIVersion()#</td>
        </tr>
        <tr>
            <td class="strong">Timestamp</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>

```

getDebugModeOnly

private string getDebugModeOnly()

Parameters:

Code:

```
<cffunction name="getDebugModeOnly" access="private" returntype="string" output="false">
    <cfreturn variables.instance.debugModeOnly />
</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._MachIITracePlugin.

Parameters:

Code:

```
<cffunction name="getIsInitialTrace" access="private" returntype="boolean" output="false"
    hint="Gets the initial trace flag from the request._MachIITracePlugin.">
    <cftry>
        <cfreturn request._MachIITracePlugin.isInitialTrace />
        <cfcatch type="expression">
            <cfset throwUsageException("Required request scope variable missing.", "Do not delete request._Ma
        </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="#ListLast(version, ".")#">
    <cfcase value="0">
      <cfset release = "Bleeding Edge Release - Unknown build" />
    </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 = "Bleeding Edge Release - Build " & ListLast(version, ".") />
    </cfdefaultcase>
  </cfswitch>
</cffunction>
```

```
        </cfdefaultcase>
    </cfswitch>

    <cfreturn Left(version, Len(version) - Len(ListLast(version, ".")) - 1) & " " & 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._MachIITracePlugin.

Parameters:

Code:

```
<cffunction name="getTick" access="private" returntype="numeric" output="false"
    hint="Gets the current tick from the request._MachIITracePlugin.">
    <cftry>
        <cfreturn request._MachIITracePlugin.tick />
        <cfcatch type="expression">
            <cfset throwUsageException("Required request scope variable missing.", "Do not delete request._Ma
        </cfcatch>
    </cftry>
```

```
</cffunction>
```

getTickStart

```
private numeric getTickStart( )
```

Gets the tick start from the request._MachIITracePlugin.

Parameters:

Code:

```
<cffunction name="getTickStart" access="private" returntype="numeric" output="false"
  hint="Gets the tick start from the request._MachIITracePlugin.">
  <cftry>
    <cfreturn request._MachIITracePlugin.tickStart />
    <cfcatch type="expression">
      <cfset throwUsageException("Required request scope variable missing.", "Do not delete request._MachIITracePlugin.tickStart") />
    </cfcatch>
  </cftry>
</cffunction>
```

getTraceInfo

```
private array getTraceInfo( )
```

Gets the trace info array from the request._MachIITracePlugin.

Parameters:

Code:

```
<cffunction name="getTraceInfo" access="private" returntype="array" output="false"
  hint="Gets the trace info array from the request._MachIITracePlugin.">
  <cftry>
    <cfreturn request._MachIITracePlugin.traceInfo />
    <cfcatch type="expression">

```

```

                <cfset throwUsageException("Required request scope variable missing.", "Do not delete request._M
            </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._MachIITracePlugin.

Parameters:

Code:

```

<cffunction name="getTraceRequest" access="private" returntype="boolean" output="false"
    hint="Gets the trace request from the request._MachIITracePlugin.">
    <cftry>
        <cfreturn request._MachIITracePlugin.traceRequest />
        <cfcatch type="expression">
            <cfset throwUsageException("Required request scope variable missing.", "Do not delete request._M
        </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(getSuppressTraceArg()))>
      <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._MachIITracePlugin.

Parameters:

Code:

```
<cffunction name="hasIsInitialTrace" access="private" returnType="boolean" output="false"
    hint="Checks if the initial trace flag exists in the request._MachIITracePlugin.">
    <cfreturn IsDefined("request._MachIITracePlugin.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 />
</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>
</cffunction>
```

```

                <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" AND ((getDebugModeOnly() AND isDebugMode()) OR NOT getDebugModeOnly())>
        <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>
```

setDebugModeOnly

```
private void setDebugModeOnly( string debugModeOnly )
```

Parameters:

string debugModeOnly

Code:

```
<cffunction name="setDebugModeOnly" access="private" returntype="void" output="false">
    <cfargument name="debugModeOnly" type="string" required="true" />
    <cfset variables.instance.debugModeOnly = arguments.debugModeOnly />
</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._MachIITracePlugin.

