SITE Executive™

SAi | SYSTEMS ALLIANCE, INC.

SiteExecutive Version 2017

Custom Security Framework Guide

Revised June 2017

Contact:
Systems Alliance, Inc.
Executive Plaza III
11350 McCormick Road, Suite 1203
Hunt Valley, MD 21031

Phone: 410.584.0595 / 877.SYSALLI
Fax: 410.584.0594
http://www.systemsalliance.com
http://www.siteexecutive.com

Table of Contents

1. Custom Security Framework Overview

The SiteExecutive Custom Security Framework allows SiteExecutive customers to integrate the SiteExecutive security model with existing user/password repositories and to specify the rules that govern access to SiteExecutive content.

SiteExecutive customers can implement the custom security framework to authenticate against any source of user information (LDAP, Active Directory, user/password repositories, etc.) and to manage administrator security, content security or both within SiteExecutive.

The Custom Security Framework may also be used to provide users with the convenience of a single login by allowing persistent storage of authentication data in any manner. This guide provides sample code that developers can use and extend to authenticate against LDAP, Active Directory and proprietary user repositories.

1.1 Custom Security Capabilities

Custom ColdFusion code must be written to authenticate users attempting to log into SiteExecutive**,** to authenticate site visitors attempting to access secured content and to validate whether the visitor has already logged in. Writing the code to authenticate admin login, visitor access and visitor login validation is a one-time effort.

1.1.1 Deployed Application Interface

When the custom security module is implemented in SiteExecutive, the only user that will bypass custom security is the administrative user (**admin**).

Users must be synchronized (manually or programmatically) so that a SiteExecutive user exists with the same username as the corresponding external username. This allows the SiteExecutive permissions model to function.

1.1.2 Securing SiteExecutive Content

To secure SiteExecutive content, the code can be written in a single .cfc template with custom logic to determine the nature of the request when the content is accessed.

The process for securing a folder or site remains the same throughout SiteExecutive, whether or not custom security is enabled. The option for custom security to be applied must be selected when folder security is implemented.

2. Enabling Custom Security

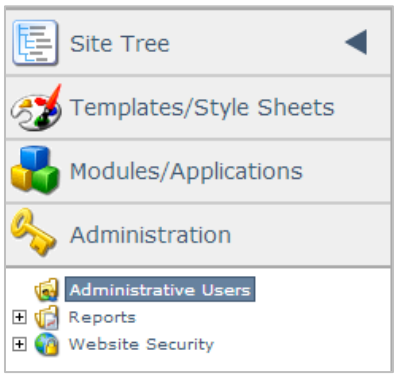
To enable custom security in SiteExecutive, the authentication templates must be stored within the instance root.

While the authentication templates can be stored in any folder within the instance root, storage in the modules folder is recommended for the purpose of consolidation.

2.1 Enabling Custom Security for Users

To enable custom security for users in SiteExecutive:

1. Select **Administration** in the **Explorer**.
2. Under **Administration**, select **Administrative Users.**



3. Select the **Security** tab.

The **Security menu** appears:

4. Click on the **Custom Security** tab.

5. Click **Edit.**
6. Select **Yes** in the **Enable Custom Security?** field.
7. Enter the path of the module that will authenticate users from the server in the **Authentication Module:** field.
8. Click **Save**.

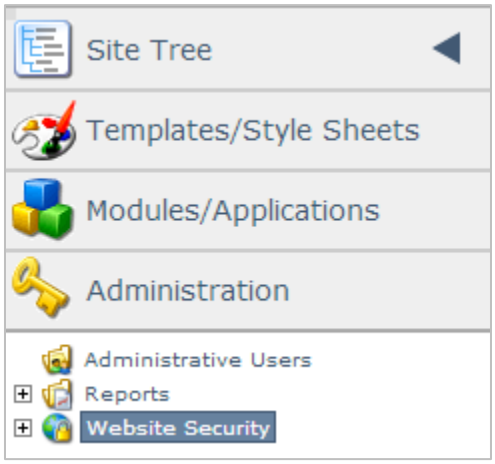
   Custom security is now enabled for user login. A SiteExecutive administrative user must be created for each user that will be granted access to the system. The SiteExecutive username must be the same as the custom username.

   The custom security logic will override the SiteExecutive login logic for all users except the admin user. This means that any users created in SiteExecutive, which do not reside in the authentication user repository will not be able to log into SiteExecutive.

