User Custom Guide

- Add new comment

This documentation covers the functionality of ExtraView User Custom programming. The User Custom extensions to ExtraView allow the programmer to extend, complement or replace the functionality of many parts of ExtraView.

If you intend to modify the behavior of ExtraView with User Custom programming, you will need to be skilled in coding with the Java and JavaScript languages. Most likely, you will also need skills in programming with SQL, to perform database access. In addition, ExtraView administration skills are required to configure many of the functions offered. If you are using User Custom to integrate ExtraView with other applications, you may also need to understand the ExtraView API. It is strongly recommended that you attend the User Custom Programming training course from ExtraView Corporation before attempting to build your own customizations within ExtraView.

ExtraView recommends strongly, that experienced programmers are utilized in developing User Custom extensions to ExtraView.

There are two principle ways of extending ExtraView with custom code:

- **User-Defined JavaScript** – this is typically used when you need to provide additional validation code, or need to submit a screen form to the server based upon a data value in your form. The characteristics are that the execution appears to be very fast to the end-user, as no trip to the server is involved, but you cannot perform coding to extract information from the database
- **User Custom Java code** – this is used to perform the "heavy-lifting" work, where you need to make appreciable alterations to the functionality of ExtraView. This work is performed on the server, after the user takes an action to submit or refresh a form within their browser.

Note: ExtraView can be customized with sophisticated rules that use a proprietary scripting language. These rules are much simpler to develop than custom code, and can perform many of the functions required to extend ExtraView. As a general principle, always consider using a rule before you consider custom code. They are much simpler and faster to develop, and are generally very efficient and simpler to maintain. Almost all common extensions can be accommodated via the use of rules.

Development Guidelines

- Add new comment

Development Instances

ExtraView recommends that customers install at least two, and preferably three instances of ExtraView, supporting web software and databases. These are typically known as the development instance, the staging instance and the production instance. The staging instance is optional. These instances should preferably be installed on separate machines, although it is common to use a single database server with multiple user accounts for the different instances.

1. **The development instance** – all development happens on this instance. When development and testing is complete, the code is moved to the staging instance
2. **The staging instance** – final testing and QA happens on this instance, after the development code is moved to this environment. Once the final testing is complete, the code is moved to the production

instance
3. **The production instance** – this is the instance of code that accesses the live database and is accessed by all end users.

The Development Environment

- ExtraView recommends a Window platform for development, although all the essential tools are available under Unix, Linux and Solaris
- Web Server – such as Apache
- Application Server – such as Apache Tomcat
- ExtraView – current version
- A supported database such as Oracle, MySQL or SQL Server
- Sun J2SE JDK
- Note: The JDK that you use to compile your user custom code must be the same version used by ExtraView to build ExtraView
- ExtraView and associated jars for developing user custom code.

See the recommended versions of the various components on the Server Requirements page.

Development tools

- Preferably use an IDE such as Eclipse
- Any text editor can be used in place of the IDE
- The ExtraView jar file - obtained from ExtraView Corporation
- **Do not develop code on your production ExtraView environment. At minimum, set up a separate development environment**
- Within ExtraView, set the behavior setting named USER_CUSTOM_CLASSNAME within the Behavior Settings Administration menu to be the name you will use for your UserCustom class, for example –
  com.extraview.usercustom.myCompanyName

Basic Development Steps

JVM Memory

1. Use javac with the classpath for your system and include the jars
2. Develop your code in a class named myCompanyName.java as the source file This class should probably extend UserCustom.java to inherit the capabilities that ExtraView ships in its best practices implementation. It is likely that you will extend the class file CustomCodeBase.java. This file itself extends UserCustom.java
3. Compile the new myCompanyName.java code
4. The myCompanyName.class is copied into the appropriate class directory. This is typically – <tomcat-base>/webapps/<instance-name>/WEB-INF/classes/com/extraview/usercustom
5. Test your code – the application server must be “bounced”, i.e., stopped and started, before you run the ExtraView application within your browser. Note that when bouncing the application server, all current interactive sessions are lost and all users are required to login into ExtraView again
6. Test the new code by running through use case scenarios. For example, add a problem to invoke the ucAddPreInsert method. Inspect the output through ExtraView and by looking at the ExtraView system log which is typically located at: <tomcat-base>/webapps/<instance-name>/WEB-INF/logs/EVJ.log
7. You should develop your code to output debug messages to the system log by invoking the system logging method
8. System.err.println messages are displayed on the application server console. This makes its use ineffective, and makes the system logging method much more preferable and convenient to use.

9. If ExtraView gets even just a single OUT OF MEMORY error, then the JVM –Xms and –Xmx arguments should be increased at the startup time of the JVM. Note that the JVM_HEAP environment variable is used to set these parameters

Coding Tips

- Add new comment

- The following resources are expensive and should be watched carefully
  - Database connections
  - Prepared statements
  - Result sets
- For any resource you use
  - After opening the resource, you must close it when you are finished with it
  - Resources should be initialized within a ‘’try” block
  - Resources should be closed or de-allocated within a ‘finally’ block.

Database Connections

- These are an expensive and finite resource
- Remember to close all connections you open, once your code has completed using them
- Failing to do so can lead to obscure problems “down the line”, when other processes run out of available connections in the available connection pool
- The number of connections and their time to live is controlled in the Configuration.properties file. This is an excerpt of the file with these settings –

```
# Connection pool settings
ConnectionPoolSize    = 10
ConnectionPoolMax     = 200
ConnectionUseCount    = 500
ConnectionTimeout     = 10
ConnectionPoolTimeout = 20
```

- Note that database connections should not use con.close
- Name the connection you open with the name of the method that opens the connection. This makes debugging using ConnectionPoolMon simpler as all connections using this name will be visible (all ExtraView connections are anonymous)
- Example -

```
public void frSelectList(SesameSession session, ValidationList selList,
                String pGlobal, HashMap selectedVals, HashMap attributes)
  {

  // code removed for clarity

  try{  con = Z.pool.getConnection();
          sql = "select security_user_id, first_name || ' ' ||
                last_name from security_user";
          pstmt = con.prepareStatement(sql);
          rs = pstmt.executeQuery();
          selList.put( "$$USER$$","* Current User Name *");
          while(rs.next()) {
                selList.put( rs.getString(1), rs.getString(2));
          }
          pstmt.close();
        }
        catch(SQLException sqle) {
            Z.log.writeToLog(Z.log.ERROR,
             "UserCustom.getAllowedValues SQLException: " + sql);
            ErrorWriter.write(sqle, ErrorWriter.LOGERR);
        }
        catch(Exception e){
            ErrorWriter.write(e, ErrorWriter.LOGERR);
        }
        finally{  if (con != null) Z.pool.close(con);
        }
      }
        return;
    }
```

Values

- Values within User Custom are generally objects such as Problem and Problem_UDF. They are stored as Strings or String Arrays in a hashmap (such as Problem Form Parameters) or as objects in the SesameSession Object
- Class and method variables can be any of these
- Generally, Strings and String arrays are used on the "user-facing" side, and Objects or ArrayLists of objects when dealing with the database
- It is generally easier to use the user facing ProblemFormParameters to manipulate data rather than the ArrayLists, so wherever possible, do the coding in these routines
- These are identifiable easily by the parameter lists
- In exits that are database oriented (primarily the ucAddPreInsert, ucEditPreUpdate ucAddPostInsert and ucEditPostUpdate routines) it is necessary to work with the Objects directly as well as the form parameters
- The session object can be used to store objects of arbitrary size, but this uses memory, and care should be taken to remove the object when it is no longer needed to prevent wasteful memory usage
- The memory will ultimately be returned when the session is invalidated, but depending on the session timeout, this can take some time.

Turning on Tracing of User Custom Calls

- To see all the calls to user custom methods and their timing information
  - As an ExtraView Administrator set USER_CUSTOM_ENABLE_METRICS = YES
  - View the log output using either tail –f evj.log (Unix / Linux) or an editor (Windows)
  - As you use ExtraView, the log will dynamically show the user custom methods called
- After Apache Tomcat (or your application server) is bounced, the log shows the current user custom class in use –

  ```
  2003-12-26 14:35:52 [  info   ]
  USER_CUSTOM_CLASSNAME=com.extraview.usercustom.myUserCustom3
  ```

- After Tomcat is bounced, the log shows the log’s name –

  ```
  2004-01-02 14:00:18 [  info   ]
  Manage Log Files Entered, prefix= art  , suffix= .log
  ```

- You can specify the name of the log file, which can make it easier to find the log you are using during development by modifying the ExtraView configuration file. The name of the log file is set in the file named Configuration.properties. This is stored in the directory named – %TOMCAT-HOME%\webapps\art\WEB-INF\configuration

  There will be a line like the following defining the log file’s name –

  LOG_FILE_PATH_NAME = logs/EVJ.log

- To make it easier to find you debug output in the log, with an editor select and delete the contents of the log before starting a debug run
- When tomcat is bounced, the log shows the database used by ExtraView and the user custom class that's currently in use.

Determining your Code's Performance

- To determine the performance impact of your user custom methods, turn on user custom metrics. Timing information will then be emitted to the ExtraView application server log
- To turn on tracing and metrics
    - Logon to ExtraView as an a user with administrator privileges
    - Select the Behavior Settings screen and set the value of the the behavior setting named **USER_CUSTOM_ENABLE_METRICS** to a value of YES
    - ExtraView will now make alog entry every time a user custom method is called and each entry has the number of milliseconds the call took to execute
    - If one of your user custom methods takes a second or longer to execute, it is probably degrading ExtraView's performance significantly.

Determining Useful Methods of a Class

- An IDE (Integrated Development Environment) like Eclipse will show all the methods in a class. The debugger will show the value of objects, such passed user custom method parameters and ExtraView objects like Z
- To determine what is either useful or available from a user custom method parameter, look at its getXXX methods. For example, the SesameSession class has the following available methods
    - getUserId
    - getUserRole
    - getRemoteAddress
    - getArea
    - getProject
    - etc.
- Looking at the class's Javadoc is another way to explore a class's available methods. The getters and setters are typically the ones of most interest to the User Custom programmer

Log Your Messages

- The user custom method w(String) is an easy way to write debugging messages to the log
- By default, the messages will be at the level of WARN messages, making them easy to find
- The Z.log.writeToLog() method also writes to the log.

Connection Pool Monitor

- Add new comment

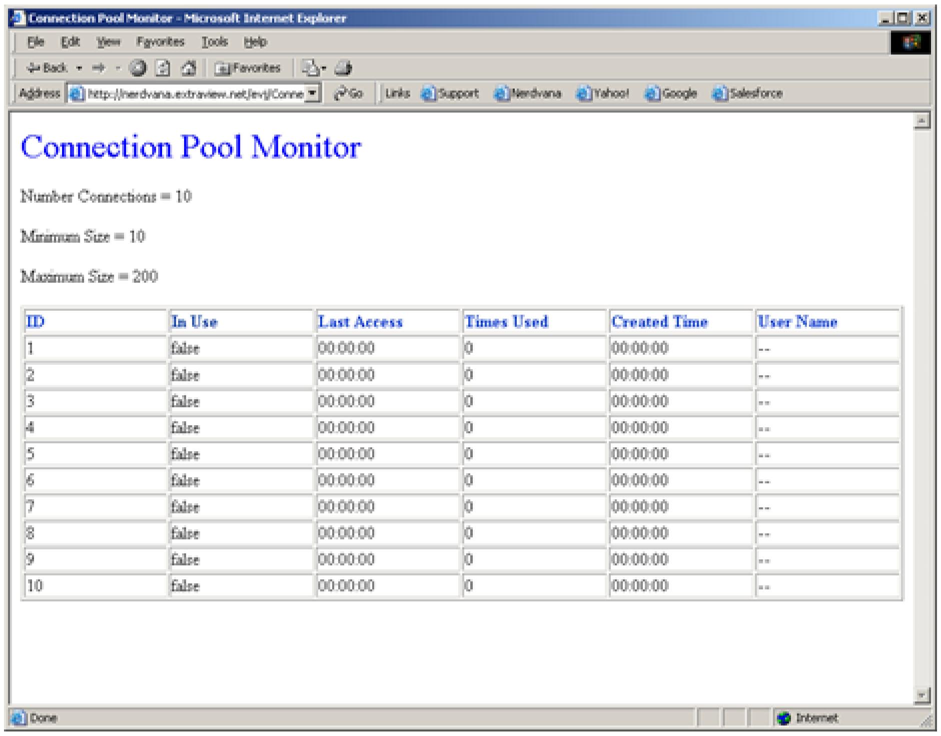
The ConnectionPoolMon utility is used to monitor all database connections in use by the ExtraView servlet. Database connections are finite in number and are created and destroyed by ExtraView as needed. The basic database connection parameters are set within the Configuration.properties file.

Running out of database connections will cause problems within your application, therefore it is important to ensure that when you create new connections in custom code, that you close them when the code completes its task.

Execute ConnectionPoolMon from a browser with request similar to –

http://www.mycompany.com/evj/ConnectionPoolMon

Following is a sample of the screen generated with ConnectionPoolMon. Note that the screen automatically refreshes every 30 seconds. You can keep this window open while debugging your application.


Connection Pool Monitor - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Favorites
Address http://nerdvana.extraview.net/evj/Conne Go
Links Support Nerdvana Yahoo! Google Salesforce
Done
Internet

Connection Pool Monitor

Number Connections = 10

Minimum Size = 10

Maximum Size = 200

ID	In Use	Last Access	Times Used	Created Time	User Name
1	false	00:00:00	0	00:00:00	--
2	false	00:00:00	0	00:00:00	--
3	false	00:00:00	0	00:00:00	--
4	false	00:00:00	0	00:00:00	--
5	false	00:00:00	0	00:00:00	--
6	false	00:00:00	0	00:00:00	--
7	false	00:00:00	0	00:00:00	--
8	false	00:00:00	0	00:00:00	--
9	false	00:00:00	0	00:00:00	--
10	false	00:00:00	0	00:00:00	--

JavaScript Interface

- Add new comment

User-defined JavaScript functions can be attached to individual fields on each *add* or *edit* screen within ExtraView. The JavaScript functions are not restricted in scope. This powerful feature allows you to generate virtually any functionality within an add or edit screen. Some typical uses of this feature are:

- Generate simple or complex data checking functions that are triggered upon selecting or editing an individual field
- The functions can refer to multiple fields on the form, thus allowing you to check for specific combinations of data. E.g. you may want to check the value of a field entered, according to the specific Product selected, one of two Statuses selected, and the Assigned To for the issue
- Alter the requiredness of a field or fields
- Alter the visibility of a field or fields
- Provide mathematical computation between fields on a form, within the browser and without using a business rule executed on the server
- Override the inbuilt ExtraView JavaScript functions

To accomplish the setting up of a user-defined JavaScript function, first note that each ExtraView form generated for most screens contains the following statement:

```
<SCRIPT LANGUAGE="JavaScript" SRC="xxxxxxx/user_javascript/UserJavaScript.js">
</SCRIPT>
```

where xxxxxxx is the environment name. This permits the programmer to reference JavaScript functions within the ExtraView environment.

Note: Browsers do not generate an error if the UserJavaScript.js file does not exist. This gives transparent functionality if no user-defined functions exist. Note: This directory exists within the web server, not the application server directory tree.

There are occasions when you might want to include an entire existing JavaScript source library, file or source tree. This is achieved with the use of the behavior setting named USER_CUSTOM_JAVASCRIPT. Put the path name of the files you want to load into the value of this setting. The path is relative to the directory named *javascript* within your ExtraView environment. Separate multiple files with a semi-colon character.

No checking is done for the presence of this file, so be careful to make sure your JavaScript file exists at the location you provide.

Creating JavaScript Methods

- Add new comment

There are several pre-defined JavaScript exits which are described on subsequent pages in this guide. In addition, you can create your own JavaScript methods which you can attach and call to any of the fields on the form, using a JavaScript event to trigger the entry to your own method.

There are two steps to creating your own JavaScript method. First, you create a link from the field on the form to your JavaScript method. Secondly, you create the JavaScript method within the UserJavaScript.js file on the server.

- Within the layout editor of ExtraView, select the field that you wish to use to trigger the user-defined JavaScript function, and create a layout element attribute of type **HTML Modifier**. For example, an HTML modifier can be created similar to –

  ```
  onchange=user_fn(paramList);
  ```

  This will call the function in the JavaScript file named user_fn on the server for operations such as validation, or altering the required or visibility attribute of fields. You can use any other JavaScript event such as onclick, onfocus, onkeyup, or onmouseover to trigger the call to the JavaScript.

- You may place any valid JavaScript methods within the UserJavaScript.js file. This allows each user-defined JavaScript method to be executed from any field that references the method by name. Example JavaScript function –

  ```
  function validateUPS(val) {
          alert("Value passed is: " + val[val.selectedIndex].text);
          return;
  }
  ```

Support Methods

customEditorToolbar

The HTML Area utility, used within HTML Area fields, the sign on message, custom email composition and preparing scheduled reports, has a toolbar which is set with the behavior setting named EDITOR_TOOLBAR.

This has 3 settings, BASIC, STANDARD and FULL, offering 3 preset arrangements of toolbar buttons. If you require to customize the buttons to provide your own arrangement, then this custom JavaScript exit enables this capability. You must set the behavior setting EDITOR_TOOLBAR to a value of CUSTOM for this method to be called. The HTML Area utility is built using the CKEditor open source utility, and you can look at www.ckeditor.com for more information on its configuration.

Signature

```
customEditorToolbar()
```

The `customEditorToolbar` method must return a string value which is the composition of the toolbar you require, with the different button groups and buttons in the appropriate order.

Example

```
function customEditorToolbar() {

  return(
  [
    { name: 'document', items : [ 'Source', '-', 'Save', 'NewPage',
'DocProps', 'Preview', 'Print', '-' ,'Templates' ] },
    { name: 'tools', items : [ 'Maximize', 'ShowBlocks', '-', 'About' ]
},
    { name: 'basicstyles', items : [ 'Bold', 'Italic', 'Underline',
'Strike', 'Subscript', 'Superscript', '-', 'RemoveFormat' ] },
    { name: 'paragraph', items : [ 'NumberedList', 'BulletedList', '-',
'Outdent', 'Indent', '-', 'Blockquote', 'CreateDiv', '-', 'JustifyLeft',
'JustifyCenter', 'JustifyRight', 'JustifyBlock', '-', 'BidiLtr',
'BidiRtl' ] },
    { name: 'links', items : [ 'Link', 'Unlink', 'Anchor' ] },
    { name: 'insert', items : [ 'Image', 'Flash', 'Table',
'HorizontalRule', 'Smiley', 'SpecialChar', 'PageBreak', 'Iframe' ] }
   ]);
}
```

The example shows the composition of a toolbar with 6 groups, each with a range of buttons. The `'-'` inserts a divider between buttons within a group. An `'/'` entry inserts a new line into the toolbar.

CKEditor Plugins

If you are using the CKEditor environment to develop your own plugin which adds additional buttons to the toolbar, you need to have a statement within your UserJavaScipt.js file which identifies the names of the plugins you are adding. You add a global variable to your UserJavaScript.js file with the syntax:

```
evPlugins = 'myPlugin1[,myPlugin2][,myPlugin3]';
```

For example, if you have created a single plugin named `specialLink`, the statement becomes:

```
evPlugins = 'specialLink';
```

Please review the documentation for CKEditor at [ckeditor.com]for information on creating plugins for the environment.

doLoad

- Add new comment

On all the key ExtraView screens, such as the *add* screen, the *edit* screen and the *query* screen, there is a standard JavaScript call in the ONLOAD attribute of the BODY tag within the HTML page. This call is to a method with the following signature:

```
doLoad(document.editForm)
```

The doLoad method initializes several required elements of the form. In addition, this method calls a function within the UserJavaScript.js file. This method has the following signature:

```
function userJavaScriptOnload(displayMode)
```

This method need not exist in the UserJavaScript.js file. If it does exist it can be used to perform initialization functions to the administrator's specification. The parameter displayMode is used to help know which screen is being called. It can have the following values:

Value	Purpose
ADD	This shows that the add issue screen is being initiated
EDIT	This shows that the edit issue screen is being initiated
REFRESH	This shows that either the add or the edit screen is being refreshed
SEARCH	This shows that the query or a report edit screen is being initiated

In most cases an ExtraView business rule can be used to initiate values on a form, but in some cases you need more interaction than a rule can provide. The following example shows how an ActiveX control is called within the loading of an add screen, and populates values into fields on the form:

```
function userJavaScriptOnload(displayMode) {
// IE only function to activate an ActiveX control and take some
// of the values and populate them into a form
// Here we are getting the network hardware and configuration
// information on the user's computer

// For this to work, the host site being accessed by the client
// must be a trusted site, and the security
// settings for the trusted site zone must be altered to allow
// ActiveX controls to run without hindrance
```

```
  if (displayMode != ‘ADD’) return;
  var locator = new ActiveXObject ("WbemScripting.SWbemLocator");
  var service = locator.ConnectServer(".");
var properties = service.ExecQuery("SELECT * FROM Win32_NetworkAdapterConfiguration");
  var d = document.editForm;
  var e = new Enumerator (properties);
  for (;!e.atEnd();e.moveNext ()) {
  // just take the first element returned and put into EV fields
    var p = e.item ();
    d.p_help_caption.value       = p.Caption;
    d.p_help_description.value   = p.description;
    d.p_help_dhcp_enabled.value  = p.DHCPEnabled;
    d.p_help_dhcp_sever.value    = p.DHCPServer;
    d.p_help_dns_host_name.value = p.DNSHostName;
    d.p_help_ip_address.value    = p.IPAddress(0);
    return;
  }
}
```

ucRGIFormLoad

The method ucRGIFormLoad will be called with the IFrame object after a Related Issue Display iframe is loaded and resized, either in the case of the initial rendering, a refresh, or a column re-sort. This method has the following signature:

```
function ucRGIFormLoad(iframeObject)
```

This method need not exist in the UserJavaScript.js file. If it does exist, it can be used to perform functions to the administrator’s specification. The parameter iframeObject is used to help know which Related Issue Display is being called. The following example shows how an alert script will show the name of the iframeObject when a Related Issue Display is loaded, sorted, or refreshed:

```
function ucRGIFormLoad(iframeObject) {
    alert('hello from rgi formload: ' + iframeObject.name);
}
```

userJavaScriptAddBtn

- Add new comment

There is an optional method within the UserJavaScript.js file named userJavaScriptAddBtn(). When this method is present you can return true or false to allow ExtraView to perform additional processing when the **Submit** button is pressed on the *add* screen. If this processing returns true, then the submit function continues, but if it returns false, then the submit is halted. Typically you will present the user with a message to indicate the error that you want them to correct. The default code for the method within the UserJavaScript.js file is:

```
function userJavaScriptAddBtn() {
    return true;
}
```

If the method is not present, no error is generated.

userJavaScriptOnloadAjax

- Add new comment

Ajax calls are used within forms for several purposes, for example to process the results for many layout cell attributes such as Required If, and Visible If, as well as to process dependencies on embedded layouts such as their visibility. There is a user custom JavaScript method which, if defined, allows you to provide additional logic following the return to the client browser from an Ajax call. The signature to the method is userJavaScriptOnloadAjax(layoutType).

layoutType is a String parameter that may have a value of ADD_PROBLEM, EDIT_PROBLEM, or SEARCH, depending on the layout type calling the method. This allows you to have different logic executed according to the layout calling the method.

userJavaScriptUpdateBtn

- Add new comment

This method is called before updating an issue on all *Edit* screens.

This is an optional method within the UserJavaScript.js file. When this method is present you can return true or false to allow ExtraView to perform additional processing when the **Update** button is pressed on the edit screen. If this processing returns true, then the submit function continues, but if it returns false, then the update is halted. Typically you will present the user with a message to indicate the error that you want them to correct.

The default code for the method within the UserJavaScript.js file is:

```
function userJavaScriptUpdateBtn() {
    return true;
}
```

If the method is not present, no error is generated.

chkRqdFlds

There are some circumstances where it is useful to create new issues within the database, but to have these flagged as being in a *draft* mode.

In this mode, no checking for required fields will be made. Before saving the issue in a non-draft mode, the issue must be edited, and all the required fields must be completed.

To enable this feature, an entry within the UserJavaScript.js file must be made, as follows:

```
    if (typeof(chkRqdFlds) != "undefined")  chkRqdFlds = "NO";
```

This call should be placed within a function, and the function called from a custom Java method utilizing the `setButtonDopeMap` method, adding a new button to the menubar on the *add* and *edit* screens, with a title of **Save As Draft** or similar, to invoke the JavaScript function.

Note that there are dangers to recognize. This practice allows issues which may not have required fields added to the database. Therefore it is recommended that you also add a **Status** of DRAFT to your system, and this is set at the same time that you call the function. Queries and reports may then filter out these issues.

evUserCustomReadOnlyFields

There is an option that can be placed into the UserJavaScript.js file to disable the ability for user clicking on read-only fields. This has the effect of making them read-only on *add* and *edit* screens.

To accomplish this, create an array of the field names that should be rendered in this way. The assumption is that the values in these fields are already rendered as toggle boxes. The name of the array you create is `evUserCustomReadOnlyFields`.

For example, within the UserJavaScript.js file, use this statement to apply the conditioning to fields named TEST_PLAN_COMPLETE, NEED_DOCUMENT:

```
var evUserCustomReadOnlyFields = ['test_plan_complete', 'need_document'];
```

userJavaScriptDeleteBtn

There is an optional method within the UserJavaScript.js file named userJavaScriptDeleteBtn(). When this method is present you can return true or false to allow ExtraView to perform additional processing when the **Delete** button is pressed on the *edit* screen. If this processing returns true, then the delete function continues, but if it returns false, then the delete operation is halted. Typically you will present the user with a message to indicate the error that you want them to correct. The optional message is handled by the method userJavaScriptDeleteMessage(). The default code for the method within the UserJavaScript.js file is:

```
function userJavaScriptDeleteBtn() {
    return true;
}
```

If the method is not present, no error is generated.

userJavaScriptDeleteMessage

There is an optional function within the UserJavaScript.js file named userJavaScriptDeleteMessage(). This method will replace the standard, inbuilt message returned by ExtraView when an issue is deleted with a string value. This method is not executed if the optional userJavaScriptDeleteBtn() function returns *false*. The default code for the method within the UserJavaScript.js file is:

```
function userJavaScriptDeleteMessage() {
    return;
}
```

Simply code your own message into the function, and return a string. If the method is not present, no error is generated.

