WEBAPPZ

LISTedit framework

Application Programmer's Interface (API) Reference Document

Version 2.0.1 – June 2004

For Java 1.4.2+ with WebObjects 5.2+ on Windows, Mac OS X, Solaris, and Linux

TABLE OF CONTENTS

Introduction

Application development framework for consistent graphical user interface

RCMListEdit.framework is a Java framework, which provides the infrastructure needed to quickly implement a variety of robust business applications. The framework operates in conjunction with Apple's WebObjects™ environment, WebObjects 5.1/5.2 on Windows 2000, MacOS X, and UNIX server platforms. RCMListEdit is fully documented, easy-to-use, and in very little time, will turn new WebObjects developers into pros. A tutorial application, called System Environment Reservation and Tracking System (SERTS), is included with the documentation of how it was implemented and all of its source code. RCMListEdit makes applications easy to develop, and ultimately easy to use by the client.

Truly rapid application development turn-around time!

A mid-sized e-business application (e.g. 12 database tables and 30 dynamic web pages) can easily be constructed in under 40 hours! Our powerful framework dramatically minimizes the amount of code that you have to write – not only the HTML and bindings, but also the Java because of the many built-in validations. Since the framework is pre-compiled, your application using the framework compiles faster too.

Turns your programmers into Pros!

RCMListEdit turns WebObjects™ into an object-oriented tool that is truly accessible to every Java/Database software developer. Even junior Java programmers need only have a basic familiarity with the EOModeler and ProjectBuilder applications (not even WOBuilder) in order to produce fully-functional applications which automatically include rock-solid data integrity protection: RCMSafeCoding™ and WEBAPPZ's database row-level record locking mechanism. Get all the power of the WebObjects libraries without the steep learning curve! The RCMListEdit framework is a re-usable compilation of all the important aspects found in WEBAPPZ's past and anticipated future e-business projects and is used by WEBAPPZ for its product development and as part of its consulting services.

Web-enable any database schema!

RCMListEdit provides all of the functions you would expect to support administration, transaction entry, and read-only users for any kind of online application! The compact nature of the coding also makes it easier to document your code. We prefer to use the Boyce-Codd Normal Form (BCNF) to ensure properly modelled databases.

Product Features

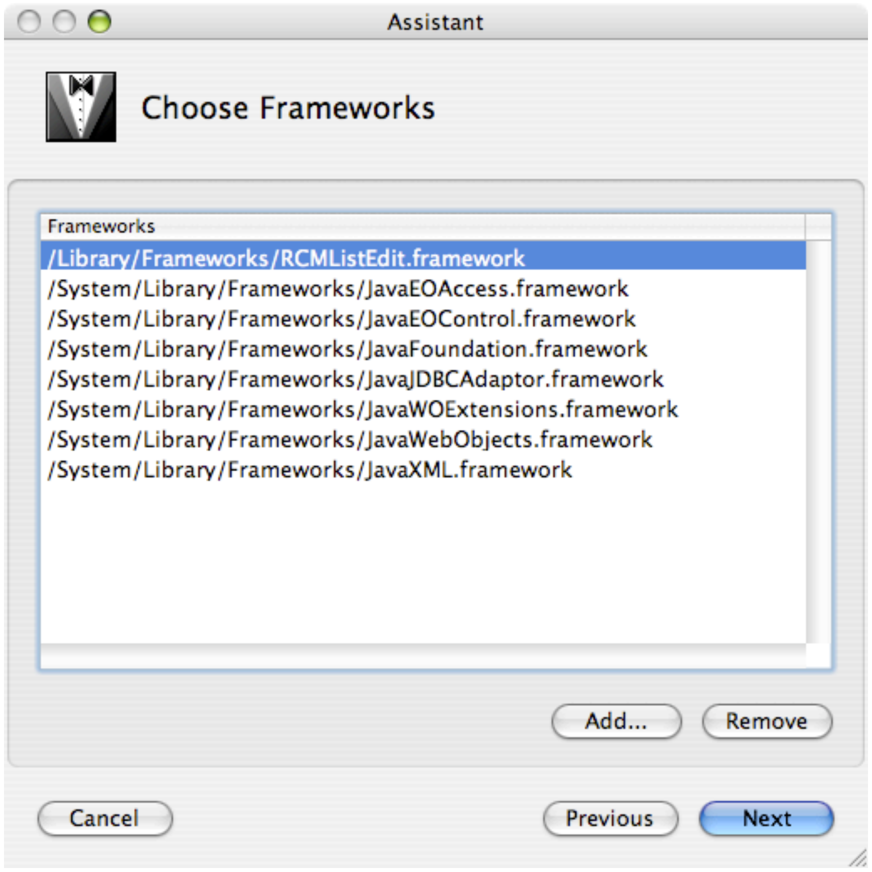
- Entire graphical user interface (GUI) is programmable with a **JAVA-based API**. No need to use fixed HTML coding templates, which is the default in WebObjects.
- Gives your web-based applications a **harmonized graphical user-interface**, using a consistent look-and-feel for all search, list, and detail editing views.
- **Data Export** button on every list and edit page allows users to export data from lists and detail edit pages to spreadsheets, like Microsoft Excel, in comma or tab separated formats.
- Print button creates professional color **reports in portable document format (PDF)**, based on the data in your lists and detail edit pages, without any additional programming.
- Help button provides intelligently generated **user-friendly help** for all your search, list and detail editing pages.
- **Automated JAVA-script calendar** to pick dates for all your data format fields, using a calendar view.
- **Sophisticated Login/Logout controls** with 8 security validations, optional cookie handling, and “expired password changing” support.
- **Login administration and password management**, including support for custom attributes such as “department_code” or “security_group”.
- **Flexible session management** including built-in functional and data security.
- **Data model independent** - Query By Example (QBE) search/select views with validated criteria for common data types; Paged lists of records that can be added, edited or deleted; Editing views with built-in validations for common data types; Selection lists with paged choices; Detail views and URL launching; Pop-ups that are simple to make yet fully featured.
- **Customization features -** Layout attributes, templates, date, and number formats that are individually unique or common
- **JavaScript rollover buttons** for 3 different purposes
- **DHTML controls** to maximize and minimize panels

Integrating RCMListEdit with your application

RCMListEdit.framework is a WebObjects Framework directory and should be installed in (/Apple)/Local/Library/Frameworks on Windows/Solaris/Linux, and /Library/Frameworks on Mac OS X along with any other frameworks that may already exist at that location.

Xcode – Mac OS X from version 10.3:

A: When you create a new “WebObjects Application” or “WebObjects Framework” type of project, add the RCMListEdit.framework as shown below.



/Library/Frameworks/RCMListEdit.framework
/System/Library/Frameworks/JavaEOAccess.framework
/System/Library/Frameworks/JavaEOControl.framework
/System/Library/Frameworks/JavaFoundation.framework
/System/Library/Frameworks/JavaJDBCAdaptor.framework
/System/Library/Frameworks/JavaWOExtensions.framework
/System/Library/Frameworks/JavaWebObjects.framework
/System/Library/Frameworks/JavaXML.framework

B: With an existing Xcode project:

- 1 Click on the “Frameworks” folder in the left folder bar.
- 2 Click on the “Project” menu, and then “Add Frameworks…”.
- 3 Select RCMListEdit.framework in /Local/Library, and click “Add”
- 4 Please ensure that the target is “Application Server”.

ProjectBuilder – Windows and Mac OS X prior to version 10.3:

In the browser of your PB.project, double-click the Frameworks suitcase to bring up the Add Frameworks panel. Browse to the RCMPDFKit.framework and click the Open button. If the path to (/Apple)/Local/Library/Frameworks is not already in your search order, you may agree with the dialog box suggesting to add it.

From Your Java Classes: `import com.webappz.listedit.*;`

In your **Application.java** file, <u>you must change</u> the following

```
public class Application extends WOApplication {
```

to

```
public class Application extends RCMApplication {
```

In your **Session.java** file, <u>you must change</u> the following

```
public class Session extends WOSession {
```

to

```
public class Session extends RCMSession {
```

This makes your Application and Session objects sub-classed from our enhanced implementations of the Application and Session classes (i.e. RCMApplication extends Apple's standard WOApplication class, and RCMSession extends the standard WOSession class), preserving the basics but adding more features).

RCMApplication (extends WOApplication)

Declare the sub-class as described previously, otherwise the RCMListEdit framework will not work properly. The following attributes may be set at any point in the source code with a reference to (RCMApplication)application(). Alternatively, in the application's creator/init method -- public Application() -- you can set attributes with a simple call to the accessor methods immediately.

Attributes

Each of the following attributes may be set and retrieved with accessor methods named according to the convention. For example:

```
// Set sessionRestorationPage in RCMApplication's framework
setSessionRestorationPage("Logout");

// Get sessionRestorationPage (pointer to a WOComponent)
pageToShowWhenFinished = sessionRestorationPage();
```

String createUserId_name (default = "z_createUserId"; OPTIONAL)

Provide the name of the attribute of *anInstance* in which to store the *session.currentUserId* on a newly created record when the Save button is pressed and all validations (RCM's and your optional custom ones) have passed successfully. The record is initially created in a temporary editing context when the Add button on a previously associated RCMList is pressed, but the createUserId is only stamped on the finally inserted record.

String createDate_name (default = "z_createDate"; OPTIONAL)

Provide the name of the attribute of *anInstance* in which to store the current date on a newly created record when the Save button is pressed and all validations (RCM's and your optional custom ones) have passed successfully. The record is initially created in a temporary editing context when the Add button on a previously associated RCMList is pressed, but the createUserId is only stamped on the finally inserted record.

String modifyUserId_name (default = "z_modifyUserId"; OPTIONAL)

Provide the name of the attribute of *anInstance* in which to store the *session.currentUserId* on an existing record when the Save button is pressed and all validations (RCM's and your optional custom ones) have passed successfully.

String modifyDate_name (default = "z_modifyDate"; OPTIONAL)

Provide the name of the attribute of *anInstance* in which to store the current date on an existing record when the Save button is pressed and all validations (RCM's and your optional custom ones) have passed successfully.

These are application-wide settings of the attributes used on all database objects/instances for audit logging when saving objects. When each RCMEdit tries to save an object, it checks these attributes and attempts them to update them for whichever table/class you are using. This feature is optional -- if the attribute identified does not exist, the save will still complete without an error. It is recommended that each entity in your model use the same attribute names so that this application-wide setting can be used by all of your application's RCMEdit objects (otherwise see the RCMEdit section below to set the auditing attribute names individually). In this example, if your objects have a createUserId, in your application's creator/init method you would use:

setCreateUserId_name("createUserId");

The UserId is taken from RCMLogin's userId_name, and is available from the session variable named currentUserId.

