EPiSERVER 4

Securing Edit and Admin

Product version:	4.50
Document version:	1.0
Document creation date:	02-05-2005

Purpose

EPiServer 4.50 introduces several new security features. Among other things it allows relocation of the edit and admin directories and configurable http ports, to make it harder for intruders to try to access sensitive resources.

This document describes how to write EPiServer applications that are somehow dependent on the edit and admin directories and also, how to ensure that the applications will continue to work even if those directories are moved to unknown locations.

Table of Contents

Relocation of edit and admin Directories

If a section begins with the text “No changes needed in existing applications", it means that the application does not need any modifications to work with relocated directories for that particular section in EPiServer. The rest of the text in the section is merely for informational purposes.

New Configuration Properties in web.config

The edit and admin directories are no longer hardcoded in EPiServer 4.50. Instead, there are two new keys in web.config that define the virtual directory locations:

EPsAdminDir	location of admin directory, for example `foo111/bar222`
EPsEditDir	location of edit directory, for example `foo333/bar444`

Applications can access these values by reading the new **EditDir** and **AdminDir** properties in the ApplicationConfiguration object. If the keys are not defined in web.config, EditDir and AdminDir will return default values “edit/” and “admin/” respectively.

The values returned by the EditDir/AdminDir properties have the format “path/”. To simplify their usage they are guaranteed to always have a trailing “/” and never a leading “/”, even if defined otherwise in web.config.

Language Keys

No changes needed in existing applications.

All language keys work as previously - they are not affected by any directory relocations. This is also true for relative language keys, i.e. keys that are prefixed with “#”.

Url Property in plugin Attributes

No changes needed in existing applications.

The EPiServer plugin loader in 4.50 knows how to handle relocated directories. Any references to “~/edit/” and “~/admin/” in the Url property in the plugin attributes are automatically mapped to the real directory locations. This is an operation that is completely transparent to the plug-ins. Therefore, plug-ins that refer to files in edit/admin in their plugin attributes can continue to do so.

Example:

```
[GuiPlugIn(DisplayName="",Area=PlugInArea.EditPanel,
Url="~/edit/EditMultiLanguage.ascx”)]
```

Hardcoded Paths to edit and admin

Hardcoded paths that refer to the edit and admin directories need to be changed. To help developers with this, the following new things are available in version 4.50:

New properties in the ApplicationConfiguration object

AdminUrl	Absolute or relative URL to the admin directory. If the admin port is the same port as in the current Http Context, the URL is relative, otherwise it is absolute. The property is especially suited for environments with proxies, where links should use relative URLs. The path is guaranteed to end with a “/”.
EditUrl	Absolute or relative URL to the edit directory. If the edit port is the same port as in the current Http Context, the URL is relative, otherwise it is absolute. The property is especially suited for environments with proxies, where links should use relative URLs. The path is guaranteed to end with a “/”.
AbsoluteAdminUrl	Absolute URL to the admin directory, including configured protocol, port and path from current configuration. The path is guaranteed to end with a “/”.
AbsoluteEditUrl	Absolute URL to the edit directory, including protocol, port and path from current configuration. Applications should use this property to get an URL to the edit directory when they are not running under an Http Context. The path is guaranteed to end with a “/”.

There are also new properties for Http ports – see the Configurable Http Ports section for the details.

A new method in the ApplicationConfiguration object

ResolveUrl(url)	Resolve an expression like "~/edit" to a relative or absolute url. The url that is created is relative as long as the Http port obtained, when resolving the url, is the same port as in the current Http Context; otherwise the created url is absolute.
ResolveUrl(url, relativeUrl)	Overloaded version of ResolveUrl above where you can control whether the created url should be relative or absolute. If the relativeUrl parameter is true, the created url is relative, otherwise it is absolute.

A new Web control:

ControlLoader	Loads a user control using an URL expression like "~/edit/FileManagement.ascx" (the real locations of expressions like “~/edit” depend on the site’s configuration and are resolved runtime).

References to User Controls in edit/admin

From a security point of view, it is best if you avoid references to user controls in the edit and admin directories altogether. But if your application requires this anyway, follow the guidelines below to ensure that your application will work even if the directories are relocated.

Replace any user control declarations using the `<%@ Register` syntax with the ControlLoader Web control. Do this when:

- You are using a user control located in the edit directory from a Web Form or user control outside edit.

  or

- You are using a user control located in the admin directory from a Web Form or user control outside admin.

Instead of:

```
<%@ Register TagPrefix="edit" TagName="FileManagement"
"Src="~/edit/FileManagementControl.ascx"%>
...
<edit:FileManagement runat="server"/>
```

write:

```
<EPiServer:ControlLoader runat="Server"
Src="~/edit/FileManagementControl.ascx"/>

Add the EPiServer declaration to your aspx/ascx file, if needed:

<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls"
Assembly="EPiServer" %>
```

If you reference a user control from a file in the same directory, it is OK to use the ASPNET user control registration syntax, as long as you do not use “~/” or “../” in the URL. What happens when you use the “~/” or “../” syntax is that when the page is loaded runtime, ASPNET resolves the URL to an address that corresponds to the file’s location at design time. This is not necessarily the same as the runtime address, since the directory that the page resides in may have been relocated.

In the following example, a file in edit references the CommandActions.ascx user control, also a file in edit:

This is OK:

```
<%@ Register TagPrefix="EPiServerSys" TagName="CommandActions"
Src="CommandActions.ascx"%>
```

These are not OK:

```
<%@ Register TagPrefix="EPiServerSys" TagName="CommandActions"
Src="~/edit/CommandActions.ascx"%>

<%@ Register TagPrefix="EPiServerSys" TagName="CommandActions"
Src="../edit/CommandActions.ascx"%>
```

References to edit and admin Using “../”

In files both inside and outside the edit/admin directories, replace any URLs that begin with “../edit/” or “../admin/” with URLs that are based on the current site’s configuration. This applies to nested directories as well, for example “../../edit/”.