HTML Modifiers & QuickEdit

- Add new comment

Quickedit reports do not handle general HTML modifiers, as it cannot be predicted what might affect the whole report or the whole page. In particular, if an HTML modifier causes a screen refresh it must not be allowed to exist within a Quickedit report. The following HTML Modifiers are supported as part of the base ExtraView functionality and work as described in the Administration Guide:

- autoSize(this)
- formatAsPhoneNum(this)
- submitChange(this.name)

To provide an alternative mechanism, there is a function within the UserJavaScript.js file that you can use to provide an exit to JavaScript to replace the HTML modifiers. The default is the following, but the function is ignored if it does not exist.

```
function ucHtmlModifierQuickEdit() {
    return true;
}
```

Modifying Calls to External URL Links

- Add new comment

When a field in the data dictionary utilizes the **Display as URL** feature, ExtraView places a link button by the field on an *add* or *edit* form, and turns the displayed value into a link to the URL on a report. It can sometimes be convenient to override this behavior and create your own link to the external URL. This operation is done with a UserJavaScript function. The method to add to the UserJavaScript.js file is named userJavaScriptRemoteLink(). If this method exists, it takes the following three parameters:

1. startURL – the url to be invoked
2. extra – the comma-separated list of parameters for the window open operation
3. fieldname – the name of the Display_as_URL field

Return code: false implies that the url should be invoked by ExtraView code as previously would have been done; this is:

```
window.open(startURL, "_blank", extra);
```

If the return code is true, then ExtraView will do nothing additional for the field and assumes that the User JavaScript issues the URL in its own way. Here is an example of userJavaScriptRemoteLink:

```
function userJavaScriptRemoteLink( startURL, extra, fieldName) {
  window.open(startURL, "_blank", "width=750,
   height=650, location=no, status=no, scrollbars=yes,
   resizable=yes, dependent=yes");
  return true;
}
```

Overriding Inbuilt Functions

- Add new comment

Occasionally it is useful to replace the standard functionality of an ExtraView inbuilt JavaScript function with a user-defined JavaScript function. You can use JavaScript's ability to return a Boolean answer in response to a query on the existence of a method. As an example, you may want to override the inbuilt ExtraView

JavaScript function named SubmitChange. This method lives within a JavaScript file that is accessed from the *Add Issue* screen, named AddEdit.js.

- The ExtraView JavaScript functions are located in the main ExtraView code directory on the application server, within the directory name javascript. Within this directory, another directory named user_javascript contains the user-defined JavaScript functions
- Create a function named userSubmitChange in the file AddEdit.js
- In the UserJavaScript.js file, create a function named _userSubmitChange, which actually performs the task needed
- Now you can call userSubmitChange, which will look like the following code snippet –

```
/**
 * this will call the _userSubmitChange if
 * it is defined, else it calls submitChange
 */
function userSubmitChange(field){
    if(_userSubmitChange){
        _userSubmitChange(field);
    } else {
        submitChange(field);
    }
}
```

If the function _userSubmitChange doesn't exist in the UserJavaScript.js file, then the standard submitChange function is used instead. Its existence is assured as it is contained in the standard ExtraView distribution file.

Replacing the Print Screen Capability

- Add new comment

When you use the browser’s **Print** button, or you use the **Print Page** button at the top or bottom of ExtraView’s main screens, the printout is handled completely by the browser. Many times this is sufficient, but often this printout is not formatted well. For example, you may have an *add* or *edit* screen that prints too wide for a printer in portrait mode, or you may not like the way that the browser cuts off the text within text area and similar fields, printing only the visible part of the text as opposed to the complete text.

There is an optional JavaScript method that will allow you to provide an alternative to the standard JavaScript window.print() method. This method is only provided for the *add* and *edit* screens.

If you provide this method, it is entirely up to you to define what you will do as an alternative; this may include calling a server function or handling your alternative print function completely in JavaScript. The signature for the method, should you provide this in the UserJavaScript.js file is:

```
function printAddEditPage() {
  // here is alternative code to print an add or edit screen
. . .
. . .
}
```

Example - Calculated Fields

- Add new comment

This example shows how to provide two calculated fields on an *Add* or *Edit* screen layout. This could be achieved by using business rules, but this examples serves the purpose of showing alternative methods of achieving similar results. The key difference is that business rules are executed via an Ajax call back to the server, while the JavaScript approach downloads the code to the client browser where it is executed.

The example involves two fields named VAL1 and VAL2. Numeric data entered into these two fields will be summed and the result placed in a third field named VAL_SUM, and the two fields will be multiplied and the result placed in a fourth field named VAL_PRODUCT. All four fields have a display type of Number. To set up this requirement, take the following steps –

- Within the data dictionary, define the fields VAL1, VAL2, VAL_SUM and VAL_PRODUCT, all with a display type of Number
- Make sure you give these fields read and write permission
- Place all the fields on an *Add* or *Edit* screen layout. Add an HTML modifier to the fields named VAL1 and VAL2. The modifier value should be onchange=calc();
- Place the following JavaScript functions within the UserJavaScript.js file. Note that the field names referenced in the layout have **p_** prefixed to their names. These are the parameter names of the fields

```
function valOf(which) {
  if (! which) return 0;
    var val = parseInt(which, 10);
  if (isNaN(val)) return 0;
    return val;
}

function calc() {
  document.editForm.p_val_product.value =
                  document.editForm.p_val1.value *
                  document.editForm.p_val2.value;
  var a = document.editForm.p_val1.valueOf;
  document.editForm.p_val_sum.value =
                  valOf(document.editForm.p_val1.value) +
                  valOf(document.editForm.p_val2.value);
  return;
}
```

- When the user changes either the field named VAL1 or VAL2, the JavaScript function is triggered, and the results are immediately displayed in VAL_SUM and VAL_PRODUCT.

Example - Field Validation & Formatting

- Add new comment

This example shows how to format and validate a US telephone number field on an *Add* or *Edit* screen layout.

The example involves a single field named tel_number. Data in the form of numbers entered into this field will automatically be formatted in the usual form of a telephone number in the USA. For example, entering the digits 1234567890 will result in a display of (123)456-7890. In addition, the function will not allow any non-numeric characters to be entered unless the user acknowledges that this is what they want.

To set up this requirement, take the following steps –

- Within the data dictionary, define the field TEL_NUMBER with a display type of Text Field
- Make sure you give these fields read and write permission
- Place this field on an *Add* screen layout. Add an HTML modifier to this field. The modifier value should be onkeypress=formatAsPhoneNumber(this);
- Place the following JavaScript functions within the UserJavaScript.js file.

```
function chkNAN(char2chk)
{
   var validNum = "0123456789";
   if (validNum.indexOf(char2chk) == "-1")
      return(confirm("You have entered a non-numeric
                      character.\nDo you want to turn off non-
                      numeric character checking?"));
}

function formatAsPhoneNum(fld)
{
   var isNamedFone = false;
   fldVal = fld.value;

   var tmpStr = "(";
   keyCount = fldVal.length;
   keyEntered =fldVal.substring(keyCount-1,keyCount);

   if (keyCount < 2)   isNamedFone = false;
   if (!isNamedFone)   isNamedFone = chkNAN(keyEntered);

   keyCount++;
   with (document.editForm)
   {
      switch (keyCount)
      {
         case 2:
            tmpStr +=  fldVal;
            fld.value = tmpStr;
            break;
         case 5:
            fld.value +=  ")";
            break;
          case 9:
            fld.value += "-";
            break;
      }
   }
}
```

- When the user enters the telephone number in the field it will be validated, and the formatting applied to the number. If they enter an invalid character, they will see a dialog box with the message **You have entered a non-numeric character. Do you want to turn off non-numeric character checking?**

Java Interface

- Add new comment

UserCustom is a Java class consisting of callback methods that are invoked by ExtraView. This class is extended to modify the functionality of ExtraView.

The class facilitates integration with 3rd party systems and/or additional data validation and provides for business rule customization over and above those provided by the base product.

UserCustom extensions allow the programmer to alter the base functionality of many parts of ExtraView. To create a user custom class, inherit from this class and override the methods of interest. ExtraView Corporation has extended the base ExtraView product with many useful functions as part of its best practices implementation. If you want to take advantage of these functions, then extend CustomCodeBase.java rather than UserCustom.java. CustomCodeBase.java itself extends UserCustom.java. Given you have extended the CustomCodeBase.java class, and you want to override the code in a method within that class as well as executing the same method name in your own class, then use the Java special variable super. This can be placed at different points in your own method, allowing you to decide whether to execute your own code first or last in sequence.

The convention for naming user custom classes is to use the company name and optionally append a version. For example, if your company is named MyCo, name your user custom class as –

MyCo.java

The beginning of a sample skeleton user custom class becomes:

```
package com.extraview.usercustom;

import java.io.*;
import java.sql.*;
import java.text.*;
import java.util.*;
import javax.servlet.http.*;
import java.text.SimpleDateFormat;

import com.extraview.applogic.admin.*;
import com.extraview.applogic.layout.*;
import com.extraview.applogic.problem.*;
import com.extraview.applogic.report.*;
import com.extraview.applogic.security.*;
import com.extraview.common.datatype.EnumeratedDataType;
import com.extraview.common.*;
import com.extraview.util.*;
import com.sesame.misc.*;
import com.sesame.template.*;
import com.extraview.dbms.Sequence;
import com.extraview.presentation.*;
import com.extraview.presentation.chart.*;
import com.extraview.presentation.problem.*;
import com.extraview.presentation.search.EVHTMLWorkerFont;
import com.extraview.presentation.usercustom.*;

public class MyCo extends CustomCodeBase{
. . . .
. . . .
}
```

Principles

The UserCustom.java class is the customer’s way of extending the functionality and validation features within ExtraView. Working with this class implies that you will have intimate knowledge of the underlying database environment, the Java programming language, and the environment that ExtraView offers to programmers. In addition, familiarity with the ExtraView schema is assumed. ExtraView has considerable experience using this interface to custom write code for methods such as:

- Providing additional business rules that are not part of the base product
- Directly linking ExtraView through sophisticated data management techniques to other Oracle database systems, such as CRM and SCM systems
- Link to remote applications of any type, through URL methods
- Allowing remote applications to call ExtraView and perform data extractions or data updates

- Allowing ExtraView to call remote applications and perform updates or data extractions. The principal functions of the UserCustom module allow the administrator to modify the behavior of ExtraView at many points during the process, for example –
    - Before you insert a new issue
    - Before you update an existing issue
    - After you insert a new issue
    - After you update an issue.

All code changes made within the UserCustom source code are limited to a single customer of ExtraView. They will not be overwritten by successive upgrades of the ExtraView product. However, care should be taken when upgrading to test that your functionality still works as expected.

The interface can be used to extend ExtraView in virtually any direction, and can call other external packages and systems. It is assumed that the programmer has a working knowledge of the following:

- ExtraView administration
- Knowledge of Java programming
- Knowledge of SQL programming
- Knowledge of HTML programming is useful, but not required
- Knowledge of web servers such as Apache is useful, but not required
- Knowledge of application servers such as Apache Tomcat is useful, but not required.

The user exits are method calls to a Java class named UserCustom.java. The method calls are made synchronously with the execution of a service in the servlet environment which runs under the application server. All code executed in the UserCustom methods is completed before responses are sent to the client browser; this should be taken into account when considering the use of long-running methods. Performance of the user interface is very important to the user’s perception of speed of the entire application.

Certain objects are passed to the UserCustom methods as defined by the UserCustom Interface object. These objects model the customer business objects within ExtraView and maintain the current object context, some of which may be modified by the customer programming the UserCustom methods. All object attribute modification must be done through setter methods; all inspection of object attributes must be done through getter methods. It is possible for the programmer to modify objects in ways that cause erroneous behavior with respect to the customer’s business model; therefore, care must be taken to test and verify any programs that will be installed in a production system. Customers can consult with ExtraView about any aspect of proper or improper use.

All code added to the UserCustom class must be thread-safe, since multiple threads will execute simultaneously on that code. This implies that all libraries called by UserCustom methods must also be thread-safe. All ExtraView methods are thread-safe.

ExtraView Internals

- Add new comment

Metadata

Metadata is data that resides within the ExtraView database. Metadata describes other data items. For example, the description for fields and their characteristics is metadata that is contained within the data dictionary. The following examples represent some of the types of metadata within ExtraView:

- Data Dictionary
- UDF Permissions

- Title Mapping for localized data
- Layouts for screens.

Metadata drives a very significant amount of functionality within ExtraView. Very little in the way of data relationships or field definitions is hard-coded within ExtraView.

For example, when you define a new User Defined Field, ExtraView does not create a new column in a database table. Metadata is created in a number of tables that describe the new field and its behavior; if the field is one of the list display types, pointers are set up to the members of the list where the values are stored.

The traditional method of setting up parent child relationships between fields in a database is to use foreign key constraints. While ExtraView makes extensive use of foreign key constraints, they are not used to manage the parent child relationships that you create yourself, between two or more of your fields. Instead, ExtraView uses metadata to describe these "Allowed Values". This avoids the need to modify the database. It is interesting to note that every single ExtraView customer has the identical schema to its database. This avoids many of the headaches that application building companies have had in the past, when different customers had different schemas.

Issue Data

Issue data is the data stored within your system that contains the values of data. For example, issue data contains the text within an issue, or the value of the status of an issue. Sample issue data includes:

- The item table which stores the key information about an issue
- item UDF’s. These are the values of the user defined fields within an issue
- product table information. This is where the product information is stored
- module table information. This is where the module information (which is a child of product information) is stored
- Repeating row records. These are structures that are dependent upon the issue data, and which can be repeated for different values within the issue.

Reports

Report data holds key information about reports held within ExtraView. For example, a report consists of a layout, filters, a title and description. There are several types of reports within ExtraView:

- Quicklist – this is a built-in report format, used as a default layout for reports, when summary report information is required
- Detailed Report – this is a built-in report format, displayed when the user wants a complete detailed record of their issues
- Columnar Reports – these are designed by the user, and can display any set of fields (assuming the user’s role has permission) with a given set of filters
- Summary Reports – these allow the summarizing of counts of issues of up to four fields, for a given set of filters
- Matrix Reports – these allow the production of a table of information using two different fields, one as the count on the X axis and one as the count on the Y axis
- Aging Reports – These provide information that summarizes the number of issues within different statuses, according to the time the issues selected by the filters for the report, have spent within the statuses. A drilldown allows the user to see the time each issue spent in all statuses up to the current point in time
- Charts – these display information in a pie, line or bar chart form. Unlike other reports, they can display information for any date in history, up to the current date
- Custom Report – allow an entry on the report menus for reports designed using techniques discussed in this document, and reports which cannot be built using the standard ExtraView reporting tools

- Layouts for Existing Reports – These allow the user to combine the results of summary and other reports together and to create a style that is used to place the results on a layout. This layout is then rendered to PDF output when the layout is placed within a Container report
- Container Reports – These are used to take the output from a Layout and place the output into a PDF file. Multiple layouts can be placed on a single Container report.

Useful Internal Methods

Printing Report Definitions

If you are creating a user custom report and wish to output the report definition, the following code fragment will be useful. This may be used when the behavior setting named GRANULAR_REPORT_DEFINITIONS has a value of YES.

```
ReportDefinition rd = new ReportDefinition();
fp.put("p_output_type", "BROWSER");  // note: need to generate unescaped HTML, esp. for PDF output
fp.put(ReportDefinition.OUTPUT_REPORT_DEFINITION_PARAM, "yes");
fp.put(ReportDefinition.OUTPUT_REPORT_PROPERTY_PARAM, "yes");
fp.put(ReportDefinition.OUTPUT_FILTER_DEFINITION_PARAM, "yes");
fp.put(ReportDefinition.OUTPUT_CUSTOM_FILTER_PARAM, "yes");
String reportDefinitionHtml  = rd.doOutputReportDefinition(session, dbconn, fp, report, null);
```

Debugging

- Add new comment

The ExtraView Log File

The ExtraView application server log is a very valuable tool in writing and debugging Java in ExtraView. It is possible to issue timestamped messages to the log file while the program is running. The following sample call is used to issue log messages:

Z.log.writeToLog (Log.DEBUG, "String to be inserted");

This will insert the parameter string into the log subject to the log filtering constraints. In general, the "log level" is set internally to a value that allows only error, critical error, and information messages to be inserted to the log. However, a simple CLI command can set the debug level for log messages:

debug 7

You can also set the log level with a URL, such as:

http://www.mycompany.com/evj/ExtraView?DEBUG=7

This sets the log level internally to allow all DEBUG messages to be stored in the log, viz., those that are issued by the program with “Log.DEBUG” as the first parameter.

Where is the log file? It is located in:

```
<tomcat-base>/webapps/<instance-name>/WEB-INF/logs/<logname>.log
```

However, there is a parameter named LOG_FILE_PATH_NAME within the Configuration.properties file that allows you to relocate the log file to a different location. The log file is a simple sequential text file that can be read using any text editor.

Debug Levels

Up to 12 levels of debugging are provided. Level 6 is the value used for production sites, and 6 means only messages with a level of 6 or less are written to the log.

Note: A value greater than 6 can significantly impact the performance of a production system, and should only be used when trying to trace a specific problem.

The following error levels are defined for the log –

DEBUG Name	DEBUG Level
ALERT	1
CRITICAL	2
ERROR	3
WARN	4
NOTICE	5
INFO	6
DEBUG	7
DEBUG2	8
DEBUG3	9
DEBUG4	10
DEBUG5	12

Monitoring and Evaluating the SQL that is Executed

If you insert the line -

PSP_LOG = YES

Into the `WEB-INF/configuration/Configuration.properties` file, then an entry is made into the log file, showing each SQL statement as it is executed, along with all the bind variables used in its execution.

Monitoring and Evaluating Ajax calls

If you insert the line AJAX_LOG = YES Into the WEB-INF/configuration/Configuration.properties file, then an entry is made into the log file, showing metrics about each Ajax call as it is executed.

Values and Parameters

- Add new comment

Values

Typically, you follow the following steps to manipulate values:

- Start by using the information contained within the with form parameters
- Add in manipulation of UDFs, and other internal objects
- Then move to read-only database access, used for, say, select lists
- Then move on to mail list updates

Defining and Manipulating Screen Parameters

You must work with fields that have been created within the Design Center or Data Dictionary. In addition, before a field appears within the parameters for a form, it must have been placed on that layout.

Value types

- String
- String array
- Calendar
- Calendar arrays.

All values are normally stored in Maps and HashMaps.

Manipulation of Values

The FormParameters value for p_assigned_to is not frequently used.

The most frequent method of manipulating values is to use a HashMap, indexed by the field name from the form on the web page. For example:

fp.get("p_assigned_to")

The key to the HashMap is the same as the data dictionary field name. For example:

values.get("ASSIGNED_TO")

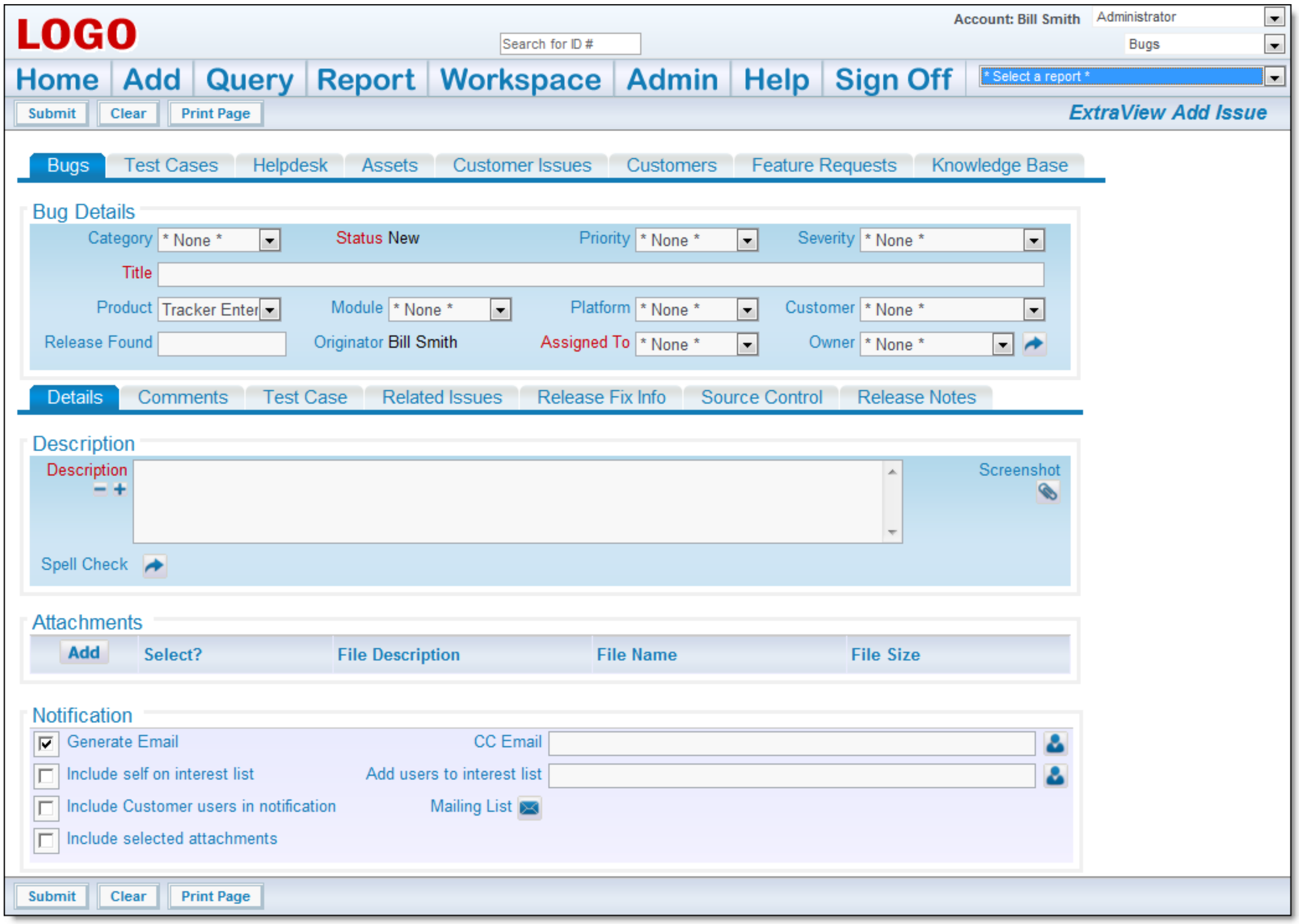
Parameters

A fundamental part of understanding how to code with User Custom methods within ExtraView is to follow how ExtraView places fields on screen layouts, such as the *add* issue screen or the *edit* screen. This is done by processing a series of parameters that are passed between the server and the client browser. The parameters are created within the ExtraView servlet, passed within HTML to the browser, handled by the browser, then submitted back to the servlet where they are processed. When processed in User Custom code, the programmer gets the parameter information from the FormParameters object that ExtraView creates. ExtraView creates a new FormParameters object for each request it receives from the web browser.

As an example, let us view how ExtraView processes the following user interactions once they have signed on to ExtraView:

- Select the **Home** navigation bar button, and navigating to the *Home Page* screen
- Select the **Add** navigation bar button, and navigating to the the *Add Issue* screen

- Complete entering fields on the *Add Issue* screen
- Select the **Add to Database** button
- The user is placed on the *New Issue Verification* screen.