2.2 Enabling Custom Security for Content

To enable custom security for content in SiteExecutive:

1. Select **Administration** in the **Explorer**.
2. Under **Administration,** select **Website Security.**

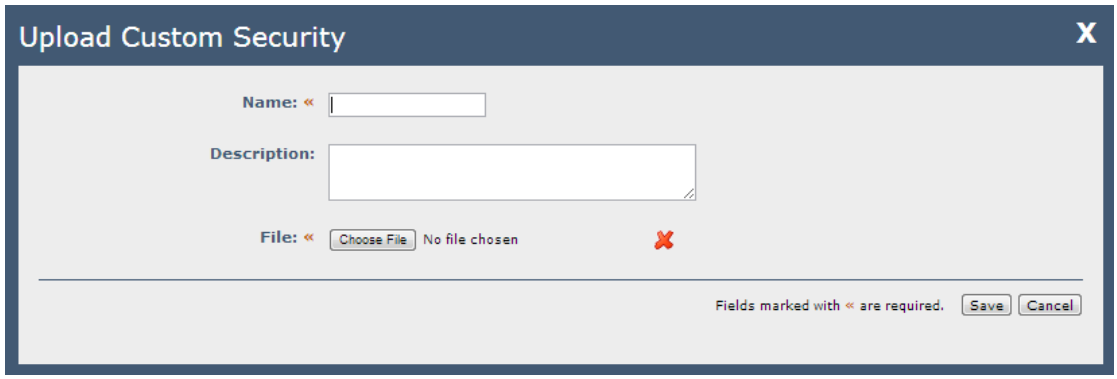


3. Select the **Properties** tab.

The **Website Security Properties** screen appears:

4. Click **New Custom Security**.

The **Upload Custom Security** dialog appears:



The available fields are:

Name: « The name of the custom security model. This will be used within SiteExecutive as an identifier for the model and will appear in the Security Model dropdown when applying custom security to folders or sites. This field is required.

Description: A short description of the custom security model.

File: « The .cfc template to upload as the custom security model. This field is required.

5. Fill in the information, upload a .cfc template, and click **Save.**

   The custom security model is uploaded into SiteExecutive.

2.2.1 Applying Custom Security to Sites

To apply custom security on content in specified sites:

1. Select the desired site in the site tree.
2. In the **Actions** panel, under **Site Administration,** select **Site Security.**

The **Site Security** dialog box appears:

3. Select the **Security Model** dropdown.

   The available security models to choose from are:

   Standard: The default SiteExecutive website security model, which draws from users and groups defined in Website Security.

   Legacy: Uses an existing ColdFusion security configuration file placed on the server prior to the installation of/upgrade to SiteExecutive 2013.

   NOTE: Any custom security models uploaded in the Security tab in Website Security will be available in the dropdown. The option will display as "Custom Security – [name of custom security model]."

4. Choose a security model from the dropdown.

   The **Site Security** dialog updates:

The available options are:

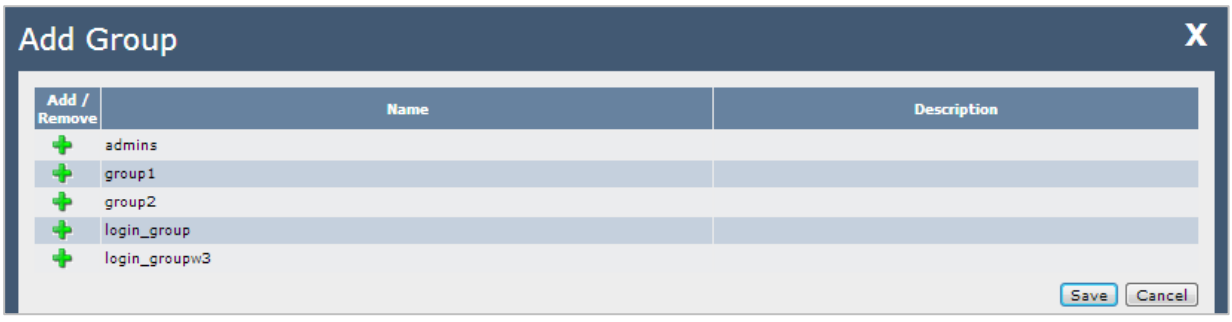
Login Page: Specifies the page where users are required to log in before they can access the folder with Site Security applied. Users can either type a search term into the box and select a page from the search results, or use the link selector to the right and choose a page.

> **NOTE: Only internal links are allowed. If no published page with a Login & Registration module is specified, then SiteExecutive will use a default system login page.**

Allow Registrations (Standard Security Model only): Specifies whether guests can register for access to the secure area from the login page.

Create Group: Users can create security groups from the Site Security dialog. See the *Site Administrator User Guide* for directions on how to create a security group.