Parameters:

boolean isInitialTrace

Code:

```
<cffunction name="setIsInitialTrace" access="private" returnType="void" output="false"
  hint="Sets the initial trace flag in the request._MachIITracePlugin.">
  <cfargument name="isInitialTrace" type="boolean" required="true" />
  <cfset request._MachIITracePlugin.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._MachIITracePlugin.

Parameters:

numeric tick

Code:

```
<cffunction name="setTick" access="private" returntype="void" output="false"
    hint="Sets the current tick in the request._MachIITracePlugin.">
    <cfargument name="tick" type="numeric" required="true" />
    <cfset request._MachIITracePlugin.tick = arguments.tick />
</cffunction>
```

setTickStart

```
private void setTickStart( numeric tickStart )
```

Sets the tick start in the request._MachIITracePlugin.

Parameters:

numeric tickStart

Code:

```
<cffunction name="setTickStart" access="private" returntype="void" output="false"
  hint="Sets the tick start in the request._MachIITracePlugin.">
  <cfargument name="tickStart" type="numeric" required="true" />
  <cfset request._MachIITracePlugin.tickStart = arguments.tickStart />
</cffunction>
```

setTraceInfo

```
private void setTraceInfo( [array traceInfo] )
```

Sets the trace info array in the request._MachIITracePlugin.

Parameters:

[array traceInfo]

Code:

```
<cffunction name="setTraceInfo" access="private" returntype="void" output="false"
  hint="Sets the trace info array in the request._MachIITracePlugin.">
  <cfargument name="traceInfo" type="array" required="false" />
  <cfset request._MachIITracePlugin.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._MachIITracePlugin.

Parameters:

[boolean traceRequest]

Code:

```
<cffunction name="setTraceRequest" access="private" returnType="void" output="false"
    hint="Sets the trace request request._MachIITracePlugin.">
    <cfargument name="traceRequest" type="boolean" required="false" />
    <cfset request._MachIITracePlugin.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 NOT IsDefined("request._MachIITracePlugin.traceRequest") OR (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" />
  <cfif arguments.point EQ "postEvent">
    <cfset appendTrace(computeEventName(arguments.eventContext, arguments.point), computeModuleName(arguments.eventContext, arguments.point)) />
  <cfelse>
    <cfset appendTrace(computeEventName(arguments.eventContext, arguments.point), computeModuleName(arguments.eventContext, arguments.point)) />
  </cfif>
</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	setBeanAutoFields(any bean, struct fieldCollection) Sets the value of fields in a bean (determined by describeBean()) using method calls setBeanField().
public void	setBeanField(any bean, string field, any value) alue).
public void	setBeanFields(any bean, string fields, struct fieldCollection) Sets the value of fields in a bean using method calls setBeanField().

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 StructKeyExists(arguments, "initArgs")>  
    <cfset bean.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 from="1" to="#ArrayLen(metaFunctions)#" index="i">
    <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>
```

setBeanAutoFields

public void setBeanAutoFields(any bean, struct fieldCollection)

Sets the value of fields in a bean (determined by describeBean()) using method calls setBeanField().

Parameters:

any bean
struct fieldCollection

Code:

```
<cffunction name="setBeanAutoFields" access="public" returntype="void" output="false"
  hint="Sets the value of fields in a bean (determined by describeBean()) using method calls setBeanField().">
  <cfargument name="bean" type="any" required="true"
    hint="The bean to populate." />
  <cfargument name="fieldCollection" type="struct" required="true"
    hint="A struct of field names mapped to values." />

  <cfset var field = 0 />
  <cfset var map = describeBean(arguments.bean) />

  <cfloop collection="#map#" item="field">
    <cfif StructKeyExists(arguments.fieldCollection, field)>
      <cfset setBeanField(arguments.bean, field, arguments.fieldCollection[field]) />
    </cfif>
  </cfloop>
</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 )
```

Sets the value of fields in a bean using method calls setBeanField().

Parameters:

any bean
string fields
struct fieldCollection

Code:

```
<cffunction name="setBeanFields" access="public" returntype="void" output="false"
  hint="Sets the value of fields in a bean using method calls setBeanField().">
  <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 list="#arguments.fields#" index="field" 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)

Method Summary

Wraps and sets caughtException (cfcatch).

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>
```

```
</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>
```

Utils

Package: MachII.util

Utility functions for the framework.

Method Summary

public Utils	init() Initialization function called by the framework.
public string	expandRelativePath(string baseDirectory, string relativePath) Expands a relative path to an absolute path relative from a base (starting) directory.
public string	listFix(string list, [string listDelimiter=",",], [string nullString="NULL"]) Fixes a list by replacing null entries.
public any	recurseComplexValues(any node) Recurse through complex values by type.

Method Detail

expandRelativePath

```
public string expandRelativePath( string baseDirectory, string relativePath )
```

Expands a relative path to an absolute path relative from a base (starting) directory.