Bug Details

Category	* None *	Status	New	Priority	* None *	Severity	* None *
Title							
Product	Tracker Enter	Module	* None *	Platform	* None *	Customer	* None *
Release Found		Originator	Bill Smith	Assigned To	* None *	Owner	* None *

Description

Description

Screenshot

Spell Check

Attachments

Select?	File Description	File Name	File Size

Notification

- [✓] Generate Email
- [ ] Include self on interest list
- [ ] Include Customer users in notification
- [ ] Include selected attachments

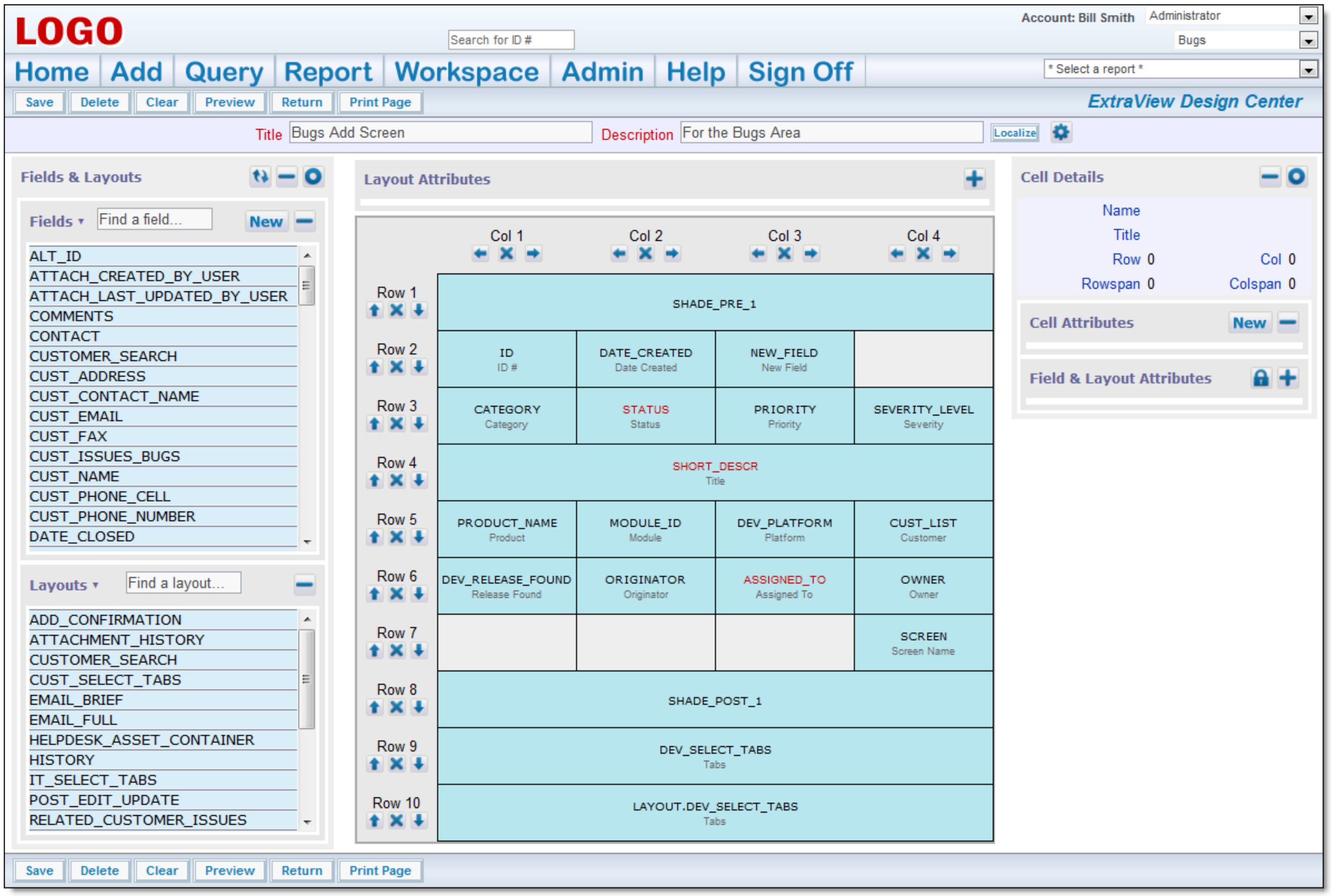
CC Email

Add users to interest list

Mailing List

The Add Issue screen

Explained fully in the Administration Guide, the format and placing of all the fields on the *Add Issue* screen is maintained in the Design Center for the *Add Issue* screen, shown following:



	Col 1	Col 2	Col 3	Col 4
Row 1	SHADE_PRE_1			
Row 2	ID ID #	DATE_CREATED Date Created	NEW_FIELD New Field	
Row 3	CATEGORY Category	STATUS Status	PRIORITY Priority	SEVERITY_LEVEL Severity
Row 4	SHORT_DESCR Title			
Row 5	PRODUCT_NAME Product	MODULE_ID Module	DEV_PLATFORM Platform	CUST_LIST Customer
Row 6	DEV_RELEASE_FOUND Release Found	ORIGINATOR Originator	ASSIGNED_TO Assigned To	OWNER Owner
Row 7				SCREEN Screen Name
Row 8	SHADE_POST_1			
Row 9	DEV_SELECT_TABS Tabs			
Row 10	LAYOUT.DEV_SELECT_TABS Tabs			

Design Center showing the Add Issue layout

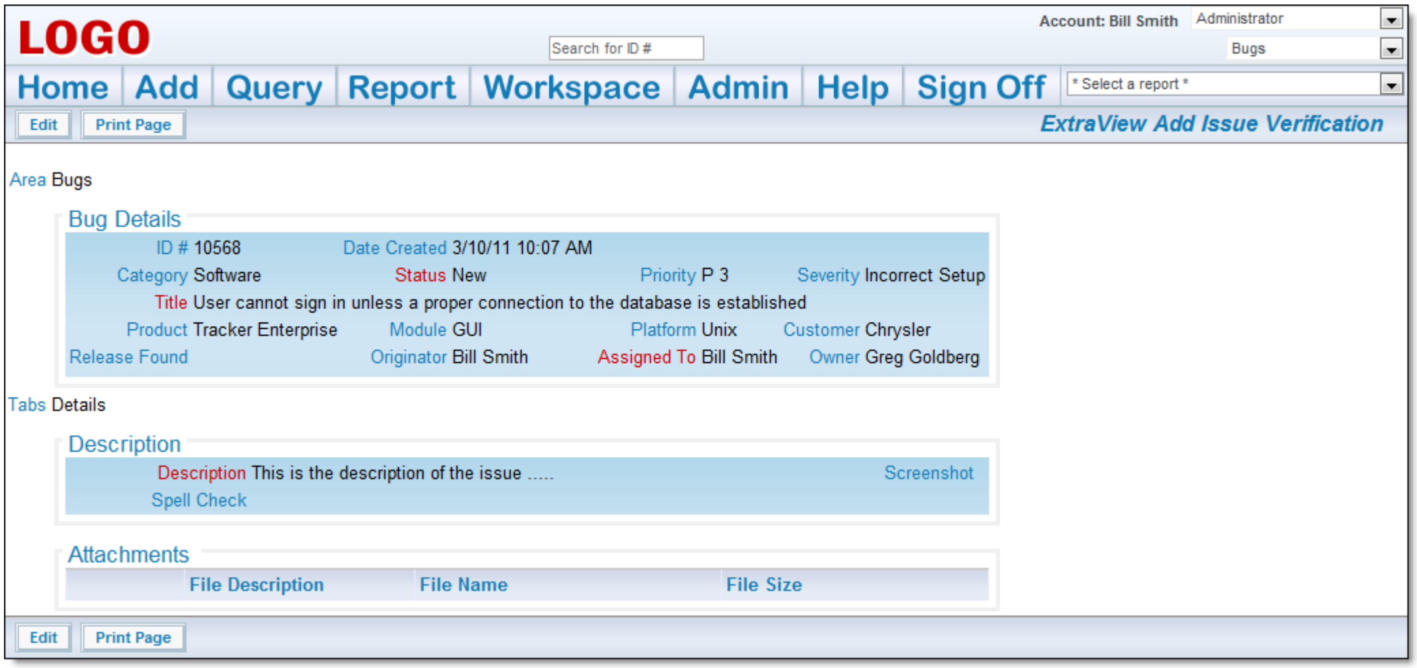
The following excerpt from a debug session shows how ExtraView moved through the sequence of drawing the *Add Issue* layout in the browser:

```
SRQ:Parms:p_option = HomeDisplay
SRQ:Parms:p_action = doDisplay
SRQ:
SRQ:Parms:p_option = security.LoginDisplay
SRQ:Parms:p_action = getMenu
SRQ:
SRQ:Parms:p_option = Display
SRQ:Parms:p_action = doAddDisplay
SRQ:
SRQ:Parms:p_email_customer = unchecked
SRQ:Parms:p_area = 5
SRQ:Parms:p_generate_email = unchecked
SRQ:Parms:p_cc_email_EVDISPLAY =
SRQ:Parms:p_customer_company = 1456
SRQ:Parms:p_category = SOFTWARE
SRQ:Parms:p_date_created =
SRQ:Parms:p_cc_email =
SRQ:Parms:p_originator = BSMITH
SRQ:Parms:p_comments =
SRQ:Parms:p_option = Display
SRQ:Parms:p_short_descr = User cannot sign in unless a proper connection to the
database is established
SRQ:Parms:finished = drawingPage
SRQ:Parms:p_date_open =
SRQ:Parms:p_project = %25SMHww
SRQ:Parms:p_module_id = 5326
SRQ:Parms:p_assigned_to = BSMITH
```

```
SRQ:Parms:p_owner = G.GOLDBERG
SRQ:Parms:p_platform = 643
SRQ:Parms:p_release_found = {NULL}
SRQ:Parms:p_description = This is the description of the issue ....
SRQ:Parms:p_product_name = TRACKER_ENT
SRQ:Parms:p_status = NEW
SRQ:Parms:p_screenshot = {NULL}
SRQ:Parms:p_action = doAddDisplay
SRQ:Parms:stateVar = insert
SRQ:Parms:p_from_option = Display
SRQ:Parms:p_priority = P3
SRQ:Parms:p_from_action = doAddConfirmDisplay
SRQ:Parms:p_timestamp =
SRQ:
SRQ:Parms:p_option = Display
SRQ:Parms:p_action = doAddConfirmDisplay
```

Note that parameters with a null value were passed with a value of {NULL}. Also notice that list values are passed with the ID value of the list, not by using the title. This is because lists may contain duplicate titles, but the list ID's are unique. However, do not rely on using list ID values within your user custom program as these will change if you export the system metadata and import this into another instances. You should use code in your user custom program to extract the list ID's from the database in the constructor of your user custom class.

Once the issue is added to the database, a verification screen is displayed, which will look similar to the following:



Area Bugs

Bug Details

ID # 10568 Date Created 3/10/11 10:07 AM
Category Software Status New Priority P 3 Severity Incorrect Setup
Title User cannot sign in unless a proper connection to the database is established
Product Tracker Enterprise Module GUI Platform Unix Customer Chrysler
Release Found Originator Bill Smith Assigned To Bill Smith Owner Greg Goldberg

Tabs Details

Description

Description This is the description of the issue ..... Screenshot
Spell Check

Attachments

File Description	File Name	File Size

Verification screen after adding a new issue

ExtraView's standard functionality after submitting a new issue is to display the verification screen. You may control this functionality in user custom code, by altering the value of the form parameter named ADD_CONFIRM_PAGE. If you set this to a value of NO, the verification screen is bypassed. You should set this value in the user custom exit named preAddDisplay.

To understand this in more detail, let’s look at one individual parameter, and follow its trail. Let’s analyze the parameter named p_short_descr which maps to the field labeled Title, and with a value of **User cannot sign in unless a proper connection to the database is established** on the *Add Issue* verification screen.

- The parameter’s name is p_short_descr

- The parameter's value is **User cannot sign in unless a proper connection to the database is established**
- The parameter corresponds to the Title field on the New Issue Summary screen
- The name in the data dictionary is SHORT_DESCR
- The display title of the field in the data dictionary is Title
- The value of the field on the *Add Issue* Summary screen is **User cannot sign in unless a proper connection to the database is established**

Parameter Naming Convention

- All parameter names start with **p_**
- The part of the name following the **p_** must be a valid database field name. For example, p_short_descr is the name of the parameter that refers to the data dictionary field named short_descr

Sample Class File

This is a very simple example of a user custom class file. It is not a real-world example, but intended to show the basic structure of a user custom class file.

```
package com.extraview.usercustom;

import java.util.ArrayList;
import com.extraview.applogic.problem.Problem;
import com.extraview.presentation.ProblemFormParam;
import com.extraview.util.SesameSession;
import com.extraview.util.TextManager;

public class MyCompany
    extends UserCustom {

  public String ucEditPreUpdate(
      Problem pOldProblem,      ArrayList pOldReleases,
      ArrayList pOldModules,     ArrayList pOldUdfs,
      Problem pNewProblem,      ArrayList pNewReleases,
      ArrayList pNewModules,    ArrayList pNewUdfs,
      SesameSession session,    ProblemFormParam pfp) {
    // this sample implements a new business rule that ensures the user
    // enters text into the COMMENTS field when altering the status of
    // an issue on the edit screen

    String errText = null;
    if (TextManager.isStringInvisible( (String) pfp.get("COMMENTS"))) {
      if (pNewProblem.getStatus() != null
       && (pOldProblem.getStatus() == null
       || !pOldProblem.getStatus().equals( pNewProblem.getStatus()))) {
        return "Comments are required when the status changes.";
      }
    }
    return null;
  }
}
```

User Custom Methods

- Add new comment

User custom methods/exits are the key ExtraView feature that allows programmatic modification and extension of ExtraView's features, capabilities, and behavior. User Custom exits are methods that are defined in the Java class com.extraview.usercustom.UserCustom and they are stubs which mean that they have empty code bodies and they are designed to be overridden. ExtraView calls User Custom methods at specific points when presenting screens and updating the database. The User Custom methods are permanently registered callback methods and only ExtraView calls them. These callback methods are also referred to as User Custom exits because they are particular points in the ExtraView's code where it exits/calls other custom/specialized code that a developer can provide. Returning from a user custom method returns to the caller, i.e. ExtraView. You add code to a use custom method by overriding it in your own user custom class. User custom methods have access to the ExtraView environment through –

- Passed parameters (e.g. ProblemFormParam)
- Global objects (e.g., Z)
- Support classes (e.g. TextManager).

The following ExtraView screens have user custom exits -

- User Sign On
- Home
- Add Issue
- Edit Issue
- Detailed Report
- Search / Report
- Quicklist
- UDF List Maintenance
- Report Group Functions
- User Account Maintenance

Getting a list of user custom methods called by each screen

This works with a Linux / Unix / Solaris operating system command only. First. turn on user custom metrics using the ExtraView administration behavior setting named USER_CUSTOM_ENABLE_METRICS, then open a command window and execute a command like -

```
tail -f evj.log | grep UserCustomMetrics
```

Navigate to the various ExtraView screens and observe the output in the command window. You will see a list of each method called in the output from the tail command. Note the class names that indicate the place in ExtraView from which the log entry originated.

addCustomButtonTemplate

Purpose

This method defines a custom template for the rendering of the buttons on the menubar on *Add* and *Edit* screens.

Applies To

Add Issue, Add Confirmation, Edit Issue screens

Signature

```
public LinkedHashMap addCustomButtonTemplate(
    Connection dbconn,// the database connection
    SesameSession session,  // the user's session
    String layoutType,  // the type of layout, e.g. ADD_PROBLEM
    int areaId,  // the ID of the area
    int projectId )  // the ID of the project
,>
```

Notes

This is used in conjunction with the methods setButtonDopeMap and buttonInclusionMap. The returned LinkedHashMap uses custom button names as the keys - the custom button names must not match any built-in button names. There is one entry for each button, in the order that the buttons should appear, subject to the position value. The Map entry value for each button name contains the entries for the button attributes. For each button, the following must be defined in the map:

- HTML Template (key “HTMLTemplate”) – static HTML and replaceable variables
- Position (key "position") – integer (0..n) specifying where in the existing buttons the new button should appear

The built in button names are:

- CLONEBTN
- DELETEBTN
- EDITBTN
- EMAILBTN
- HISTORYBTN
- INTERESTLISTTOGGLEBTN
- RELGRPADMINBTN
- RETURNBTN
- UPDATEBTN
- UPDATECONTINUEBTN
- UPDATENEXTBTN
- UPDATEPREVBTN

buttonInclusionMap

Purpose

This method defines a map that defines whether or not to include a menubar button for each of the named buttons.

Applies To

Add Issue, Add Confirmation, Edit Issue screens

Signature

```
public HashMap buttonInclusionMap(
                     Connection dbconn, // the database connection
                     SesameSession session, // the user's session
                     String layoutType, // the type of layout, e.g. ADD_PROBLEM
                     String displayMode, // the current display mode, e.g. ADD
```

```
                              HashMap selectedVals, // the current form values
                              HashSet buttonNames ) // the names used as keys for the HashMap ,>
```

Notes

This is used in conjunction with the methods setButtonDopeMap and addCustomButtonTemplate. The method returns a HashMap with the desired list in the form buttonName --> true (for inclusion) or false (for exclusion). The button names can be:

- CLONEBTN
- DELETEBTN
- EDITBTN
- EMAILBTN
- HISTORYBTN
- INTERESTLISTTOGGLEBTN
- RELGRPADMINBTN
- RETURNBTN
- UPDATEBTN
- UPDATECONTINUEBTN
- UPDATENEXTBTN
- UPDATEPREVBTN

ldapIdSearch

- Add new comment

Purpose

Given a userId, this method will return the name/value HashMap containing the user profile information.

Applies To

LDAP

Signature

```
public HashMap ldapIdSearch(String userId)
```

Notes

This single HashMap should be in the same format as the name/value HashMaps returned by ldapSearch(). This will only be called if ldapOverride() returns true.

Example

This shows how when provided with a userId, information about that user is returned such as their name, company information, and any pre-defined user defined fields. Please note that this search stops when 1 match is found as there will only be 1 match per unique userID.

```
public HashMap ldapIdSearch (String userId) {
    HashMap nameVals1 = new HashMap();
    DirContext ctx = null;
    NamingEnumeration results = null;

    try {
        String filter = "(" + Z.lu.LDAP_PRIMARYKEY + "=" + userId + ")";

        String[] upsertFields = {Z.lu.LDAP_COMMONNAME,
                                 Z.lu.LDAP_GIVENNAME,
                                 Z.lu.LDAP_SURNAME,
                                 Z.lu.LDAP_EMAIL,
                                 Z.lu.LDAP_STREET,
                                 Z.lu.LDAP_PHONE,
                                 Z.lu.LDAP_POSTALCODE,
                                 Z.lu.LDAP_STATE,
                                 Z.lu.LDAP_CITY,
                                 Z.lu.LDAP_COUNTRY,
                                 Z.lu.LDAP_FAX,
                                 Z.lu.LDAP_TITLE};

        // Specify the scope of the search
        // subtree: starts at the base entry and searches
        // everything below it, including the base entry
        SearchControls constraints = new SearchControls();
        constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);

        // Perform the actual search
        String dn = Z.lu.getInitMgrDn();
        String password = Z.lu.getInitPswd();

        ctx = this.getContext(dn, password);
        results = ctx.search(Z.lu.getInitSearchBase(), filter, constraints);

        // Now Stepping through the search results
        int count = 0;

        while (results != null && results.hasMore() && count  0) {
            tmp.append(", ");
        }
        tmp.append(((String) nEnum.next()).trim());
        if (TextManager.isStringInvisible(tmp.toString()) ) {
            tmp = new StringBuffer(LdapUtil.NA);
        }
        nameVals1.put(upsertFields[i], tmp.toString());
        count++;
        // If the count was 0 then no results were returned.
        // Send back a null HashMap otherwise Search ldap will assume
        // a user was found and attempt to insert a null record.
        if (count == 0) { nameVals1 = null; }
    } catch(Exception e) {
      Z.log.writeToLog(Z.log.ERROR, "UC: ldapIdSearch Exception: " + e);
      ErrorWriter.write(e, ErrorWriter.LOGERR);
    } finally {
        try {
            if (results != null) { results.close(); }
            if (ctx != null) { ctx.close(); }
        } catch (Exception e) {
            Z.log.writeToLog(Z.log.ERROR, "UC: ldapIdSearch Exception: " +
                "Could not close context or results - " + e);
            ErrorWriter.write(e, ErrorWriter.LOGERR);
        }
    }
    return nameVals1;
}
```

ldapOverride

- Add new comment

Purpose

This permits the uc to override the usual search operations in LDAP by returning a true value to this method. A return of true implies that the ldapSearch and ldapIdSearch operations are implemented inside uc.

Applies To

LDAP

Signature

```
public boolean ldapOverride()
```

Notes

Example

```
public boolean ldapOverride() {
    // Changed to true to imply that the ldapSearch and ldapIdSearch
    // are implemented in User Custom code
    return true;
}
```

ldapSearch

- Add new comment

Purpose

This method, when given a firstName, lastName, and userId, will search for and return user matches in place of a normal LDAP search.

Applies To

LDAP

Signature

```
public TreeMap ldapSearch(String firstName,
                               String lastName,
                               String userId,
                               boolean exactMatch,
                               String sortBy)
```

Notes

The exactMatch specifies if the match is exact, the sortBy can be by "last", "first", "userId" or "" signifying how the returned TreeMap should be sorted. The values in the returned TreeMap must be HashMap's with the name/value pairs used by ldap. The names can be retrived from the Configuration.properties. The names may include wildcards (*), and this method is only called if ldapOverrive() returns true.

Example

```
public TreeMap ldapSearch (String firstName,
                           String lastName,
                           String userId,
                           boolean exactMatch,
                           String sortBy)
{
    TreeMap sortedResults = new TreeMap();
    boolean extraFilter = false;
    String extraFilterString = "";
    ArrayList searchAttrs = new ArrayList();
    ArrayList searchVals = new ArrayList();

    if (TextManager.isStringVisible(firstName)) {
        searchAttrs.add(Z.lu.LDAP_GIVENNAME);
        searchVals.add(firstName);
    }
    if (TextManager.isStringVisible(lastName)) {
        searchAttrs.add(Z.lu.LDAP_SURNAME);
        searchVals.add(lastName);
    }
    if (TextManager.isStringVisible(userId)) {
        searchAttrs.add(Z.lu.LDAP_PRIMARYKEY);
        searchVals.add(userId);
    }
    if (searchAttrs.size() == 0) {
        // We should never fall into this else but if we do here is an alert
        Z.log.writeToLog(Z.log.ERROR, "UC: ldapSearch: " +
            "Cannot seach LDAP with null information");
        return sortedResults;
    }

    extraFilterString = Z.config.getConfigValue("LDAP_SEARCH_FILTER");

    if (TextManager.isStringVisible(extraFilterString)) {
        extraFilterString.trim();
        extraFilter = true;
    }

    // Setting up the filter
    StringBuffer filter = new StringBuffer();

    if (searchAttrs.size() == 1 && ! extraFilter)
    {
        // a = b
        filter.append("(");
        filter.append((String)searchAttrs.get(0));
        filter.append("=");
        filter.append((String)searchVals.get(0));

        if (! exactMatch) {
            filter.append("*");
        }

        filter.append(")");
    }
    else
```

```
        {
            // (&(a=b)(c=d)(e=f))
            filter.append("(&");

            if (extraFilter) {
                filter.append(extraFilterString);
            }

            for (int x = 0; x < searchAttrs.size(); x++)
            {
                filter.append("(");
                filter.append((String)searchAttrs.get(x));
                filter.append("=");
                filter.append((String)searchVals.get(x));

                if (!exactMatch) {
                    filter.append("*");
                }

                filter.append(")");
            }

            filter.append(")");
        }

        // Specify the scope of the search
        // subtree: starts at the base entry and searches
        // everything below it, including the base entry
        SearchControls constraints = new SearchControls();
        constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);

        try
        {
            String dn = Z.lu.getInitMgrDn();
            String password = Z.lu.getInitPswd();

            DirContext ctx = null;
            ctx = this.getContext(dn, password);

            // Perform the actual search
            // We pass the searchbase, filters and constraints
            // containing the scope of the search
            NamingEnumeration results = ctx.search(
                Z.lu.getInitSearchBase(), filter.toString(), constraints
            );

            // default sort is by lastname
            String sortKey1 = Z.lu.LDAP_SURNAME;
            String sortKey2 = Z.lu.LDAP_GIVENNAME;
            String sortKey3 = Z.lu.LDAP_PRIMARYKEY;

            if ("first".equalsIgnoreCase(sortBy)) {
                sortKey1 = Z.lu.LDAP_GIVENNAME;
                sortKey2 = Z.lu.LDAP_SURNAME;
                sortKey3 = Z.lu.LDAP_PRIMARYKEY;
            }
            else if ("id".equalsIgnoreCase(sortBy)) {
                sortKey1 = Z.lu.LDAP_PRIMARYKEY;
                sortKey2 = Z.lu.LDAP_SURNAME;
                sortKey3 = Z.lu.LDAP_GIVENNAME;
            }

            HashMap nameVals = null;

            // Now Stepping through the search results
            while (results.hasMoreElements())
            {
                nameVals = new HashMap();
                SearchResult sr = (SearchResult) results.next();
                Attributes attrs = sr.getAttributes();
                HashMap hm = Z.lu.getLdapFields();
                Iterator it = hm.keySet().iterator();

                while (it.hasNext())
                {
                    String key = (String) hm.get((String)it.next());

                    Attribute attr = attrs.get(key);
                    StringBuffer tmp = new StringBuffer();

                    if (attr != null)
                    {
                        NamingEnumeration nEnum = attr.getAll();
                        int cnt = 0;
                        tmp.setLength(0);

                        while (nEnum.hasMoreElements()) {
                            if (cnt++ > 0) { tmp.append(", "); }
                            tmp.append(((String) nEnum.next()).trim());
                        }
                    }

                    if (TextManager.isStringInvisible(tmp.toString()) ) {
                        tmp = new StringBuffer(LdapUtil.NA);
                    }
                    if (Z.lu.LDAP_PRIMARYKEY.equals(key)) {
                        nameVals.put(key,tmp.toString().toUpperCase());
                    } else {
                        nameVals.put(key,tmp.toString());
                    }
                }

                sortedResults.put(TextManager.catStrings((String)nameVals.get(sortKey1),
                    " ", (String)nameVals.get(sortKey2),
                    " ", (String)nameVals.get(sortKey3)), nameVals
                );
            }

            results.close();
            ctx.close();

        } catch (javax.naming.PartialResultException pre) {
            // Don't throw and exception here...this is for MS Active Server stuff
            // throwing this exception probably because of a referral problem...
            // but in any case the results are OK according to the data retrieved
            Z.log.writeToLog(Z.log.WARN, "Partial Result Exception: " + pre.toString());

        } catch (Exception e) {
            Z.log.writeToLog(Z.log.ERROR, "UC: ldapSearch Exception: " + e);
            ErrorWriter.write(e, ErrorWriter.LOGERR);
        }

        return sortedResults;
    }
```

massUpdateSelectList

Purpose

This method filters the fields in the select list presented to the user for mass update, in the selection of which fields to update.

Applies To

Mass Update screen

Signature

```
    public TreeMap massUpdateSelectList(
      Connection dbconn,      // database connection
      SesameSession session,  // the session
      TreeMap list,           // the values, in the form title --> ddname
      boolean clone);         // if true, this is a "clone" operation, not a mass update
,>,>
```

Notes

This method returns a Treemap with the desired list in the form title --> ddname.

scanUploadFile

Purpose

This method allows user custom code to scan an incoming attachment file for potentially dangerous characters and reject or allow the file to be inserted as an attachment to the database. For example, you might want to reject files that are considered dangerous due to cross-site scripting or XSS exploits. Many installations require files that contain these characters to be uploaded normally; some organizations have a policy not to upload files that contain these characters.

Signature

```
public String scanUploadFile( File attachmentFile,
                              SesameSession session,
                              Connection dbconn,
                              String fileName,
                              String fileCharset,
                              String contentType) throws Exception
```

Notes

The return value from the method should be either a null or a blank string to accept the attachment. If you want the method to reject the attachment, the return should be a message that will be displayed to the end user providing an explanation that their file is not being uploaded.

The specific algorithm written in user custom code for a specific customer is as follows:

1. Detect if file extension is in whitelist of allowed file extensions. If not, reject the attachment.

2. If **text**, **html**, or **js** MIME type, as determined by the stated file type, try to convert to character stream using specified charset; if conversion works, use the Java built-in class named **Scanner** to find the script tag

3. If not **text**, **html** or **js**, get the contained text and JavaScript based on document type, for example, Excel spreadsheet; then employ existing full text search means (e.g., using POI) - use Scanner to find script tag in the text

4. If no match so far, the extension may be incorrect, so assume that it is a binary, unrecognized MIME type file; search bytes for pattern using KMPMatch (a java class that matches strings inside binary files) with extensions for upper/lower case. The search will include two patterns - the **<script** pattern without binary zeroes and the **<ZsZcZrZiZpZt** pattern, where **Z** == binary zero. The latter pattern is used to identify **script** in text that is encoded with UCS-2 or UTF-16 (double byte character encoding)

5. Conversion of data through different transfer-encodings: this is unnecessary because ExtraView does not use other transfer-encodings to send an attachment to the browser.There is no use case where ExtraView would send a base64-transfer-encoded document to the browser as an attachment, for example. The transfer-encoding is not an attribute of the attribute file; it is specified in the header of the message sent to the browser or received from the browser.)

KMPMatch Method

```
/**
 * Knuth-Morris-Pratt Algorithm for Pattern Matching
 */
class KMPMatch {
    /**
     * Finds the first occurrence of the pattern in the text.
     */
    public int indexOf(byte[] data, byte[] pattern) {
        int[] failure = computeFailure(pattern);

        int j = 0;
        if (data.length == 0) return -1;

        for (int i = 0; i < data.length; i++) {
            while (j > 0 && pattern[j] != data[i]) {
                j = failure[j - 1];
            }
            if (pattern[j] == data[i]) { j++; }
            if (j == pattern.length) {
                return i - pattern.length + 1;
            }
        }
        return -1;
    }

    /**
     * Computes the failure function using a boot-strapping process,
     * where the pattern is matched against itself.
     */
    private int[] computeFailure(byte[] pattern) {
        int[] failure = new int[pattern.length];

        int j = 0;
        for (int i = 1; i < pattern.length; i++) {
            while (j > 0 && pattern[j] != pattern[i]) {
                j = failure[j - 1];
            }
            if (pattern[j] == pattern[i]) {
                j++;
            }
            failure[i] = j;
        }
```

setButtonDopeMap

Purpose

This method defines a map that overrides the rendering of the buttons on the menubar on *Add* and *Edit* screens.