Add Existing Group: Displays the **Add Group** dialog box, where users can add or remove existing security groups at the Site Security level:



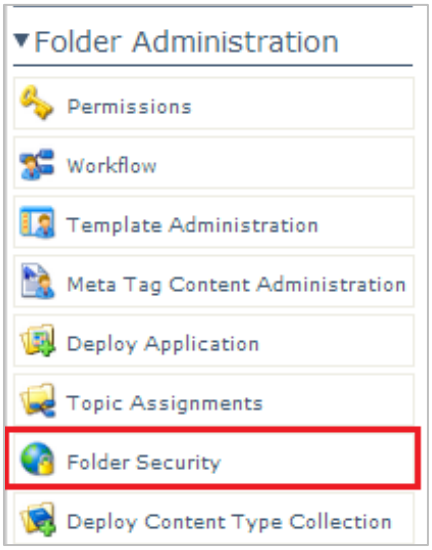
5. Specify settings for Site Security and click **Save.**

   Site Security is defined, and the user is returned to the SiteExecutive explorer at the site level.

2.2.2 Applying Custom Security to Folders

To apply custom security on content in specified folders:

1. Select the desired folder in the site tree.
2. In the **Actions** panel, under **Folder Administration,** select **Folder Security.**



The **Folder Security** dialog box appears:

NOTE: If security is defined for the site or the parent folder, the security settings will cascade down to the given folder as well. In this case, users will have the option to "Override Existing Security" in the Folder Security dialog box.

3. Select the **Security Model** dropdown.

   The available security models are:

   Standard: The default SiteExecutive website security model, which draws from users and groups defined in Website Security.

   Legacy: Uses an existing ColdFusion security configuration file placed on the server prior to the installation of/upgrade to SiteExecutive 2013.

NOTE: Any custom security models uploaded in the Security tab in Website Security will be available in the dropdown. The option will display as "Custom Security – [name of custom security model]."

4. Choose a security model from the dropdown.

   The **Folder Security** dialog updates:

The available options are:

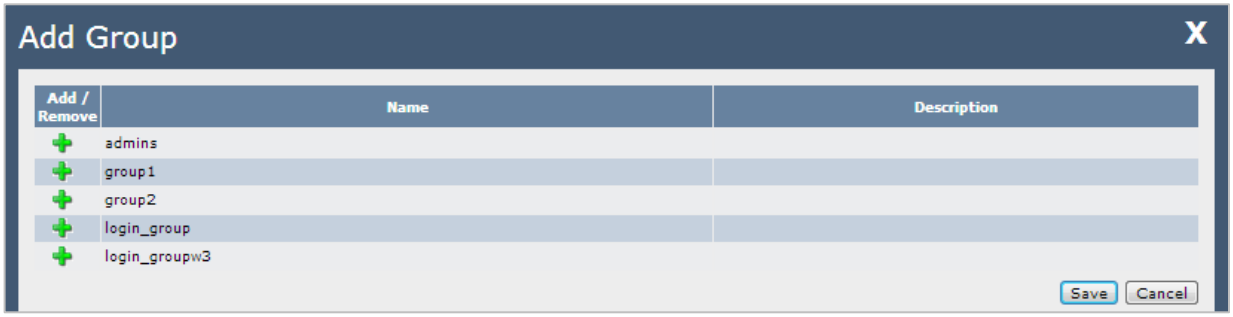
Login Page: Specifies the page where users are required to log in before they can access the folder with Site Security applied. Users can either type a search term into the box and select a page from the search results, or use the link selector to the right and choose a page.

> **NOTE: Only internal links are allowed. If no published page with a Login & Registration module is specified, then SiteExecutive will use a default system login page.**

Allow Registrations (Standard Security Model only): Specifies whether guests can register for access to the secure area from the login page.

Create Group: Users can create security groups from the Folder Security dialog. See the *Site Administrator User Guide* for directions on how to create a security group.

Add Existing Group: Displays the **Add Group** dialog box, where users can add or remove existing security groups at the Folder Security level:



5. Specify settings for Folder Security and click **Save.**

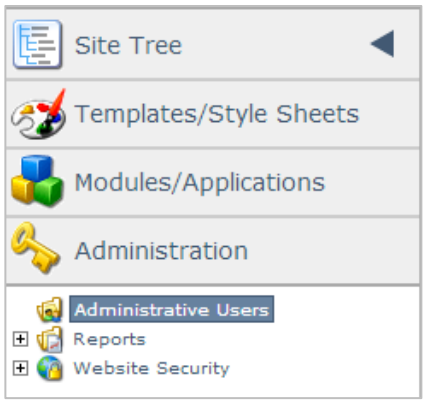
   Folder Security is defined, and the user is returned to the SiteExecutive explorer at the folder level.

