University of California ~ Effort Reporting System Customization Guide

Release 2
April 10, 2006

Application Technology Services
Information Resources & Communications
Office of the President

Contents

Introduction

The Effort Reporting System was designed to be configurable and customizable to support the specific needs of each installation. In preparing to customize the system, it is useful to review the types of configuration and customization that might be needed for a particular installation.

Configuration

Configuration means changing system behavior by modifying one or more options that are provided by the application configuration file. Configuration can be done by the installer or a programmer after installation. For ERS, configuration is accomplished by editing settings in the application configuration file ***ERSConfig.XML***

Customization

Customization means changing the system "look and feel" by modifying the fonts, colors or graphics used for the web application. This process is commonly known as "skinning". Customization may be done by a web designer or HTML programmer. This task requires editing CSS files, possibly editing HTML or JSP files, creating new graphics and installing these modified objects in the appropriate application folders.

Modification

Modification means changing the system behavior by modifying source code. This type of customization will require Java programming skills and the changes are introduced by editing source code, recompiling and redeploying the application in WAR format. In order to modify the ERS source code, the installation must establish the required development environment with all of the products described in the "ERS Software Specification" document.

Configuration File: ERSConfig.XML

The ERS configuration file contains parameters which control the behavior of many aspects of the Effort Reporting System. It is a structured XML document that will allow the installer to set up the Effort Reporting System to meet installation specific needs.

The root element of this document is the "Configuration" element. The "root" element (Configuration) is only mentioned in the first entry.

Throughout this section, the element descriptions will use a dotted notation to refer to the sub-elements. For example, an entry in the XML file with the following structure:
`<GlobalItems><ApplicationBaseURL>XXX</ApplicationBaseUrl></GlobalItems>`

will be shown as:
`GlobalItems.ApplicationBaseURL` with a supplied value of: `XXX`

Configuration Element	**Supplied Value**
Description	
Configuration	url=http://www.ucop.edu
<string> the URL of the campus	
	campusName=UC Office of the President
<string> text name of the campus	
GlobalItems.ApplicationBaseURL	http://localhost:8080/EffortReporting
<string> URL of index.jsp for the application	
GlobalItems.TrainingBaseURL	http://localhost:8080/EffortReporting/Training
<string> URL of Training website for linking from online help	
GlobalItems.SupportEmail.Name	Effort Reporting System
<string> text Name of the Support Email User	
GlobalItems.SupportEmail.Address	ersadmin@ucop.edu
<string> Email Address of the Support Email User	
AccessControl.Authentication.internal-authentication	true
whether the authentication method used is internal (true) or external (false)	
AccessControl.Authorization.internal-authorization	true
whether the authorization method used is internal (true) or external (false)	
AccessControl.Authorization.authorization-package	edu.ucop.ers.access
<string> package name containing authorization classes	
AccessControl.Authorization.authorization-class	UCOPAuthorizer
<string> class that will check requests to perform functions	
AccessControl.Authorization.authorizer-mgr-package	edu.ucop.ers.access
<string> package name containing the authorization classes	
AccessControl.Authorization.authorizer-mgr-class	UCOPAuthorizerManagerImpl
<string> name of the class that implements the IAuthorizer Interface to determine whether a logged-in user may perform a given function on any one of a list of specific resources	