Applies To

Add Issue, Add Confirmation, Edit Issue screens

Signature

```
public HashMap setButtonDopeMap(
    Connection dbconn,        // the database connection
    SesameSession session,    // the user's session
    String buttonName,        // the name of the button
    String layoutType,        // the type of layout, e.g. ADD_PROBLEM
    HashMap buttonDopeMap )   // the current values to be used for rendering the button
                              // this includes:value, title, onclick, class,
                              // and options using those keys
```

Notes

This is used in conjunction with the methods addCustomButtonTemplate and buttonInclusionMap. The method returns a HashMap with the values corresponding to the buttonDopeMap keys to be used in rendering the button. The values in the HashMap can be modified to change the value, title, onclick, class and/or options. The button names can be:

- CLONEBTN
- DELETEBTN
- EDITBTN
- EMAILBTN
- HISTORYBTN
- INTERESTLISTTOGGLEBTN
- RELGRPADMINBTN
- RETURNBTN
- UPDATEBTN
- UPDATECONTINUEBTN
- UPDATENEXTBTN
- UPDATEPREVBTN

ucAddInit

- Add new comment

Purpose

ucAddInit is called as the Add routine is first entered. This method can be used to provide processing before the *add issue* screen is launched.

Applies To

Add Issue screen

Signature

```
 public void ucAddInit(SesameSession session,  // contains values of form elements
                       HashMap values)         // selected Vals hashmap
```

Notes

The business rule directive <== load ==> in conjunction with the SCREEN_NAME equaling ADD provides a similar opportunity to influence the *add* screen before it is loaded.

Example

<== load ==> if (SCREEN_NAME = 'ADD') { FIELD_NAME = VALUE; } This example shows how to set the values of data on the form after reading data from the both the current user and that user's manager's account. This will happen before the *add issue* screen is entered, thus pre-populating the "Manager's Work Phone" and "Manager's Cell Phone" fields, depending upon the user accessing the screen.

```
 public void ucAddInit(SesameSession session,  // contains values of form elements
                       HashMap values)         // selected Vals hashmap
{
    // Call Business Rules
    super.ucAddInit(session, values);

    // Get area & project
    String areaId = (String) values.get("AREA");
    String projectId = (String) values.get("PROJECT");
    projectId = projectId.substring(projectId.indexOf("|") + 1, projectId.length());
    Z.log.writeToLog(Z.log.WARN, "UC: ucAddInit - areaId=" + areaId +
        ", projectId=" + projectId);

    // Check for specific area & project
    if (A_CUSTOMER_ISSUES.equals(areaId) && P_CUSTOMER_ISSUES_DATA.equals(projectId) )
    {
        Connection dbConn = null;
        SecurityUser originator = null;
        SecurityUser manager = null;
        String wPhone = null;
        String cPhone = null;

        try
        {
            // Get database connection
            dbConn = Z.pool.getConnection("ucAddInit");

            // Lookup originator's manager User ID,
            // which is stored in the account USER_DEFINED_1 field
            originator = SecurityUser.getReference(dbConn, session.getUserId());
            manager = SecurityUser.getReference(dbConn, originator.getUserField1());
            // If manager's account found, get work and cell phone info
            if (manager != null) {
                wPhone = manager.getWorkTelephone();
                cPhone = manager.getCellPhone();
            }
            Z.log.writeToLog(Z.log.WARN, "UC: ucAddInit - wPhone=" + wPhone +
                ", cPhone=" + cPhone);

            // Set form data
            values.put("MANAGER_WORK_PHONE", wPhone);
            values.put("MANAGER_CELL_PHONE", cPhone);

        } catch (Exception e) {
            Z.log.writeToLog(Z.log.ERROR, "UC: ucAddInit Exception: " + e);
            ErrorWriter.write(e, ErrorWriter.LOGERR);

        } finally {
            // Close the database connection
            if (dbConn != null) { Z.pool.close(dbConn); }
        }
    }
}
```

ucAddPostInsert

- Add new comment

Purpose

This allows additional error reporting by returning an error message.

Applies To

Add Issue screen

Signature

```
public void ucAddPostInsert (
      Problem pProblem,         // the problem or case
      ArrayList pReleases,      // the list of releases
      ArrayList pModules,       // the list of modules
      ArrayList pUdfs,          // the list of UDF's
      SesameSession session)    // the current sesame Session
```

Notes

This is called after a record has been successfully added to the database.

Example

This example shows how after the record has been added, information pertaining to the record is read and an update is in turn made to another record. Technically, this same example could be achieved in Business Rules within the <== postupdate ==> directive, in conjunction with a <== link ==> rule.

```
public void ucAddPostInsert (
        Problem pProblem,    // the problem or case
        ArrayList pReleases,    // the list of releases
        ArrayList pModules,    // the list of modules
        ArrayList pUdfs,    // the list of UDF's
        SesameSession session)    // the current sesame Session
{
    // Call Business Rules
    super.ucAddPostInsert(pProblem, pReleases, pModules, pUdfs, session);

    String status = pProblem.getStatus();
    String projectId = pProblem.getProjctId();
    Z.log.writeToLog(Z.log.WARN, "UC: ucAddPostInsert - projectId=" + projectId);

    // PROJ__BUGS_DATA is a static String object,
    // populated in the class constructor and if status closed
    if (P_BUGS_DATA.equals(projectId) && "OPEN".equals(status) )
    {
        Connection dbConn = null;
        String childId = null;
        HashMap fp = new FormParameters();

        try
        {
            // Get database connection
            dbConn = Z.pool.getConnection("ucAddPostInsert");

            // Iterate through udf's to find special child id
            Iterator itr = pUdfs.iterator();
            while (itr.hasNext())
            {
                Problem_UDF udf = (Problem_UDF) itr.next();

                if ("SPECIAL_CHILD_ID".equals(udf.getName()) )
                {
                childId = udf.getValue();
                break;
                }
            }

            // Populate form parameter object as needed
            fp.put("STATUS", "DUPLICATE");
            fp.put("COMMENTS", "This record was marked as Duplicate because it was " +
            "the 'Special Child' of a new Bug record: " +
                pProblem.getId() + " that was Opened.");

            // If child records not null then update using another User Custom method
            if (childId != null) {
                apiEditItem(fp, dbConn, session, childId);
            }

        } catch (Exception e) {
            Z.log.writeToLog(Z.log.ERROR, "UC: ucAddPostInsert Exception: " + e);
            ErrorWriter.write(e, ErrorWriter.LOGERR);

        } finally {
            // Close the database connection
            if (dbConn != null) { Z.pool.close(dbConn); }
        }
    }
}
```

ucAddPreDisplay

- Add new comment

Purpose

This method can be used to determine if the add screen should be displayed.

Applies To

Add Issue screen

Signature

```
public boolean ucAddPreDisplay (
        ProblemFormParam values,            // contains values of form params
        HttpServletRequest request,         // the servlet request
        HttpServletResponse response,       // the servlet response
        Connection conn,                    // the database connection
        SesameSession session)              // the session object
```

Notes

This method returns a Boolean value to ExtraView. Database objects are sustained at this point. Changes made to the pfp will not affect them at this point. Updates dependent upon the insert should be done in ucAddPostInsert.

Example

This example returns false, thus preventing the display of the add screen, if the user is in the "GUEST" User Role and the user's ID is either "EXTRNL_USER" or "BILL.SMITH".

```
public boolean ucAddPreDisplay (
        ProblemFormParam values,            // contains values of form params
        HttpServletRequest request,         // the servlet request
        HttpServletResponse response,       // the servlet response
        Connection conn,                    // the database connection
        SesameSession session)              // the session object
{
    boolean displayAdd = true;
    String userId = null;
    String userRole = null;

    // Lookup originator's user ID and User Role,
    // to prevent Add screen loading for some users when in a certain role
    userId = session.getUserId();
    userRole = session.getUserRole();

    if ("GUEST".equals(userRole) && userId != null &&
        ("EXTRNL_USER".equals(userId) || "BILL.SMITH".equals(userId))) {
        displayAdd = false;
    }

    Z.log.writeToLog(Z.log.WARN, "UC: ucAddPreDisplay - displayAdd=" + displayAdd);
    return displayAdd;
}
```

ucAddPreInsert

- Add new comment

Purpose

This allows additional error reporting by returning an error message.

Applies To

Add Issue screen

Signature

```
public String ucAddPreInsert (
       Problem pProblem,        // the problem or case
       ArrayList pReleases,     // the list of releases
       ArrayList pModules,      // the list of modules
       ArrayList pUdfs,         // the list of UDF's
       SesameSession session,   // current session
       ProblemFormParam pfp)    // values of from the form
```

Notes

This is called during the Add process before any updates are performed, and allows a final manipulation of parameters prior to attempting the add operation. There is a point to explain about the form parameters which are passed into this method, and that you can manipulate these parameters to stop the display of the new issue verification screen. Add the following line into this method:

```
values.put("ADD_CONFIRM_PAGE", "NO");
```

This will cause the new issue verification screen to be suppressed.

Example

This example shows how ucAddPreInsert is called before any updates to a record are made. Based upon the customer requirements, if the record is in the "Test Case" or "Test Plan" area and project, the issue is not saved upon the add operation, and an error message is displayed.

```
public String ucAddPreInsert (
       Problem pProblem,        // the problem or case
       ArrayList pReleases,     // the list of releases
       ArrayList pModules,      // the list of modules
       ArrayList pUdfs,         // the list of UDF's
       SesameSession session,   // current session
       ProblemFormParam pfp)     // values of from the form
{
    String errText = rules.ucAddPreInsert(item,
                                          releases,
                                          modules,
                                          udfs,
                                          session,
                                          values);
    if (errText != null) { return errText; }

    // If we are in the test case area or test plan project,
    //  do not allow issue to be saved
    if (A_TEST_CASES.equals(values.get("AREA")) && (A_TEST_CASES + "|" +
            A_TEST_PLAN_PROJECT_ID).equals(values.get("PROJECT")))
    {
        return Z.m.msg(session, "You cannot save a test plan as an issue. " +
            "Use this screen to generate a test plan from test cases you select.");
    }

    return null;
}
```

ucAddRefresh

- Add new comment

Purpose

This is called by ExtraView during each Add Screen refresh.

Applies To

Add Issue screen

Signature

```
public String ucAddRefresh (
      SesameSession session,  // current session
      ProblemFormParam pfp)   // contains values of form params
```

Notes

This method typically clears dependent values, performs lookups, and sets values from ExtraView or external sources.

Example

This example demonstrates how when the "System" (PROJECT_SYSTEM_NAME_RR) Repeating Row field is changed, we reset the "Primary Row" (PROJECT_SYSTEM_NAME_PRIMARY_RR) field value to the value of Yes" in only the first row, while setting all of the other values to null.

```
public String ucAddRefresh (
      SesameSession session,  // current session
      ProblemFormParam pfp)   // contains values of form params
{
    String areaId = (String)values.get("AREA");
    String projectId = (String)values.get("PROJECT");
    String whatChanged = whatChanged(values);

    // Write WARN message in log
    w("whatChanged: " + whatChanged);

    // Set Primary to first of Repeating Rows on Change Request,
    // when "System" field changed on a Row
    if ("P_PROJECT_SYSTEM_NAME_RR".equalsIgnoreCase(whatChanged) &&
        (A_CR_ID.equalsIgnoreCase(areaId) && P_CR_ID.equalsIgnoreCase(projectId)) )
    {
        String[] primaryRowsA = values.getArray("PROJECT_SYSTEM_NAME_PRIMARY_RR");

        // 12845 is the udfListId for the value of "Y" in this instance
        String[] primaryS = new String[primaryRowsA.length];
        for (int i=0; i < primaryS.length; i++)
        {
            if (i == 0) {
                primaryS[i] = "12845";
            } else {
                primaryS[i] = "{NULL}";
            }
        }
        values.put("PROJECT_SYSTEM_NAME_PRIMARY_RR", primaryS);
    }
}
```

ucAddSubmit

Purpose

This method is called after the form is submitted, during the Add process and before the umbrella of problem objects is created. It allows access to the parameters before they have been manipulated.

Applies To

Add Issue screen

Signature

```
public String ucAddSubmit (
      SesameSession session,      // current session
      ProblemFormParam values)    // contains values of form params
```

Notes

Any string returned will cause the process to stop and the value of that string will be displayed to the user in the form of a JavaScript alert box.

Example

This example checks to see if the 'Password' field value meets the configured password requirements. If it does not, then a JavaScript alert box will inform the user that there is an error, and the 'Password' and 'Verify Password' field values will be cleared.

```
public String ucAddSubmit (
      SesameSession session,      // current session
      ProblemFormParam values)    // contains values of form params
{
    String areaId = (String) values.get("AREA");
    String projId = (String) values.get("PROJECT");
    Z.log.writeToLog(Z.log.DEBUG, "UC: ucAddSubmit - areaId=" + areaId +
        ", projId=" + projId);

    // Call Business Rules
    String errMsg = super.ucAddSubmit(session, values);
    if (errMsg != null) {
        return errMsg;
    }

    if (AREA__CUSTOMER_ISSUES.equals(areaId) )
    {
        boolean validPassword = false;
        String userPwd = null;
        String userId = session.getUserId();

        userPwd = (String) values.get("CONTACT_PASSWORD");
        if (userPwd != null) { userPwd = userPwd.trim(); }

        validPassword = UserAccountUtils.isValidPassword(userPwd, session, userId);

        if (validPassword == false) {
                errMsg = "ERROR: The 'Password' does not meet the " +
                    "minimum requirements established by your System Administrator.";
        }

        // Clear out the 'Password' & 'Verify Password' field values,
        // and return error if password was invalid
        if (errMsg != null)
        {
            Iterator iter = values.entrySet().iterator();
            while (iter.hasNext()) {
                String key = (String) iter.next().toString();
                if (key.equals("CONTACT_PASSWORD") ||
                        key.equals("CONTACT_PASSWORD_VERIFY") ) {
                    values.remove(key);
                }
            }

            return errMsg;
        }
    }

    return null;
}
```

ucAdjustWorkdays

- Add new comment

Purpose

This is used to allow special date arithmetic. A true value is returned if a change has been made.

Applies To

Add Issue screen

Signature

```
 public boolean ucAdjustWorkdays (int action, DateCalculator dc)
```

Notes

Example

```
// Setup global vars used by ucAdjustWorkdays
```

```
Calendar [] holidays = null;
static final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;

// Setup methods used by ucAdjustWorkdays

int myCompare (Calendar c1, Calendar c2) {
    // Note: This looks like it's trying to get the same day
    // by dividing by MILLIS_PER_DAY,
    // but that can cross date boundaries;
    // probably due to the TZ offset, don't want to change this
    // at the moment as it seeems to wrork for what is needed.
    long t1 = c1.getTimeInMillis() / MILLIS_PER_DAY;
    long t2 = c2.getTimeInMillis() / MILLIS_PER_DAY;
    if (t1 > t2)      { return 1;  }
    else if (t1 < t2) { return -1; }

    return 0;
}


void backupOneBusinessDay (Calendar st, Calendar start)
{
    st.add(Calendar.DATE, -1);
    start.add(Calendar.DATE, -1);

    if (st.get(Calendar.DAY_OF_WEEK) > 6) {    // saturday
      st.add(Calendar.DATE, -1);
      start.add(Calendar.DATE, -1);
    }
    if (st.get(Calendar.DAY_OF_WEEK) == 1) {    // sunday
      st.add(Calendar.DATE, -2);
      start.add(Calendar.DATE, -2);
    }
}


void advanceOneBusinessDay (Calendar st, Calendar start)
{
    st.add(Calendar.DATE, 1);
    start.add(Calendar.DATE, 1);

    if (st.get(Calendar.DAY_OF_WEEK) == 7) {    // saturday
      st.add(Calendar.DATE, 2);
      start.add(Calendar.DATE, 2);
    }
    if (st.get(Calendar.DAY_OF_WEEK) == 1) {    // sunday
      st.add(Calendar.DATE, 1);
      start.add(Calendar.DATE, 1);
    }
}


public void setHolidays (Connection con, SesameSession session)
throws Exception
{
    holidays = new Calendar[9];

    DbTime t =  new DbTime(session, con);
    Calendar c = t.getUserNow() ;
    int i = 0;

    c.set(Calendar.AM_PM, Calendar.PM);
    // New Year’s Day  Friday, January 1, 2010
    c.set(2010, 0, 1, 0, 0,  0);
    c.set(Calendar.MILLISECOND, 0);
```

```
        c.add(Calendar.DATE, 1);
        c.add(Calendar.DATE, -1);
        holidays[i++] = (Calendar) c.clone();

        // President’s Day  Monday, February 15, 2010
        c.set(2010, 1, 15);
        c.add(Calendar.DATE, 1);
        c.add(Calendar.DATE, -1);
        holidays[i++] = (Calendar) c.clone();

        // Memorial Day  Monday, May 31, 2010
        c.set(2010, 4, 31);
        c.add(Calendar.DATE, 1);
        c.add(Calendar.DATE, -1);
        holidays[i++] = (Calendar) c.clone();

        // Independence Day  Monday, July 5, 2010
        c.set(2010, 6,  5);
        c.add(Calendar.DATE, 1);
        c.add(Calendar.DATE, -1);
        holidays[i++] = (Calendar) c.clone();

        // Labor Day  Monday, September 6, 2010
        c.set(2008, 8,  6);
        c.add(Calendar.DATE, 1);
        c.add(Calendar.DATE, -1);
        holidays[i++] = (Calendar) c.clone();

        // Thanksgiving Day  Thursday, November 25, 2010
        c.set(2010,10, 25);
        c.add(Calendar.DATE, 1);
        c.add(Calendar.DATE, -1);
        holidays[i++] = (Calendar) c.clone();

        // Day after Thanksgiving  Friday, November 26, 2010
        c.set(2010,10, 26);
        c.add(Calendar.DATE, 1);
        c.add(Calendar.DATE, -1);
        holidays[i++] = (Calendar) c.clone();

        // Christmas Eve  Friday, December 24, 2010
        c.set(2010,11, 24);
        c.add(Calendar.DATE, 1);
        c.add(Calendar.DATE, -1);
        holidays[i++] = (Calendar) c.clone();

        // Christmas Day Monday, December 27, 2010
        c.set(2010,11, 27);
        c.add(Calendar.DATE, 1);
        c.add(Calendar.DATE, -1);
        holidays[i++] = (Calendar) c.clone();

        // for (i=0; i < holidays.length; i++) w("holiday[" + i + "]: " + dmpD(holidays[i]));
    }


    public boolean ucAdjustWorkdays (int action, DateCalculator dc)
    throws Exception
    {
        boolean changed = false;

        if (holidays == null)
        {
            SesameSession session = SesameSession.getSession();
            Connection con = null;
            try {
                con = Z.pool.getConnection("ucAdjustWorkdays");
                setHolidays (con, session);

            } catch (Exception e) {
                holidays = null;
                Z.log.writeToLog(Z.log.ERROR, "UC: ucAdjustWorkdays Exception: " + e);
                ErrorWriter.write(e, ErrorWriter.LOGERR);

            } finally {
                if (con != null) { Z.pool.close(con); }
            }
        }

        Calendar start = dc.getStartDate();
        Calendar end = dc.getEndDate();
        double interval = dc.getInterval();
        boolean isReversed = false;
        // w("action ==? : " + (action == DateCalculator.SET_INTERVAL));
        // if we're setting the interval, check for holidays inside...
        if (action == DateCalculator.SET_INTERVAL)
        {
            if (start.before(end)) {
                Calendar t = start;
                start = end;
                end = t;
                interval = - interval;
                isReversed = true;
            }

            for (int i=0; i < holidays.length; i++)
            {
                // w("holiday: " + dmpD(holidays[i]) + "; start: " + dmpD(start) +
                // "; end: " + dmpD(end) + "; mycomp: " + myCompare(start, holidays[i]) +
                // "; comp2: " + myCompare(end, holidays[i]));
                if (!start.before(holidays[i]) && !end.after(holidays[i])) {
                    interval -= 1.d;
                }
            }

            if (isReversed) { interval = - interval; }

            if (dc.getInterval() != interval) {
                dc.setInterval(interval);
                return true;
            }

            return false;
        }

        //
        if (dc.getInterval() == 0.d) { return false; }

        // w("start: " + dmpD(start) +"; end: " + dmpD(end) + "; interval: " + interval);
        // workaound for rules bug passing in interval with wrong sign in 6.0.1 code.
        // this can be removed in 6.1.
        if (start.after(end) && interval > 0.d || start.before(end) && interval < 0.d ) {
            interval = -interval;
        }

        Calendar st = (Calendar) start.clone();
        st.set(Calendar.AM_PM, Calendar.PM);
        st.set(Calendar.HOUR, 0);
        st.set(Calendar.MINUTE, 0);
        st.set(Calendar.SECOND, 0);
        st.set(Calendar.MILLISECOND, 0);
        st.add(Calendar.DATE, 1);
        st.add(Calendar.DATE, -1);

        Calendar en = (Calendar) end.clone();
        en.set(Calendar.AM_PM, Calendar.PM);
        en.set(Calendar.HOUR, 0);
        en.set(Calendar.MINUTE, 0);
        en.set(Calendar.SECOND, 0);
        en.set(Calendar.MILLISECOND, 0);
        en.add(Calendar.DATE, 1);
        en.add(Calendar.DATE, -1);

        // if these are reversed, then swap the dates
        // for comparison, so the first date is always
        // before the second.  when we adjust below,
        // we take this into account.
        boolean reversed = en.before(st);

        if (reversed) {
            Calendar h = st;
            st = en;
            en = h;
        }

        int s = -1;
        int e = -1;

        for (int i=0; i= 0 ) {
                s = i;
                break;
            }
        }

        // if all holidays are before start date, nothing to do.
        if (s == -1) { return false; }

        for (int i= holidays.length; i-- > 0;) {
            if (myCompare (holidays[i], en) <= 0) {
                e = i;
                break;
            }
        }

        if (e < s) { return false; }    // no holidays in between

        // number of holidays is the number in between
        int num = e - s + 1;

        // step through adding the days one at a time so we
        // can skip over additional holidays or weekends
        // as we go.  note that we are adjusting the date in the date calculator
        // which is actually a reference (pointer) to the original
        // object, so we are always adjusting the end date
        for (int i=0; i 0 && myCompare (st, holidays[s-1]) == 0  ) {
                    s--;
                    num++;
                }
            }
            else
            {
                advanceOneBusinessDay(en, end);
                // if new day is a holiday, skip another one.
                if (e < holidays.length && myCompare(en, holidays[e+1]) == 0 ) {
```

```
                num++;
                e++;
            }
        }
    }

    return changed;
}
;>;>
```

ucAdminPreUdfListTransaction

- Add new comment

Purpose

ucAdminPreUdfListTransaction is called to allow updates of related lists as part of the same transaction that maintains the base list. This is done prior to the DB update of the primary list.

Applies To

Administration Methods

Signature

```
public void ucAdminPreUdfListTransaction(
                  Connection con,       // database connection
                  UdfList listEntry,
                  String op )           // A = add, M = modify, D = delete
```

Notes

No commit should be done in this method. Error conditions should throw an exception, which will abort the underlying transaction

Example

```
 public void ucAdminPreUdfListTransaction(
                  Connection con,       // database connection
                  UdfList listEntry,
                  String op)            // A = add, M = modify, D = delete
                  throws Exception {
                         //Call Best Practices:
                         super.ucAdminPreUdfListTransaction(con, listEntry, op);
    }
```

ucAdminUtilityButtons

Purpose

This user custom exit is used to add new buttons with custom code as their action, to the administration utilities.

Applies To

Administration utilities

Signature

```
public String ucAdminUtilityButtons( String utility,
                 SesameSession session,
                 Connection dbconn,
                 HashMap formValues) throws Exception
```

Notes

The parameters are as follows:

-- utility = name of the utility, e.g., "FILE_IMPORT"

-- session = the current session (used for building URL's, for example)

-- dbconn = current database connection

-- formValues = the form values from the request initiating the utility

The return value from the method must be a well-formed HTML string that will appear within the menubar of the utility screen. The button's target URL must follow the usual rules for ExtraView if the ExtraView instance is the target, that is, the session_id, p_action and p_option paramters must be included.

ucAllowAttachmentOperation

- Add new comment

Purpose

Determines permissibility of a GUI attachment operation, such as: ADD button, EDIT button, DESCRIPTION modification, or DELETE button

Applies To

Attachment Methods

Signature

```
   public boolean ucAllowAttachmentOperation(
        String operation,       // ADD, EDIT, MODIFY_DESCRIPTION, DELETE
        String layoutType,      // ADD_PROBLEM, EDIT
        Attachment attachment,  // if available, an attachment id
        SesameSession session,  // current session
        Connection dbconn,      // DB connection
        Map selectedVals        // current issue state
    ) throws Exception;
```

Notes

There are three environments (layoutType) that this may be invoked from ADD_PROBLEM screen, EDIT_PROBLEM screen, EDIT_ATTACHMENT popup. The Attachment object is passed in when available. It is not available for the ADD button, otherwise, it is non-null, then the usual parameters are passed in. This method RETURNS true if operation permitted, false if not.

Example

```
public boolean ucAllowAttachmentOperation(
        String operation,       // ADD, EDIT, MODIFY_DESCRIPTION, DELETE
        String layoutType,      // ADD_PROBLEM, EDIT
        Attachment attachment,  // if available, an attachment id
        SesameSession session,  // current session
        Connection dbconn,      // DB connection
        Map selectedVals)       // current issue state
throws Exception
{
    String role = session.getUserRole();
    String userId = session.getUserId();

    // TESTER role should only have the ability to delete attachements
    // submitted by same user for issues that in queue.
    if ("DELETE".equals(operation) && "TESTER".equals(role) )
    {
        String attachmentUserId = attachment.getUserId();

        if (! userId.equals(attachmentUserId) )
        {
                Z.log.writeToLog(Z.log.WARN, "Disallow delete of attachment submitted by " +
                                attachmentUserId + " for user " + userId);
            return false;
        }
    }

    return true;
}
```

ucAllowViewDocumentOperation

Purpose

This method is called when a user accesses a document field, attempting to view the document. This gives control within user custom code to selectively allow different documents to be downloaded by users according to criteria defined by the administrator.

Applies To

Add Issue, *Edit Issue* screens and Reports

Signature

```
public boolean ucAllowViewDocumentOperation(
        HttpServletRequest request,
        HttpServletResponse response,
        Connection dbconn,
        SesameSession session,
        Map selectedVals)
    throws Exception {
      return true;
    }
```

Notes

- The method returns `true` if operation is permitted, `false` if not
- You should set a value into `ERR_MSG` in the selectedVals. The `ERR_MSG` may be a text message or an `ALERT` message
- The base code will display the content of `ERR_MSG`
- The method is called when the user clicks the link to view the contents of the Document field.

ucAllowedValuesChanges

Purpose

Informs user custom code about changes made to the Allowed Values combinations from within administration screens.