2.3 Active Directory Integration

Administrators can specify settings for Active Directory integration with SiteExecutive in the Security menu.

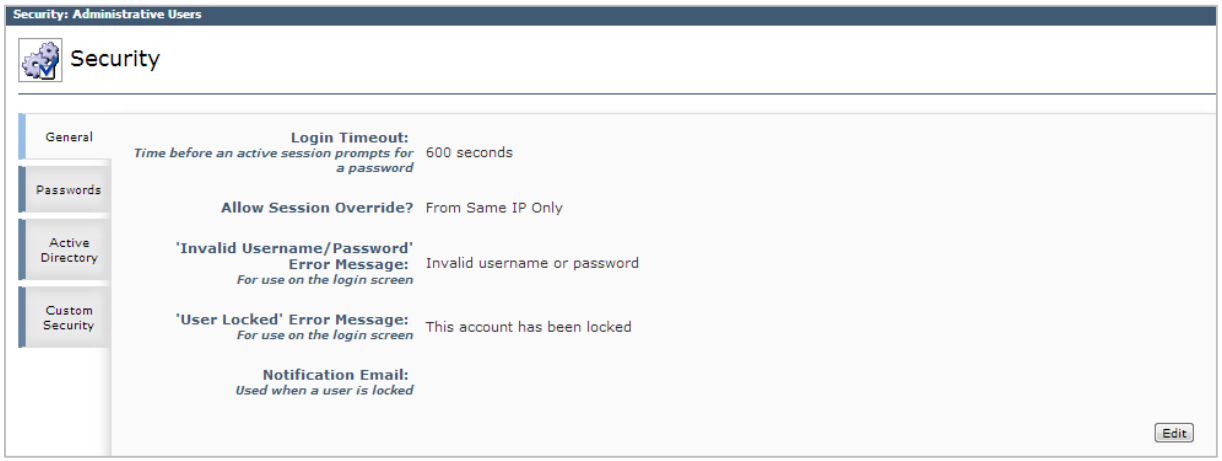
To modify Active Directory settings in SiteExecutive:

1. Select **Administration** in the **Explorer**.
2. Under **Administration**, select **Administrative Users.**



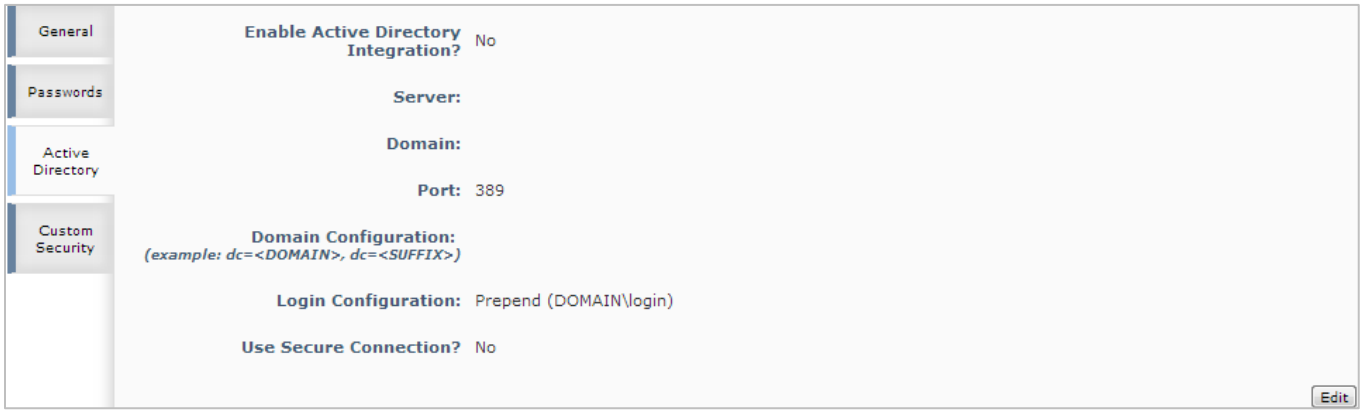
3. Select the **Security** tab.

   The **Security menu** appears:



4. Select **Active Directory.**

The **Active Directory configuration menu** appears:



5. Click **Edit.**

   The available options are:

   Enable Active Directory Integration? Specifies whether to enable integration with an Active Directory.

   Server: The address of the Active Directory server.

   Domain: The domain where the Active Directory is located.

   Port: The port number used by the Active Directory.

   Domain Configuration: Specifies how the domain syntax is configured in the Active Directory integration framework.