Configuration Element	Supplied Value
Description	
AccessControl.Authorization.my-report-authorizer-package	edu.ucop.ers.access
<string> name of the package containing the authorization classes	
AccessControl.Authorization.my-report-authorizer-class	UCOPMyReportAuthorizerImpl
<string> name of the class of the AuthorizerManager that takes care of reports designated as My Reports	
AccessControl.Authorization.my-project-authorizer-package	edu.ucop.ers.access
<string> package name containing the authorization classes	
AccessControl.Authorization.my-project-authorizer-class	UCOPMyProjectAuthorizerImpl
<string> name of the class of the AuthorizerManager that takes care of reports designated My Projects.	
AccessControl.Authorization.my-roles-checker-package	edu.ucop.ers.access
<string> package name containing the authorization classes	
AccessControl.Authorization.my-roles-checker-class	UCOPMyRolesCheckerImpl
<string> name of the class of the AuthorizerManager that determines which role(s) a user possesses.	
AccessControl.Authorization.uc-builder-package	edu.ucop.ers.access
<string> package name containing the authentication classes	
AccessControl.Authorization.uc-builder-class	UCOPUserContextBuilderImplementation
<string> name of the class that builds the User Context consistent with the IUserContext Interface. Whether you use the UCOP Authorization or not, your implementation must satisfy IUserContext. If you use UCOP authorization, your implementation must additionally satisfy IUCOPUserContext.	
CostSharing.DrillDownActive	False
Controls whether cost sharing numbers on an effort report will (true) or will not (false) provide an HTML link to an external cost sharing system	
CostSharing.DrillDownURL	http://www.ucop.edu/costsharing
<string> the URL that will be used to construct a link to the external cost sharing system. This link will be passed two parameters, the employee ID for the effort report and the project ID that is selected via the HTML link.	
Notification.MailServer	smtp.ucop.edu
<string> The hostname of an SMTP server that will accept mail from the application server.	
Notification.MailAgentHeader	Windows Eudora Version 6.1.2.0
<string> An RFC 2821 Mail User Agent acceptable to the SMTP server from above.	
Notification.AdministratorName	Effort Reporting System
<string> The text Name portion of the email address from which notifications are sent (From:)	
Notification.AdministratorEmail	ersadmin@ucop.edu
<string> The actual email Address of the email User from which notifications are sent. Insure that intended recipients do not consider this a SPAM from address.	
Notification.Reminder.RDUE	30
<number> This parameter defines the number of days prior to the certification due date that a reminder notification will be sent to the ER coordinator. A value of -1 will prevent reminder notifications of this type from being sent.	
Notification.Reminder.CDUE	15
<number> This parameter defines the number of days prior to the certification due date that a reminder notification will be sent to the PI. A value of -1 will prevent reminder notifications of this type from being sent.	
Interface.Filepath	Z:\Interfaces\Test Files\
<string> The path to campus interface files, from the perspective of the application server.	

Configuration Element	Supplied Value
Description	
DatabaseConfiguration.PropFiles.PropFile	hibernate.base.properties
<string> Common Hibernate configuration properties used by all databases.	
DatabaseConfiguration.PropFiles.PropFile	hibernate.tst01.properties
<string> Specific Hibernate configuration properties used for a particular database and database management system. Note that the PropFiles items are read hierarchically.	
DatabaseConfiguration.MapFile	mappings.xml
<string> A list (in xml format) of Hibernate class mapping files (HBM xml). Shared by all DBMS and databases. A copy may be used to test a new mapping.	
Utility.fau-formatter.package	edu.ucop.ers.utility
<string> name of the package that contains the FAU formatter class	
Utility.fau-formatter.class	FauFormatterAdapter
<string> name of the class used to format the FAU in a way that makes sense to a particular campus.	
Branding.BrandDelimiter	/campus/brand/
<string> The string that will be replaced in Tiles definitions with the CampusBrand below.	
Branding.CampusBrand	/campus/generic/
<string> The replacement values to be used for branding the web pages with the local campus page elements.	
Constants.Classname	edu.ucop.ers.ERSConstants
<string> The name of the class that will be used to load constants into the application context during application server initialization for the web application.	
Externalized.Strings.FileName	ExternalizedStrings.xml
<string> Identifies an XML file that contains customizable string literals for various parts of the web application.	
Options.allow-save-out-of-balance	False
Web application option that controls whether an out of balance effort report can (true) or cannot (false) be saved to the database.	
Options.search-fund-tab-visible	True
Web application option that controls whether the funding source search tab is displayed.	
Options.search-alt-tab-visible	True
Web application option that controls whether the alternate projects search tab is displayed.	
Options.search-primary-filter3-visible	False
Web application option that controls whether the third filter field is visible on the primary project search tab.	
Options.search-alternate-filter3-visible	True
Web application option that controls whether the third filter field is visible on the alternate project search tab.	