The default attribute names begin with z_ so that in EOModeler, which sorts alphabetically, the auditing attributes are the last ones shown. Hint: if the database column names vary across entities but have the same type and length, you may want to simply rename the corresponding attribute names in the EOModel so that you can take advantage of this feature!

String sessionRestorationPage (default = null)

This is where you define the page the user will see if the session times out or has an error, and you want to present a friendly message rather than allow the intimidating WebObjects default error page to appear. If not set, the framework will display its own RCMLogout component as a self-containing page. For more information about this page, refer to the RCMLogout explanation.

It is recommended that you make your own page, e.g. Exit, which contains an RCMLogout component, declared in your Exit.wod file, and set this as the sessionRestorationPage.

String passwordEncryptionKey (default = application name)

The default encryption key is a string, which is used in the calculation of encrypted values for fields such as the password. If not set, the key defaults to the string that is the name of your Application (based on the Project name). This key is used in the

RCMLogin component, the RCMLoginAdmin component, and the ‘password’ type of RCMEditCell.

NSGregorianDate maxAllowedDate (default = December 31, 2099)

The largest acceptable date across the whole application, defined in the NSGregorianDate type. This is used in the validation of date fields, primarily in the RCMEdit component.

Replace Framework Buttons

void set*Function*Button (new RCMJavaButton ("pic_up.gif","pic_dn.gif,width,height))

The images used for the following standard function buttons can be over-ridden with user-supplied images in your application's WebServerResources directory: Search, Save, Reset, Cancel, Return, Delete, Clear, Add, Next, Prev, SaveSelected, Change, Login. For example:

```
setSearchButton(new RCMJavaButton("rcm_edit_search_up.gif",
                                  "rcm_edit_search_dn.gif",107,21));
setSaveButton(new RCMJavaButton("rcm_edit_save_up.gif",
                                "rcm_edit_save_dn.gif",107,21));
```

When any of these are used, each RCMJavaButton's attribute useImageFromGrandparent will be set to true.

Methods

public String passwordEncrypt(String password)

This is the application-wide way to encrypt a given plaintext string using the passwordEncryptionKey, and return an encrypted string. This method is used by other components in the RCMListEdit framework for encrypting fields that need have their data kept confidential in the database, such as password fields.

public WOResponse handleSessionRestorationErrorInContext(WOContext aContext)

This method is used the RCMListEdit framework and does not need to be explicitly called by your application. It tries to show the restoration page when required, otherwise it shows the RCMLogout page.

RCMSession

Declare the sub-class as described previously, otherwise the RCMListEdit framework will not work properly. The following attributes may be set at any point in the source code with a reference to (RCMSession)session(). In the session's init method you can set attributes with a simple call to the accessor methods immediately.

Attributes

Each of the following attributes may be set and retrieved with accessor methods named according to the convention (e.g. setDateFormat(newDateFormat); and dateFormat = dateFormat();).

Security Tracking

String currentUserId (Default = "no_user_defined")

A string representation of the current login's userID, so that you don't have to message the currentLogin object to get it (since the ID may be used frequently by the RCMFramework for audit logging, i.e. z_createUserId, z_modifyUserId, etc.). It will be set by RCMLogin.

Object currentLogin (Default = null)

An object that is the current login. This is determined the RCMLogin based on successful login. This currentLogin is the object corresponding to the record in the table which used to validate logins.

For example, if you want to retrieve another attribute from that record, such as the user's first name, use a reference such as `((Employee) (RCMSession) session().currentLogin()).firstName().` In the WOD file for a WOString, you can access the same information with value = session.currentLogin.firstName;

String currentSourceIp (Default = "unknown")

A string of the IP address that originated the current session (once the application is deployed), which is set by RCMLogin. The returned value is "unknown" if not available from the web server environment variables.

Layout

The following attributes allow you to customize the appearance of all of the visual components in the framework, such as RCMLogin, RCMLogout, RCMList, and RCMEdit. You can change these settings at any point in the execution of your application – session formats can be overridden in the individual components! It is not necessary to change the defaults, which provide a clear and visually appealing layout for you already. These defaults will be rewritten as HTML tag attributes, so you do not need to wrap them in HTML tag markers (less-than "<" or greater-than ">" symbols).

String listOutsideTableFormat
String listTableFormat (Default = "WIDTH=100% CELLPADDING=2 CELLSPACING=2 BORDER=0")

The layout of the entire table used by an RCMList. Here you provide attributes that would normally have gone into a <TABLE> tag, such as Width (in pixels or percent), Cell Padding (in pixels), Cell Spacing (in pixels), and Border Width (in pixels).

String listHeadingFont (Default = "FACE=arial,helvetica SIZE=2")

The font and size to use in the table heading row (first row above the data). For example, by specifying both Arial and Helvetica, the browser will use whichever is available on that operating system. The size is in the 1-6 scale where 1 is largest and 6 is smallest to provide a typical table heading font style.

String listHeadingColor (Default = "#BFBFBF")

A string containing the background color to use in the table heading cells (and the navigation footer of each list table). The color is defined in standard hexadecimal RGB values. The default is a light grey color.

String listRowFont (Default = "FACE=arial,helvetica SIZE=1";)

The font and size to use in the data rows of a table used for a list.

String listRowColor (Default = "#CCCCFF"")
String listAltRowColor (Default = "#CCCCEE"")

A string containing the background colour for use in the data rows of a table used for a list. The color is defined in standard hexadecimal RGB values. The default is a light blue color. Alt is for odd rows

String listSelectedRowColor (Default = "#FFFF99"")

A string containing the background colour for use the selected row of a list table. The default is a pale yellow color.

Each of the following attributes operate identically to the list…. formatting methods described above, except that they apply only to the RCMEdit components in Edit mode:

String editOutsideTableFormat
String editTableFormat (Default = "WIDTH=100% CELLPADDING=2 CELLSPACING=2 BORDER=0")
String editHeadingFont (Default = "FACE=arial,helvetica SIZE=2")
String editHeadingColor (Default = "#BFBFBF")
String editRowFont (Default = "FACE=arial,helvetica SIZE=1")
String editRowColor (Default = "#CCCCFF"")
String editAltRowColor (Default = "#CCCCEE"")

Each of the following attributes operate identically to the list…. formatting methods described above, except that they apply only to the RCMEdit components in Select mode (e.g. in a Search panel for Query By Example searches):

String selectOutsideTableFormat
String selectTableFormat (Default = "WIDTH=100% CELLPADDING=2 CELLSPACING=2 BORDER=0")
String selectHeadingFont (Default = "FACE=arial,helvetica SIZE=2")
String selectHeadingColor (Default = "#BFBFBF")
String selectRowFont (Default = "FACE=arial,helvetica SIZE=1")
String selectRowColor (Default = "#CCFFCC"")
String selectAltRowColor (Default = "#CCFFCC"")

General Formatting Variables

The following session attributes can be used to make certain numeric and date formats consistent throughout your application. The framework will also rely on these formats. For an explanation of these formats, refer to the WebObjects documentation.

String dateFormat (Default = "%d/%m/%Y")
String currencyFormat (Default = "#,##0.00")
String numberFormat (Default = "0.000")
String currencyCode (Default = "US$")

The default currency label to put in front of monetary amounts (a LISTedit feature).

Inter Edit-List Variables

A number of attributes control how information is passed between the list components and the related edit components. After these variables have been accessed, they will be cleaned up automatically. The following should not be used directly by your application, therefore they are not documented in detail here: nextPage, prevPage, selectedInstance, needsToBeAdded, tempEC, selectQualifier, detailKey, and masterObject.

Methods

void addMessage (String message)

This will add a message to an array of messages which are queued for display within an RCMMessageWindows component. Generally RCMList and RCMEdit will display these messages because they already have this messaging component embedded within themselves. If you need to inform the user of urgent information during the session, you should use this method (i.e. "You cannot delete this Business since it is used by one or more Licenses which require a Business"). See later section for RCMMessageWindows.

int nextInt ()

Returns the next available integer (unless the MAX_VALUE has been reached, in which case it returns 0). You do not need this method to assign integer primary keys, since EOF will do that for you, but you may want to use it for other purposes in your code.

Object valueForObjectAndKeys (EOEnterpriseObject object, String keys)

Returns the accessed value (could be any object, which must be cast to the right type for use, e.g. String) for the provided recursively traversed keys (one or more, either accessor method name(s) or variable name(s), e.g. toContact.address.cityName) in the given object (e.g. myBusiness). For more information refer to the Key/Value coding topic in the WebObjects documentation – e.g. if you use this method for a single key, it is essentially the same as `Object valueForKey(String aKey)`.

void takeValueForObjectAndKeys (Object value, EOEnterpriseObject object, String keys)

Sets a value (e.g. value = "Vancouver" cast back to Object) for the provided *recursively traversed* keys (one or more, ending with either set method name(s) or variable name(s), e.g. toContact.address.cityName) in the given object (e.g. myBusiness). For more information refer to the Key/Value coding topic in the WebObjects documentation – e.g. if you use this method for a single key, it is essentially the same as `Object takeValueForKey(Object value, String aKey)`.

void saveChanges ()

Whenever you (or the framework) want to update your objects to the database, invoke this method. Code in here safely ensures that the changes are fully committed and that any errors are trapped without showing a WebObjects exception page.

void resetChanges ()

Performs a roll-back on the default editing context by undoing all changes (invalidateAllObjects) since the last successful saveChanges.

boolean exclusiveAttributeForNameAndInstance (String attributeValue, attributeName, anInstance)

Returns TRUE or FALSE to say whether the string provided already exists as the value for the attribute name in another instance. For example, to check for duplicate user names during RCMLoginAdmin, this method is called as:

exclusiveAttributeForNameAndInstance ("Frank", "userName", selectedUser)

It will return FALSE if there are one or more other instances that also have "Frank" as their "userName".

EOEnterpriseObject getTempECCopy (EOEnterpriseObject sourceObject)

Make a new copy of the sourceObject in the session's current temporary editing context. This is useful in RCMList's addingAction to hook up relationships between saved objects and new objects that are being added ('faulting' with the temporary editing context is performed).

RCMMessageWindows

The RCMMessageWindows component provides a mechanism to display queued messages in the session to the user. JavaScript alert panels ("OK" button only) are created for each message that was pending in the session. You usually don't need to use this, since both RCMList and RCMEdit already have this component embedded in them.