   Login Configuration: Specifies the format with which users must enter their username when logging in via the Active Directory. The options are:

   - **Prepend (DOMAIN\login)**
   - **Append (login@DOMAIN)**
   - **None**

   Use Secure Connection? Specifies whether the Active Directory connection is secured with SSL.

6. Fill in the information and click **Save.**

   Active Directory integration is configured based on the settings specified.

3. Custom Security Framework APIs

This section contains API guides for implementing custom logic on the SiteExecutive server, along with sample ColdFusion templates. The topics covered are: creating a website security framework and creating a custom administrative security framework for the authoring server.

3.1 Website Security Framework

3.1.1 Basic Usage

To implement a custom website security framework on the server, create a .cfc file which:

- **Implements** the following:

```
com.sysalli.se.security.websitesecurity.IWebsiteSecurity
```

- **Extends** the following:

```
com.sysalli.se.security.websitesecurity.abstractWebsiteSecurity
```

Sample code snippet:

```
<cfcomponent
extends="com.sysalli.se.security.websitesecurity.abstractWebsiteSecurity"
implements="com.sysalli.se.security.websitesecurity.IWebsiteSecurity">
```

3.1.2 Required Methods to Override

The following methods **must** be overridden in a custom website security framework:

3.1.2.1 Authenticate

General notes:

- Access: public
- Returns: (boolean) *true* or *false*, indicating whether the user has successfully logged in.

- This method is called whenever a user submits the login form.
- This method should perform whatever is necessary to ensure that the next call to `isAuthenticated()` returns *true*.
- If this method is going to return *false*, the developer can set a custom error message by calling `setErrorMsg("custom error text")`.

Arguments:

- **USERNAME:** (string) The username that was entered on the login form.
- **PASSWORD:** (string) The password that was entered on the login form.
- **SECUREDOBJECT**: (struct) The secured SEObject that the user is attempting to visit
- **ACCEPTABLEGROUPS**: (array) An array of structs with the following keys:
  - GROUP: the ID of a website security group
  - PRIMARYGROUP: a boolean value indicating if the group is the primary group defined in the website security dialog
- **SECURITYOPTIONS**: (struct) The keys are:
  - LOGINPAGE: the ID of the login page defined in the website security dialog
  - *REDIRECT: Depreciated and can be ignored*
  - *LIMITFAILURES: Depreciated and can be ignored*
  - REGISTRATIONGROUP: the ID of the website security group that was selected as a primary in the website security dialog. (this is a possibility for future expansion)

Sample code snippet:

```
<cffunction name="authenticate" returntype="boolean" access="public"
output="false">
      <cfargument name="username" required="true" type="string" />
      <cfargument name="password" required="true" type="string" />
      <cfargument name="securedObject" required="true" type="struct" />
      <cfargument name="acceptableGroups" required="true" type="array" />
      <cfargument name="securityOptions" required="true" type="struct" />
```

3.1.2.2 IsAuthenticated

General notes:

- Access: public
- Returns: (boolean) *true* or *false* indicating if the user is authorized to access the secured content.
- If this method fails, the user will be presented with the login page.
- This method is called every time a user accesses a page within the website security area.
- This method can be used to check and see if the user is authorized to view content (in addition to authenticating his/her credentials).

Arguments:

- **SECUREDOBJECT:** (struct) The secured SEObject that the user is attempting to visit.
- **ACCEPTABLEGROUPS:** (array) An array of structs with the following keys:
    - GROUP: the ID of a website security group
    - PRIMARYGROUP: a boolean value indicating if the group is the user's primary group as defined in the website security dialog

Sample code snippet:

```
<cffunction name="isAuthenticated" returntype="boolean" access="public"
output="false">
      <cfargument name="securedObject" required="true" type="struct" />
      <cfargument name="acceptableGroups" required="true" type="array" />
```

3.1.3 Optional Methods to Override

The following methods can be overridden in a custom website security framework to give additional functionalities.

3.1.3.1 Logout

General notes:

- Access: public

- Returns: (void) Nothing is returned from the logout method. Exceptions caused by the logout method will be logged, but will not prevent the rendering of a page.
- Whenever a user logs out of website security, the logout method is called on **all custom security models.**

Arguments:

- None

Sample code snippet:

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

3.1.3.2 ForgotPasswordAllowed

General notes:

- Access: public
- Returns: (boolean) *true* or *false* indicating if the forgot password functionality should be presented to the user (and is implemented).
- If this method returns *true*, the ForgotPassword method should be implemented.