Applies To

Administration Methods

Signature

```
   public boolean ucAllowedValuesChanges(
        Connection dbconn,      // DB connection
        AllowedValues av        // allowed values objejct
    ) throws Exception;
```

Notes

This method is called by ExtraView from within the Administration utility that allows the adding, editing and deleting of allowed value combinations. The method is called for each and every change to the allowed value combination made by the user within each update operation. For example, if the user adds 2 allowed values and deletes one allowed value, the method is called 3 times. The method is also called for changes to the sort sequence.

The method can never alter the stored allowed value combinations.

ucApiSubmit

- Add new comment

Purpose

This user exit that allows modification of values passed in during an API call.

Applies To

API

Signature

```
public String ucApiSubmit (
      SesameSession session,   // the current sesameSession
      HashMap params,
      Connection dbconn)
```

Notes

Values are passed in a hashmap and the method returns a hashmap.

Example

```
 public HashMap ucApiSubmit (SesameSession session,
                                 HashMap params,
                                 Connection dbconn) throws Exception;
```

ucChangeEmailHeader

Purpose

This user exit performs two tasks:

The user can modify the values for EMAIL_FROM_USER_NAME and EMAIL_FROM_USER_ID. These values are populated by default from behavior settings with these same names and they are used as the FROM name and address within standard outgoing email headers.

The user can specify 2 new variables to be used as a REPLY-TO value in the outgoing standard email header. These have the names;

EMAIL_REPLY_TO_NAME
EMAIL_REPLY_TO_ADDRESS

Applies To

API

Signature

```
public void ucChangeEmailHeader( SesameSession session,
                                 Connection dbconn,
                                 String problemId,
                                 HashMap emailVariables )
```

Notes

Values are passed in a hashmap and the method returns a hashmap.

Example

```
// just supply REPLY_TO header info
public void ucChangeEmailHeader( SesameSession session,
                                 Connection dbconn,
                                 String problemId,
                                 HashMap emailVariables ) {
    emailVariables.put("EMAIL_REPLY_TO_ADDRESS", "replytoaddress@my_company.com");
    emailVariables.put("EMAIL_REPLY_TO_NAME", "myname");
}
```

ucCliCustom

- Add new comment

Purpose

Exit to control customizing calls by way of the Application Programming Interface, or the Command Line Interface.

Applies To

CLI Methods

Signature

```
public void ucCliCustom (
      HashMap parms,
      PrintWriter out,
      String delim,
      Connection dbconn,
      SesameSession session)
```

Notes

This method is called from the CLI when a statevar of custom is passed. Results are written to the PrintWriter and returned to the CLI as output. These can be plain text, HTML, XML, or other formats.

Example

```
public void ucCliCustom (HashMap params,
                         PrintWriter out,
                         String delim,
                         Connection dbconn,
                         SesameSession session)
{
    String op = (String) params.get("OPERATION");

    if ("credit_card_transaction".equalsIgnoreCase(op) )
    {
        // doCreditCardTransaction(params, out, dbconn, session);

        String issueId = (String) params.get("ID");
        String result = (String) params.get("RESULT");

        if (issueId == null || result == null) {
            out.println("ERROR: Missing one or more required parameters: ID, RESULT!");
            return;
        }

        // verify result values and map to CREDIT_CARD_RESULT field
        String creditCardResult = null;

        if (result.equals("yes") ) {
            creditCardResult = "4515";
            // These static vars (holding UDF_LIST_ID values)
            // should be setup in the class constructor
            // creditCardResult = CREDIT_CARD_RESULT__SUCCESS;
        } else if (result.equals("no") ) {
            creditCardResult = "4520";
            // These static vars (holding UDF_LIST_ID values)
            // should be setup in the class constructor
            // creditCardResult = CREDIT_CARD_RESULT__FAIL;
        } else {
            out.println("ERROR: RESULT parameter specifies invalid value: " + result);
        }

        if (creditCardResult != null)
        {
            boolean updated = false;

            try
            {
                // verify that id exists
                if (! Problem.isProblemPermitted(dbconn, issueId, false) ) {
                    // false == readOnly flag
                    out.println("ERROR: Record " + issueId +
                                     " does not exist or is not accessible!");
                }
                else
                {
                    // update payment record with credit card result
                    HashMap updateMap = new HashMap();
                    updateMap.put("CREDIT_CARD_RESULT", creditCardResult);

                    if (apiEditItem(issueId, updateMap, dbconn, session) == 0) {
                        out.println("ERROR: Failed to update record " + issueId +
                                         "!  See logs for details.");
                    } else {
                        updated = true;
                    }
                }
            } catch (Exception e) {
                ErrorWriter.write(e, ErrorWriter.LOGERR);
                out.println("ERROR: General error condition: " + e.getMessage() );
            }

            // write success response
            if (updated == true) { out.println("Record " + issueId + " updated."); }
        }
    }

    super.ucCliCustom(params, out, delim, dbconn, session);
}
```

ucCliGetTemplate

- Add new comment

Purpose

ucCliGetTemplate returns a user custom template based on its name.

Applies To

CLI Methods

Signature

```
public com.sesame.template.Template ucCliGetTemplate (
      String templateName)
```

Notes

Example

```
public Template ucCliGetTemplate(String templateName) {
        return null;
  }
```

ucCloneGenAltId

- Add new comment

Purpose

This can be used to generate the ALT_ID value to store an issue when cloning an issue.

Applies To

Edit screen buttons

Signature

```
public HashMap ucCloneGenAltId(
      ProblemFormParam values,    // contains values of form params
      FormParameters fp,          // contains values of form params
      HashMap msg,                // contains msg that goes into COMMENTS field
      Connection con,             // the database connection
      SesameSession session)      // the session object
```

Notes

It can also be used to manipulate the cloning message that goes into the inbuilt COMMENTS field. For example, if you want to insert a message that is different from the standard “Issue # xxxxx cloned from Issue # yyyyy”, then use this method to overwrite the value for the COMMENTS parameter.

Example

This method shows how a cloning message can be changed from the standard. Instead, it overwrites the value for the comments parameter, and returns the modified message.

```
public HashMap ucCloneGenAltId(
      ProblemFormParam values,    // contains values of form params
      FormParameters fp,          // contains values of form params
      HashMap msg,                // contains msg that goes into COMMENTS field
      Connection con,             // the database connection
      SesameSession session) throws Exception  // the session object
{
    String project_prefix = null;
    String area = null;
    String project = null;
    String new_alt_id = null;
    ProblemFormParam npfp = (ProblemFormParam) session.getAttribute("CLONEE_PFP");

    if (npfp == null) {
       project = (String) fp.get("p_project");
    } else {
       project = (String) npfp.get("PROJECT_ID");
    }

    int index1 = project.indexOf("|");
    if (index1 > -1) {
        area = project.substring(0, index1);
        project = project.substring(index1 + 1, project.length());
    }

    // Get current issue's ALT_ID
    String alt_id = (String) values.get("ALT_ID");
    if (alt_id == null) {
        Problem p = Problem.getReference(con, (String) values.get("ID"));
        alt_id = p.getAltId();
    }

    if (area == null) { area = "AREA"; }
    if (project == null) { project = "PROJECT"; }

    project_prefix = area + "_" + project;

    String preAltId = null;
    String seq = (String) values.get("ID");

    int len = seq.length();
    // fill with preceeding zeros if length is less than 5 digits
    while (len < 5) {
        seq = "0" + seq;
        len++;
    }

    preAltId = project_prefix + seq;
    new_alt_id = preAltId;

    msg.put("alt_id", new_alt_id);

    String attr = Z.appDefaults.getAttribute("ITEM_ID_DISPLAY");
    attr = attr.trim().toUpperCase();

    if (attr.equals("ALT_ID"))
    {
        String idTitle = Z.dictionary.getTitle(con, "ALT_ID");
        String problemTitle = Z.dictionary.getTitle(con, "PROBLEM");
        SesameMessageFormat smf = new SesameMessageFormat(con, session);
        smf.clear();
        smf.append("This ");
        smf.appendString(problemTitle);
        // put in original problem#
        smf.append(" ");
        smf.appendString(alt_id);
        smf.append(" has been cloned. The new ");
        smf.appendString(idTitle);
        smf.append(" is ");
        smf.appendString(new_alt_id);
        smf.append("\n");
        smf.append("\n");

        String s = smf.getFormattedMessage();
        msg.put("cloned_msg", s);
    }

    return msg;
}
```

ucCloneSetParams

- Add new comment

Purpose

This returns a new updated PFP before the cloning of an issue.

Applies To

Edit Issue screen

Signature

```
public ProblemFormParam ucCloneSetParams(
      ProblemFormParam pfp,
      SesameSession session,
      Connection dbconn )
```

Notes

Any specific adjustments needed in the parameters can also be done here.

Example

The following example clears out a couple of fields in the newly cloned record and sets the STATUS to OPEN.

```
public ProblemFormParam ucCloneSetParams( ProblemFormParam pfp,
             SesameSession session,
             Connection dbconn )
             throws Exception
{
    // Modify form parameters
    pfp.remove("DETAIL_DISPLAY");
    pfp.remove("CLONE_AS");
    pfp.put("STATUS", "OPEN");

   return pfp;
}
```

ucColumnReportInit

- Add new comment

Purpose

This method is called when you execute a column report. The method is used to initialize any user custom code you require for the report

Applies To

Search & Reporting screens

Signature

```
public void ucColumnReportInit()
```

Notes

The method is also called when the user refreshes a report

Example

```
public HashMap ucModifyColumnReportRow (HashMap tags)
throws Exception
{
    // Check the report output type: only implement custom color coding scheme if
    // output type if HTML (not for Word or Excel)
    SesameSession session = SesameSession.getSession();
    String outputType = (String) session.getAttribute("REPORT_OUTPUTTYPE");

    if ( ! "HTML".equalsIgnoreCase(outputType) ) { return tags; }

    // Resource Interactive CUSTOM CODE:
    // Set the color code according to the issue priority:
    String priority = (String) tags.get("PRIORITY");

    if (priority != null)
    {
        String fontColor = null;

        if ( priority.equals("Monday") ) {
            fontColor = "#FF0000";
        } else if ( priority.equals("Tuesday") ) {
            fontColor = "#CF2904";
        } else if ( priority.equals("Wednesday") ) {
            fontColor = "#CF4E04";
        } else { // Friday or Thursday
            fontColor = "#000000";
        }

        // Iterate through all tag values and set their font color
        Set keys = tags.keySet();
        Iterator iterator = keys.iterator();

        while (iterator.hasNext() )
        {
            String key = (String) iterator.next();

            // Don't change special tags (i.e., $$ROWNUM_START$$, $$ROWNUM_SIZE$$)
            // NOTE: Noticed that if changing the ID value,
            // then an Exception is generated in the log file:

            if ( key.startsWith("$$") || key.equals("ID") ) {
                continue;
            }

            String value = (String) tags.get(key);
            tags.put(key, " < font color='" + fontColor + "' >" + value + "< /font>");
        }
    }

    return tags;
}
```

ucDeleteAttachment

- Add new comment

Purpose

This method deletes the passed in file attachment from the local file system.

Applies To

Attachment Methods

Signature

```
public boolean ucDeleteAttachment(Attachment attachment)
```

Notes

Example

```
public boolean ucDeleteAttachment(Attachment attachment)
{
    String filepath = attachment.getExternalAttachmentIdent();
    if (filepath == null) { return true; }

    File file = new File(filepath);

    if (file.delete() != true) {
        Z.log.writeToLog(Z.log.ERROR, "ucDeleteAttachment: could not delete file " +
            filepath);
        return false;
    }

    return true;
}
```

ucEditAttachFdf

- Add new comment

Purpose

This is used to save an Adobe format FDF file as an attachment to the issue.

Applies To

Adobe Fdf screen

Signature

```
public boolean ucEditAttachFdf (
      Connection c,             // database connection
      FdfFile fdf,              // the fdf file
      String problemId,         // the issue number
      Map params)               // Parameter map
      throws Exception
```

Notes

Example

```
public boolean ucEditAttachFdf (
      Connection c,             // database connection
      FdfFile fdf,              // the fdf file
      String problemId,         // the issue number
      Map params)               // Parameter map
      throws Exception
{
    Map p = new HashMap();
    p.put("ID", itemId);
    p.put("ATTACH_DESC", "Sample form: " + (String) values.get("SHORT_DESCR"));
    p.put("FILE_NAME", "SampleForm.fdf");
    Attachment a = new Attachment(con);
    File f = new File(fdf.getFileLocation());

    w("fileLocation: " + fdf.getFileLocation());

    boolean status = a.insert(f, p, "text/plain");
    fdf.delete();

    return status;
}
```

ucEditAutoAttachFdf

- Add new comment

Purpose

This method determines if you should attach an Adobe FDF format field to an issue, and performs the attachment if needed.

Applies To

Adobe Fdf screen

Signature

```
public boolean ucEditAutoAttachFdf (
      SesameSession session,    // Session object
      Map params,               // parameters
      Connection dbconn)        // database connection
          throws Exception
```

Notes

The FdfFile class is a light wrapper around Adobe's fdf toolkit object

Example

```
public boolean ucEditAutoAttachFdf(SesameSession session,
            Map values,
            Connection con)
throws Exception
{
    boolean status = false;
    Z.log.writeToLog(Z.log.DEBUG, "autoAttachFdf params " + values);

    if (session.getAttribute("GENERATE_PDF") != null && values.get("ID") != null)
    {
        String v = (String) session.getAttribute("GENERATE_PDF");
        session.removeAttribute("GENERATE_PDF");

        if ("Y".equals(v)) {
            FdfFile fdf = fdf(session, values);
            status = attachFdf(con, fdf, (String) values.get("ID"), values);
        }
    }

    return status;
}
```

ucEditClone

- Add new comment

Purpose

ucEditClone is called during the issue clone process, after the new database objects have been set up but before rendering the edit screen for the new issue

Applies To

Edit Issue screen

Signature

```
public void ucEditClone (
      Problem item,          // the item or case
      ArrayList releases,    // the list of releases
      ArrayList modules,     // the list of modules
      ArrayList udfs,        // the list of UDF's
      SesameSession session, // the current session
      Connection con,        // DB connection for ExtraView db.
      String parentId,       // id of the cloning bug
      String clonedId,       // new id of the cloned bug.
      HashMap values)        // the selected values from the form
```

Notes

Example

In this example, if the record is in the Resolved Status, upon executing a clone, then it's moved to Closed.

```
public void ucEditClone (
            Problem item,           // the item or case
            ArrayList releases,     // the list of releases
            ArrayList modules,      // the list of modules
            ArrayList udfs,         // the list of UDF's
            SesameSession session,  // the current session.
            Connection con,         // DB connection for ExtraView db.
            String parentId,        // id of the cloning bug
            String clonedId,        // new id of the cloned bug.
            HashMap values) {       // the selected values from the form
    String area = item.getAreaId();
    String project = item.getProjctId();

    if (AREA__BUGS.equals(area) && PROJ__BUGS_DATA.equals(project) )
    {
        String status = (String) values.get("STATUS");

        // If the status is Resolved when we clone the record, go ahead and close it
        if (status != null && "RESOLVED".equals(status) ) {
            values.put("STATUS", "CLOSED");
        }
    }
}
```

ucEditClose

- Add new comment

Purpose

ucEditClose allows additional processing upon closing an edit screen.

Applies To

Edit Issue screen

Signature

```
public void ucEditClose(
      ProblemFormParam values, // contains values of form params
       SesameSession session)  // current session
```

Notes

Example

This method shows how upon the closure of an edit screen, the users role is reset to their default role.

```
public void ucEditClose (
         ProblemFormParam values,    // contains values of the form params
         SesameSession session) {    // current session.

        super.ucEditClose(values, session);
        resetRole(session);
    }
```

ucEditDelete

- Add new comment

Purpose

ucEditDelete is called during the delete process prior to the actual delete.

Applies To

Edit Issue screen

Signature

```
public String ucEditDelete(
      Problem item,            // the item object
      ArrayList attachments,   // the list of attachments
      ArrayList interestList,  // the interest list
      ArrayList releases,      // the list of release
      ArrayList modules,       // the list of modules
      ArrayList udfs,          // the list of udfs
      SesameSession session,   // current session
      ProblemFormParam values) // values of from the form
```

Notes

Returning a message from this method will prevent the delete occurring and will display the message as an alert.

Example

This example shows how an error message is returned, and the deletion of the issue is prevented if the status of the issue is not "In Work", or the person trying to delete the issue is not the Originator of the issue.

```
public String ucEditDelete (
                                Problem item,              // the item object
                                ArrayList attachments,     // the list of attachments
                                ArrayList interestList,    // the interest list
                                ArrayList releases,        // the list of release
                                ArrayList modules,         // the list of modules
                                ArrayList udfs,            // the list of udfs
                                SesameSession session,     // current session
                                ProblemFormParam values) { // values of from the form.

        // For Bugs allow delete when the status = Work and user is the Originator
        String areaId = item.getAreaId();
        String status = item.getStatus();
        String originator = item.getOriginator();
        String userId = session.getUserId();
        String role = session.getUserRole();
        String error = null;

        if (AREA__BUGS.equals(areaId) && ! ROLE__ADMIN.equals(role)
           && ! ROLE__COORDINATOR.equals(role)
           && ! ROLE__IMPORT_SECURITY.equals(role) ) {
            if (! ("IN_WORK".equals(status) || "SUBMIT".equals(status)) ) {
                return Z.m.msg(session,
                "Sorry, only records In Work may be deleted.
                Since this is an official record on the Master Bug Log,
                this issue can't be deleted.");
            } else if (! userId.equals(originator) ) {
                return Z.m.msg(session,
                "Sorry, only the originator may delete the record!");
            }
        }
        error = deleteRecordHistory(item, attachments, interestList, releases,
                                                 modules, udfs, session, values);
        if (error == null)
            removeCurrentEditSession(item, session);
        return error;
    }
```

ucEditGenerateFdf

- Add new comment

Purpose

Used to generate an Adobe FDF file (PDF form data).

Applies To

Adobe Fdf screen

Signature

```
public FdfFile ucEditGenerateFdf (
      SesameSession s,  // Session object
      Map params)       // Parameter map
```

Notes

Example

```
public FdfFile ucEditGeneratefdf(SesameSession session,
                                   Map values) throws Exception {

          FdfFile f = new FdfFile();
          f.setFileLocation(session);
          Z.probe("entering FdfFile");
          Connection dbconn = null;
          String SYSDATE = Z.dbms.CURRENT_TIMESTAMP();
          String today = "";
          String sql = null;
          SimpleDateFormat sdf = null;
          String problemId = (String) values.get("ID");
          HashMap udfHm = new HashMap();
          try {
                dbconn = Z.pool.getConnection();
                Z.log.writeToLog(7, "PARAMS ARE >>>" + values);
                Z.log.writeToLog(7, "MDR problemId " + problemId);
                Z.log.writeToLog(7, "MDR udfHm " + udfHm);

        udfHm = setMDRValues (session, values, dbconn);

                f.setFields(udfHm);
                StringBuffer templateLocation = new StringBuffer(Z.extSiteURL);
                templateLocation.append("/pdf_templates/");
                templateLocation.append("Medwatch3500A.pdf");
                f.getFdfDoc().SetFile(templateLocation.toString());
                f.write();
          } catch (SQLException sqle) {
                Z.log.writeToLog(Z.log.ERROR, "SQLException: " + sql);
                ErrorWriter.write(sqle, ErrorWriter.LOGERR);
          } catch (Exception e) {
                ErrorWriter.write(e, 0);
          } finally {
                if (dbconn != null) {
                  Z.pool.close(dbconn);
                }
          }
          return f;
        }
```

ucEditInit

- Add new comment

Purpose

This allows you to manipulate the parameters before they are rendered, with the exception of area and project.

Applies To

Edit Issue screen

Signature

```
public void ucEditInit (
      SesameSession session,    // current session
      HashMap values)           // contains values of form elements
```

Notes

This is called as the Edit routine is first entered.

Example

This method demonstrates how just as an Edit screen is entered, the status of the issue is checked. If the status is no longer in draft mode, the Approver role can have access to edit the issue. If the status of the issue is still in draft, Approvers have no ability to edit.

```
public void ucEditInit (SesameSession session, HashMap values) {
        String status = (String)values.get("STATUS");
        //only set the access to approve if the status is not longer in draft mode.
        if(statusPendingApproval.equals(status)){
            setApproverAccess(values,session.getUserId());
        }
                super.ucEditInit(session, values);
    }
```

ucEditPostUpdate

- Add new comment

Purpose

Update of external database objects or anything that requires the change to be committed is best done at this point

Applies To

Edit Issue screen

Signature

```
public void ucEditPostUpdate (
      Problem pProblem,         // the problem or case
      ArrayList pReleases,      // the list of releases
      ArrayList pModules,       // the list of modules
      ArrayList pUdfs,          // the list of UDF's
      SesameSession session)    // the current Sesame Session
```

Notes

This is called during the edit process after the main problem update, but before email is generated

Example

```
  public void ucEditPostUpdate (
                     Problem item,                   // the item or case
                     ArrayList releases,             // the list of releases
                     ArrayList modules,              // the list of modules
                     ArrayList udfs,                 // the list of UDF's
                     SesameSession session) {        // the current sesameSession

        super.ucEditPostUpdate(item, releases, modules, udfs, session);

        String areaId = null;
        String projectId = null;
        String status = null;
        String category = null;

        if (item != null) {
            areaId = item.getAreaId();
            projectId = item.getProjctId();
            status = item.getStatus();
            category = item.getCategory();
        }

        String oldStatus = (String) session.getAttribute("UC__OLD_STATUS");
        session.removeAttribute("UC__OLD_STATUS");

        // send email only on new closure of record
        if (A_IT.equals(areaId) && "CLOSED".equals(status)
            && ! "CLOSED".equals(oldStatus)
            && "IT_LICENSE_KEY_DELIVERY".equals(category) ) {
            sendLicenseKeyEmail(item, udfs, session);
        }
    }
```

ucEditPreDisplay

- Add new comment

Purpose

This is called before the drilling down to edit an issue.

Applies To

Edit Issue screen

Signature

```
public boolean ucEditPreDisplay (
      SesameSession session,    // the current Sesame session
      Problem prob,             // the current problem
      HttpServletRequest request,
      PrintWriter out)
```

Notes

Returning false will prevent the issue from being edited.

Example

```
  public boolean ucEditPreDisplay (
                         SesameSession session, // the current Sesame session
                         Problem item, // the current item
                         HttpServletRequest request,
                         PrintWriter out) {
       // check if clone was just performed in order to perform after-clone operations
        if (session.getContainsKey("UC_CLONE_INFO") ) {
            ArrayList cloneInfo = (ArrayList) session.getAttribute("UC_CLONE_INFO");

            if (cloneInfo != null) {
                String parentId = (String) cloneInfo.get(0);
                String clonedId = (String) cloneInfo.get(1);
                String groupName = (String) cloneInfo.get(2);
                String title = item.getShortDescr();

                cloneRelationshipGroupChildren(groupName, parentId, clonedId);
                cloneInterestList(parentId, clonedId, title, session);
            }
        }
        return super.ucEditPreDisplay(session, item, request, out);
    }
```

ucEditPreUpdate

- Add new comment

Purpose

This allows additional error reporting by returning an error message.

Applies To

Edit Issue screen

Signature

```
public  String ucEditPreUpdate (
      Problem pOldProblem,      // the problem or case
      ArrayList pOldReleases,   // the list of releases
      ArrayList pOldModules,    // the list of modules
      ArrayList pOldUdfs,       // the list of UDF's
      Problem pNewProblem,      // the problem or case
      ArrayList pNewReleases,   // the list of releases
      ArrayList pNewModules,    // the list of modules
      ArrayList pNewUdfs,       // the list of UDF's
      SesameSession session,    // current session
      ProblemFormParam pfp)     // values of from the form
```

Notes

This is called during the Edit update process before any updates are done. It also allows a final manipulation of values prior to attempting the update.

Example

```
 public String ucEditPreUpdate (
                  Problem oldItem,             // the item or case
                  ArrayList oldReleases,       // the list of releases
                  ArrayList oldModules,        // the list of modules
                  ArrayList oldUdfs,           // the list of UDF's
                  Problem newItem,             // the item or case
                  ArrayList newReleases,       // the list of releases
                  ArrayList newModules,        // the list of modules
                  ArrayList newUdfs,           // the list of UDF's
                  SesameSession session,  // current session
                  ProblemFormParam values) {   // values of from the form.

        String error = super.ucEditPreUpdate(oldItem, oldReleases,
                 oldModules, oldUdfs, newItem, newReleases,
                 newModules, newUdfs, session, values);
       if (error != null) return error;

       executeIntegration(oldItem, newItem, values);

       return null;
   }
```

ucEditRefresh

- Add new comment

Purpose

This method is called during every edit screen refresh.

Applies To

Edit Issue screen

Signature

```
public void ucEditRefresh (
SesameSession session,  // current session
ProblemFormParam pfp)   // contains values of form params
```

Notes

If you are providing dependent values, you will need to maintain them when the parent field changes.

Example

```
public void ucEditRefresh (
                       SesameSession session,      // current session
                       ProblemFormParam values) {  // contains values of form params
        super.ucEditRefresh(session, values);
        String whatChanged = whatChanged(values);

        if ("ADD_SELECTED_BTN".equals(whatChanged) ) {
            doAddSelectedItems(session, values);
        }
    }
```

ucEditSubmit

- Add new comment

Purpose

This provides an easier and quicker way to manipulate data instead of using the ucEditPreUpdate method.

Applies To

Edit Issue screen

Signature

```
public String ucEditSubmit (
      SesameSession session,    // current session
      ProblemFormParam pfp)     // contains values of form params
}
```

Notes

This is called after the form is submitted during the Edit process and before the umbrella of problem objects is created. Any String returned by this method will cause the process to stop and the value of the String to be displayed to the user in the form of a JavaScript Alert Box. It allows access to the parameters before they have been manipulated.