If there are no messages to display, the result of this component will be blank (no additional HTML will be generated as part of the response that creates the current page). The messages will be shown in entered order:

Example: session.addMessage(“Thanks for completing this registration form”);

RCMLogin

The RCMLogin component provides a complete set of features for allowing valid users to log in, including options such as account/password expiry, password change self-service panel and cookie management.

The fields prompted for on the component are: "user id:" and "password:". If both fields are empty, the cursor will focus on the user id field – if the user id field has already been correctly set (i.e. by a cookie) then the cursor will focus on the password field. The IP address is also shown to discourage hackers, since they will see that their attempts are being monitored.

For example, in the web page of a component, all you need to do to put a login panel on your page is this:

```
     <WEBOBJECT NAME=MyLogin></WEBOBJECT>
```

All the variables are passed in to the component from the WOD file where you call the RCMLogin.

```
MyLogin: RCMLogin {
     pageAfterLogin = "WelcomePage";
     entity = "Contact";
     userId_name = "login_userId";
     password_name = "login_Password";
     dateSuspended_name = "login_Date_Sus";
     dateExpires_name = "login_Date_Exp";
     dateLast_name = "login_Date_Last";
     datePasswordExpires_name = "login_Date_PassExp";
     attemptsGood_name = "login_Attempts_Good";
     attemptsBad_name = "login_Attempts_Bad";
     cookieAppId = "MyGreatTestApplication";
}
```

Note: you are not required to use all of the above parameters should you not wish to do so.

Attributes

String entity (REQUIRED)

This is the table name that is used to fetch login-related attributes (i.e. a class that has at least a username and password attributes). You may want to make this entity useful for a purpose other than just logins (i.e. Contact, Employee, etc.), by having other non-login type attributes such as firstName, lastName, and a Social Security Number or Social Insurance Number, if the entity were called "Person" for example.

String userId_name (REQUIRED)

This is the entity's attribute name that is used to store the user's unique identifier (e.g. user_id, login_name, etc.). Currently the framework validates this field to be no more than 15 characters.

String password_name (REQUIRED)

This is the attribute name that is used to store the user's password as an encrypted string (based on the encryptionKey in the RCMApplication). Currently the framework validates this field to be no more than 15 characters.

String dateSuspended_name (OPTIONAL)

This is the attribute name that is used to store the date on or after which the account was disabled (Note: the Administrator can still re-activate the account, but the user cannot). The date will be set at the time that the maximum allowable number of failed login attempts (maxBadAttempts) has been exceeded (e.g. 3 times).

String dateLast_name (OPTIONAL)

This is the attribute name that is used to store the date of the most recent successful login.

String dateExpires_name (OPTIONAL)

This is the attribute name that is used to store the date on or after which the account will be completely disabled (Note: the Administrator can still re-activate the account, but the user cannot). For example, the date may be set manually to reflect a pre-paid, time-limited subscription.

String datePasswordExpires_name (OPTIONAL)

This is the attribute name that is used to store the date on or after which the user is forced to change his or her password. Note: the password change panel will be shown if the password has expired. Upon successful entry of a new password, the date will be set to today's date plus the number of days specified in passwordExpiryDateAdvanceDays (see below).

String attemptsGood_name (OPTIONAL)

This is the attribute name that is used to store the number of successful logins. This will increment automatically after every successful login.

String attemptsBad_name (OPTIONAL)

This is the attribute name that is used to store the number of unsuccessful logins. This will increment automatically after every failed login. Note: if this exceeds the maxBadAttempts, the account will be disabled automatically (this is <u>not dependent</u> on the dateSuspended_name feature being used).

int maxAttemptsBad (Default = 3)

This is an integer that defines the maximum number of unsuccessful login attempts that will be allowed for the account. WARNING: if attemptsBad_name is not used, this test only applies to the current session!

int passwordExpiryDateAdvanceDays (Default = 30)

This is an integer that defines the number of days that will be added to the current date for the purpose of resetting the value in the datePasswordExpires_name.

String encryptionKey (Default = null; OPTIONAL)

This string is used only for encrypting the password field on the login panel, using the standard UNIX crypt() command. If you leave this set to null (<u>strongly recommended</u>), then the application-wide encryption methods will be used with the application's encryption key instead. To completely disable password encryption (temporarily or permanently), use `encryptionKey = "";`

String cookieAppId (Default = null; OPTIONAL)

This string is used to uniquely identify cookies associated with your application. If you set this attribute to an arbitrary string (e.g. "MyGreatTestApplication"), then the userId of the last successful login will be used to set a cookie for the user to login more easily next time (user name only). Note: if an application has more than one kind of login panel, be sure to name this attribute uniquely! If the value is null, then cookies are not used at all.

String pageAfterLogin (REQUIRED)

The name of a WOComponent to display after the user has logged in successfully.
The following formatting attributes default to the values that have already been set as session attributes, but they can be overridden here.

RCMLogout

The RCMLogout component is used to display a confirmation page saying that your session has ended for this application (by name) and you have been logged out. The component optionally may contain a logo. A link is provided to re-enter the application.

It is recommended that you make your own Exit page (not necessary to be named Exit exactly) which includes the RCMLogout component. You should also set your Exit page as the setSessionRestorationPage in your Application.

Similarly to RCMLogin, the following attributes should be set in your Exit.wod. All attributes are optional – the page will function correctly with the defaults included in the framework. If you wish to include a logo, then you must use all four logo-related attributes. The height and width of each image should be the same so that the JavaScript rollover does not cause a rescaling of the image. For example:

```
MyLogout: RCMLogout {
     logoutLogoUp = "goodbye.gif";
     logoutLogoDown = "goodbye-pressed.gif"
     logoutLogoHeight = 275;
     logoutLogoWidth = 326;
}
```

The logo will be displayed as a RCMJavaButton (see later section).

Attributes

String logoutLogoUp (REQUIRED if logo is used)
The filename of the image to display when the pointer is not over the logo.

String logoutLogoDown (REQUIRED if logo is used)
The filename of the image to display when the pointer is over the logo.

String logoutLogoHeight (REQUIRED if logo is used)
The height of both logo images in pixels.

String logoutLogoWidth (REQUIRED if logo is used)
The width of both logo images in pixels.

String applicationName (Default = application.name())
The phrase to display after the "Thank you for using …" and "here to re-enter …" wording on the exit page.

String selfContained (Default = false)

Set this to TRUE if you want to dynamically generate an RCMLogout as a page instead of including it in a .wod. This allows you to make unique exit pages with varying logos, for example:

```
RCMLogout nextPage // assume this exists
nextPage = pageWithName("RCMLogout");
nextPage.setSelfContained(true);
nextPage.applicationName("My Great Test Application");
```

RCMLoginAdmin

The RCMLoginAdmin component provides a complete set of features for managing users and their login-related attributes. For example, the user administration web page should include:

```
    <WEBOBJECT NAME=MyLoginAdmin></WEBOBJECT>
```

All the variables are passed in to the component from the WOD file where you call the RCMLoginAdmin.

```
MyLoginAdmin: RCMLoginAdmin {
     entity = "Contact";
     userId_name = "login_userId";
     password_name = "login_Password";
     dateSuspended_name = "login_Date_Sus";
     dateExpires_name = "login_Date_Exp";
     dateLast_name = "login_Date_Last";
     datePasswordExpires_name = "login_Date_PassExp";
     attemptsGood_name = "login_Attempts_Good";
     attemptsBad_name = "login_Attempts_Bad";
}
```

The RCMLoginAdmin page includes panels for Search, List and Edit. The Search panel allows you to find users by User ID, date attributes (Account Expiry, Password Expiry, Last Login, Suspended), or number of attempts (Good or Bad). The List panel lets you sort the results in ascending or descending order, and delete or edit each one. The Edit panel allows you add or change each attribute, with validation on each.

Attributes

The following attributes function identically to the corresponding ones in the RCMLogin. Note that not all are required.

String entity (REQUIRED
String userId_name (REQUIRED)
String password_name (OPTIONAL)
String dateSuspended_name (OPTIONAL)
String dateLast_name (OPTIONAL)
String dateExpires_name (OPTIONAL)
String datePasswordExpires_name (OPTIONAL)
String attemptsGood_name (OPTIONAL)
String attemptsBad_name (OPTIONAL)
String attemptsBad_name (OPTIONAL)

<u>Maintaining Custom Attributes</u>

The following attributes are provided to allow you to extend RCMLoginAdmin to allow the searching/listing/editing of custom attributes such as firstName, lastName, departmentCode, etc. In each case, the Java file corresponding to the page that uses the RCMLogin should define an RCMEdit or RCMList containing RCMEditCells in the usual manner, for the custom attributes only.

NSMutableArray searchPreAttributes (OPTIONAL. Default = null)

The attributes constructed will be shown <u>before</u> the basic and auditing attributes in the RCMEdit in Search mode.

NSMutableArray searchPostAttributes (OPTIONAL. Default = null)

The attributes constructed will be shown <u>after</u> the basic and auditing attributes in the RCMEdit in Search mode.

NSMutableArray listPreAttributes (OPTIONAL. Default = null)

The attributes constructed will be shown <u>before</u> the basic and auditing attributes in the RCMList.

NSMutableArray listPostAttributes (OPTIONAL. Default = null)

The attributes constructed will be shown <u>after</u> the basic and auditing attributes in the RCMList.

NSMutableArray editPreAttributes (OPTIONAL. Default = null)

The attributes constructed will be shown <u>before</u> the basic and auditing attributes in the RCMEdit in Edit mode.

NSMutableArray editPostAttributes (OPTIONAL. Default = null)

The attributes constructed will be shown <u>after</u> the basic and auditing attributes in the RCMEdit in Edit mode.