Arguments:

- **SECUREDOBJECT:** (struct) The secured SEObject that the user is attempting to visit.
- **ACCEPTABLEGROUPS:** (array) An array of structs with the following keys:
  - GROUP: The ID of a website security group.
  - PRIMARYGROUP: A boolean value indicating if the group is the user's primary group as defined in the website security dialog.
- **SECURITYOPTIONS:** (struct) The keys are:
  - LOGINPAGE: The ID of the login page defined in the website security dialog (or the site address, if using the default login page)
  - *REDIRECT: Depreciated and can be ignored*
  - *LIMITFAILURES: Depreciated and can be ignored*

- REGISTRATIONGROUP: The ID of the website security group that was selected as a primary in the website security dialog. (This is a possibility for future expansion)

Sample code snippet:

```
<cffunction name="forgotPasswordAllowed" returntype="boolean"
access="public" output="false">
	<cfargument name="securedObject" required="true" type="struct" />
	<cfargument name="acceptableGroups" required="true" type="array" />
	<cfargument name="securityOptions" required="true" type="struct" />
```

3.1.3.3 ForgotPassword

General notes:

- Access: public
- Returns: (string) returns a message to display the user when the user clicks "Submit" on the forgot password form.
- This method should do whatever is necessary to help the user recover the password, including reminding the user of the password, resetting the password, and so forth. (For instance, the method could send an email or update a user record and display a message.)

Arguments:

- **USERNAME:** (string) The username entered on the forgot password form.
- **EMAIL:** (string) The email address entered on the forgot password form.
- **FORGOTPASSWORDURL:** (string) The URL of the login page (or the site, if using the default login page).

Sample code snippet:

```
<cffunction name="forgotPassword" returntype="string" access="public"
output="false">
	<cfargument name="username" required="true" type="string" />
	<cfargument name="email" required="true" type="string" />
	<cfargument name="forgotPasswordURL" required="true" type="string" />
```

3.1.4 Inherited Methods

The following methods are public but **should not be overridden**, as this may impact the functionality of the custom security model.

3.1.4.1 init

General notes:

- Access: public
- Returns: (Object) The website security object. This is a constructor.

Arguments:

- **None**

3.1.4.2 setUserID

General notes:

- Access: public
- Returns: Nothing

Arguments:

- (string) The user ID.

3.1.4.3 getUserID

General notes:

- Access: public
- Returns: (string) The user ID.

Arguments:

- **None**

3.1.4.4 setErrorMsg()

General notes:

- Access: public

- Returns: Nothing

Arguments:

- (string) The error message.

3.1.4.5 getErrorMsg()

General notes:

- Access: public
- Returns: (string) The error message.

Arguments:

- **None**

3.1.5 Sample Code Template

This is a sample template for website security. It covers the processes for:

- authenticating a user that is trying to access protected content
- testing if a user has already been authenticated for a piece of protected content
- determine if the forgot password functionality is implemented
- the forgot password functionality
- the logout functionality