Parameters:

string baseDirectory
string relativePath

Code:

```
<cffunction name="expandRelativePath" access="public" returntype="string" output="false"
  hint="Expands a relative path to an absolute path relative from a base (starting) directory.">
  <cfargument name="baseDirectory" type="string" required="true"
    hint="The starting directory from which relative path is relative." />
  <cfargument name="relativePath" type="string" required="true"
    hint="The relative path to use." />

  <cfset var combinedWorkingPath = arguments.baseDirectory & arguments.relativePath />
  <cfset var pathCollection = 0 />
  <cfset var resolvedPath = "" />
  <cfset var hits = ArrayNew(1) />
  <cfset var offset = 0 />
  <cfset var i = 0 />

  <cfset combinedWorkingPath = Replace(combinedWorkingPath, "\", "/", "all") />
  <cfset combinedWorkingPath = Replace(combinedWorkingPath, "\\.\/", "/", "all") />
  <cfset pathCollection = ListToArray(combinedWorkingPath, "/") />

  <cfloop from="1" to="#ArrayLen(pathCollection)#" index="i">
    <cfif pathCollection[i] IS "..">
      <cfset ArrayAppend(hits, i) />
    </cfif>
  </cfloop>
  <cfloop from="1" to="#ArrayLen(hits)#" index="i">
    <cfset ArrayDeleteAt(pathCollection, hits[i] - offset) />
    <cfset ArrayDeleteAt(pathCollection, hits[i] - (offset + 1)) />
    <cfset offset = offset + 2 />
  </cfloop>

  <cfset resolvedPath = ArrayToList(pathCollection, "/") />

  <cfif Left(arguments.baseDirectory, 1) IS "/">
    <cfset resolvedPath = "/" & resolvedPath />
  </cfif>
```

```
<cfif Right(arguments.relativePath, 1) IS "/">
    <cfset resolvedPath = resolvedPath & "/" />
</cfif>

<cfreturn resolvedPath />
</cffunction>
```

init

```
public Utils init( )
```

Initialization function called by the framework.

Parameters:

Code:

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

listFix

```
public string listFix( string list, [string listDelimiter=",", [string nullString="NULL"] )
```

Fixes a list by replacing null entries.

Parameters:

```
string list
[string listDelimiter="
"]
[string nullString="NULL"]
```

Code:

```

<cffunction name="listFix" access="public" returntype="string" output="false"
  hint="Fixes a list by replacing null entries.">
  <cfargument name="list" type="string" required="true" />
  <cfargument name="listDelimiter" type="string" required="false" default="," />
  <cfargument name="nullString" type="string" required="false" default="NULL" />

  <cfset var delim = arguments.listDelimiter />
  <cfset var special_char_list = "\,+,*?,.,[,],^,$,(,),{,},|,-" />
  <cfset var esc_special_char_list = "\\,\\+,\\*,\\?,\\.\\,\\[,\\],\\^,\\$,\\(,\\),\\{,\\},\\|,\\-" />
  <cfset var i = "" />

  <cfif findNoCase(left(arguments.list, 1), delim)>
    <cfset arguments.list = arguments.nullString & arguments.list />
  </cfif>
  <cfif findNoCase(right(list,1), delim)>
    <cfset arguments.list = arguments.list & arguments.nullString />
  </cfif>

  <cfset i = len(delim) - 1 />

  <cfloop condition="i GTE 1">
    <cfset delim = mid(delim, 1, i) & "_Separator_" & mid(delim, i+1, len(delim) - (i)) />
    <cfset i = i - 1 />
  </cfloop>

  <cfset delim = ReplaceList(delim, special_char_list, esc_special_char_list) />
  <cfset delim = Replace(delim, "_Separator_", "|", "ALL") />

  <cfset arguments.list = rereplace(arguments.list, "(" & delim & ")" & delim & ")", "\\1" & arguments.nullString &
  <cfset arguments.list = rereplace(arguments.list, "(" & delim & ")" & delim & ")", "\\1" & arguments.nullString &

  <cfreturn arguments.list />
</cffunction>

```

recurseComplexValues

public any recurseComplexValues(any node)

Recurses through complex values by type.

Parameters:

any node

Code:

```
<cffunction name="recurseComplexValues" access="public" returntype="any" output="false"
  hint="Recurses through complex values by type.">
  <cfargument name="node" type="any" required="true" />

  <cfset var value = "" />
  <cfset var child = "" />
  <cfset var i = "" />

  <cfif StructKeyExists(arguments.node.xmlAttributes, "value")>
    <cfset value = arguments.node.xmlAttributes["value"] />
  <cfelse>
    <cfset child = arguments.node.xmlChildren[1] />
    <cfif child.xmlName EQ "value">
      <cfset value = child.xmlText />
    <cfelseif child.xmlName EQ "struct">
      <cfset value = StructNew() />
      <cfloop from="1" to="#ArrayLen(child.xmlChildren)#" index="i">
        <cfset value[child.xmlChildren[i].xmlAttributes["name"]] = recurseComplexValues(child.xmlChildren[i]) />
      </cfloop>
    <cfelseif child.xmlName EQ "array">
      <cfset value = ArrayNew(1) />
      <cfloop from="1" to="#ArrayLen(child.xmlChildren)#" index="i">
        <cfset ArrayAppend(value, recurseComplexValues(child.xmlChildren[i])) />
      </cfloop>
    </cfif>
  </cfif>

  <cfreturn value />
</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>
```