For example, to enable editing of the first name and last name (search and list follow the same concept):

In MaintainPerson.wod, after the usual attributes inside the RCMLoginAdmin declaration, add:

```
          editPreAttributes = editPreAttributes;
```

In MaintainPerson.java:

```
     protected final NSArray editPreAttributes = new NSArray(new
     RCMEditCell[] {
             (new RCMEditCell("Edit the Name of this Person")),
             (new RCMEditCell("First name:","name_First",

     "string;notNull;max=32;width=32;")),
             (new RCMEditCell("Last name:","name_Last",
                                  "string;notNull;max=32;width=32"))
     });
```

RCMJavaButton

The RCMJavaButton component displays a rollover JavaScript button which can be used for Actions, Form Submissions, or HTML redirection. To use it, simply include the following in your page's HTML:

```
<WEBOBJECT NAME=MyJavaButton></WEBOBJECT>
```

All the variables are passed in your component's WOD file where you call the RCMJavaButton. This example invokes an action "goCancel" when the button is pressed:

```
MyCancelButton: RCMJavaButton {
      buttonAction = "goCancel"; // assume your page has this method
      height = 29;
      width = 80;
      filenameUp = "cancel_normal.gif";
      filenameDown = "cancel_pressed.gif";
      alt = "Cancel and return to previous page";
}
```

Here the button is used to submit data from a form:

```
MySaveButton: RCMJavaButton {
      height = 29;
      width = 80;
      filenameUp = "save_normal.gif";
      filenameDown = "save_pressed.gif";
      alt = "Save and continue to next page";
      onClick = "document.MyForm.submit();return false;";
      // assumes MyForm exists and is uniquely named in the HTML
      // don't forget the --> return false; <-- this is essential!!
}
```

The last example shows an HTML re-direct:

```
MyURLButton: RCMJavaButton {
      href = "http://www.webappz.com";
      height = 29;
      width = 80;
      filenameUp = "webappz_logo_normal.gif";
      filenameDown = "webappz_logo_pressed.gif";
      alt = "Visit the WEBAPPZ web site now!";
}
```

Attributes

NOTE: The height and width of each image should be the same so that the JavaScript rollover does not cause a rescaling of the image.

int height (Default = 100)

The height of both images in pixels.

int width (Default = 100)

The width of both images in pixels.

String filenameUp (REQUIRED)

The filename of the image to display when the pointer is not over the button.

String filenameDown (REQUIRED)

The filename of the image to display when the pointer is over the button.

String buttonAction (OPTIONAL)

The name of the action method in your source code to invoke when the button is clicked.

String pageName (OPTIONAL)

The name of the page (a WOComponent) to go to when this button is clicked. If you use provide both a buttonAction and a pageName by mistake, only the buttonAction will be executed.

String href (OPTIONAL. Default = "#")

The URL to redirect the current browser window to when the button is clicked. The default target of "#" is necessary for forms.

String onClick (OPTIONAL)

Additional JavaScript to be executed when the button is clicked. For example, on a form, the following JavaScript code submits the form with the required return value:

```
document.MyForm.submit();return false;";
```

For a reset button, just change submit() to reset().

String alt (OPTIONAL)

The alternate text to display when the pointer hovers over this button. Will be shown in a silver DHTML layer.

boolean useImageFromGrandparent (OPTIONAL. Default = false)

The default of FALSE means that image filenames are taken from your Project's WebServer Resources. If you are using the RCMFramework from within your own framework, set this attribute to TRUE. *NOTE: currently only one framework between the app and RCMListEdit.*

String keepName (OPTIONAL. Default = false)

If TRUE, does not increment the numeric portion of the dynamically generated image file names (may allow web browser caching to improve screen rendering speed).

RCMList

The RCMList component uses an array of RCMListCell objects. This fully implemented component provides functionality related to the batched and sorted listing of objects from a database table, using a preferred format and with the appropriate security controls for adding, deleting and modifying objects.

Attributes

String entity (REQUIRED)

The name of the class that corresponds to a table in your EOModel which you want to list items from, e.g. Customer. This could be any class from your EOModel.

NSMutableArray attributes (REQUIRED)
assumes an NSMutableArray/NSArray of RCMListCell objects

An array of RCMListCell objects corresponding the attributes of the entity that you want to list. See RCMListCell documentation later on how to construct these. In the example of a Customer entity, the array would include items corresponding to attributes such as firstName, lastName, accountNumber, etc. Use NSMutableArray if you would like your cells/lists to change behaviour while being used.

boolean allowEditing (Default = false)

If TRUE, you must set the pageEditing attribute, in which case the Edit column of the list will show the "writing hand" icon with alternate text that says "Edit this {entity}". When a list item’s "writing hand" icon is clicked, an Inter Edit-List variable will be set to the selectedInstance for editing, and the page specified in the pageEditing attribute will be launched. It is recommended that your editing page have an RCMEdit component in it to receive the selectedInstance.

boolean allowDeleting (Default = false)

If TRUE, the Del column of the list will show the "trashcan" icon with alternate text that says "Delete this {entity}". When a list item’s "trashcan" icon is clicked, a JavaScript dialog box will appear saying "This {entity} will be deleted. Do you wish to continue?" If you hit the [Cancel] button, no action will be taken. If you hit the [OK] button, the selectedInstance will be deleted, and the page containing the list will be refreshed.

In case of a many-to-many relationship, only the relationship will be deleted, not the object itself.

boolean allowDisplaying (Default = false)

If TRUE, you must set the pageDisplaying attribute, in which case the Disp column of the list will show the "pointing hand" icon with alternate text that says "Display details for this {entity}". When a list item’s "pointing hand" icon is clicked, an Inter Edit-List variable will be set to the selectedInstance for displaying, and the page specified in the pageDisplaying attribute will be launched. It is recommended that your details page have an RCMEdit component in it to receive the selectedInstance, with the attribute allowEditing = 0 (false).

boolean allowSelecting (Default = false)

Used for many-to-many selections, as a more flexible alternative to a browser widget (in theory, it can also be used in one-to-many relationships, but that wouldn't be recommended from a user interface perspective). If true, it provides a check-box for each item listed, and some addtional attributes are required (selectingMasterObject and selectingDetailKey).

In the case of multiple persons belonging to multiple groups (via a Group_Person join table) you would flatten the relationships. Then you would allowSelecting in either the Group maintenance function ("toPersons" means 'persons in this group') or in the Person maintenance ("toGroups" means 'member of groups'). For example, in a Group maintenance page, where you want to assign persons to a group, the code would look like the following:

In MaintainGroupsPersons.wod:

```
PersonList: RCMList {
     entity = "Person";
     attributes = listAttributes;
     allowSelecting = YES;
     selectingMasterObject = selectedGroup;
     // assume declared in .java
     selectingDetailKey = "toPersons";
}
```

boolean allowAdding (Default = false)

If TRUE, you must set the pageEditing attribute, and the footer row of the list shows an "Add" button with alternate text that says "Create a new {entity}". When the "Add" button is clicked, a temporary editing context will be created for this new record, and the page specified in the pageEditing attribute (same as for editing) will be launched. RCM Safe Coding ensures that only successfully validated insertions are made into the real editing context for saving to the database. Several Inter Edit-List variables are set to facilitate the process of safely adding a new object.

NOTE: In your entity’s Java code, you may want to implement an awakeFromInsertion method which allows you to set default values for certain attributes.

boolean allowSorting (Default = false)

If TRUE, sorting controls are displayed next to the header text of each column in the list. The control is a drop-down list containing a "+" to sort in ascending order, "-" for descending, and a " " is unsorted. When more than one column is sorted, the sort levels are nested in priority from the farthest right column to the left. If FALSE, the list would still be sorted according to the RCMListCell’s attribute sortOrder if set, but the user cannot change the sort order.

String pageEditing (REQUIRED if allowEditing = true)

The name of your page (a WOComponent) to launch when the Edit icon is clicked. Your page should probably contain an RCMEdit component with RCMEditCell objects. If you decide not to use an RCMEdit component, you could write your own by accessing the Inter Edit-List session variables: selectedInstance, nextPage, prevPage, needsToBeAdded, tempEC, detailKey, masterObject. See allowEditing above and the tutorial for more info.

String pageDisplaying (REQUIRED if allowDisplaying = true)

The name of your page (a WOComponent) to launch when the Display icon is clicked. Although you are only displaying details, your page should probably still contain an RCMEdit component with RCMEditCell objects. If you decide not to use an RCMEdit component, you could write your own by accessing the Inter Edit-List session variables: selectedInstance and prevPage (since all you want to do it show attribute values and know which page to make the target of your return/cancel button. See the allowDisplaying explanation above for more info.

<u>Edit and List on the same Page</u>

RCMEdit editComponent (REQUIRED if an RCMEdit component for this list is on the same page)

An RCMEdit component (a pointer to one, set in the WOD file) used for editing attributes of a selectedInstance when the Edit icon for an item in the list is clicked. The RCMList and RCMEdit components need to know about each other using the "this" attribute/accessor method so that pointers can be exchanged between them. For example:

In MyCustomerAdmin.java:
```
          private WOComponent myEditComponent, myListComponent;
```

In MyCustomerAdmin.html:
```
          <WEBOBJECT NAME=CustomerList></WEBOBJECT>
          <WEBOBJECT NAME=CustomerEdit></WEBOBJECT>
```

In MyCustomerAdmin.wod:
```
          CustomerList: RCMList {
                …various attributes of RCMList of your choice, but especially in this case…
                this = myListComponent;
                // 'this' sets myListComponent to have a pointer to CustomerList
                editComponent = myEditComponent;
                // tell the CustomerList which RCMEdit will be used for editing/displaying items on the list
          }

          CustomerEdit: RCMEdit {
                …various attributes of RCMList of your choice, but especially in this case…
                this = myEditComponent;
                // 'this' sets myEditComponent to have a pointer to CustomerEdit
                listComponent = myListComponent;
                // tell the CustomerEdit which RCMList to refresh after editing an item on the list
          }
```

WOComponent this (Accessor method)

Returns a pointer to itself, like "self" in Objective-C.

Batching

If the number of list records exceeds the initial batch size, a navigation header row will appear showing (listComponent) buttons for Previous (batch), Next (batch), Page [Page #] of [Total Pages], and Display [batchSize] rows a time.

int initialBatchSize (Default = 50)

The number of list records to display on one page. The user may change this later in the batch navigation bar.

int maxBatchSize (Default = 1000)

The maximum allowable number of list records to display on one page (since the user can adjust the current batch size in the navigation header).

Defining List Data

EOQualifier listQualifier (Default = null;)

Use this if the list records are to be limited using criteria other than the results from a search panel (an RCMEdit in search mode) such as data security rules, etc. For example, in a Function maintenance component where you only want to show "active" records (in MaintainFunctions.java), you could use the following:

```
EOQualifier limitation =
      EOQualifier.qualifierWithQualifierFormat("active = 1", null);
```

In your RCMList in the MaintainFunctions.wod, add the following additional line:

```
listQualifier = limitation;
```

NOTE: If you use a Search panel (RCMEdit in search mode) to populate this list, the selectQualifier (session Inter Edit-List variable) AND the listQualifier must both be satisfied for records to be displayed in the list.