Customization / Branding / Skinning

Overview

The ERS application was designed so that the look of the application could quickly and easily match the established web style of each deploying campus.

There are basically four things that can be customized, campus headers and footers, CSS, graphics, and HTML.

Headers and Footers

Campus specific headers and footers are defined in files separately from the main content of the ERS application. This allows you to completely replace those files with our own files and have almost no impact on the structure of the application.

CSS

CSS is used to define the look, and to a certain extent the layout, of ERS. With a little ingenuity you could even design your own tabs and buttons.

The cascading style sheets are structured so that you can override the default styles for the application in your own CSS file and never touch the default styles. This makes it easy to maintain your campus styles and see what new styles you need to update when the application is updated.

Graphics

Graphics used in ERS are stored in campus specific folders. This will allow you to maintain your own folder and see what changes are made to the default (generic) folder when the application is updated.

Note: since dimensions are declared for graphics in the HTML source, if you diverge from the default size of an image, you will need to modify the HTML source to correctly render the image. Specifying image dimensions allows the browser to more quickly render an HTML page, although this improved performance may be negligible with fast processors and broadband network connections.

HTML

The actual HTML structure of the application should not need to be modified in order to brand the application for your campus. However, if you need to change the structure of the pages, for the most part you can. It is not encouraged for reasons that we will discuss later, along with giving you some strategies to do so effectively and safely.

Tools

You may find that the following tools will help you as you customize ERS for your own campus:

- Mozilla's Firefox browser

Firefox extensions:

- View Formatted Source Code
  allows you to view the CSS applied to a specific page element.
- IE View
  allows you to quickly view the page in IE.
- Web Developer
  does so much, but specifically has shortcuts to viewing CSS and JavaScript, disabling styles, clearing caches. All around great tool!

- Microsoft's Internet Explorer browser

Getting Started: Setting up a Campus folder

In order to make ERS easy to customize and easy to upgrade later, we have engineered the application to use a special "campus" folder that will contain your headers and footers, your style overrides, and your graphic files.

This allows you to easily switch back and forth between your instance of the application and the generic instance. This is useful when debugging your campus customization and is also very useful when upgrading ERS with a new version, since you can use the changes to the generic folder as a template for the changes that you need to make to your own folder.

Note: There isn't an override scheme for applications HTML files. HTML changes are done right in the source. For that reason, changing HTML is much more difficult to maintain when a new version of the application comes along.

We've also provided you with a sample campus folder called *ucop*. The ucop folder was used during development to test our customization scheme and may not be completely up-to-date. However, we felt that it does provide and interesting example of a campus customization.

Setting up a new campus folder

Inside the */**campus*** directory, you will find two folders, ***generic***, and ***ucop***.
Duplicate the ***generic*** directory.
Rename the duplicate directory to some useful name such as the name or initials for your campus.
Once you set up the new directory, any changes to headers, footers, CSS or graphics should be done in this folder.

Modify settings

Once you set up the new directory, you need to make one small modification to ***ERSConfig.xml*** for the application to start using your campus folder. ***ERSConfig.xml*** can be found at ***/EffortReporting/WEB-INF/classes/ERSConfig.xml***.

In ***ERSConfig.xml*** replace the word ***generic*** in the following XML with the name that you gave to your campus directory. (Do not modify the ***BrandDelimiter*** line.)

```
<Branding>
```

```
        <BrandDelimiter>/campus/brand/</BrandDelimiter>
        <CampusBrand>/campus/generic/</CampusBrand>
</Branding>
```

Restart your server and you're ready to develop your own customized ERS application.