Example

```
   public String ucEditSubmit(
              SesameSession session,
              ProblemFormParam values){
        String status = (String)values.get("STATUS");
        String msg = "";
        w("ucEditSubmit before rules: " + status);

        if(statusPendingApproval.equals(status)){
            doApproverLogic(values);
        }

        // execute the rules
        msg = super.ucEditSubmit(session,values);

        if (TextManager.isStringVisible(msg)){
            return msg;
        }
        status = (String)values.get("STATUS");
        w("ucEditSubmit after rules: " + status);

        //only check to set the status to approve if the request is in pending approval.
        if(statusPendingApproval.equals(status)){
            setNextApprover(values);
        }
        // w("Additional Approals required: " + additionalApprovalRequired);
        return msg;
    }
```

ucEmailFilter

- Add new comment

Purpose

This method allows you to pre-populate or change the values that are displayed on the CustomEmail Filter screen

Applies To

Mail screen

Signature

```
public void ucEmailFilter( SesameSession session,   //current session
                        ProblemFormParam values)    // form params to populate the form
```

Notes

Example

```
public void ucEmailFilter(
                       SesameSession session,      // current session
                       ProblemFormParam values) {  // form params to populate the form
        return;
  }
```

ucExpiredPasswordExit

- Add new comment

Purpose

ucExpiredPasswordExit is called from an unsecured environment when the password for the user is determined to be expired. This is currently used to invoke the RequestPasswordDisplay class which allows the user's password to be generated or modified.

Applies To

User Authentication Methods

Signature

```
public boolean ucExpiredPasswordExit(
                   HttpServletRequest request,      // the servlet request
                   HttpServletResponse response,    // the servlet response
                   Connection dbconn,               //the database connection
                   SesameSession session)
                       throws Exception             // current session
```

Notes

Return Boolean true if the expired password exit was handled. Return false if nothing is done – this will cause the calling class (LoginDisplay) to go to the Expired Password screen

Example

```
public boolean ucExpiredPasswordExit(
                HttpServletRequest request,             // the servlet request
                HttpServletResponse response,           // the servlet response
                Connection dbconn,                      // the database connection
                SesameSession session) throws Exception { // current session

        if (Z.config.getConfigValue("Random_Password_Pass".toUpperCase()) != null) {
            RequestPasswordDisplay.doDisplay(request, response, dbconn, session);
            return true;
        }
        return false;
    }
```

ucExportImportTables

- Add new comment

Purpose

Applies To

Administration Methods

Signature

```
    public Map ucExportImportTables()
                    throws Exception { return null; }
```

Notes

Example

```
 public Map ucExportImportTables() throws Exception {
        HashMap eximMap = new HashMap();
        eximMap.put("USER_GROUP",
                      new UserGroupBuilder("USER_GROUP"));
        eximMap.put("USER_GROUP_USER",
                      new UserGroupUserBuilder("USER_GROUP_USER"));
        eximMap.put("UC_PROJECT_PRIVACY_GROUP",
                      new ProjectPrivacyGroupBuilder("UC_PROJECT_PRIVACY_GROUP"));
        eximMap.put("UC_PROJECT_USER_GROUP",
                      new ProjectUserGroupBuilder("UC_PROJECT_USER_GROUP"));
        eximMap.put("UC_USER_AREA_ROLE",
                      new UserAreaRoleBuilder("UC_USER_AREA_ROLE"));
        eximMap.put("UC_USER_GROUP_PRIVACY_GROUP",
                      new UserGroupPrivacyGroupBuilder("UC_USER_GROUP_PRIVACY_GROUP"));
        return eximMap;
    }
```

ucFilterPopup

- Add new comment

Purpose

ucFilterPopup is called before rendering the values in Popup list fields and allows the list of values to be modified.

Applies To

Field Rendering Methods

Signature

```
public void ucFilterPopup(SesameSession session,      // current session
                              ValidationList selList, // the items to be displayed
                              String pGlobal,         // ddname
                              HashMap selectedVals,   // selected values in the list
                              HashMap attributes,     // attributes of the list
                              String mode,            // where it's called from...
                              String layoutType);
```

Notes

Example

ucFilterUserPopup

- Add new comment

Purpose

ucFilterUserPopup is called when the USER_LIST_DISPLAY behavior setting has a value of POPUP, and can be used to filter the results of a user search by modifying the TreeMap parameter.

Applies To

Field Rendering Methods

Signature

```
public void ucFilterUserPopup (SesameSession session,
                               String ddName,
                               TreeMap userMap)
```

Notes

Example

```
 public void ucFilterUserPopup (SesameSession session,
                                String ddName,
                                TreeMap userMap) {
        String role = session.getUserRole();

        // only perform check for Customer role
        if (! "CLIENT".equals(role) ) return;

        // For Customer (CLIENT) role, filter search results to only users
        // with same Company Name as the CUSTOMER_ID field value on the record
        // NOTE: THIS FILTERING AFFECTS ALL USER FIELDS, AS OPPOSED TO THE
        // SPECIFIC FIELDS LISTED IN THE checkCustomerFields METHOD.
        HashMap tabVals = (HashMap) session.getAttribute("TAB_VALS");
        String companyName = null;
        String customerUdfListId = null;

        // first, get the Company Name from the CUSTOMER_ID field
        if (tabVals != null) {
            customerUdfListId = (String) tabVals.get("CUSTOMER_ID");
            Connection con = null;

            try {
                con = Z.pool.getConnection("ucFilterUserPopup");
                UdfList ul = UdfList.getReference(con, customerUdfListId);
                companyName = ul.getTitle();
            }
            catch (Exception e) {
                ErrorWriter.write(e, ErrorWriter.LOGERR);
            }
            finally {
                Z.pool.close(con);
            }
        }
        // otherwise, get the Company Name from the current user
        else {
            SecurityUser su = (SecurityUser) session.getAttribute("USER");

            if (su != null) {
                companyName = su.getCompanyName();
            }
        }
        if (companyName == null) {
            Z.log.writeToLog(Z.log.WARN,
            "ucFilterUserPopup: NO COMPANY NAME DETECTED FOR USER SEARCH FILTERING");
            userMap.clear();
        }
        // go through list of users and remove the ones not matching the Company Name
        Iterator i = userMap.keySet().iterator();
        ArrayList removeList = new ArrayList();

        while (i.hasNext() ) {
            String key = (String) i.next();
            HashMap hm = (HashMap) userMap.get(key);
            String userCompany = (String) hm.get("COMPANY_NAME");

            if (! companyName.equals(userCompany) ) {
                removeList.add(key);
            }
        }
        for (int j = 0; j < removeList.size(); j++) {
            userMap.remove(removeList.get(j));
        }
    }
```

ucFormatLogAreaTimestamp

- Add new comment

Purpose

This exit allows UC to override the user's preferred date format and to render whatever date format is necessary.

Applies To

Add Issue screen

Signature

```
public String ucFormatLogAreaTimestamp( String dateInUsersFormat,
                                            Calendar theTimestamp,
                                            SesameSession session,
                                            Connection dbconn) {
        return null; // this means "respect the user's preferences in date formatting"
    }
```

Notes

While it may seem trivial to format the date, this method is responsible for ensuring that it is sensitive to the user's locale, which may dictate date formats with different ordering for the fields, without which sensitivity there will likely be serious misunderstandings. It should be understood that any formatting done here that is in conflict with the user's preferred format will be inconsistent with all other dates seen by the user. ExtraView therefore disavows any responsibility for the misunderstandings and inconsistencies and any damage incurred therefrom due to the insertion of customer logic in this method.

Example

```
public String ucFormatLogAreaTimestamp (String dateInUsersFormat,
                                            Calendar theTimestamp,
                                            SesameSession session,
                                            Connection dbconn) {
        String strDate = this.getStringFromCalendar(
                 theTimestamp,"MMM dd, yyyy HH:mm:ss");
        StringBuffer dateFormat = new StringBuffer();
        dateFormat.append("");
     dateFormat.append(strDate);
     dateFormat.append("");

        return strDate;
    }
    /**
     * Calendar --> String converter
     */
    protected static String getStringFromCalendar (
             java.util.Calendar cal, String format) {
        if (cal == null) return null;
        Timestamp ts = null;
        String s = null;
        try {
            ts = Convert.toDate(cal);
            SimpleDateFormat sdf = new SimpleDateFormat(format, Locale.US);
            s = sdf.format(ts);
        }
        catch (NullPointerException e) {
            ErrorWriter.write(e, ErrorWriter.LOG);
            s = null;
        }
        return s;
    }
```

ucGetAjaxDependencies

- Add new comment

Purpose

ucGetAjaxDependencies allows custom code to use ajax refreshes by specifying fields that are either required or to be set by custom code from an ajax refresh.

Applies To

Layout rendering

Signature

```
public void ucGetAjaxDependencies( SesameSession session,
                            String area,
                            String project,
                            String layoutType,  // ADD_PROBLEM, EDIT_PROBLEM, SEARCH_XXXXX
                            ArrayList ucParent,
                            ArrayList ucChild)
```

Notes

This exit allows converting page refreshes for custom code to ajax calls. The base code calls this method, and the user returns the parent and child fields in an arraylist.

Example

ucGetAttachmentFileName

- Add new comment

Purpose

Applies To

Attachment Methods

Signature

```
    public String ucGetAttachmentFileName(
                              Attachment attachment) {
        return attachment.getAttachmentId();
    }
```

Notes

Example

```
    public String ucGetAttachmentFileName(
                              Attachment attachment);
```

ucGetContentType ForExtension

- Add new comment

Purpose

Applies To

Administration Methods

Signature

```
public String ucGetContentTypeForExtension(
                 String fileName,
                 String currentContentType)
```

Notes

Example

ucGetRemoveAdv SearchFields

- Add new comment

Purpose

When creating Advanced Search Filters this method is used to remove the fields passed into the method.

Applies To

Search & Reporting Methods

Signature

```
public String[] ucGetRemoveAdvSearchFields(
                        Connection dbconn,
                        SesameSession session)
```

Notes

Example

This example of ucGetRemoveAdvSearchFields shows how when called, the method will go through all of the user defied fields that exist in the Data Dictionary, and be able to pick out certain fields if they are configured for a project that is either not the default project nor the current user's project. It then removes fields that do not exist in the default project, or the users current project, to enable them to not have advanced search access to those particular fields.

```
public String[] ucGetRemoveAdvSearchFields (Connection dbconn, SesameSession session) {
       String userRole = session.getUserRole();
       String projectId = Integer.toString(session.getProject() );

       if (ROLE__ADMIN.equals(userRole)
        || ROLE__COORDINATOR.equals(userRole)
        || ROLE__IMPORT_SECURITY.equals(userRole) )
           return null;

       HashMap udfNameIds = DataDictionary.getUdfNameIds();

       if (udfNameIds == null) {
           Z.log.writeToLog(Z.log.WARN, "DataDictionary UDF Name list is empty!");
           return null;
       }
       // go through list of UDF fields in Data Dictionary and collect all
       // fields configured for a project different than current,
       // ignoring default project
       ArrayList fieldList = new ArrayList();
       Iterator i = udfNameIds.keySet().iterator();

       while (i.hasNext() ) {
           String udfName = (String) i.next();
           DDEntry dde = DataDictionary.getDDEntry(dbconn, udfName);

           if (dde != null) {
                          String udfProjectId = dde.getProjectId();

               if (! "0".equals(udfProjectId) && ! projectId.equals(udfProjectId) ) {
                   fieldList.add(udfName);
               }
           }
       }

       String[] fieldsToRemove = arrayListToStringArray(fieldList);
       Z.log.writeToLog(Z.log.WARN,
           "Removing fields from Advanced Search: " + fieldList);

       return fieldsToRemove;
   }
```

ucGetRoleMenu

- Add new comment

Purpose

Applies To

Administration Methods

Signature

```
    public String ucGetRoleMenu(
                Connection dbconn,
                SesameSession session)
```

Notes

Example

```
 public String ucGetRoleMenu(Connection con, SesameSession session) throws Exception {

        if ("ADMIN".equals(session.getUserId())
           || "SYSTEM".equals(session.getUserId()))return null;
        String myRole = (String) session.getAttribute("USER_ROLE");

        SecurityGroup sg = SecurityGroup.getReference(con, myRole);
        String myRoleTitle = "";
        if (sg != null) myRoleTitle = sg.getTitle();
        SesameMessageFormat smf = new SesameMessageFormat(con, session);
        smf.append("Current Role:");
        smf.appendString(myRoleTitle);
        return "\n<i title=\ Z.m.msg(session, "This is your current role")>"
                             + TextManager.htmlescape(smf.getFormattedMessage())
                             + "</li>";
    }
```

ucGetSequenceInfo

- Add new comment

Purpose

Applies To

Administration Methods

Signature

```
public Map ucGetSequenceInfo(
                Connection dbconn )
```

Notes

Example

```
public Map ucGetSequenceInfo(Connection dbconn) throws Exception {
        HashMap sequences = new HashMap();
        if (dbconn == null)throw new NullPointerException("Connection is null.");
        String sql =
            " select name, table_name, column_name, cache_size from ev_sequence
              where name like '%$_ALT$_SEQ' escape '$'";
        PreparedStatement ps = null;
        ResultSet rs = null;
        List updateStatements;
        try {
            ps = dbconn.prepareStatement(sql);
            rs = ps.executeQuery();
            while (rs.next()) {
                String sequenceName = rs.getString("name");
                int pos = sequenceName.indexOf("_");
                String altIdPrefix = sequenceName.substring(0, pos);
                String tableName = rs.getString("table_name");
                String columnName = rs.getString("column_name");
                String cache = rs.getString("cache_size");
                if (tableName != null && columnName != null && "ITEM".equals(tableName)
                   && "ALT_ID".equals(columnName)) {
                    StringBuffer cookedName = new StringBuffer("cast( ");
                    cookedName.append(Z.dbms.substr("ALT_ID", "4", "5"));
                    cookedName.append(" as numeric )");
                    sequences.put(sequenceName,
                                  getSequenceUpdateStatement(tableName,
                                     cookedName.toString(), cache, sequenceName,
                        altIdPrefix));
                }
            }
        } catch (Exception e) {
            ErrorWriter.write(e, ErrorWriter.LOG);
            throw e;
        } finally {
            if (rs != null)try {
                rs.close();
            }
            catch (Exception basura) {}
            if (ps != null)try {
                ps.close();
            }
            catch (Exception basura) {}
        }
        return sequences;
    }
```

ucGetTableExportInfo

- Add new comment

Purpose

Applies To

Administration Methods

Signature

```
public ArrayList ucGetTableExportInfo()
```

Notes

Example

```
public ArrayList ucGetTableExportInfo() throws Exception {
        TableExportInfo tei = new TableExportInfo();
        tei.tableName = "USER_GROUP";
        tei.isMetadata = true;
        tei.isItemData = false;

        // array of tableDope entries...
        tei.tmkInfo = null;
        // array of softReference entries
        tei.softReferenceInfo = new String[][] {
            {"UDF_LIST_ID", "UDF_LIST", "UDF_LIST_ID"}
        };
        tei.metadataFamilies = new String[] {
            "F_REPORTS", "F_SECURITY_USERS"};
        tei.itemDataFamilies = null;
        ArrayList al = new ArrayList();
        al.add(tei);
        tei = new TableExportInfo();
        tei.tableName = "USER_GROUP_USER";
        tei.isMetadata = true;
        tei.isItemData = false;

        // array of tableDope entries...
        tei.tmkInfo = null;

        // array of softReference entries
        tei.softReferenceInfo = null;
        tei.metadataFamilies = new String[] {
            "F_REPORTS", "F_SECURITY_USERS"};
        tei.itemDataFamilies = null;
        al.add(tei);
        tei = new TableExportInfo();
        tei.tableName = "UC_PROJECT_PRIVACY_GROUP";
        tei.isMetadata = true;
        tei.isItemData = false;

        // array of tableDope entries...
        tei.tmkInfo = null;

        // array of softReference entries
        tei.softReferenceInfo = null;
        tei.metadataFamilies = new String[] {
            "F_REPORTS", "F_SECURITY_USERS"};
        tei.itemDataFamilies = null;
        al.add(tei);
        tei = new TableExportInfo();
        tei.tableName = "UC_PROJECT_USER_GROUP";
        tei.isMetadata = true;
        tei.isItemData = false;

        // array of tableDope entries...
        tei.tmkInfo = null;

        // array of softReference entries
        tei.softReferenceInfo = null;
        tei.metadataFamilies = new String[] {
            "F_REPORTS", "F_SECURITY_USERS"};
        tei.itemDataFamilies = null;
        al.add(tei);
        tei = new TableExportInfo();
        tei.tableName = "UC_PROJECT_USER_GROUP";
        tei.isMetadata = true;
        tei.isItemData = false;

        // array of tableDope entries...
        tei.tmkInfo = null;

        // array of softReference entries
        tei.softReferenceInfo = null;
        tei.metadataFamilies = new String[] {
            "F_REPORTS", "F_SECURITY_USERS"};
        tei.itemDataFamilies = null;
        al.add(tei);
        tei = new TableExportInfo();
        tei.tableName = "UC_USER_AREA_ROLE";
        tei.isMetadata = true;
        tei.isItemData = false;

        // array of tableDope entries...
        tei.tmkInfo = null;

        // array of softReference entries
        tei.softReferenceInfo = null;
        tei.metadataFamilies = new String[] {
            "F_REPORTS", "F_SECURITY_USERS"};
        tei.itemDataFamilies = null;
        al.add(tei);
        tei = new TableExportInfo();
        tei.tableName = "UC_USER_GROUP_PRIVACY_GROUP";
        tei.isMetadata = true;
        tei.isItemData = false;

        // array of tableDope entries...
        tei.tmkInfo = null;

        // array of softReference entries
        tei.softReferenceInfo = null;
        tei.metadataFamilies = new String[] {
            "F_REPORTS", "F_SECURITY_USERS"};
        tei.itemDataFamilies = null;
        al.add(tei);
        return al;
    }
```

ucGetUFIColumnDope

- Add new comment

Purpose

Applies To

User Account Display

Signature

```
    public ArrayList ucGetUFIColumnDope(
                Connection dbconn,
                SesameSession session )
```

Notes

Example

```
 public ArrayList ucGetUFIColumnDope(
                 Connection dbconn,
                 SesameSession session ) throws Exception {
      ArrayList vodafoneDope  = new ArrayList();
      //build an arrayList of UFIColumnDope objects - one for each table
```

```
        //USER_GROUP_USER table
        UFIColumnDope  ucd = new UFIColumnDope();
        ucd.setTableName("USER_GROUP_USER");
        ucd.setIsRepeating(true);
        ucd.setIsEnumerated(true);
        ucd.setColumnName("USER_GROUP_ID");
        ucd.setColumnTitle("User Group ID");
        ucd.setColumnLength(getFieldLength("USER_GROUP_ID",
                        "USER_GROUP_USER",
                         dbconn));

        //build the Enumerated list
        ValidationList userGroupList = UserGroup.getUserGroupsForImport(dbconn);
        ucd.setValuesTitles(userGroupList);

        vodafoneDope.add(ucd);


        //UC_USER_AREA_ROLE table
        ucd = new UFIColumnDope();
        ucd.setTableName("UC_USER_AREA_ROLE");
        ucd.setIsRepeating(true);
        ucd.setIsEnumerated(true);
        // security_group_id column
        ucd.setColumnName("UC_USER_AREA_ROLE.SECURITY_GROUP_ID");
        ucd.setColumnTitle("User Area Role - Role Id");
        ucd.setColumnLength(getFieldLength("SECURITY_GROUP_ID",
                                "UC_USER_AREA_ROLE",
                                 dbconn));
        vodafoneDope.add(ucd);

        //build the Enumerated list
        ValidationList roleVL = SecurityGroup.getSecurityGroups(dbconn,
                         session.getUserId());
        ucd.setValuesTitles(roleVL);

        // area_id column
        ucd = new UFIColumnDope();
        ucd.setTableName("UC_USER_AREA_ROLE");
        ucd.setIsRepeating(true);
        ucd.setIsEnumerated(true);
        ucd.setColumnName("UC_USER_AREA_ROLE.AREA_ID");
        ucd.setColumnTitle("User Area Role - Area Id");
        ucd.setColumnLength(getFieldLength("AREA_ID",
                             "UC_USER_AREA_ROLE",
                             dbconn));
        vodafoneDope.add(ucd);

        //build the Enumerated list

        ValidationList areaVL = DataDictionary.executePrimarySql(dbconn, "AREA");
        ucd.setValuesTitles(areaVL);

        return vodafoneDope;
      }
```

ucHomeDisplay

- Add new comment

Purpose

This is called to allow display to the home page, and allows full customization of the Home Page.

Applies To

Home Page screen

Signature

```
public String ucHomeDisplay (
      SesameSession session)   // the current sesameSession
```

Notes

This method is only called if the behavior setting named USE_ABBREVIATED_HOME_PAGE is set to NO. It allows HTML to be sent directly to the Home Page and allows up to three report_id's to be specified. These reports will be rendered on the Home Page. This method also allows dynamic substitution of the report_id's used to display reports on the Home Page.

```
public void ucHomeDisplay (
      SesameSession session,   // the current sesameSession
      HashMap reports)
```

Example

This example demonstrates how the home page is customized based upon role. The behavior setting USE_ABBREVIATED_HOME_PAGE is set to No, and this method ensures that the Client role also is automatically assigned the Approver role if it is not already set in their user profile.

```
    public String ucHomeDisplay (SesameSession session)  {   // the current sesameSession

       // if the user has the Client role, make sure they also have the Approver role
        checkClientApproverRole(session);

        return super.ucHomeDisplay(session);
    }
     /**
     * This method checks whether the current user has the Client user role
     * assigned to their account; if it does, then make sure that the Approver
     * user role is also assigned, and if not assigned then add it to their account.
     */
    private void checkClientApproverRole (SesameSession session) {
        String role = session.getUserRole();

        // check if the user is currently in the Approver role
        if (ROLE__APPROVER.equals(role) ) return;

        String userId = session.getUserId();
        Connection con = null;

        try {
            con = Z.pool.getConnection("checkClientApproverRole");

            // get all roles for current user
            ArrayList roles = SecurityGroupUser.getSecurityGroupUsersIDs(con, userId);

            // check if current user is in the Client role or has the Client role
            if (ROLE__CLIENT.equals(role) || roles.contains(ROLE__CLIENT) ) {
                // check if current user does not have the Approver role assigned
                if (! roles.contains(ROLE__APPROVER) ) {
```

```
                // add Approver role to current user
                String[] group = new String[] {ROLE__APPROVER};
                SecurityGroupUser.setSecurityGroupUser(con, group, userId, false);
                // false=add new role
                con.commit();

                Z.log.writeToLog(Z.log.INFO, "Assigned role " + ROLE__APPROVER
                                + " to user " + userId);
            }
        }
    } catch (Exception e) {
        ErrorWriter.write(e, ErrorWriter.LOGERR);
    } finally {
        if (con != null) Z.pool.close(con);
    }
}
```

ucInsertAttachmentContent

- Add new comment

Purpose

Applies To

Attachment Methods

Signature

```
public Attachment ucInsertAttachmentContent(
                              Connection dbConn,
                              Attachment attachment)
```

Notes

Example

```
public Attachment ucInsertAttachmentContent(Connection dbConn,Attachment attach){

        boolean toBeStoredInternal = attach.getToBeStoredInternal();

        if (!toBeStoredInternal) {
        String attachId = attach.getAttachmentId();
        String problemId = String.valueOf(attach.getProblemId());

        String zeros = "00000000";

        String zeroId = zeros.substring(problemId.length()) + problemId;

        String millions = zeroId.substring(0, 2);
        String thousands = zeroId.substring(2, 5);

        String rootDir = "";
        rootDir = Z.config.getConfigValue("ATTACHMENT_ROOT_DIR");
        //String rootDir = "C:/Documents and Settings/...";
        if (rootDir == null) {
            rootDir = System.getProperty("java.io.tmpdir");
        }
        String attachDir = rootDir + "/" + millions + "/"
                + thousands + "/" + problemId + "/" + attachId;
        String fileName = attach.getFileName();

        try {
            String extIdent = attach.getExternalAttachmentIdent();
            File inputFile = new File(attach.getPath());
            File outputDir = null;
            File outputFile = null;

            boolean status = false;

            outputDir = new File(attachDir);
            // Create the output directory
            if (!outputDir.exists()) {
                outputDir.mkdirs();
            }
            if (outputDir.exists()) {
                outputFile = new File(attachDir, fileName);
            }
            Z.log.writeToLog(Z.log.INFO,
                  "THE File is : " + outputFile.getAbsolutePath());
            if (inputFile != null && outputFile != null) {
                status = this.copyFile(inputFile, outputFile);
            }
            if (status) {
                attach.setExternalAttachmentIdent(outputFile.getAbsolutePath());
                attach.setStoredInExternalSystem(true);
                attach.setToBeStoredInternal(false);
                attach.setStoredLocal("N");
            }
        }
        catch (Exception e) {
            ErrorWriter.write(e, ErrorWriter.LOG);
        }
        }
        return attach;
    }
```

ucLdapSetSession SecurityUser

Purpose

ucLdapSetSessionSecurityUser sets session variables using the SecurityUser Object.

Applies To

Miscellaneous Methods

Signature

```
public boolean ucLdapSetSessionSecurityUser (
      SecurityUser su)
```

Notes

Any changes made to the SecurityUser object at this point will be reflected in the SecurityUser object and related attributes in the session. If this method returns true, the object will update the database.

Example

```
  public boolean ucLdapSetSessionSecurityUser(SecurityUser su) {
        return false;
  }
```

ucLoginAuthenticateUser

- Add new comment

Purpose

Provides an exit to authenticate users outside of ExtraView,

Applies To

User Authentication Methods

Signature

```
public boolean ucLoginAuthenticateUser (
      String id,                        // the User ID
      String pswd)                      // the password
```

Notes

Typically used when interfacing with a directory service such as LDAP.