A good rule: files inside the edit and admin directories should always refer to other files using URLs that are built from EditUrl/AdminUrl/ResolveUrl in ApplicationConfiguration. As a rule, avoid references to files using the "../" syntax altogether.

For example, in files inside edit/admin:

Instead of:

```
Response.Redirect("~/Edit/Default.aspx?id=" + pageLink.ID.ToString());

Response.Redirect("~/Admin/FileManagement.aspx");
```

write:

```
Response.Redirect(Configuration.EditUrl +  "Default.aspx?id=" +
pageLink.ID.ToString());

Response.Redirect(Configuration.EditUrl + "FileManagement.aspx");
```

For example, in files outside edit/admin:

Instead of:

```
Response.Redirect("../Edit/Default.aspx?id=" + pageLink.ID.ToString());

Response.Redirect("../Admin/FileManagement.aspx");
```

write:

```
Response.Redirect(Configuration.EditUrl + "Default.aspx?id=" +
pageLink.ID.ToString());

Response.Redirect(Configuration.AdminUrl + "FileManagement.aspx");
```

References to files outside edit and admin Using "../"

When referencing files that are outside secured directories, the URLs can be relative to the site's root (protocol and port information not necessary). Make sure that the URLs are relative to the site's root and do not use expressions like "../dir1/dir2/file.css".

For example, in aspx/ascx files inside edit and admin:

Instead of:

```
<link rel="stylesheet" type="text/css" href="../util/styles/system.css">

<script type='text/javascript' src="../util/javascript/system.js">
</script>
```

write:

```
<link rel="stylesheet" type="text/css"
href="<%=Configuration.RootDir%>util/styles/system.css">

<script type='text/javascript'
src="<%=Configuration.RootDir%>util/javascript/system.js">
</script>
```

Configurable HTTP Ports

Overview

The HTTP ports used to access edit and admin directories are configurable in EPiServer 4.50. Applications that rely on the default ports 80 (default HTTP port) and 443 (default HTTPS port) will probably need to be modified to handle unknown ports.

New Configuration Properties in web.config

There are new values in web.config for HTTP ports configuration:

EPnEditHttpPort	HTTP port required to access the edit directory, for example 80 or 887.
EPnAdminHttpPort	HTTP port required to access the admin directory, for example 80 or 887.

The port values can be accessed using properties in the ApplicationConfiguration called **AdminHttpPort** and **EditHttpPort**.

Requirements

To enable configurable HTTP ports in EPiServer 4.50, the spidersupport HTTP module must be defined in web.config. The module is defined by default in standard installations.

Dynamic CSS Files

Overview

Since the edit and admin directories can be relocated in EPiServer 4.50, all references to and from files in those directories must be dynamic. This is also true for style sheet files, since they can be included by files in relocated directories and have references to images, HTCs and other files. However, stylesheet files are static by nature, so to solve this problem a new concept, dynamic CSS files, is introduced in version 4.50. Basically, it allows you to insert predefined tags into CSS files and the tags are parsed runtime.

The dynamic CSS files technology was designed with four goals in mind:

1. A concept that is easy to grasp and use.
2. It should be totally optional – if developers do not need it, they can just ignore it. If they want to make only a single or a few CSS files dynamic, they can do so.
3. Dynamic CSS files should be completely transparent to browsers – the browsers should still think that they are dealing with static CSS files, when they are in fact dynamic on the server.

4. Performance. The cost of using dynamic CSS files should be as low as possible. Even if the files have dynamic content, once parsed on the server they should be cached in browsers just like static CSS files, which makes them scale well.

Requirements

To enable dynamic CSS files in EPiServer 4.50, the spidersupport HTTP module must be defined in web.config. The module is defined by default in standard installations.

Using Dynamic CSS Files

To make a dynamic CSS file, just create a copy of an existing stylesheet file and rename it by adding the string "_template" after the file name, for example, copy of `styles.css` is renamed as `styles_template.css`. When a browser makes a request for a CSS file, EPiServer will intercept ASPNET's CSS loading procedure and check if there is a dynamic CSS file available. If it finds one, the CSS file is read, parsed and returned. If no dynamic CSS file is found, ASPNET continues its standard CSS loading procedure.

Example:

1. We want to make a dynamic CSS file from mystyles.css, so we copy mystyles.css to mystyles_template.css.
2. Next, a Web request comes in, asking for mystyles.css.
3. EPiServer looks if there is a dynamic CSS file and finds mystyles_template.css.
4. The dynamic CSS file is parsed and returned to the browser, who still sees the file as mystyles.css. The CSS file has the standard MIME header and datestamp for CSS files and is cached by the browser.

Predefined CSS Tags

The syntax of a tag is "tag", the tag name is case-insensitive. The entire expression is replaced with the actual value runtime.

The number of predefined tags is kept to a minimum, due to security reasons. There may be more tags later, but currently only the following ones are supported:

$RootDir$	Virtual path to the application's root, e.g. "/EPiServerSample/".
$UploadDir$	Virtual path to the application's root dir, e.g. "/upload/".

Examples:

```
.EPEdit-tabbackground
{
 background-image:url($RootDir$Util/images/tabrow_background.gif);
}
```

```
.EPEdit-inputNumber
{
 border:solid 1px #6D8CA8;
 behavior:url($RootDir$Util/javascript/changedinput.htc);
}
```