A Customization Strategy

We suggest that you customize your instance of ERS in the following order: headers and footers, css, graphics, and finally, and only if absolutely necessary, HTML.

Note: While it may be tempting to just make changes to the CSS and .jspf files outside of your campus folder, doing so will make it more difficult to keep your installation up-to-date.

Headers and Footers

Whether you're making minor modifications to the header and footer provided or you're completely replacing the provided header and footer with your own, the process is pretty much the same. Simply edit or replace the content of the current header and footer files.

Self-contained headers and footers or wrapper

Your headers and footers can be self-contained or can act as a *wrapper* for the rest of the application. You must decide whether you want to have your headers and footers to be self-contained or have them wrap the main content.

Self-contained headers

With self-contained headers and footers, the tags for each section are complete; the header or footer could stand on its own and be valid XHTML. The header section, footer section, and main section each have their own styles defining their width and position on the page. If done correctly, they'll all line up and look right.

```
<div id="header">
</div>
```

```
<div id="main">


</div>
```

```
<div id="footer">
</div>
```

Wrapping headers and footers

Wrapping headers and footers give you slightly more control over the layout of the page by using the header section and the footer section to ***wrap*** the main section that contains the application.
The header initiates the wrapper div and the footer closes it.
Widths and positioning on the page is controlled by the wrapper – the enclosed elements are contained and constrained by the wrapper.

```
<div id="wrapper">
    <div id="header">
    </div>
```

```
    <div id="main">


    </div>
```

```
    <div id="footer">
    </div>
</div><!-- end of wrapper -->
```

Note: According the Struts documentation, you're supposed to close all tags within a given file, so we're officially cheating when we do this. But if done correctly it doesn't break anything and it buys us a lot of control.

Liquid vs. Fixed Width Layout

While not strictly a header and footer issue, this is a good place to talk about liquid verses fixed-width layout. Liquid layouts resize as the user resizes the browser. Fixed-width layout, as the name implies, have a fixed width. Both styles have their advantages and ERS supports both. Which one you choose will depend mostly on preference or precedence.

The generic layout is fixed width. To change your layout to liquid, you'll want to change the width attribute for the wrapper element to a percentage instead of a pixel width.

Exercise

If you're not sure what we're talking about here, just try this simple exercise:

1. Start up your server and open your instance of the ERS in a browser.
2. Sign-in to the Report List view.
3. In your editor, open the file styles.css in your campus folder, and change the line width: 740px; to width: 97%;, Save and refresh the page.
4. Now, try resizing the browser, and notice that the content of the page resizes with the browser.

Customizing the default header and footer files

You can modify the header and footer files pretty much anyway you like as long as all of your changes are self-contained to the header and footer files.

If you're doing a very light customization, you may want to just edit the header and footer files that are provided; change the colors, add your own logo, etc. If you would like to add links to the header file to other online campus resources, there are already some commented out links in the default header file.

Styles for the header and footer files

The styles for the header and footer provided can be found in the ***styles.css*** file in your *campus* folder. The styles in this file define only the header and footer and the wrapper div for the application.

To link to a different style sheet, such as a style sheet already in use at your campus, modify the stylesheet reference link in ***campus/<yourcampus>/head_tags.jspf.***

```
<link href='<html:rewrite page="/${campusBrand}styles/campus_styles.css" />'
rel="stylesheet" type="text/css" media="all" />
```

CSS

This guide assumes that you already know a fair amount about CSS; how it works and why it good. If you need a refresher course on CSS, you might start with: http://www.mako4css.com/Basics.htm, or http://www.cssbasics.com/.

ERS contains some very complex styling. We're not claiming that it's the "be all and end all" of CSS design—there were a lot of changes and modifications that crept into the styles that could have been more artfully done. However, we've attempted to follow a best practices approach to CSS while also making it easy for each campus to have a high-level of control when customizing the look of ERS.

The tools mentioned above, especially the Firefox ***View Rendered Source*** extension, will help you identify how the styles are applied to each element.