Example

ucLoginAuthenticateUser implements the following custom authentication routine:search the LDAP directory for the uid provided; if only one entry exists, then obtain the uniqueIdentifier attribute and perform the authentication; if multiple entries exist for the same uid, then write to the log and fail the authentication.

```
    public boolean ucLoginAuthenticateUser (String id, String pswd) {
        boolean authenticated = false;
        Connection con = null;

        // Use this for LDAP Authentication
        SearchLDAP slap = new SearchLDAP();

        try {
            // Error check
            if (id == null || id.trim().length() == 0)
                throw new Exception("Empty user id!");

            if (pswd == null || pswd.trim().length() == 0)
                throw new Exception("Empty user password!");

            // For the Admin user configured for ADMIN_USER_ID parameter in
            // Configuration.properties bypass the custom authentication and
            // return false. If the CUSTOM_AUTHENTICATION behavior property
            // is set to HYBRID, then extraview will use the inbuilt authentication
            // after returning from this method.

            String adminUserId = Z.config.getConfigValue("ADMIN_USER_ID");
            if(adminUserId !=null && adminUserId.equals(id)){
                return false;
            }

            // Search the directory for the id provided
            String distinguishedName = searchForDistinguishedName(id);

            // Authenticate user
            DirContext ctx = getContext(distinguishedName.toString(), pswd);

            if (ctx != null) {
                // do a little test to see if we authenticated....
                ctx.lookup("");
                authenticated = true;

                try {
                    // we connected  - upsert the data now
                    con = Z.pool.getConnection();
                    slap.doUpsert(con, ctx, id, true);
                }
                catch (Exception e) {
                    Z.log.writeToLog(Z.log.ERROR, "UPSERT FAILED: " + e);
                    ErrorWriter.write(e, ErrorWriter.ERR);
                }
                finally {
                    ctx.close();
                    if (con != null) Z.pool.close(con);
                }
            }
        } catch (Exception e) {
            Z.log.writeToLog(Z.log.ERROR, "authentication error: " + e);
            ErrorWriter.write(e, ErrorWriter.LOGERR);
        }
        return authenticated;
    }

    /**
     * searchForDistinguishedName searches the LDAP directory for the
     * given user id and returns the distinguishedName attribute value
     * if only one exists; if multiple entries exist with the same uid,
     * then write out the info to the log and throw an exception.
     */

    private String searchForDistinguishedName (String uid)
    throws Exception {
        // Get simple directory context for searching
        Hashtable env = new Hashtable();
        env.put(Context.PROVIDER_URL, Z.lu.getInitHost() );
        env.put(Context.INITIAL_CONTEXT_FACTORY, initContextFactory);
        env.put(Context.SECURITY_AUTHENTICATION, securityAuth);
        env.put(Context.SECURITY_PRINCIPAL,Z.lu.getInitMgrDn());
        env.put(Context.SECURITY_CREDENTIALS,Z.lu.getInitPswd());
        DirContext ctx = new InitialDirContext(env);

        String primaryKeyAttr =
               Z.config.getConfigValue("LDAP_PRIMARYKEY");
        String firstNameAttr =
               Z.config.getConfigValue("LDAP_GIVENNAME");
        String lastNameAttr =
               Z.config.getConfigValue("LDAP_SURNAME");
        String distinguishedNameAttr =
               Z.config.getConfigValue("LDAP_DISTINGUISHEDNAME");
        String activeUserFilterAttr =
               Z.config.getConfigValue("LDAP_USER_FILTER_ATTR");
        String activeUserFilterAttrCriteria =
                Z.config.getConfigValue("LDAP_USER_FILTER_ATTR_CRITERIA");

        // Prepare for searching
        String filter = "(&("+primaryKeyAttr+"=" + uid + ")("+firstNameAttr+"=*)("
                +lastNameAttr+"=*)("+activeUserFilterAttr+"=*"
                +activeUserFilterAttrCriteria+"*))";
        SearchControls controls = new SearchControls();
        controls.setSearchScope(SearchControls.SUBTREE_SCOPE);

        // Search directory and collect unique identifiers
        ArrayList distinguishedNames = new ArrayList();
        NamingEnumeration ne = ctx.search(Z.lu.getInitSearchBase(), filter, controls);


        while (ne.hasMore() ) {
            SearchResult sr = (SearchResult) ne.next();
            Attributes atts = sr.getAttributes();
            Attribute att = atts.get(distinguishedNameAttr);
            String distinguishedNameString = (String)att.get();
                distinguishedNames.add(distinguishedNameString);
        }
        ctx.close();
        // Check how many entries were found
        String msg = null;

        if (distinguishedNames.isEmpty() ) {
            msg = "No entry found in LDAP for "+primaryKeyAttr+"=" + uid;
            Z.log.writeToLog(Z.log.WARN, msg);
        }
        else if (distinguishedNames.size() > 1) {
            msg = "Multiple entries found in LDAP for "+primaryKeyAttr+"=" + uid;
            Z.log.writeToLog(Z.log.ERROR, msg);
            Z.log.writeToLog(Z.log.ERROR,
               "distinguishedNames found: " + distinguishedNames);
        }

        // Either throw exception or return unique identifier
        if (msg != null)
            throw new Exception(msg);

        return (String) distinguishedNames.get(0);
    }
```

ucLoginScreenButtons

- Add new comment

Purpose

LoginScreenButtons is called from the generation of the login screen to insert buttons into the login display screen beneath the user name and password input fields.

Applies To

User Authentication Methods

Signature

```
public String ucLoginScreenButtons(
              SesameSession session)
```

Notes

Generally, the buttons should direct the service to the unsecuredServiceEntry method so that the button handling can be done in User Custom code. The return string is inserted into the HTML generated on the login screen.

Example

```
 public String ucLoginScreenButtons(SesameSession session) {
        // move this logic to user custom code for Vodafone
        String mAppHome = Z.appDefaults.getAttribute("APP_HOME");
        if (Z.config.getConfigValue("Random_Password_Pass".toUpperCase()) != null) {
            String requestPassword = "<span class=\"text\"><a class=\"text\" href=\""
                + session.rewriteURL(mAppHome
                + "ExtraView?p_option=security.LoginDisplay&p_action=
                          unsecuredServiceEntry&stateVar=doDisplay")
                + "\" target=_sel>"
                + Z.m.msg(session, "Request Password")
                + "</a>\n";
            return requestPassword + "<br>\n";
        }
        return "";
    }
```

ucMailListUpdate

- Add new comment

Purpose

This exit allows modification or replacement of the Mailing List that is created by ExtraView.

Applies To

Add Issue screen

Signature

```
public java.util.ArrayList ucMailListUpdate(
      java.sql.Connection dbcon,
      java.lang.String problemId,
      Problem prob,
      SesameSession session,
      java.util.ArrayList mailList,
      ProblemFormParam pfp)
```

Notes

It should return an ArrayList of MailList Objects. If null is returned then the normal Notification Mailing List will be generated.

Example

```
public ArrayList ucMailListUpdate (Connection con,
                                   String itemId,
                                   Problem item,
                                   SesameSession session,
                                   ArrayList mailList,
                                   ProblemFormParam values)  {
        // NOTE: THE VALUES PARAMETER SEEMS TO BE EMPTY OR CONTAIN JUST ccMail
        String requestReview = values.getString("REQUEST_REVIEW");
        String addendumAlert = values.getString("ADDENDUM_ALERT");
        String areaId = values.getString("AREA");
        String projectId = getProjectId(values);

        // THEREFORE, GET FIELDS FROM RECORD SAVED IN DATABASE
        ArrayList udfs = Problem_UDF.getReference(con, itemId);

        for (int i = 0; i < udfs.size(); i++) {
            Problem_UDF pudf = (Problem_UDF) udfs.get(i);
            String name = pudf.getName();

            if ("REQUEST_REVIEW".equals(name) )
                requestReview = pudf.getUdfListId();
            if ("ADDENDUM_ALERT".equals(name) )
                addendumAlert = pudf.getUdfListId();
        }

        if (item != null) {
            areaId = item.getAreaId();
            projectId = item.getProjctId();
        }

        // if Request Review or Addendum Alert is On, then add project Coordinators
        // and backup Coordinators to mailing list
        if (REQUEST_REVIEW__ON.equals(requestReview)
         || ADDENDUM_ALERT__ON.equals(addendumAlert) ) {
            String projectTitle = getProjectTitle(con, areaId, projectId);
            String projectAdminId = getProjectAdminId(con, projectTitle);
            ArrayList coordinators = getProjectUsers(con,
                          projectAdminId,
                          "COORDINATORS");
            String usersList = "";

            for (int i = 0; i < coordinators.size(); i++) {
                String user = (String) coordinators.get(i);
                usersList += user + ";";
            }

            Z.log.writeToLog(Z.log.INFO,
             " Request Review/Addendum Alert is On, so send email to coordinators: "
              + coordinators;

            ArrayList backupCoordinators = getProjectUsers(con,
                      projectAdminId,
                      "BACKUP_COORDINATORS");

            for (int i = 0; i < backupCoordinators.size(); i++) {
                String user = (String) backupCoordinators.get(i);
                usersList += user + ";";
            }

            Z.log.writeToLog(Z.log.INFO,
            "Request Review/Addendum Alert is On, so send email-backup coordinators: "
             + backupCoordinators);
            // also, add project Translators to mailing list when Request Review is On
            if (REQUEST_REVIEW__ON.equals(requestReview) ) {
                ArrayList translators = getProjectUsers(con,
                         projectAdminId,
                         "TRANSLATORS");

                for (int i = 0; i < translators.size(); i++) {
                    String user = (String) translators.get(i);
                    usersList += user + ";";
                }

                Z.log.writeToLog(Z.log.INFO,
                    "Request Review is On, so send email to translators: "
                     + translators);
            }

            Pr_Notify prNotify = new Pr_Notify(con, session);
            ArrayList newMailList = prNotify.getCCMailList(usersList);

            for (int i = 0; i < newMailList.size(); i++) {
                mailList.add(newMailList.get(i) );
            }
        }
        return mailList;
    }
    /**
     * NOTE: This method is called by base code (instead of ucMailListUpdate), so
     * re-directing to new method name implemented above.
     */
    public ArrayList replaceMailList (Connection dbcon,
                                      String problemId,
                                      Problem prob,
                                      SesameSession session,
                                      ArrayList mailList,
                                      ProblemFormParam pfp) {
        return ucMailListUpdate(dbcon, problemId, prob, session, mailList, pfp);
    }
```

ucMassUpdatePreUpdate

- Add new comment

Purpose

Applies To

Administration Methods

Signature

```
    public void ucMassUpdatePreUpdate(
                 Connection dbconn,
                 SesameSession session,
                 ProblemFormParam pfp,
                 FormParameters fp,
                 String pGlobal,
                 String newArea,
                 String newProject)
```

Notes

Example

```
    /*
     *  Create CLONEE_PFP so that ucEditClone method can use it to set
     *  STATUS and PRIVACY fields correctly.
     *
     */
    public void ucMassUpdatePreUpdate(Connection dbconn,
                                      SesameSession session,
                                      ProblemFormParam pfp,
                                      FormParameters fp,
                                      String pGlobal,
                                      String newArea,
                                      String newProject){
         if (pGlobal.equals("PROJECT")){
           ProblemFormParam npfp = new ProblemFormParam(fp);
           npfp.put("AREA_ID",newArea);
           String project = newArea + "|" + newProject;
           npfp.put("PROJECT_ID",project);

           session.setAttribute("CLONEE_PFP", npfp);
         }
    }
```

ucModifyColumn ReportRow

- Add new comment

Purpose

ucModifyColumnReportRow is used to modify the results displayed on each row of a column report.

Applies To

Search & Reporting screens

Signature

```
public HashMap ucModifyColumnReportRow(HashMap tags) throws Exception
```

Notes

A hashmap of the results is passed into the method, for each row of the result set. The hashmap contains all the objects to be displayed on the report, with the data dictionary name being the key to each entry. You may modify any of the entries in the hashmap and then return the hashmap to be processed by the report renderer.

Example

```
  public HashMap ucModifyColumnReportRow (HashMap tags)
    throws Exception {
        if (tags.containsKey("TIME_CRITICAL") ||
            tags.containsKey("TIME_HIGH") ||
            tags.containsKey("TIME_MEDIUM") ||
            tags.containsKey("TIME_LOW") ||
            tags.containsKey("TIME_TOTAL") ||
            tags.containsKey("CUSTOMER_TIME_CRITICAL") ||
            tags.containsKey("CUSTOMER_TIME_HIGH") ||
            tags.containsKey("CUSTOMER_TIME_MEDIUM") ||
            tags.containsKey("CUSTOMER_TIME_LOW") ||
            tags.containsKey("CUSTOMER_TIME_TOTAL") )
        {
            HashMap oldTags = (HashMap) tags.clone();

            calculateCustomerTimes(tags);

            formatTags ("TIME_CRITICAL", tags, oldTags);
            formatTags ("TIME_HIGH", tags, oldTags);
            formatTags ("TIME_MEDIUM", tags, oldTags);
            formatTags ("TIME_LOW", tags, oldTags);
            formatTags ("TIME_TOTAL", tags, oldTags);
            formatTags ("CUSTOMER_TIME_CRITICAL", tags, oldTags);
            formatTags ("CUSTOMER_TIME_HIGH", tags, oldTags);
            formatTags ("CUSTOMER_TIME_MEDIUM", tags, oldTags);
            formatTags ("CUSTOMER_TIME_LOW", tags, oldTags);
            formatTags ("CUSTOMER_TIME_TOTAL", tags, oldTags);
           }
        return tags;
    }
```

ucModifyDetailed ReportRow

- Add new comment

Purpose

ucModifyDetailedReportRow is used to modify the results of a detailed report

Applies To

Search & Reporting screens

Signature

```
public HashMap ucModifyDetailedReportRow(
      Connection dbConn,     // database connection
      SesameSession session, // the current session object
      HashMap row)           // the hashmap containing the fields
                       throws Exception
```

Notes

The results set is passed into the method as a hashmap for each record to be displayed. The hashmap contains all the objects to be displayed on the report, with the data dictionary name being the key to each entry. You may modify any entry in the hashmap and then you return the hashmap for the results to be rendered

Example

```
public HashMap ucModifyDetailedReportRow (Connection dbConn,
                       SesameSession session,
                    HashMap row) {
        // If we're outputting to the browser or to Word,
        // we will send out the mini history html
        if ("HTML".equalsIgnoreCase(session.getAttribute(
                   "REPORT_OUTPUTTYPE").toString() ) ||
            "WORD".equalsIgnoreCase(session.getAttribute(
                    "REPORT_OUTPUTTYPE").toString() ) ) {
            if (row.get("MINI_HISTORY.TEXT") != null) {
                row.put("MINI_HISTORY", getMiniHistory(dbConn, session, row) );
                return row;
            }
        }
        return super.ucModifyDetailedReportRow(dbConn, session, row);
    }
```

ucModifySpecialCauseTests ForControlChart

Purpose

This exit provides the ability to alter the Special Cause Tests that exist on cCharts and uCharts.

- Default values for the rules can be set upon report creation. It does not override existing values.
- Tests can be checked or unchecked
- Tests can be disabled.

Applies To

cCharts and uCharts

Signature

```
public void ucModifySpecialCauseTestsForControlChart(
               SesameSession session,
               SearchChartReport scr,
               ArrayList<HashMap<String,String>> specialCauseTestAttribues,
               FormParameters fp) throws Exception
```

Notes

With the following example, expect to see Test 3 being checked, read-only and with a default value of 2.

Example

```
public void ucModifySpecialCauseTestsForControlChart(
               SesameSession session,
               SearchChartReport scr,
               ArrayList<HashMap<String,String>> specialCauseTestAttribues,
               FormParameters fp)
    throws Exception{
        specialCauseTestAttributes.get(2).put("CHECKED","Y");
        specialCauseTestAttributes.get(2).put("READONLY","Y");
        specialCauseTestAttributes.get(2).put("DEFAULT","2");
}
```

ucNavBarExtension

- Add new comment

Purpose

This exit is used to place an additional element on the navigation bar.

Applies To

Administration Methods

Signature

```
    public String ucNavBarExtension(
                 Connection dbconn,
                 SesameSession session)
                        throws Exception { return null; }
```

Notes

The exit is expected to return HTML that will render within a TABLE tag.

Example

```
public String ucNavBarExtension(Connection dbconn, SesameSession session) throws Exception {
   // String to embed meebo looks like
   //<embed src="http://widget.meebo.com/mm.swf?iXnWgTEroI"
     type="application/x-shockwave-flash" width="190" height="275"></embed>

   //Put a div and a button that opens a window containing meebo

   StringBuffer html = new StringBuffer("<div style=\"position:absolute;
      left:620px; top:22px\">");

   //<input value="Live Chat" onclick="openPopupWin(300,300,'http://widget.meebo.com
     /mm.swf?iXnWgTEroI','chat');" type="button">

      html.append(" <input type=\"button\" value=\"Live Chat\" ");
      html.append(" onClick=");
      html.append("window.open('http://widget.meebo.com
        /mm.swf?iXnWgTEroI','chat','width=300,height=300')");
      html.append(">");
      html.append("</div>");

    return html.toString();
}
```

ucNavBarReplace

- Add new comment

Purpose

This exit is used to replace the standard inbuilt navigation bar with a navigation bar that is built entirely in user custom code.

Applies To

Administration Methods

Signature

```
public String ucNavBarReplace(
                 Connection dbconn,
                 SesameSession session)
```

Notes

It is the responsibility of the programmer to provide the entire HTML that will be rendered within the navigation bar. The navigation bar is still set to the size controlled by the behavior setting named MENU_SIZE, and the background image named BannerBackground.gif is still placed on the navigation bar. The standard JavaScript methods in the Onload and Onunload attributes of the BODY tag of the BODY statement are still called.

Example

```
public String ucNavBarReplace(Connection dbconn,
                              SesameSession session) throws Exception;
```

ucPostDelete

Purpose

This allows additional processing after an issue is deleted from the database.

Applies To

Add Issue and *Edit Issue* screens

Signature

```
public void ucPostDelete(
                Problem prob,
                Connection dbconn,
                SesameSession session
            ) throws Exception;
```

Notes

This is called after an issue has been deleted. Note there is no return from this method, but it allows further processing after the deletion process.

ucPreClone

- Add new comment

Purpose

This is used to preprocess the clone operation after the Clone button is pressed, and before the ExtraView internal cloning process starts.

Applies To

Edit screen buttons

Signature

```
public boolean ucPreClone(
      ProblemFormParam values,        //contains values of form params
      HttpServletRequest request,     // the servlet request
      HttpServletResponse response,   // the servlet response
      Connection con,                 // the database connection
      SesameSession session)          // session object
```

Notes

You may change any of the values in the **values** passed into the method.

Example

```
public boolean ucPreClone(
      ProblemFormParam values,        //contains values of form params
      HttpServletRequest request,     // the servlet request
      HttpServletResponse response,   // the servlet response
      Connection con,                 // the database connection
      SesameSession session) throws Exception  // session object
{
    // check to see if we have the value for the area and project for this clone.
    // if not, send the area/project prompts like in the preAdd display
    if (pfp.containsKey("uc_pre_add_indicator") &&
            "true".equals(pfp.getString("uc_pre_add_indicator")))
    {
        // get the area and project as specified by the user...
        ProblemFormParam npfp = new ProblemFormParam(pfp);
        session.setAttribute("CLONEE_PFP", npfp);

        if (session.getAttribute("CLONING_PFP") instanceof ProblemFormParam) {
            pfp.clear(); // discard preclone form parameters...
            pfp.putAll( (ProblemFormParam) session.getAttribute("CLONING_PFP"));
            return true;
        }
    }
    session.setAttribute("CLONING_PFP", new ProblemFormParam(pfp));
    return this.addMenuForClone(pfp, request, response, dbconn, session);
}
```

ucPreDelete

- Add new comment

Purpose

This is called at the beginning of the delete process prior to the actual delete.

Applies To

Edit Issue screen

Signature

```
public String ucPreDelete ( String probId,              // the problem id
                            Connection dbconn,           // a Connection
                            SesameSession session,       // current session
                            ProblemFormParam pfp )       // values from the form
```

Notes

It can return an error message to halt the process with an error condition, and its typical use is to prevent the deletion of an issue if a child issue exists.

ucReauthorizeGetParams

- Add new comment

Purpose

ucReauthorizeGetParams will return cached form parameters from a previous request, if needed.

Applies To

User Authentication Methods

Signature

```
public ProblemFormParam ucReauthorizeGetParams (
      ProblemFormParam pfp,
      SesameSession session,
      Connection dbconn )
      throws Exception
```

Notes

Additionally, any customer specific adjustment needed in the parameters is done here.

Example

```
 public ProblemFormParam ucReauthorizeGetParams(
           ProblemFormParam pfp,
           SesameSession session,
           Connection dbconn) throws Exception {
```

```
        // is there a cached_req parameter in the form parameters? If so, we
        // will be using cached form parameters. if not, we will be returning
        // the form parameters passed into method.
        String cr = pfp.getString("CACHED_REQ");

        if (TextManager.isStringVisible(cr)) {
            StatusSignatureHandler ssh = StatusSignatureHandler.getCached(cr, session);

            if (ssh != null) {
                pfp = ssh.getProblemFormParameters();

                // remove signature handler from session
                session.removeAttribute(ssh.getCacheId());
            }
        }
```

ucReauthorizeUser

- Add new comment

Purpose

ucReauthorizeUser used to redirect user to a reauthentication server, if this is directed by the workflow of processing an issue.

Applies To

User Authentication Methods

Signature

```
public boolean ucReauthorizeUser (
      HttpServletRequest request,
      HttpServletResponse response,
      ProblemFormParam pfp,
      SesameSession session,
      int sshState )
```

Notes

It is currently used when the user’s permissions in an SSO environment are not sufficient to perform an action. This routine can be used to reauthenticate the user with a different permission set.

Example

```
 public boolean ucReauthorizeUser(
               HttpServletRequest request,
               HttpServletResponse response,
               ProblemFormParam pfp,
               SesameSession session,
               int sshState) throws Exception {

        // set up a signature handler object
        StatusSignatureHandler ssh;
        // first, is there a CHANGE: state var with one of the key fields?
        String sv = pfp.getString("stateVar");

        // Log the state var
        Z.log.writeToLog(Z.log.DEBUG, "STATEVAR IN doReauth(): " + sv);

        // if you change this you also must change the list of esignature fields below.
        String fieldPat = "^CHANGE:" +
            "(P_RISK_SIG_REQUIRED)|" +
            "(P_GOVT_DCSNTREE_SIG_REQUIRED)|" +
            "(P_GOVT_DSCNTREE_RVW_SIG_REQ)|" +
            "(P_GOVT_MDR_DEC_SIG_REQ)|" +
            "(P_GOVT_REVIEW_SIG_REQ)|" +
            "(P_GOVT_OTH_DEC_SIG_REQ)|" +
            "(P_GOVT_AUS_DEC_SIG_REQ)|" +
            "(P_GOVT_CAN_DEC_SIG_REQ)|" +
            "(P_GOVT_VSR_DEC_SIG_REQ)|" +
            "(P_GOVT_VIGIL_REV_SIG_REQUIRED)|" +
            "(P_GOVT_MHLW_DEC_SIG_REQUIRED)|" +
            "(P_GOVT_MHLW_DEC_REV_SIG_REQ)" +
            "_\\d+$";

        Regex fieldParser = new Regex(fieldPat);
        boolean reauthFieldExists = fieldParser.search(sv.toUpperCase());

        // Is this refresh fired by a reauth field?
        Z.log.writeToLog(Z.log.DEBUG,
               "REFRESH FIRED BY A REAUTH FIELD: " + reauthFieldExists);

        // if no reauthFieldExists, there is no reason to do anything else, we
        // can return "false" immediately.
        if (!reauthFieldExists) {
            return false;
        }

        // next, is there a p_cached_req parameter?
        String cr = pfp.getString("CACHED_REQ");
        boolean crFieldExists = TextManager.isStringVisible(cr);

        // did we get the cached request parameter?
        Z.log.writeToLog(Z.log.DEBUG,
               "CACHED REQUEST PARAMETER EXISTS: " + crFieldExists);
        Z.log.writeToLog(Z.log.DEBUG,
               "CACEHD REQUEST PARAMETER: " + cr);

        // if reauthFieldExists and no crFieldExists, we have not yet been
        // through reauthentication. we must create the ssh object, redirect to
        // the SSO server, and return a true (we are doing reauth). Otherwise,
        // reauthField and crField both exist, get cached ssh, then check the
        // SSO authorization status.
        if (reauthFieldExists && !crFieldExists) {

            // log that we are making a new StatusSignatureHandler
            Z.log.writeToLog(Z.log.DEBUG, "MAKING NEW StatusSignatureHandler.");

            ssh = new StatusSignatureHandler(request, session, sshState);
            if (TextManager.isStringVisible(sv)) {
                ssh.setStateVar(sv);
            }

            // ssh reauth status before cache
            Z.log.writeToLog(Z.log.DEBUG,
                "SSH REAUTH STATUS BEFORE CACHE: " + ssh.isReauthorizedCheck(session));

            // cache ssh in session
            ssh.cache();

            // ssh reauth status after cache
            Z.log.writeToLog(Z.log.DEBUG,
                "SSH REAUTH STATUS AFTER CACHE: " + ssh.isReauthorizedCheck(session));
        } else {

            // we are retrieving a cached ssh
            Z.log.writeToLog(Z.log.DEBUG,
                  "GETTING CACHED StatusSignatureHandler.");

            ssh = StatusSignatureHandler.getCached(cr, session);

            // check the cached ssh for reauthorization
            // if we are not using SSO, just reauthorize
            // them. otherwise, check the SSO reauth.
            boolean sso = "YES".equalsIgnoreCase(
                       Z.appDefaults.getAttribute("SSO_STATE"));

            // log sso state
            Z.log.writeToLog(Z.log.DEBUG, "SSO_STATE IS ON: " + sso);

            if (!sso) {
                // log what we do with respect to SSO
                Z.log.writeToLog(Z.log.DEBUG, "SSO NOT ON, REAUTHORIZING.");

                ssh.reauthorize();
            } else if (com.extraview.presentation.security.LoginDisplay.
                           doSSOReAuthorization(request)) {
                // log what we do with respect to SSO
                Z.log.writeToLog(Z.log.DEBUG, "SSO ON,
                      doSSOReAuthorization PASSED, REATHORIZING.");

                ssh.reauthorize();
            } else {
                // log what we do with respect to SSO
                Z.log.writeToLog(Z.log.DEBUG,
                      "SSO ON, doSSOReAuthorization FAILED, NOT REATHORIZING.");
            }
        }

        // Log wheter or not we are reauthorized
        Z.log.writeToLog(Z.log.DEBUG,
               "SSH SAYS WE ARE REAUTHORIZED: " + ssh.isReauthorizedCheck(session));

        // if we are reauthorized, do NOT redirect and return a status of false,
        // otherwise redirect and return a status of true (for redirected).
        if (ssh.isReauthorized(session)) {
            return false;
        } else {

            String reloginUrl = ssh.reloginUrl();

            // log reloginUrl
            Z.log.writeToLog(Z.log.DEBUG,
                  "REAUTH URL WITH TARGET IMMEDIATELY BEFORE REDIRECTING: "
                 + reloginUrl);

            response.sendRedirect(reloginUrl);
            return true;
        }
    }
```

ucRefreshOnContinue

Purpose

This exit is called by ExtraView after an Update and Continue operation on an *edit* screen, allowing the user to perform actions that are specific to continuing operation after an update to the database. For example, the user may want to change some field values after the update and before the user continues to enter or update data through the user information.

Applies To

Edit Issue screen

Signature

```
public void ucRefreshOnContinue( Connection dbconn,
                                 SesameSession session,
                                 HashMap values,
                                 ArrayList refreshList) {
    }
```

Notes

Note that you may alter the contents of the `values` within the HashMap or the contents of the `refreshList`. The `refreshList` is the list of fields that will be refreshed with an Ajax call at the appropriate time(s) when editing the issue.