NOTE: Whenever a user logs out of website security, the logout method on ALL custom security models is called.

```
<cfcomponent      output="false"
            accessors="true"
            implements="com.sysalli.se.security.websitesecurity.IWebsiteSecur
      ity"

      extends="com.sysalli.se.security.websitesecurity.abstractWebsiteSecurit
      y">

      <!--- use your own datasource here if necessary, otherwise just remove
      this line --->
      <Cfset variables.customDatasource = "CustomSecurity">
```

```
	<cffunction name="authenticate" output="false" access="public"
	returntype="boolean" hint="authenticate the user and return if they are
	allowed to access this content">
		<cfargument name="username" type="string" required="true"/>
		<cfargument name="password" type="string" required="true"/>
		<cfargument name="securedObject" type="struct" required="true"
			hint="the SiteExecutive object that is secured" />
		<cfargument name="acceptableGroups" type="array" required="true"
			hint="an array of structs with a website security
			group ID and a primary group flag" />
		<cfargument name="securityOptions" type="struct" required="true"
			hint="a struct with the settings for the secured
object" />

		<cfset local.retval = false />

		<!--- query the user table for a user with this password --->
		<cfquery name="local.getUser"
		datasource="#variables.customDatasource#" maxrows="1">
			SELECT	username
			FROM	customusertable
			WHERE	username = <cfqueryparam
				value="#arguments.username#"
				cfsqltype="CF_SQL_VARCHAR" />
				AND	password = <cfqueryparam
					value="#arguments.password#"
					cfsqltype="CF_SQL_VARCHAR" />
		</cfquery>

		<cfif local.getUser.recordCount gt 0>
			<!--- I found a user --->


			<!---
				Tell the system this user is logged in.
				This is what you will test for in isAuthenticated()
			--->


			<!--- BEGIN: In this example we will set a cookie --->
			<cfset local.cookieval = structNew() />
			<cfset local.cookieval.when = now() />
			<cfset local.cookieval.who = arguments.username />
			<cfset cookie.SECustomWebsiteSecurityUser =
			SerializeJSON(local.cookieval) />
			<!--- END: In this example we will set a cookie --->
```

```
                  <!--- we need to return a true value to indicate the user
                  logged in --->
                  <cfset local.retval = true />
            <cfelse>
                  <!--- could not find a user --->

                  <!--- set an error message to display to the user --->
                  <cfset setErrorMsg("Invalid Username or Password") />

                  <!--- we need to return a false value to indicate the user
                  did not log in --->
                  <cfset local.retval = false />
            </cfif>

            <cfreturn local.retval />
      </cffunction>

      <cffunction name="isAuthenticated" output="false" access="public"
      returntype="boolean" hint="test to see if a user is already
      authenticated.">
            <cfargument name="securedObject" type="struct" required="true"
                        hint="the SiteExecutive object that is secured" />
            <cfargument name="acceptableGroups" type="array" required="true"
                        hint="an array of structs with a website security
                        group ID and a primary group flag" />

            <!---
                  check to see if the user is logged in by checking
                  whatever you set in the Authenticate() method in a prior
request
            --->
            <cfif structKeyExists(cookie,"SECustomWebsiteSecurityUser")>
                  <cfset local.retval = true />
            <cfelse>
                  <cfset local.retval = false />
            </cfif>

            <cfreturn local.retval />
      </cffunction>

      <cffunction name="logout" output="false" access="public"
returntype="void"
                        hint="a function that is called on logout">
```

```
            <!---
                  Do whatever needs to be done to make sure the next call to
                  isAuthenticated() will return false
            --->
            <cfset structDelete(cookie,"SECustomWebsiteSecurityUser",false)
/>

            <cfreturn />
      </cffunction>


      <cffunction name="forgotPasswordAllowed" output="false" access="public"
      returntype="boolean" hint="should the forgot password link be shown on
      the login page?">
            <cfargument name="securedObject" type="struct" required="true"
                              hint="the SiteExecutive object that is secured"
/>
            <cfargument name="acceptableGroups" type="array" required="true"
                              hint="an array of structs with a website
                              security group ID and a primary group flag" />
            <cfargument name="securityOptions" type="struct" required="true"
                        hint="a struct with the settings for the secured
            object" />

            <!--- this is likely to just be true or false --->
            <cfreturn true />
      </cffunction>

      <cffunction name="forgotPassword" output="false" access="public"
      returntype="string" hint="returns a message string to display after
      sending the password to the user. If you do not want to support the
      forgot password functionality, you do not have to include this
      method.">
            <cfargument name="username" type="string" required="true"
                              hint="the username the user entered on the
form"/>
            <cfargument name="email" type="string" required="true"
                              hint="the email address the user entered on the
                              forgot password form"/>
            <cfargument name="forgotPasswordURL" type="string"
required="true"
                              hint="the URL the user was attempting to go to
                              when they forgot their password"/>

            <cfset local.retval = "" />
```

```
            <!--- do something when the user claims to have forgotten the
password --->

            <!--- in this example we'll check to see if the username/email
            combo exists --->
            <cfquery name="local.getUser"
            datasource="#variables.customDatasource#" maxrows="1">
                  SELECT      username
                  FROM        customusertable
                  WHERE       username = <cfqueryparam
                        value="#arguments.username#"
                        cfsqltype="CF_SQL_VARCHAR" />
                        AND email = <cfqueryparam value="#arguments.email#"
                        cfsqltype="CF_SQL_VARCHAR" />
            </cfquery>

            <cfif local.getUser.recordCount eq 0>
                  <!--- the combo does not exist so discourage them --->
                  <cfset local.retval = "That username/email address combo is
                  not in our system." />

            <cfelse>
                  <!--- the combo does exist so send them an email --->
<CFMAIL     FROM="#arguments.email#"
            TO="#arguments.email#"
            SUBJECT="Forgot your password?"
            REPLYTO="#arguments.email#">
You really shouldn't have forgotten your password, #arguments.username#!

#arguments.forgotPasswordURL#

If you can't tell, this message is a test and not a REAL I forgot my password
message.
</CFMAIL>
                  <!--- after the email tell them we sent it. --->
                  <cfset local.retval = "An email has been sent with
                  instructions to get a password" />
            </cfif>

            <cfreturn local.retval />
      </cffunction>
</cfcomponent>
```

3.2 Custom Administrative Security

3.2.1 Basic Usage

To implement a custom administrative security framework on the server, create a .cfc file which **extends** the following:

```
com.sysalli.se.users.abstractSecurity
```

Sample code snippet:

```
<cfcomponent extends="com.sysalli.se.users.abstractSecurity">
```

3.2.2 Required Method to Override

The following method **must** be overridden in a custom administrative security framework:

3.2.2.1 PasswordIsValid

General notes:

- Access: public
- Returns: (boolean) *true* or *false*, indicating if the password is valid for the user that is attempting to log in.
- This method can get the username by calling `getUserName()`. It can change the username of the user logging in (as a translation layer) by calling `setUserName("username")` with a parameter of the username that you want to use.
- If this method is going to return *false*, the developer can set a custom error message by calling `setErrorMsg("custom error text")`.