Note: As you make changes to the CSS, make sure to check your changes in the live application with both IE and Firefox.

CSS Structure: How to override our styles with your styles

In CSS, styles cascade from enclosing element to enclosed element. However, you can also think of styles as cascading through the various CSS documents and style declarations that are applied to the page. Just as you can declare a style to a parent element and then override that style with a style attached to a child element, you can also declare a style for an element in one style sheet and then override that style in another style sheet, on the actual page or element.

This is the hierarchy of the css files attached to every page in ERS.

`<your campus CSS file for your chrome>`

`ers_styles.css` *:the bulk of the ERS CSS is here*

`campus_styles.css` *:but you'll modify it here*

Editing CSS

Approach modifying the styles from the outside in, starting with enclosing elements (parents) and then modifying the styles attached to child elements, starting with the body and working your way inward to the smaller more localized elements on the page.

Another reason to start with enclosing objects is that an element on a page may have the same class or id as a similar element on a different page. However, because of styles applied to the enclosing element the styles applied any given element may look and behave quite differently from page to page.

The best way to learn is go in and make changes. Go ahead, be bold! Make a change and then quickly take a stroll through the app to see if your change has had the effect that you expected. You can always rollback to a previously saved version.

Tricks

Image references in the CSS files

In order to have image references in the CSS reference image files inside your campus folder, all image references are declared in the campus override CSS file in your campus folder, and not in the main CSS file. There is a comment calling this out wherever it happens.

IE Hacks

Not all CSS standards are implemented exactly the same way from browser to browser. Luckily, with most modern browsers, CSS Level 2 for XHTML documents is implemented pretty consistently. However, there are a couple of styles that are modified to support quirks in Microsoft's Internet Explorer.

There are a number of techniques to apply different styles for different browsers. Each technique has its advantages and disadvantages, and its supporters and detractors debating *ad nauseam* in the blogsphere. We have chosen to hide styles from IE using a child selector, ">".

Example

```
/* Text Buttons */
span.textbutton {
	position: relative;
	top: -2px;
	border: black 1px solid;
	padding: 0;
	margin: 0;
}

/* selector hack to hide from IE */
html>body span.textbutton {
	padding: 1px 0px 2px;
	top: 0px;
	font-size: 12px;
}
```

The child selector is part of the W3 CSS2 specification. Conveniently, IE does not support the child selector and thusly ignores any CSS statements defined in this way.

There is a danger that some day Microsoft will release a version of IE that will support this part of the W3 specification. However, it is hoped that they will clean up their CSS bugs at the same time.

A few other CSS files

There are a couple of other CSS files that you will need to know about.

Printing styles are defined in ***/styles/ers_print.css.*** You can also override printing styles in your campus_styles.css file using the `@media print` declaration.

The styles for the tooltip style popups are declared in ***/styles/nicetitles.css***.

Graphics

Your graphics in your folder

All the graphics files for ERS are in the images folder inside the campus folder that you set up. You can easily change the look of your application by replacing or modifying these images.

Sizes

However, don't change the size of a particular graphic element unless you have to. Most of the images in the source are specified with height and width attributes to increase performance. If you change the size of an image, you will need to change it's size in the source.

HTML

Changing the HTML really should be done only in the last resort. Changing the HTML in the source files must be done with great care since it may change the behavior of the page. And, changes to source will need to be resolved each time the application is updated.

Hide don't delete fields

When changing HTML on pages that use forms, do not delete any fields, as these fields are referenced by the application both in the Java and, in many cases, by the JavaScripts on the page. Missing fields will result in errors. To remove a field, change the type of the field to hidden. In most cases this is done by changing the struts tag from `html:text` to `html:hidden`. You may also need to attach a default value to the hidden field as well in order to pass the right value to the application.

Comment, Comment, Comment

Don't forget to comment your changes to the source. Use a standard format and a key word in your comments so you can easily search for them when doing updates.