EOEnterpriseObject masterObject (REQUIRED if detailKey is used)

An object which is the master entity in the master/detail relationship that is the basis for showing detail entities in the list (e.g. myDepartment).

String detailKey (REQUIRED if masterObject used)

The name of the relationship to the detail entity which will be displayed in the list (e.g. "toEmployee_s" where the master entity is myDepartment) based on a master/detail relationship.

EOEnterpriseObject selectingMasterObject (REQUIRED if allowSelecting = true)

An object which is the master entity in a many-to-many master/detail relationship from which we spawn the flattened relationship (selectingDetailKey) which is the basis for showing detail entities in the list. From the Group/Person example, this object would be the Group instance (selectedGroup) that you are currently assigning persons to.

String selectingDetailKey (REQUIRED if allowSelecting = true)

The name of the flattened many-to-many relationship to the detail entity which will be displayed in the list (e.g. "toPersons" refers to list of "Person" entities which are connected to the master entity of currentGroup).

String addingAction (OPTIONAL)

The method in your Java class to invoke when the 'Add' button is pressed. There you can program new object defaults or other custom events. NOTE: the framework still inserts the object for you in a newly created Temporary Editing Context first. You can access the object that the framework has created for you by casting the session.selectedInstance(). You may want to use session.getTempECCopy() to create a copy of an object that you want to establish a relationship with (since both objects must be in the same temporary editing context to do so).

For example, if you have a list of Licenses to which you can add new ones, then in the WOD where you call your RCMList, you could add a line like `addingAction = "addLicense";` and your Java could look like:

```
// this method is called when a new license will be added
public void addLicense(); {       // in the temporary editing
context

     License license = (License) session.selectedInstance();
     license.setDate(todaysDate);

     Business business .... // assume you have this
     Business businessCopy = (Business)
session.getTempECCopy(business);
     license.setToBusiness(businessCopy); // hook up the
relationship
```

```
        // more start-up default setting of license attributes here
    }
```

The awakeFromInsertion method in the object's class is still called before this point (the normal WebObjects way to set defaults) but you may find it convenient to use this technique to implement custom logic.

String deletionAction (OPTIONAL)

Similarly to addingAction above, you may write a function in your WOComponent to refuse deletion under certain circumstances. This is the delegated method in your Java class to invoke when the 'Delete' button is pressed. As well, the object that is about to be deleted can be accessed using session.selectedInstance() but you have to cast this object to the appropriate class before you try to use it. There you can perform custom validations or related events, such as:

return true if there is a reason that the deletion should be disallowed
(i.e. based on your test for a relationship integrity violation; consider using session.addMessage("Unable to delete ...") to inform the user of the reason)
return false to allow deletion (i.e. after successfully logging the deleted object's content to an audit table)

boolean showSelectCriteria (OPTIONAL; Default = false)

If true, an additional heading row will contain the selectCriteria in a readable format to inform the user about what records are being displayed in the RCMList. These selectCriteria are usually set by an RCMEdit in Select mode. This automatically generated string will be based on the title attributes of RCMEditCells and the query operators and values.

For example:

Results of search for ***Server=***"FINDB05B_SYB" and **Reservation Date**>8/9/2001

Layout

The following formatting attributes default to the values that have already been set as session attributes, but they can be overridden here.

String listTableFormat (Default = session.listTableFormat)

String listOutsideTableFormat (Default = session.listOutsideTableFormat)

String listHeadingFont (Default = session.listHeadingFont)

String listHeadingColor (Default = session.listHeadingColor)

String listRowFont (Default = session.listRowFont)

String listRowColor (Default = session.listRowColor)

String listSelectedRowColor (Default = session.listSelectedRowColor)

String listSelectedRowColor (Default = session.listSelectedRowColor)

RCMListCell (a class, not a component)

The RCMListCell class is used to the define the columns displayed in an RCMList. Unlike a component, which is instantiated via the WOD file, you must instantiate these objects yourself in the Java code for your listing page. You must construct an array of RCMListCell objects and pass that array to the RCMList.

For example, in MyListPage.java:

```
// The init method of your page:
public class MyListPage extends WOComponent {
        protected final NSArray myAttributes = new NSArray(
                new RCMListCell[] {
                        (new RCMListCell("First Name","firstName","string;width=100")),
                        (new RCMListCell("Last Name","lastName","string;width=150;sortOrder=+")),
                        (new RCMListCell("Status","statusCode","boolean")),
                        (new RCMListCell("Create Date","z_createDate","date;dateFormat=%Y-%m-%d"))
                }
        );
}
```

This example creates four columns, from left to right: First Name as a string with a column width of 100 pixels, Last Name as a string with a column width of 150 pixels, Status as a Yes/No boolean, and Create Date as a date with a special "dash separated" date format. The rows will be sorted in ascending order by Last Name.

Constructors

RCMListCell (String title, String attributeName, String type)

A convenience constructor method to instantiate an RCMListCell object with its required attributes: column title, entity’s attribute name (corresponding to a column in the database), and parsed string of various additional attributes (see the "type" documented below).

RCMListCell ()

If you use this to construct each object with no parameters, you have to set each of the attributes manually later.

Attributes

Cell Definition Attributes

String title (OPTIONAL, but the Default = "undefined")

The label for a column heading, which will be shown in bold, using the listHeadingFont.

String attributeName (REQUIRED. Default = "")

The entity's attribute name from the EOModel corresponding to the database column containing the data you want to display in a cell. Multiple relationships may be traversed by separating them using a dot (e.g. toContact.address.cityName).

String type (REQUIRED. Default = "empty")

A semi-colon separated list of the specific attributes described below, except you don't include the "displayAs" portion of the attribute's name. The list is parsed to set the attributes for you, but if you don't use this, you can set each attribute manually by calling the set method corresponding to each attribute. See above example.

Display Attributes

Only one of the following attributes can TRUE at a time, while all the others remain FALSE by default.

boolean displayAsString ("string" in the type attribute)

Simple text string

boolean displayAsHyperLink ("hyperLink" in the type attribute)

A simple text string wrapped in an <A HREF="string" target=new>string</A> tag.

boolean displayAsBoolean ("boolean" in the type attribute)

If the data is Yes, the cell background color is set to green. If the data is No, the background is made red.

boolean displayAsNumber ("number" in the type attribute)

Display a number as an integer, with negatives in brackets.

boolean displayAsCurrency ("currency" in the type attribute)

Show monetary amounts using currencyFormat (see below).

boolean displayAsDecimal ("decimal" in the type attribute)

Display an amount as a decimal using numberFormat (see below).

boolean displayAsDate ("date" in the type attribute)

Show a date using dateFormat (see below).

boolean displayAsContent ("content" in the type attribute)

Based on the mime type (e.g. application/pdf), display the appropriate icon, wrapped in an <A HREF="{dynamic URL to data object}" target=new> tag. *NOT YET IMPLEMENTED.*

boolean displayAsImage ("image" in the type attribute)

Display an image in a cell using an <IMG SRC> or <EMBED> tag for a dynamically generated URL to the data object. *NOT YET IMPLEMENTED.*

boolean displayAsEmpty ("empty" in the type attribute)

Don’t show anything, for example in the case of undisplayable erroneous data.

<u>Miscellaneous Attributes</u>

Also separated by semi-colons, the following attributes are used in the same way as the Display Attributes described above.

String width (Default = null)

The width of a column, e.g. "30%" or "300" (pixels). Null means not applicable.

String mimeType (Default = null)

A mime type, e.g. "application/pdf" (only applies to displayAsContent or displayAsImage above). Null means not applicable.

String numberFormat (Default = session.numberFormat();)

The format of numbers displayed as decimals. The session default (e.g. ###.### or whatever you have changed it to) will be used unless specified here.

String currencyFormat (Default = session.currencyFormat();)

The format of monetary amounts. The session default (e.g. #,###,## without a $ or whatever you have changed it to) will be used unless specified here.

String dateFormat (Default = session.dateFormat();)

The format of a date. The session default (e.g. %d/%m/%Y or whatever you have changed it to) will be used unless specified here.

String currencyCode (Default = session.currencyCode();)

The currency label to put in front of monetary amounts (e.g. CAD, US$, or Hfl.). The session default of "US$" will be used unless specified here.

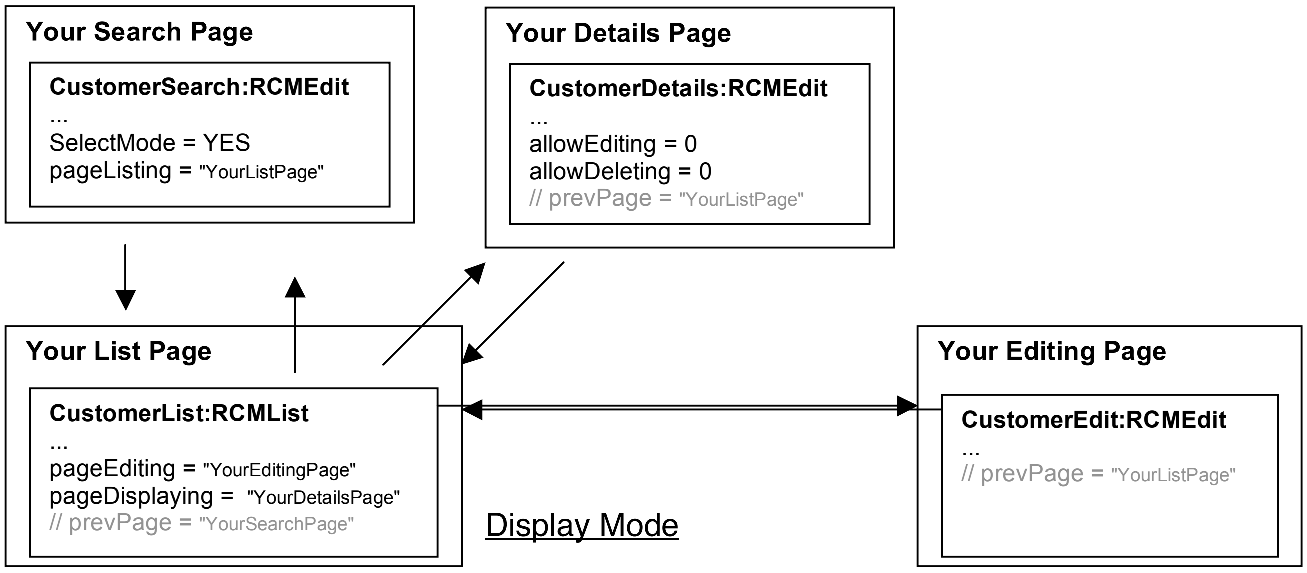
String sortOrder (Default = " ")

The sorting order "+" to sort in ascending order, "-" for descending, and a " " (the default) is unsorted. See above for explanations of the evaluation of the sort order in the RCMList.