Arguments:

- **PASSWORD:** (string) The password that was entered on the login form.

Sample code snippet:

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

```
<cfargument name="password" required="true" type="string" />
```

3.2.3 Inherited Methods

The following methods are public but **should not be overridden**, as this may impact the functionality of the custom security model.

3.2.3.1 init()

General notes:

- Access: public
- Returns: (Object) The object. This is a constructor.

Arguments:

- USERNAME: (string) The username that was entered in the login form.

3.2.3.2 setUsername()

General notes:

- Access: public
- Returns: Nothing

Arguments:

- (string) The username.

3.2.3.3 getUsername()

General notes:

- Access: public
- Returns: (string) The username.

Arguments:

- **None**

3.2.3.4 setErrorHTML()

General notes:

- Access: public
- Returns: Nothing

Arguments:

- (string) The HTML code of the error message.

3.2.3.5 getErrorHTML()

General notes:

- Access: public
- Returns: (string) The HTML code of the error message.

Arguments:

- **None**

3.2.4 Sample Code Template

This sample template for administrative custom security supports:

- Checking the password for the username/password being attempted
- Changing the username of the user logging in
- Setting a custom error message.

```
<cfcomponent output="false" accessors="true"
extends="com.sysalli.se.users.abstractSecurity">
      <cffunction name="passwordIsValid" output="false" access="public"
      returntype="boolean" hint="should use getUserName() and CAN use
      setUserName() and setErrorHTML()">
            <cfargument name="password" type="string" required="true"/>

      <!---
      Test Conditions:
      this presupposes you have the users 'greg' and 'tom' in Administrative
Users

      username          password          allowed in        final SE Username
   ---------------------------------------------------------------------------
-------
      fakegreg    <greg's SE Password>    yes                    greg
      fakegreg    <anything else>         no                     n/a
      greg        <greg's SE Password>    yes                    greg
      greg        <anything else>         no                     n/a
```

```
tom         admin                   yes                     admin
tom         password                no                      n/a
tom         <tom's SE Password>           yes                     tom
tom         <anything else>         no                      n/a

--->

      <cfset local.retval = false />
      <cfset local.checkPW = true />


      <!--- if username is fakegreg, we really mean greg --->
      <cfif getUserName() eq "fakegreg">
            <cfset setUsername("greg")>
      </cfif>

      <cfif getUserName() eq "tom">
            <cfif arguments.password eq "admin">
                  <!--- if username is tom and password is admin, we
            mean admin and pw is fine --->
                  <cfset setUserName("admin") />
                  <cfset local.retval = true />
                  <cfset local.checkPW = false />
            <cfelseif arguments.password eq "password">
                  <!--- if username is tom and password is password, i
            don't care what you set it to, don't let him in --->
                  <cfset local.retval = false />
                  <cfset local.checkPW = false />
                  <cfset setErrorHTML("tom would <em>never</em> have a
                  password of 'PASSWORD'")>
            </cfif>
      </cfif>

      <cfif local.checkpw>
            <!--- actually check the pw against SE --->
            <cfmodule   template="/#request.semap#/obj/becomeadmin.cfm"
                        mode="start">
            <cftry>
                  <cfmodule
template="/#request.SEMap#/obj/getbyname.cfm"

parentid="#request.magicnumber.seobject.usercolid#"
                                    objectname="#getUserName()#"
                                    objectvar="local.user"
                                    throwonerror="Yes" />

                  <cfset local.retval = local.user.password eq
                  hash(arguments.password) />

                  <cfcatch>
                        <!--- couldn't find the user so no soup for you
--->
```

```
                    </cfcatch>
                </cftry>
                <cfmodule template="/#request.semap#/obj/becomeadmin.cfm"
                mode="end">

                <cfif !local.retval>
                    <cfset setErrorHTML("I'm sorry that password is
invalid.") />
                </cfif>
            </cfif>

            <cfreturn local.retval />
      </cffunction>

</cfcomponent>
```