Example:

```
<%-- CHANGE 2/17/06 Eli: Changed case of field value to upper --%>
```

Test

I guess it goes without saying that any change that you make, you should test thoroughly. Make sure to test in all the browsers that you plan to support.

Authentication

Authentication in ERS is the process of making the decision whether a user can access the web application.

ERS supports two modes of authentication, Internal and External. As released, ERS is configured to use internal authentication.

Internal Authentication

Internal authentication uses an application login page, and compares the entered user ID and password against an application table, ERSUser. The user interface supports user-requested password resets via email, and the System Administration module also allows a password to be reset by a central administrator.

After the login page validates the user's password, it constructs an object known as a UserContext. This object contains information about the logged in user, such as their user ID, full name, email address and employee ID among other items. This object is key to the operation of the web application and is required to be present for all application users.

External Authentication

An external authentication mechanism may be used for ERS. The login process would be managed by another application which would have responsibility for user ID and password verification.

Using an external authentication mechanism requires that the authenticator fulfill several responsibilities prior to attempting to access ERS on behalf of their user. These steps are:

1. construct a UserContext object
2. populate it's fields with the user's attributes
3. place the object into the session object field USER_CONTEXT
4. transfer control to the ERS main entry point, EnterERS.

If the UserContext is null or otherwise invalid, the ERS entry point EnterERS will exit without allowing access to the application.

Authorization

Authorization in ERS is the process of making decisions about a user performing an action on a specific application resource, and in some cases, obtaining email and full name information about the user.

The ERS authorization scheme consists of several interfaces that capture the types of decisions the system must make.

The interfaces are:

IAuthorizer
IMyProjectsAuthorizer
IMyReportsAuthorizer
IMyRolesChecker
IEmployeeInfo

ERS provides internal implementations of all these interfaces. Campuses can choose to rely entirely upon the ERS internal system, they can completely rely on an external system, or they can choose to use a mix of both internal and external implementations.

There is support for implicit permissions, and exclusion lists from the implicit permissions. The implicit permissions are termed "My Projects" and "My Reports". The "My Projects" implicit permission means a principal investigator has permission to view, edit, certify and view payroll data for any report that contains at least one of his/her projects. The "My Report" permission means any user with a report in the system can view, edit, certify, and view payroll data for their own reports. Users who would otherwise have these permissions can be explicitly excluded from them. Furthermore, users can be excluded from certifying their own reports, while still retaining the other permissions. Internally this is done by maintain lists that are managed through the System Administration module.

Interfaces

IAuthorizer

The core interface is IAuthorizer. A campus that implements this interface can support all authorization requirements of the application.

Every ERS application instance must provide an implementation of this interface. The implementing class must be specified in ERSConfig.xml with the following items:

AccessControl.Authorization.authorization-package
AccessControl.Authorization.authorization-class

ERS provides a concrete implementation of this interface for internal authorization:

edu.ucop.ers.access.UCOPAuthorizerImpl

IMyProjectsAuthorizer

To enforce the "My Projects" business rule (i.e. any principal investigator can perform any intent on any report that contains at least one of his/her projects unless explicitly excluded otherwise) when performing an authorization check, ERS makes a call on this interface.

Every ERS installation will have an implementing class, either a custom-build class that connects to the campus-specific authorization application that can return a decision based on the provided parameters, or one of the two ERS-provided implementations. The implementing class must be specified in ERSConfig.xml with the following items:

AccessControl.Authorization.my-project-authorizer-package
AccessControl.Authorization.my-project-authorizer-class

Since this permission can be explicitly denied to specific users, the interface call relies on another interface, IMyRolesChecker, to enable that business rule.

ERS provides a trivial implementation of this interface,

edu.ucop.ers.access.SimpleMyProjectsAuthorizerAdapter

for external implementations that wish to use IAuthorizer to handle the self-permission role.