RCMEdit

The RCMEdit component contains an array of RCMEditCell objects, and your own optional editing fields wrapped within the component, plus buttons at the bottom for Save, Reset, Cancel, Delete. RCMEdit provides complete editing functionality within each RCMEditCell (explained below), including section titles, field labels, field descriptions, required field indicators, and data entry validations with field-specific error messages. An RCMEdit can be used to make an editing, displaying or searching panel on a page.

This figure explains the three Modes for RCMEdit and how it could fit with your list page (RCMEdit and RCMList should always be components on your pages):



Note: you don't need to actually set "prevPage" as the framework does this for you through the Inter Edit-List variables.

Attributes

NSMutableArray attributes (REQUIRED) // assumes an NSMutableArray of RCMEditCell objects

An array of RCMEditCell objects corresponding the attributes of anInstance that you want to edit. See RCMEditCell documentation later on how to construct these. In the example of a selected Customer, the array would include items corresponding to attributes such as firstName, lastName, accountNumber, etc.

boolean allowEditing (Default = true)

If TRUE (Editing Mode), the Save and Reset buttons will be shown with alternate text that says "Save Changes to the database" and "Reset Changes on this form" respectively. In Editing Mode, each RCMEditCell whose allowEditing attribute is set to TRUE (the default) will be modifiable, and each one that is set to false will read-only. If FALSE (Display/Details Mode), all cells will be made non-modifiable even if the allowEditing attribute on each RCMEditCell had been set to true.

boolean allowDeleting (Default = true)

If TRUE, the Delete button will be shown with alternate text that says "Delete this {entity} from the database". When Delete button is pressed, a JavaScript dialog box will appear saying "This {entity} will be deleted. Do you wish to continue?" If you hit the [Cancel] button, no action will be taken. If you hit the [OK] button, anInstance will be deleted, and the user will be taken to the page specified in the prevPage attribute. In case of a many-to-many relationship, only the relationship will be deleted, not the object itself.

String actionSaving (OPTIONAL. Default = null)

The name of your method in the calling WOComponent's class to be invoked after successfully saving the RCMEdit. Your method must return a WOComponent representing a page to be displayed. If you return null, the same page will be refreshed. You may want to use this if you want to perform a specific action before returning a specific page.

String pageSaving (OPTIONAL. Default = null)

The name of a new page (a WOComponent) to go to after successfully saving the RCMEdit. You may want to use this on a page containing an RCMEdit to return to a page containing the related RCMList.

boolean goPrevPageAfterSaving (OPTIONAL. Default = false)

If true, pressing the Save button results in the previous page being displayed after the RCMEdit has been successfully saved. NOTE: in the unusual case (strongly discouraged) where you use more than one of the above two attributes, actionSaving is executed before pageSaving, and pageSaving is executed before goPrevPageAfterSaving.

Select Mode

boolean selectMode (Default = false)

If TRUE, the component is transformed into a Search panel, containing buttons for Search, Reset, Clear and Cancel.

When the Search button is clicked:

an Inter Edit-List variable in RCMSession called selectQualifier will be set (using AND criteria narrowing the search using all of the entered data values), and

the page specified in the *pageListing* attribute will be launched.

The Reset button causes the editing fields to be restored to the values that were on the page when it was most recently displayed (e.g. previous search criteria after a Re-Select). The Clear button restores the editing fields to their default empty values. The Cancel button returns the user to the page specified in the *prevPage* attribute.

String pageListing (REQUIRED if selectMode = true)

The name of your page (a WOComponent) to launch when the Search button is clicked. Your page should probably contain an RCMList component with RCMListCell objects. If you decide not to use an RCMList component, you could write your own by accessing the Inter Edit-List session variables: selectQualifier and prevPage. See selectMode above and the tutorial for more info.

RCMList listComponent (REQUIRED if an RCMList with the results of this Search is on the same page)

An RCMList component (a pointer to one, set in the WOD file) used for displaying records resulting from a search, on the same page as the search.The RCMList and RCMEdit components need to know about each other using the "this" attribute/accessor method so that pointers can be exchanged between them. For example:

In MyCustomerAdmin.java:

```
    private WOComponent myListComponent;
```

In MyCustomerAdmin.html:

```
    <WEBOBJECT NAME=CustomerSearch></WEBOBJECT>
    <WEBOBJECT NAME=CustomerList></WEBOBJECT>
```

In MyCustomerAdmin.wod:

```
    CustomerSearch: RCMEdit {
    …various attributes of RCMList of your choice, but especially in this case…
          selectMode = YES;
          listComponent = myListComponent;
          // tell the CustomerSearch which RCMList to refresh using the results of the search
    }

    CustomerList: RCMList {
          …various attributes of RCMList of your choice, but especially in this case…
          this = myListComponent;
          // 'this' sets myListComponent to have a pointer to CustomerList
    }
```

Note: if this page is in Edit Mode with an RCMList, your CustomerList should be set up as explained in the RCMList example so that the list can be automatically updated after being edited (not just after the result of a search).

WOComponent this (Accessor method)

Returns a pointer to itself, like "self" in Objective-C.

Validation and Auditing

String validationAction (OPTIONAL; DELEGATION. Default = null)

Provide the name of your custom validation method on the parent component/page (your page) which will be executed. Your method should return a boolean representing whether there was an error (TRUE) or not (FALSE). You must also use setErr to an error string explaining why your function did not consider the cell to contain valid data.

For example:

In MyCustomerEdit.java:

```
public boolean validateCustomer() {
     boolean error = false;
          RCMEditCell postalCell = (RCMEditCell)
               editAttr.objectAtIndex(6);
     // get a pointer to the 6th editable cell for postal code, which you want to test
     if (postalCell.stringValue().indexOf("V")==-1) {
          postalCell.setErr("Postal code must contain a V.");
          error = true;
     }

if ( ... )
  // more validations go here, assuming you instantiated RCMEditCell objects for them too.

     return error;
}
```

The following auditing attributes default to the values that have already been set as application attributes, but they can be overridden here if the entity that your RCMEdit in Edit mode is working with has its auditing attributes named differently than other entities in your EOModel. For example, a Journal entity may have an authoredBy attribute where you want to store the session.currentUserId. See the RCMApplication section earlier for more information.

String createUserId_name (OPTIONAL. Default = application.createUserId_name();)

String createDate_name (OPTIONAL. Default = application.createDate_name();)

String modifyUserId_name (OPTIONAL. Default = application.modifyUserId_name();)

String modifyDate_name (OPTIONAL. Default = application.modifyDate_name();)

EOEditingContext defaultEditingContext
(OPTIONAL. Default = session.defaultEditingContext();)

You may want to provide a custom editing context to use for saving to the database after all validations have been passed. In rare cases you may want to have a temporary editing context hold the objects that the RCMEdit is working on while you have the user edit a related object simultaneously (NOTE! This is advanced functionality for experienced users only).

Layout

The following formatting attributes default to the values that have already been set as session attributes, but they can be overridden here for the layout of an RCMEdit in Edit mode.

String editOutsideTableFormat (Default = session.editOutsideTableFormat)

String editTableFormat (Default = session.editTableFormat)

String editHeadingFont (Default = session.editHeadingFont)

String editHeadingColor (Default = session.editHeadingColor)

String editRowFont (Default = session.editRowFont)

String editRowColor (Default = session.editRowColor)

String editAltRowColor (Default = session.editAltRowColor)

The following formatting attributes default to the values that have already been set as session attributes, but they can be overridden here for the layout of an RCMEdit in Search mode (selectMode = true).

String selectOutsideTableFormat (Default = session.selectOutsideTableFormat)

String selectTableFormat (Default = session.selectTableFormat)

String selectHeadingFont (Default = session.selectHeadingFont)

String selectHeadingColor (Default = session.selectHeadingColor)

String selectRowFont (Default = session.selectRowFont)

String selectRowColor (Default = session.selectRowColor)

String selectAltRowColor (Default = session.selectAltRowColor)

<u>Inter Edit-List Variables</u>

A number of attributes control how information is passed between the list components and the related edit components. After these variables have been accessed, they will be cleaned up automatically. The following <u>should not be used directly</u> by your application, therefore they are not documented in detail here (and each one defaults from the corresponding session attribute): nextPage, prevPage, anInstance, needsToBeAdded, tempEC, selectQualifier, detailKey, and masterObject.

If you want to make your own editing cells within an RCMEdit (rather than, or in addition to, using RCMEditCell objects), you will need *anInstance* (set to session.selectedInstance by an RCMList) in order to identify the object that you are editing. For example, in the WOD declaration of an RCMEdit, you could have anInstance = myCustomer. Your custom editing cells would then be able to edit the attributes of myCustomer.

The variable nextPage is the target page to be shown to the user after successfully saving the edited record to the database.

RCMEditCell (a class, not a component)

The RCMEditCell class is used to define the rows in an RCMEdit component containing a title, a field (either editable or non-editable), and an optional description. Unlike a component, which is instantiated via the WOD file, you must instantiate these objects yourself in the Java code for your editing/displaying/searching page. You must construct an array of RCMEditCell objects and pass that array to the RCMEdit component.

For example, MyEditPage.java incorporates many different attributes that can be used in an RCMEditCell:

```
// The init method of your page:
public class MyEditPage extends WOComponent {
  protected final NSArray myAttributes = new NSArray(
    new RCMEditCell[] {
        (new RCMEditCell("Personal Information"),
        (new RCMEditCell("First Name","firstName","string;min=2;max=20","Your first name")),
        (new RCMEditCell("Last Name","lastName","string;min=2;max=35","Your last name")),
        (new RCMEditCell("E-mail","email","string;notNull;eMail","Address to send e-mail to")),
        (new RCMEditCell("Home page","homePage","hyperlink;width=40;max=128","Go to URL")),
        (new RCMEditCell("Birthdate","birthDate","date;max=1982/12/31;min=1900/1/1;notNull",
                         "Your birthday")),
        (new RCMEditCell("System Information"),
        (new RCMEditCell("Status","statusCode","boolean;notNull","Active or Inactive")),
        (new RCMEditCell("Defaut payment","payment","currency;min=0;max=500;notNegative",
                         "Preferred amount to pay each month")),
        (new RCMEditCell("Create Date","z_createDate","date;dateFormat=%Y-%m-%d;noEditing"))
    }
  );
}
```

This example creates 10 rows, from top to bottom:

A title row that says "Personal Information"
First Name, a string cell where at least 2 but no more than 20 characters are required
Last Name, a string cell where at least 2 but no more than 35 characters are required
E-mail, a string cell that must contain a string that looks like an e-mail address
Home page, a hyperlink cell that is 40 characters wide but may contain up to 128 characters
Birthdate, a date cell where the entered date must be later than January 1, 1900 but earlier than Dec. 31, 1982.
A title row that says "System Information"
Status, a Yes/No boolean radio button where you must pick one or the other
Default payment, a cell requiring a positive currency amount between 0 and 500
Create Date, a display-only field showing the date that this record was created, in a dash separated format (e.g. 2000-11-29).

Constructors

RCMEditCell (String title, String attributeName, String type, String description)

A convenience constructor method to quickly instantiate an entire RCMEditCell object with its attributes: row label, entity's attribute name (corresponding to a column in the database), a parsed string of various additional attributes (see the "type" documented below), and a row description.

RCMEditCell (String title, String attributeName, String type)

Same as above, but with the row description left blank.

RCMEditCell (String title)

To make a title editing cell using editHeadingFont and editHeadingColor to separate logical groupings of RCMEditCell rows. The *displayAsHeading* attribute will be set by this automatically.

RCMEditCell ()

If you use this to construct each object with no parameters, you have to set each of the attributes manually later.

Attributes

Cell Definition Attributes

String title (OPTIONAL, but the Default = "undefined")

The label for the first cell in the row, as a prompt for the attribute name that is being edited. For example "First name" would be a helpful prompt for an attributeName of "firstName".

String attributeName (REQUIRED except if used only as a title. Default = "")

The entity's attribute name from the EOModel corresponding to the database column containing the data you want to edit in a cell.

String type (REQUIRED except if used only as a title. Default = "empty")

A semi-colon separated list of the specific attributes described below, except you don't include the "displayAs" portion of the attribute's name. The list is parsed to set the attributes for you, but if you don't use this, you can set each attribute manually by calling the set method corresponding to each attribute. See above example.

String description (OPTIONAL except disallowed when used as a title. Default = "")

Additional explanatory text displayed to the right of the editing control (field, radio button, etc.) to help the user while editing this attribute.

Display Attributes

Only one of the following attributes can be true at a time, while all the others remain false by default.

boolean displayAsHeading ("heading" in the type attribute)

Same in Edit, Display and Search modes. Used for displaying a title bar using the editHeadingColor and editHeadingFont. Includes optional minimize/maximize functionality to show or collapse the attributes that are listed after this heading and before the next heading (or the end of the RCMEdit's list of editing attributes).

It is recommended to use the one-argument constructor for RCMEditCell where you want the default behaviour; i.e. where all of the following RCMEditCells are shown and the minimize button is available. If you want to disable minimizing, you can set `noMinimizing = true`. If you wish it to appear in a minimized state (e.g. for unimportant optional attributes) you can set `initiallyMinimized = true` using the 3-argument constructor, such as:

```
      (new RCMEditCell("Search for Persons","","heading;initiallyMinimized"))
```

In this case, the second argument (the attribute name) is not needed, since this is just a heading.

boolean displayAsString ("string" in the type attribute)

Edit mode: edit a simple text string in an <INPUT> box. The *width* attribute here is the size of the input box in characters. The *min* attribute is the minimum number of required characters in the validation. The *max* attribute is the maximum number of characters allowed in the validation (this should match your database structure).

Display mode: show a simple text string.
Search mode: a case-insensitive "like" search (a substring anwhere in the attribute) will be done on text entered.

boolean displayAsStringBox ("stringBox" in the type attribute)

Edit mode: edit a simple text string in a <TEXTAREA> box. The *width* attribute here is the size of the textarea box in characters. The *min* attribute is the minimum number of required characters in the validation. The *max* attribute is the maximum number of characters allowed in the validation (this should match your database structure). The *rows* attribute is the number of vertical rows for the textarea to display before scrolling is necessary (default of 0 means not used, see later explanation).
Display mode: show a simple text string in the cell (same as for "string").
Search mode: a case-insensitive "like" search will be done on text entered.

boolean displayAsHyperLink ("hyperLink" in the type attribute)

Editing mode: display a URL in an input box with a "GO" link next to it, which launches the entered URL in a new window. The attributes min, max and width are the same as in "string" above. Validations will be done similarly to the "URL" type (at least one "." and "//")
Display mode: A simple text string wrapped in an <A HREF="string" target=new>string</A> tag.
Search mode: a case-insensitive "like" search will be done on text entered, and a "GO" link will be put next to it.

boolean displayAsPassword ("password" in the type attribute)

Edit mode: edit a simple text string in an <INPUT type=password> box, which masks entry with asterixes for each character typed. The *width* attribute here is the size of the input box in characters. The *min* attribute is the minimum number of required characters in the validation. The *max* attribute is the maximum number of characters allowed in the validation (this should match your database structure). When saved, a newly entered password will be encrypted using the application-wide encryption method. When it is loaded from an existing record, the encrypted data will also be shown masked as asterixes. Unchanged passwords are not re-encrypted while saving.
Display mode: show an encrypted text string in the cell.
Search mode: a case-insensitive "like" search will be done on the <u>decrypted</u> text entered (generally not useful now, since the decryption method is not yet implemented).

boolean displayAsBoolean ("boolean" in the type attribute)

Edit mode: a radio button pair with the cell background color set to green for Yes, and red for No.

Display mode: If the data is Yes, the cell background color is set to green. If the data is No, the background is made red.
Search mode: same as Edit Mode; searches for TRUE/YES or FALSE/NO. If neither radio button is selected, this attribute is not included in the search criteria.

boolean displayAsNumber ("number" in the type attribute)

Edit mode: an input box. The optional *min* attribute is the minimum allowed number value. The optional *max* attribute is the maximum number allowed value. The optional *notNegative* attribute can validate that the entered number is 0 or positive. The optional *width* attribute is the width of the input box in characters.
Display mode: show a number as an integer, with negatives in brackets.
Search mode: a drop-down list of relative operators (see relOperator below) will be shown before the input box, and a "REPEAT..." link will be shown after the input box. The repeat feature refreshes the RCMEdit component with an additional similar RCMEditCell below the current one – this provides additional selection criteria on a single attribute (e.g. to search for numbers greater than 5 and less than 10).

boolean displayAsCurrency ("currency" in the type attribute)

Similar to displayAsNumber above, except that the *currencyCode* will be displayed in front of the number (e.g. USD).
Edit mode: an input box. The optional *min* attribute is the minimum allowed number value. The optional *max* attribute is the maximum number allowed value. The optional *notNegative* attribute can validate that the entered number is 0 or positive. The optional *width* attribute is the width of the input box in characters.
Display mode: Show monetary amounts using currencyFormat (see below).
Search mode: same functionality as for displayAsNumber above, except for currency amounts.

boolean displayAsDecimal ("decimal" in the type attribute)

Same as displayAsCurrency above, except displays an amount as a decimal using numberFormat (see below).

boolean displayAsDate ("date" in the type attribute)

Edit mode: an input box. The optional *min* attribute is the minimum (oldest) allowed date according to the layout Year/Month/Day/Hour/Minute/Second. The optional *max* attribute is the maximum (youngest) allowed date according to the same layout as for *min*. The optional *width* attribute is the width of the input box in characters.
Display mode: Show a date using dateFormat (see below).
Search mode: a drop-down list of relative operators (see relOperator below) will be shown before the input box, and a "REPEAT..." link will be shown after the input box.

The repeat feature refreshes the RCMEdit component with an additional similar RCMEditCell below the current one – this provides additional selection criteria on a single attribute (e.g. to search for records with dates greater than 28/2/2000 and less than 30/9/2000).

boolean displayAsPopUp ("popup" in the type attribute)

Edit mode: a pop-up selector gadget. See below for a full description of the related variables: entity (required), displayAttribute (required), listQualifier (optional), listQualifierValue (optional), detailKey (optional), masterObject (optional), masterObjectValue (optional).
Display mode: a string representation of the pop-up's selected item
Search mode: a pop-up will be shown the same way as in Edit mode. The RCMEdit component, in this case, provides the selected item for the search criteria.

boolean displayAsContent ("content" in the type attribute)
NOT YET IMPLEMENTED.

boolean displayAsImage ("image" in the type attribute)
NOT YET IMPLEMENTED.

boolean displayAsEmpty ("empty" in the type attribute)
Don’t show anything, for example in the case of undisplayable erroneous data.

Edit/Display Attributes

Also separated by semi-colons, the following attributes are used in the same way as the Display Attributes described above.

boolean allowEditing (Default = true)
Do not set this directly (it is set to FALSE by noEditing – see below). RCMEdit's *allowEditing = false* (Display Mode) overrules this cell-based value.

boolean noEditing (Default = false. In the type attribute, just use "noEditing")
Parsed in the *type* attribute, makes the cell non-modifiable in Edit and Search mode (not necessary in Display mode).

Validation Attributes

In each of the following, a failed validation will invoke *setErr* with an explanation of the reason why validation was not passed, and *validationError* variable will be set to true which prevents the record from being saved.

int min (Default = 0)

In the case of a string, stringBox, hyperLink and password, this is the minimum allowable length in characters (causes a "star" icon to appear in front of the cell label, indicating that entry is required). In the case of a number, currency or decimal, this is the smallest acceptable number. In the case of a date, this is the oldest date allowed within the application-wide date limits. 0 means not applicable.

int max (Default = 0)

Same as above, except it is the maximum/largest/youngest value. 0 means not applicable.

boolean notNull (Default = false. In type, use "notNull")

Causes a "star" icon to appear in front of the label, indicating that user entry is required. FALSE means not applicable.

boolean notNegative (Default = false. In type, use "notNegative")

If you enter a value in the cell, it may not be a negative number. FALSE means not applicable.

boolean unique (Default = false. Only for data types string, stringBox, hyperLink, and password — use "unique")

For validation purposes the value provided for this attribute must be unique among all records in the table. FALSE means not that this validation is not performed. For example, you probably will want to use this feature to avoid a WebObjects error in a session where the user tries to add a conflicting value in a character-based field which is the primary key.

boolean eMail (Default = false. In type, use "email")

If you enter some text, it must look like an e-mail address (contains an "@" and at least one ".").FALSE means not applicable.

boolean URL (Default = false. In type, use "URL")

If you enter a value in the cell, it must look like a Universal Resource Locator (contains "//" and at least one "."). FALSE means not applicable.