ERS also provide the class

edu.ucop.ers.access.UCOPMyProjectAuthorizerImpl

which is a full implementation of the My Projects interface, supported internally by a persisted list of excluded users. This list is populated via the System Administration module.

IMyReportAuthorizer

To enforce the "My Reports" business rule (i.e. any ERS user with a generated report can perform any intent on that report unless explicitly excluded otherwise) when performing an authorization check, ERS makes a call on this interface.

Every ERS installation will have an implementing class, either a custom-built class that connects to the campus-specific authorization application that can return a decision based on the provided parameters, or one of the two ERS-provided implementations. The implementing class must be specified in ERSConfig.xml with the following items:

AccessControl.Authorization.my-reports-authorizer-package
AccessControl.Authorization.my-reports-authorizer-class

Since this permission can be explicitly denied to specific users, the interface call relies on another interface, IMyRolesChecker, to enable that business rule.

ERS provides a trivial implementation of this interface,

edu.ucop.ers.access.SimpleMyReportsAuthorizerAdapter

for external implementations that wish to bypass the implicit my reports permission and pass through to the IAuthorizer call.

ERS also provides the class

edu.ucop.ers.access.UCOPMyReportAuthorizerImpl

which is a full implementation of the my report interface, supported internally by a persisted list of excluded users.

IMyRolesChecker

This interface works in conjunction with the IMyProjectsAuthorizer and IMyReportsAuthorizer interfaces to enforce the self-permission business rules described above.

As in the My Reports/My Projects interfaces, ERS provides trivial adapter implementations that allow campus customizations to just use the authorization call in the IAuthorizer.

The implementing class must be specified in ERSConfig.xml with the following items:

AccessControl.Authorization.my-roles-checker-package
AccessControl.Authorization.my-roles-checker-class

IEmployeeInfo

This is a simple struct-like type that associates an employee's list of email addresses with a formatted version of the employee's name. Since it is only ever used in the context of a return value from one of two method calls defined in IAuthorizer, there is no need for it to be described by the configuration scheme.

Levels of Customization

There is a true/false flag configuration item in ERSConfig.xml that indicates whether the application should be run under internal authentication or external authentication:

AccessControl.Authentication.internal-authentication

External

IAuthorizer/IEmployeeInfo

If a campus's authorization system can provide support for the all authorization decisions the system requires, including the "My Projects" and "My Reports" business rules, then its customization will consist solely of implementing the IAuthorizer and IEmployeeInfo interfaces. As part of this customization, the campus will specify in ERSConfig.xml that the system should use the trivial bypass implementations of the IMyReportsAuthorizer, IMyProjectsAuthorizer, and IMyRolesChecker interfaces:

Interface	Trivial Adapter
edu.ucop.ers.access.IMyProjectsAuthorizer	edu.ucop.ers.access.SimpleMyProjectsAuthorizerAdapter
edu.ucop.ers.access.IMyReportsAuthorizer	edu.ucop.ers.access.SimpleMyReportsAuthorizerAdapter
edu.ucop.ers.access.IMyRolesChecker	edu.ucop.ers.access.SimpleMyRolesCheckerAdapter

This approach will require the implementation of the IEmployeeInfo interface, but this is expected to be trivial.

Full External Implementation

A campus can also choose to use their external system to implement the IMyReportsAuthorizer, IMyProjectsAuthorizer, and IMyRolesChecker interfaces if it makes sense when analyzing the resources of the campus's own authentication system.

Mixed Implementation

If a campus's authorization system cannot provide support for the "My Projects" and/or "My Reports" business rules, then it can customize the authorization by implementing IAuthorizer while relying on the internal implementations of the self-role business rule(s) as needed. This approach then requires the campus to use the administration user interface to populate the exclusion tables with the user that should be excluded from the self-roles as needed.