<u>Miscellaneous Attributes</u>

int width (Default = 20)

In the case of a string, stringBox, hyperLink, password, number, currency, decimal and date, this is the width of the <INPUT> field in characters (‘cols’ in the case of stringBox).

int rows (Default = 0)

In the case of a stringBox this is the numbers of ‘rows’ in the <INPUT type=textarea> field in characters. 0 means not applicable.

String mimeType (Default = null)
NOT YET IMPLEMENTED.

String numberFormat (Default = session.numberFormat();)

The format of numbers displayed as decimals. The session default (e.g. ###.###) will be used unless specified here.

String currencyFormat (Default = session.currencyFormat();)

The format of monetary amounts. The session default (e.g. #,###,## without a $) will be used unless specified here.

String dateFormat (Default = session.dateFormat();)

The format of a date. The session default (e.g. %d/%m/%Y) will be used unless specified here.

String currencyCode (Default = session.currencyCode();)

The currency label to put in front of monetary amounts (e.g. CAD, US$, or Hfl.). The session default of "US$" will be used unless specified here.

String relOperator (Default = null)

The relative operator used for quantitative searches. For number, currency, decimal and date:

>	Greater than (later than, in the case of a date)
>=	Greater than or equal to
=	Equal to
<	Less than (earlier than, in the case of a date)
<=	Less than or equal to

For example, you could make two similar Date Search cells "Min Date" with ">=" and "Max Date" with "<="

Pop-up Attributes

Also separated by semi-colons, the following attributes are used in the *popup* type.

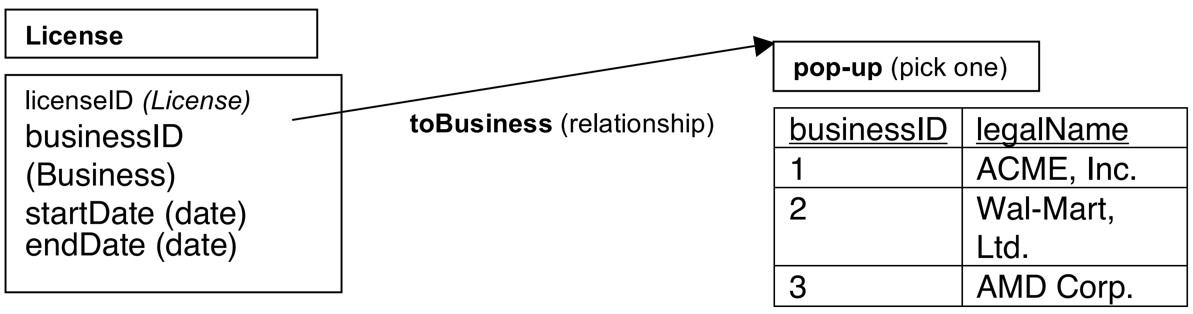
For example, assume that we have a License maintenance function where we want to pick one business from a pop-up list of businesses.

```
(new RCMEditCell("Business:","toBusiness",
   "popup;entity=Business;displayAttribute=legalName;",
   "Pick a business for this license"))
```

The Java code is essentially the same for both the Search and Edit modes. The first argument is a field prompt, and the 4th argument is a helpful explanation. The second argument of the RCMEditCell "toBusiness" is the relationship which connects the attribute being populated to the class of the entity that we are picking from. The 3rd argument (type) includes:

popup, which simply specifies that this is a pop-up type of editing cell
entity, in this case the Business class is where we iterate/scroll through
displayAttribute, is used to show the full legalName attribute for each Business in the pop-up.

The following diagram shows how a pop-up is used by an RCMEditCell. NOTE: you never need to refer to the businessID attribute (the primary key for the Business object) anywhere in your code, since the toBusiness relationship is used to take the businessID of the item you pick and put it into the businessID of the License object!



businessID	legalName
1	ACME, Inc.
2	Wal-Mart, Ltd.
3	AMD Corp.

String entity (REQUIRED for displayAsPopUp. Default = null)

The class name, same as in RCMList and RCMEdit. This is the object class that will be used by the pop-up.

String displayAttribute (REQUIRED for displayAsPopUp. Default = null)

The attribute of the above entity to display in the pop-up gadget for each item. If the attribute you want to show is not a String, you should write an accessor method to return a string representation suitable for use in a pop-up.

EOQualifier listQualifier (OPTIONAL for displayAsPopUp. Default = null)

This attribute needs to be set in the cell directly in order to limit the items that are shown in the pop-up. For example, `businessPopUpCell.setListQualifier(limitation);`

String listQualifierValue (OPTIONAL for displayAsPopUp. Default = null)

Same purpose as above, except that you pass a String containing the name of an accessor method from your code which will return a qualifier. For example inside the type attribute in the RCMEditCell, "...;`listQualifierValue=limitation`;..." refers to a method or variable "`EOQualifier limitation = ...`" in your Java code.

String detailKey (OPTIONAL for displayAsPopUp. Default = null)

A String (e.g. a relationship name) to be used in combination with masterObject or masterObjectValue to provide another way of qualifying the items shown in a pop-up. For example, `toLicenses` would support only showing licenses for the current object (e.g. currentBusiness) being edited.

EOEnterpriseObject masterObject (OPTIONAL for displayAsPopUp. Default = null)

This attribute needs to be set in the cell directly in order to limit the items that are shown in the pop-up, in combination with the detailKey set above. For example, `businessPopUpCell.setMasterObject(currentBusiness);`

String masterObjectValue (OPTIONAL for displayAsPopUp. Default = null)

Same purpose as above, except that you pass a String containing the name of an accessor method from your code which will return a masterObject. For example inside the type attribute in the RCMEditCell, `masterObjectValue=currentBusiness;`

Use this if you would like the RCMEdit's selectedInstance() to be the masterObject (for convenience to have the framework substitute this for the name of the instance being edited). For example inside the type attribute in the RCMEditCell, `masterObjectValue=this;`

Heading Window Control Attributes

The following attributes, also separated by semi-colons, provide the minimize/maximize functionality only in the *heading* type.

boolean initiallyMinimized (OPTIONAL for heading type. Default = false)

If true, when the heading is first shown on a page it will collapse (hide) the attributes that are listed after this heading and before the next heading (or the end of the RCMEdit's list of editing attributes). The maximize button will be shown in the heading so that the user can click on it to expand (show) the attributes that had been previously suppressed from view. This may be helpful for search panels where you don't want to waste vertical screen real estate unless the user wants an "advanced" search.

boolean noMinimizing (OPTIONAL for heading type. Default = false)

If TRUE, the heading does not include a minimize/maximize button, and the attributes that are listed after this heading and before the next heading are always shown.

Detail Edit Control Attributes

Also separated by semi-colons, the following attributes can be used in any type where you want to have another select/list/edit page related to a particular RCMEditCell. The "editing hand" icon (detailEdit) will be shown only after the instance being edited has been successfully saved for the first time (so that a valid master object exists in the database before working with the detail object/relationship).

It is recommended that you use this for many-to-many list selections rather than a WOBrowser, e.g. for selecting multiple users in a Group maintenance function (using RCMList in Select mode). Another example would be to allow users to actually enter a Contact associated with a Business, even though Contacts are stored in their own table. You are responsible to ensure that the Contact record is created (either in your actionEditing method described below, the addingAction in RCMList, or in awakeFromInsertion() in the object's Java class) before you continue to the next page.

When the detailEdit button is pressed, the session.selectedInstance() will be set to the object that the user is currently editing in the RCMEdit. Your detail editing page should use session.selectedInstance() to identify the masterObject from the referring page. Firstly, actionEditing will be attempted (if found) and whichever page that method returns will be shown. Then, pageEditing will be shown. Generally you should use either actionEditing or pageEditing, not both.

Example 1:

In MaintainGroups.java:

```
(new RCMEditCell("Group Members:","personList",
  "string;detailEdit;pageEditing=MaintainGroupsPersons",
            "People who are in this Group"))
```

In the Person object we have a personList method which concatenates meaningful names from multiple selected Persons so that it can be displayed in the cell controlled by this detailEdit. You can leave this argument blank if you don't want to display anything as a cell content (e.g. there isn't a meaningful string representation for a group of sounds). This string cell now becomes non-editable because you will use the MaintainGroupsPersons page select the Persons for this group.

Afterwards, in MaintainGroupsPersons.java, you would get the Group being edited using:

```
public Group selectedGroup = (Group) session.selectedInstance();
```

In MaintainGroupsPersons.wod in your RCMList:

```
allowSelecting = YES;
selectingMasterObject = selectedGroup;
selectingDetailKey = "toPersons";
```

The MaintainGroupsPersons page uses the toPersons relationship identified in the selectingDetailKey to automatically insert/delete records in the flattened Person_Group join table to connect Persons to the current Group.

Example 2:

```
(new RCMEditCell("Contact:","toContact.lastName",
  "string;detailEdit;actionEditing=goContact;notNull",
            "Main Business Contact"))
```

This string cell now becomes non-editable because you will use the page returned by the goContact method to edit the Contact. The notNull property means that the lastName attribute must not contain a null, otherwise an error message will be displayed. The goContact method is responsible for inserting a new Contact record into the database (e.g. inside the addingAction of masterObject's RCMList, then by using an RCMEdit on the returned detail editing page).

boolean detailEdit (OPTIONAL for all types. Default = false)

To activate the Detail Edit controls. If true, an "editing hand" icon will be shown at the end of the cell contents. This cell will now always be shown using a non-editable field.

String actionEditing (OPTIONAL if detailEdit is used, REQUIRED if pageEditing is NOT USED. Default = null)

The name of a method to be invoked if the user clicks on the "editing hand" icon. Your method should return a WOComponent of a detail page after performing any necessary prepatory functions.

String pageEditing (OPTIONAL if detailEdit is used, REQUIRED if actionEditing is NOT USED. Default = null)

The name of a WOComponent of a detail page to show if the user clicks on the "editing hand" icon. Use this if you can go to the straight to the next page without invoking an action first.

Support

Need help:

visit us at http://www.webappz.com/products/LISTedit/index.html, or
email us at support@webappz.com

Although rights to the source code are not included, WEBAPPZ is open to any suggestions for continuous framework improvements. Updated versions will be released regularly.

- www.webap.com -

WEBAPPZ Systems, Inc.
# 726 – 1489 Marine Drive
West Vancouver, BC
V7T 1B8 CANADA
604.921.1333