Interface	Trivial Adapter
edu.ucop.ers.access.IMyProjectsAuthorizer	edu.ucop.ers.access.UCOPMyProjectsAuthorizerImpl
edu.ucop.ers.access.IMyReportsAuthorizer	edu.ucop.ers.access.UCOPMyReportsAuthorizerImpl

Internal Implementation

If a campus lacks an authorization system or it cannot support the decisions represented by the IAuthorizer interface, then it can rely on the internal ERS authorization system. This requires using the administration user interface to assign roles, permissions, etc to users, and to populate the exclusion tables.

Account Number (FAU) Display

ERS provides a feature to allow customization of the FAU presentation when this data element is displayed in the web application. Although the FAU has a common definition of 30 characters, across installations the values in this field can vary in their format and interpretation.

ERS provides an interface called IFauFormatter for the purpose of formatting the FAU for display. A trivial implementation of this interface is provided in class FauFormatterAdapter. This implementation may be modified to meet installation-specific requirements.

If a customized version of the FauFormatterAdapter class is used, it must be identified in the ERS configuration file, ERSConfig.XML

Relabeling Fields and Customizing Message Text

ExternalizedStrings.XML
Several field labels in the web application are customizable to allow installations to choose different wording to refer to various elements of the effort report. This set of labels can be customized by editing the configuration file ExternalizedStrings.XML.

The supported customizable field labels and their default values are:

Field Name	Default Value
ProjectName	Sponsored Project Name
ProjectNameAlt	Grant Title
ProjectId	Sponsored Project ID
ProjectIdAlt	Alternate Project ID
SponsoredTitleLabel	Sponsored Projects
SponsoredTotalLabel	Total Sponsored Projects
SponsoredNoteLabel	requiring certification
OtherTotalLabel	Total Other Effort
OtherNoteLabel	not requiring certification
GrandTotalLabel	Grand Total
NegCSLabel	Cost Sharing Offset Against Other Sponsored Projects
ERcol1Head1	Effort report column heading
ERcol1Head2	Effort report column heading
ERcol2Head1	Effort report column heading
ERcol2Head2	Effort report column heading
ERcol3Head1	Effort report column heading
ERcol3Head2	Effort report column heading
ERcol4Head1	Effort report column heading
ERcol4Head2	Effort report column heading
ERcol5Head1	Effort report column heading
ERcol5Head2	Effort report column heading
ERcol6Head1	Effort report column heading
ERcol6Head2	Effort report column heading
OriginalLabel	Effort report drill-down label
PeopleTabLabel	Label for the Person search tab
DeptsTabLabel	Label for the Departments search tab
PrimaryTabLabel	Label for the primary (first) projects search tab
PrimaryFilter1Label	Label for the first filter field on the primary projects tab
PrimaryFilter2Label	Label for the second filter field on the primary projects tab
PrimaryFilter3Label	Label for the third filter field on the primary projects tab
AlternateTabLabel	Label for the alternate (second) projects search tab
AlternateFilter1Label	Label for the first filter field on the alternate projects tab
AlternateFilter2Label	Label for the second filter field on the alternate projects tab
AlternateFilter3Label	Label for the third filter field on the alternate projects tab

FundTabLabel	Label for the Funding Sources search tab

Note: Changes to this file requires a restart of the application server to reload the new file contents.

ApplicationResources.Properties
All strings in the application, including field labels, message texts, prompts, etc., are defined in the file **`ApplicationResources.Properties`** which is located in the source package **`edu.ucop.ers.struts`**. Further *modification* of the application can be accomplished by editing this file with a normal text editor.

Note: Changes to ApplicationResources.Properties are considered *modifications* and will not be preserved across releases. Campuses modifying this file will be responsible for merging local changes after installation of a new release.

Document Change Log

Date	Build #	Changes
2/15/06	Release 0	Initial draft for technical review
2/17/06	Release 1	Finalization of the customization section
4/10/06	Release 2	Document new page label customization

Changes to this document from the prior release are indicated by revision bars in the left margin. Minor editorial corrections have not been flagged with revision marks.