Example

This example simply adds an extra field to the refreshList. Note that custom fields are not automatically refreshed, therefore this example demonstrates how you can force a refresh of a custom field.

```
public void ucRefreshOnContinue( Connection dbconn,
                                 SesameSession session,
                                 HashMap values,
                                 ArrayList refreshList) {

    refreshList.add("CUSTOM_FIELD");
    return;
}
```

ucRelationshipGroup BeginEmailFilter

- Add new comment

Purpose

ucRelationshipGroupBeginEmailFilter allows you to pre-populate or change the values that are displayed on the CustomEmail filter screen

Applies To

Relationship Group Methods

Signature

```
public void ucRelationshipGroupBeginEmailFilter (
      SesameSession session,   // current session
      ProblemFormParam pfp)    // problem form params to populate
                               // the form
```

Notes

Example

ucRelationshipGroup ReplayHelp

- Add new comment

Purpose

ucRelationshipGroupReplayHelp returns a string which is the content of the help description for the replay functionality that is displayed on the Manage Relationship Groups Screen.

Applies To

Relationship Group Methods

Signature

```
public String ucRelationshipGroupReplayHelp (
      SesameSession session)    // Session object
```

Notes

Example

```
public String ucRelationshipGroupReplayHelp(SesameSession session) {
        return null;
  }
```

ucRelationshipGroup SplitMergeCommon

- Add new comment

Purpose

When merging or splitting item ID’s will either move from an existing group (update) or move into a new group (insert).

Applies To

Relationship Group Methods

Signature

```
public boolean ucRelationshipGroupSplitMergeCommon (
      SesameSession session,            // current session
      Connection con,                   // Database Connection
      RelationshipGroupPersist inserts, // list of ids to be
                                        // removed from the group
      RelationshipGroupPersist updates, // list of ids to be
                                        // moved to a new group
      String relationshipGroupId)       // relationship group
                                        // id of the group items are
                                        // being moved into
```

Notes

This method provides the list of ID’s that are being inserted and updated to the relationship_group_item table. This method returns true on success. DO NOT COMMIT within this method. ExtraView will handle the output, based on the return value. If false is returned any transactions performed in this method will be rolled back. Returning false will also rollback the split or merge of problems in the main code

Example

```
public boolean ucRelationshipGroupSplitMergeCommon(
     SesameSession session,            // current session
     Connection con,                   // Database Connection
     RelationshipGroupPersist inserts, // list of ids to be removed from the group.
     RelationshipGroupPersist updates, // list of ids to be moved to a new group.
     String relationshipGroupId) {     // relationship group id of the group items
                                       // are being moved into.
        return true;
  }
```

ucRelationshipGroup UpdateParent

- Add new comment

Purpose

This method checks takes in the RelationshipGroup object so that you can check to see if parent issue ID has changed.

Applies To

Relationship Group Methods

Signature

```
public void ucRelationshiptGroupUpdateParent (
      RelationshipGroup rg,     // Relationship group object
      Connection con,           // Database connection
      SesameSession session)    // Session object
```

Notes

This method will then update the new parent problemId with fields from the old one, pass in the connection so that this transaction can be transactional. Don't write changes if there is a problem saving the Relationship Group change.

Example

```
public void ucRelationshipGroupUpdateParent(RelationshipGroup rg,
                                              Connection con,
                                              SesameSession session) {
        return;
  }
```

ucRelationshipGroupCustom

- Add new comment

Purpose

ucRelationshipGroupCustom is the controller method for customer-specific Relationship Group display functions

Applies To

Relationship Group Methods

Signature

```
public void ucRelationshipGroupCustom (
      Connection dbconn,                // database connection
      SesameSession session,            // Session object
      HttpServletRequest request,
      HttpServletResponse response )
```

Notes

Example

```
 public void ucRelationshipGroupCustom(Connection con,
                                       SesameSession session,
                                       HttpServletRequest request,
                                       HttpServletResponse response) throws
          Exception {
        return;
  }
```

ucRelationshipGroupDelete

- Add new comment

Purpose

This method provides the list of items to be removed from a relationship group.

Applies To

Relationship Group Methods

Signature

```
public boolean ucRelationshipGroupDelete (
      SesameSession session,            // Session object
      List removeIds,                   // list of problem ids to be removed
      Connection con)                   // Connection object
```

Notes

This method is used for additional validation, and returns true on success. Do not use a COMMIT statement within this method. ExtraView will handle the COMMIT, based on the return value. If false is returned any transactions performed in this method will be rolled back. Returning false will also rollback the removal of items in the core software

Example

```
public boolean ucRelationshipGroupDelete(
                     SesameSession session,    // Session object
                     List removeIds,           // list of item ids to be removed
                     Connection con) {         // Connection object
        return true;
  }
```

ucRelationshipGroupPreAdd

- Add new comment

Purpose

This method provides the list of items that are added to a relationship group.

Applies To

Relationship Group Methods

Signature

```
public boolean ucRelationshipGroupPreAdd (
      SesameSession session,            //Session object
      List addIds,                      //list of ids to be added
      String relationshipGroupId,       //relationshipGrpId of group
                                        // to be added to
      Connection con)                   //Connection object
```

Notes

This method can be used for additional validation, and returns true on success. Do not use a COMMIT within this method. ExtraView will handle the COMMIT based on the return value. If false is returned any transactions performed in this method will be rolled back. Returning false will also rollback the addition of items to the group in ExtraView

Example

```
public boolean ucRelationshipGroupPreAdd(
        SesameSession session,      // Session object
        List addIds,                // list of ids to be added.
        String relationshipGroupId, // relationshipGrpId of group to be added to.
        Connection con) {           // Connection object
    return true;
  }
```

ucRelationshipGroupSplit

- Add new comment

Purpose

ucRelationshipGroupSplit receives the list ID’s to be split from a relationship Group and a list of ID’s that should stay in the group.

Applies To

Relationship Group Methods

Signature

```
public boolean ucRelstionshipGroupSplit (
      SesameSession session,     // current session
      Connection con,            // Database Connection
      List removeList,           // list of ids to be removed from
                                 // the group
      List keepList)             // list of ids that stay in the
                                 // original group
```

Notes

These ID’s can be used for additional processing on ID’s. Database connection that is being used to pass in these issues is used to make this transactional with the Main Split update. This method returns a Boolean which returns True if no exceptions occur in processing otherwise false. False will void the entire split transaction, and True will cause a commit.

Example

```
public boolean ucRelationshipGroupSplit(
        SesameSession session,   // current session
        Connection con,          // Database Connection
        List removeList,         // list of ids to be removed from the group.
        List keepList) {         // list of ids that are staying in the original group
    return true;
  }
```

ucRenderEmbeddedObject

- Add new comment

Purpose

This exit allows user custom code to generate arbitrary HTML that is rendered in line in a layout by coding a display type of Embedded Object.

Applies To

Field Rendering Methods

Signature

```
public String ucRenderEmbeddedObject (
                      Connection dbconn,
                      SesameSession session,
                      String layoutType,
                      DDEntry dde,
                      String ddName,
                      HashMap selectedVals,
                      String selectedVal,
                      String[] multipleVals,
                      HashMap attributes,
                      String prefix,
                      LayoutElement le,
                      int row,
                      String styleVal,
                      boolean doHiddenInput)
```

Notes

This allows a great deal a flexibility when creating layouts, but caution should be used, as there is no validity checking of the string returned, and incorrectly formed HTML can break standard functionality within the layout. Note that the prefix and doHiddenInput parameters are currently not supported, but are included for future compatibility. Likewise, the selectedVal and muitipleVals are not currently populated but provided for future expansion.

Example

```
 public String ucRenderEmbeddedObject (Connection dbconn,
                                         SesameSession session,
                                         String layoutType,
                                         DDEntry dde,
                                         String ddName,
                                         HashMap selectedVals,
                                         String selectedVal,
                                         String[] multipleVals,
                                         HashMap attributes,
                                         String prefix,
                                         LayoutElement le,
                                         int row,
                                         String styleVal,
                                         boolean doHiddenInput) {
        if (ddName.equals("SELECT_ALL_COUNTRY_CUSTOM") )
```

```
            return selectAllCustomButton(
                        dbconn,
                        session,
                        ddName);

        if ("STATUS_TRANSITION".equals(ddName) )
            return transitionBtns(
                        dbconn, session,
                        layoutType,
                        dde,
                        ddName,
                        selectedVals);

        if ("MINI_HISTORY".equals(ddName) )
            return getMiniHistory(
                        dbconn,
                        session,
                        selectedVals);

        return super.ucRenderEmbeddedObject(
                        dbconn,
                        session,
                        layoutType,
                        dde,
                        ddName,
                        selectedVals,
                        selectedVal,
                        multipleVals,
                        attributes,
                        prefix,
                        le,
                        row,
                        styleVal,
                        doHiddenInput);
    }
```

ucRenderListOptions

- Add new comment

Purpose

This method can be used to remove the None or Any items dynamically, for the select list that is generated. Note that the values can be removed permanently from lists with layout cell attributes in the web-based layout editor.

Applies To

Field Rendering Methods

Signature

```
public void ucRenderListOptions (
      HashMap results,       // hashmap of boolean add and none values
      String fieldName )     // the name of the field on the
                             // form (eg p_name)
```

Notes

The session object is not passed in. It can be referenced using – SesameSession s = SesameSession.getSession(); To get the ddname of the fieldName use – TextManager.getGlobalName(fieldName;

Example

```
public void ucRenderListOptions (
                   HashMap results,    // hashmap of boolean add and none values
                   String ddName ) {   // the name of the field on the form (eg p_name)
        if ("duplicate_bugs".equalsIgnoreCase(ddName) ) {
            results.put("addNone", new Boolean(false));
            results.put("addAny", new Boolean(false));
        }
    }
```

ucRenderListValues

- Add new comment

Purpose

ucRenderListValues allows any list, popup, or tab display to be modified by changing or replacing the ValidationList.

Applies To

Field Rendering Methods

Signature

```
public void ucRenderListValues (
      SesameSession session,    // current session.
      ValidationList selList,   // the items to display in the list or popup
      String pGlobal,           // ddname
      HashMap selectedVals,     // values selected in the list.
      HashMap attributes)       // attributes of the list. EG size.
```

Notes

Change the ValidationList to do this. The validation list is a Hashtable, and the internal value and display title pairs used to construct the html input control. The key is the udfList ID and the value is the Name that appears in the list. Do not use .clear() to clear the contents. Make a clone and use this to iterate through and remove elements from the bottom up.

Example

```
public void ucRenderListValues(
      SesameSession session,            // current session
      ValidationList selList,           // the items to be displayed in the list/popup
      String ddName,                    // ddname
      HashMap selectedVals,             // values that are selected in the list
      HashMap attributes) {             // attributes of the list. EG size
    if ("CATEGORY_SELECTOR".equals(ddName)) {
      String layoutType = (String) session.getAttribute("LAYOUT_TYPE");
      if (!TextManager.isStringInvisible(layoutType)
              && layoutType.startsWith("ADD")
              && selList != null) {
        selList.remove(UL_CATEGORY_SELECTOR_CAPA);
        selList.remove(UL_CATEGORY_SELECTOR_REG_SUB);
        selList.remove(UL_CATEGORY_SELECTOR_GROUP);
        selList.remove(UL_CATEGORY_SELECTOR_HISTORY);
      }
    }
  }
```

ucRenderLogArea

- Add new comment

Purpose

This method supports the customer platform that is created using UserCustom.

Applies To

Field Rendering Methods

Signature

```
public String ucRenderLogArea (
      String ddname,            // data dictionary name of customer log area
      SesameSession session)    // sesame session object
```

Notes

Customer fields are automatically refreshed from the main customer value and can be set using the pfp. Log Areas are not listed here. History is generated when the log area is rendered. This method will pass back the problem ID of the problem where the history should be obtained from.

Example

```
 public String ucRenderLogArea(
           String ddName,            // data dictionary name of the customer log area
           SesameSession session) {  // sesame session object
        return null;
  }
```

ucRenderUserPopup

- Add new comment

Purpose

The ucRenderUserPopup method can return a new displayVal string. This will replace the string that ExtraView was to render with the string value returned by this method.

Applies To

Field Rendering Methods

Signature

```
public String ucRenderUserPopup( SesameSession session,
                          String pGlobal,
                          String fieldName,
                          String layoutType,
                          String selectedVal,
                          String[] vals,
                          HashMap selectedVals,
                          HashMap attributes,
                          int row,
                          boolean multiple)
```

Notes

Example

```
public String ucRenderUserPopup(SesameSession session,
                                  String pGlobal,
                                  String fieldName,
                                  String layoutType,
                                  String selectedVal,
                                  String[] vals,
                                  HashMap selectedVals,
                                  HashMap attributes,
                                  int row,
                                  boolean multiple) {

        return null;
  }
```

ucReportPostamble

- Add new comment

Purpose

ucReportPostamble is used to put a postamble message on reports

Applies To

Search & Reporting screens

Signature

```
public String ucReportPostamble(
      Template tDDNames, // the template that generates the report
      int size,          // contains the current page size in rows
      int start,         // contains the row number at the beginning of the page
```

Notes

The method is called for each end-of-page event, so the size, start and total record count need to be examined or the returned string will go into each of the page endings. The string returned should map according to the template passed in, allowing columnar totals or other niceties. The method should return a string that contains the footer to be displayed on the output, just before the report footer.

Example

```
 public String ucReportPostamble (Template tDDNames,
                                    int size,
                                    int start,
                                    int total)
    throws Exception {
        SesameSession session = SesameSession.getSession();
        String reportTitle = "";
        String statusTotalFormat = "HMS";
        Report report = (Report) session.getAttribute("REPORT");

        if (report !=  null) {
            reportTitle = report.getTitle();

            if (TextManager.isStringVisible(reportTitle)
                 && reportTitle.indexOf("EXCEL:mins")>-1) {
                statusTotalFormat = "mins";
            }
        }
        return com.extraview.usercustom.TimeInStatus.ucReportPostamble(
                                   tDDNames,
                                   size,
                                   start,
                                   total,
                                   statusTotalFormat);
    }
```

ucReportPreDisplay

- Add new comment

Purpose

This routine is called before the Detailed report prints the issue.

Applies To

Search & Reporting screens

Signature

```
public boolean ucReportPreDisplay (
      SesameSession session,    // the current Sesame session
      HttpServletRequest request,
      String prob_id,
      PrintWriter out)
```

Notes

Security could be checked at this point, and returning false will prevent the report from being displayed

Example

```
 public boolean ucReportPreDisplay(SesameSession session, //the current Sesame session
                                   HttpServletRequest request,
                                   String itemId,
                                   PrintWriter out)  {
        String area = (String) session.getAttribute ("VF_UC_AREA_FILTER");
        if (A_DOC.equals(area) )
          return dh.ucReportPreDisplay(session, request, itemId,  out);

        return true;
    }
```

ucReportSetFilters

- Add new comment

Purpose

ucReportSetFilters adds filters and filter criteria to a report before the query is processed. This method is also called when a Related Issue Display is displayed within an *add* or *edit* screen.

Applies To

Search & Reporting screens

Signature

```
public void ucReportSetFilters (
      FilterGroup fg)
```

Notes

This affects all types of reports (summary, list, charts), allowing data to be segregated efficiently, for example by Customer or Privacy.

Example

```
public void ucReportSetFilters (FilterGroup fg,
                                    String mode) {
        // requirement: Customer-role user can only see own (originator) issue
        // Get current session and user
        SesameSession session = SesameSession.getSession();
        String userId = session.getUserId();
        String userRole = session.getUserRole();

        if (! "GUEST".equals(userRole) )
            return;

        // Get current filters
        ArrayList filters = fg.getFilters();

        // Create originator-specific filter
        Filter originator = new Filter("ORIGINATOR");
        originator.setOperator("in");
        originator.setConjunction("AND");
        ArrayList criteria = new ArrayList();
        criteria.add(userId);

        try {
            originator.setFilterCriteria(criteria);
        } catch (Exception e) {}

        // If no filters exist, then just add originator filter
        if (filters.isEmpty() ) {
            filters.add(originator);
            fg.setFilters(filters);
            return;
        }
        // When filters already exist, need to add originator filter to all
        // "AND" filter sets (standard or advanced)

        // Iterate through current filters and add the originator filter to
        // the end of "AND" only sets, in case
        // any filters have "OR" conjuction (which cause groups of "AND"
        // filters to be evaluated together due to precedence rules)
        ArrayList newFilters = new ArrayList();

        for (int i = 0; i < filters.size(); i++) {
            Filter f = (Filter) filters.get(i);
            String conjunction = f.getConjunction();

            if ("OR".equals(conjunction) ) {
                // Found a filter with an "OR" conjunction:
                // add originator filter
                newFilters.add(originator);
            }
            newFilters.add(f);
        }

        // Add originator filter as last filter
        newFilters.add(originator);

        // Set report filters to newly modified ones
        fg.setFilters(newFilters);
    }
```

ucReportSetUdfMultiSql

- Add new comment

Purpose

ucReportSetUdfMultiSql is used to override the SQL statement used to look up multi-value UDF’s.

Applies To

Search & Reporting screens

Signature

```
public String ucReportSetUdfMultiSql (
      String[] udfName,         // database name of the UDF
      String sqlStatement,      // standard sql statement
      String problemKey,        // alias for the problem id
      String idList,
      boolean doTitleMap)       // in clause for problem ids
```

Notes

This can be used to retrieve arbitrary information to display, using the multi value UDF list handling.

Example

```
public String ucReportSetUdfMultiSql(
                           String[] udfName, // database name of the UDF
                           String sqlStatement, // standard sql statement
                           String itemKeyAlias, // alias for the item id
                           String idList, // in clause for item ids.
                           boolean doTitleMap) {
      if (udfName == null || udfName.length() == 0)
       udfName = "RELATED_ISSUES";

      if (!"RELATED_ISSUES".equalsIgnoreCase(udfName) )
       return sqlStatement ;

      return "select i.item_id " + problemKey + ", '" + udfName
                      + "' UDF_NAME, i2.item_id TITLE " +
        " from (select item_id from item where item_id in " + idList + ") i,
                      item i2, " +
        " (select item_id, value_number from udf, item_udf " +
        "   where udf.name = 'PARENT_ID' and item_udf.udf_id = udf.udf_id) iu, " +
        " (select item_id, value_number from udf, item_udf " +
        " where udf.name = 'PARENT_ID' and item_udf.udf_id = udf.udf_id) iu2" +

        // include dummy parameter bind condition which will do nothing
        " where ? = '" + udfName + "'" +

        // exclude the current bug from the list
        "  and i2.item_id != i.item_id " +
        "  and (" +
        " (i2.item_id = iu.item_id and iu.value_number
               = i.item_id and iu2.item_id
               = iu.item_id and iu2.value_number= iu.value_number) " +// direct children

          "or (i2.item_id = iu.value_number and iu.item_id
               = i.item_id and iu2.item_id = iu.item_id
              and iu2.value_number= iu.value_number) " +// parent
          "or (i2.item_id = iu2.item_id and iu2.value_number
               = iu.item_id and iu.value_number = i.item_id) " + // direct grandchildren
          " or (i2.item_id = iu2.item_id and iu2.value_number
               = iu.value_number and iu.item_id = i.item_id) " + // peers
           " or (i2.item_id = iu2.value_number and iu2.item_id
               = iu.value_number and iu.item_id = i.item_id) " + // grandparent
        " ) order by " + problemKey + ",UDF_NAME, TITLE" ;
     }
    return sqlStatement;
  }
```

ucSAMLUpdateRedirectURL

Purpose

ucSAMLUpdateRedirectURL is called during the sign on process when SAML is enabled. It allows for the modification of the redirectURL used after a successful SAML assertion.

Applies To

The Sign On screen

Signature

```
public String ucSAMLUpdateRedirectURL(
                      HttpServletRequest  request,
                      HttpServletResponse response,
                      String redirectURL) throws Exception {
     return redirectURL;
}
```

Notes

Example

In this example, user is redirected to the extraview.com website, after a successful sign on via SAML.

```
public String ucSAMLUpdateRedirectURL(
                      HttpServletRequest  request,
                      HttpServletResponse response,
                      String redirectURL) throws Exception {
     return "https://www.extraview.com/";
}
```

ucSSOInsertUser

Purpose

This method allows for the modification of the SecurityUser object prior to commiting the entry to the database.

Signature

```
    public SecurityUser ucSSOInsertUser( HttpServletRequest request,
                                         Connection dbconn,
                                         SecurityUser su,
                                         String userId) throws Exception {
      return su;
    }
```

Notes

You may use the method to cancel a transaction by setting the `SecurityUser` object to a `null` value.

ucSearchLDAPDisplay

- Add new comment

Purpose

Applies To

Administration Methods

Signature

```
public boolean ucSearchLDAPDisplay(
                  HttpServletRequest request,
                  HttpServletResponse response,
                  Connection dbconn,
                  SesameSession session,
                  String layoutType,
                  String mode,
                  String pClass)
                       throws Exception
```

Notes

Example

```
public boolean ucSearchLDAPDisplay( HttpServletRequest request,
                                        HttpServletResponse response,
                                        Connection dbconn,
                                        SesameSession session,
                                        String layoutType,
                                        String mode,
                                        String pClass) throws Exception {
        GemsSearchLDAPDisplay.doDisplay(request, response, dbconn, session);
        return true;
    }
```

ucSearchPlanningReport

Purpose

Applies To

Custom Planning Style Reports. Custom planning style reports are typically embedded within an *add* or *edit* screen. This user custom exit allows these reports to be run outside of these screens. The base code triggers the request to run the report from an internal flag and this cannot be accessed outside the *add* and *edit* pages. A UDF field is used to pass this information and to act as a filter.

Signature

```
public boolean ucSearchPlanningReport(
                  HttpServletRequest request,
                  HttpServletResponse response,
                  Connection dbconn,
                  SesameSession session)
                       throws Exception
```

Notes

Example

- Create a user defined field to be used as the flag.
  - For example, create a field in the data dictionary as follows:

    Fixed Name = UC_PLAN_RUN
    Title to display = Execute UC Plan Report
    Display Type = Text Field
    Filter Criteria = Yes

- Modify or create a Planning Report
  - Add the following filter:

    "Execute UC Plan Report" (UC_PLAN_RUN) = Y

  - Save the Report.

- Modify the user custom exit `ucSearchPlanningReport` as follows:

```
Report report = (Report)session.getAttribute(PlanningController.SES_PLAN_MASTER_REPORT);
String title = report.getTitle();
String relGroupId = report.getHierarchyId();
if ("2".equals(relGroupId)) {
  FilterGroup filterGroup = report.getFilterGroup();
  Filter f = filterGroup.getFilter("UC_PLAN_RUN");

  // do not run unless the runUC flag has been set.
  if(f==null || (f!=null && !"Y".equals(f.getFilterCriteria().get(0)))) return;

  // check to see if the
  f = filterGroup.getFilter("REPORT_START_DATE");
  if(f==null) return;
  String reportStartDate= (String)f.getFilterCriteria().get(0);
  f = filterGroup.getFilter("REPORT_END_DATE");
  if(f==null) return;
  String reportEndDate = (String)f.getFilterCriteria().get(0);

  // set start date in session so that the report begins 7 days before the delivery date.
  try {
    if (TextManager.isStringVisible(reportStartDate)) {
    Calendar dateCal = Convert.getCalendarDayFromString(session, reportStartDate);
    dateCal.add(Calendar.DATE,-7);
    Date startDt = dateCal.getTime();
    session.setAttribute(PlanningController.SES_START_DT, startDt);
  }
} catch (Exception subE) {
  Z.logtrace(subE, Log.ERROR);
}
// exit if the report start and end dates are not set.
if(   TextManager.isStringInvisible(reportStartDate)
   || TextManager.isStringInvisible(reportEndDate)) return;
// remove these fields from the report because we do not want them used
// to filter the results.
filterGroup.removeFilter("REPORT_START_DATE");
filterGroup.removeFilter("REPORT_END_DATE");
filterGroup.removeFilter("UC_PLAN_RUN");
// set the sort order based on Product Name.
SortOrder so = new SortOrder();
SortOrderField sof = new SortOrderField();
sof.setConnection(dbconn);
sof.setDDName("PRODUCT_NAME_UDF");
sof.setOrderRank(0);
sof.setSortDirection("ASC");
so.addField(sof);
report.addSortOrder(so);

// the Report Start Date and the Report End Date are required by UserJavaScript.
// Place these values in session, so that they are placed on the planning report.
String ucParams = "={p_from_date_delivery : \""+deliveryDate+"\",
                  p_to_date_pickup : \""+collectionDate+"\"};";
session.setAttribute("UC_RUN_PLANNING_CUSTOM",ucParams);
}
```

- `Execute custom JavaScript based on the contents of the ucParams above.`

```
/** * ucPlanningReportPostLoad */
function ucPlanningReportPostLoad(colMove, pageMove) {
  var runUC = false;
  runUC = (typeof(ucParams) != 'undefined'
           && typeof(ucParams.p_from_date_delivery) != 'undefined');
  try {
    if ( runUC) {
      // do stuff
      alert(" Calling from ucPlanningReportPost Load");
      // changing background colors based on a custom ajax call
    } catch (err) {
      // ignore the error for now...
      alert('ucPlanningReportPostLoad:'+ err);
    }
```

ucSearchPostSubmit

- Add new comment

Purpose

This can be used to pass in custom arguments into a Quicklist search.

Applies To

Search & Reporting screens

Signature

```
public void ucSearchPostSubmit (
      SesameSession session,  // contains values of form elements
      FormParameters fp,      // contains values of form params
      Connection con )        // DB connection
```

Notes

Used in custom buttons.

Example

```
public void ucSearchPostSubmit(
              SesameSession session, // contains values of form elements
              FormParameters fp,     // contains values of form params
              Connection con ) {     // DB connection.
        //Call Best Practices:
        super.ucSearchPostSubmit(session, fp, con);
    }
```

ucSearchUserCustom

- Add new comment

Purpose

ucSearchUserCustom is a general pass through method that is called by Search.

Applies To

Search & Reporting screens

Signature

```
public  void ucSearchUserCustom(HttpServletRequest request,
                  HttpServletResponse response,
                  Connection dbconn,
                  SesameSession session) throws Exception {
```

Notes

Example

```
public  void ucSearchUserCustom (
                             HttpServletRequest request,
                             HttpServletResponse response,
                             Connection dbconn,
                             SesameSession session)
                             throws Exception {
        FormParameters fp = new FormParameters(request);

        if (fp.containsKey("report_id") ) {
            session.setChildSession();

            // add html tags around response output
            PrintWriter out = response.getWriter();
            out.println("");
            DashboardReport dbr = new DashboardReport(session, request, response);
            dbr.doRunReport(response, dbconn, session); // add html tags around response output
            out.println("");
            session.setParentSession();
        } else {
            super.ucSearchUserCustom(request, response, dbconn, session);
        }
}
```

ucSelfRegistrationEmailList

Purpose

The ucSelfRegistrationEmailList method is used to manipulate the list of email recipients following a user performing a self-registration from the sign on screen.

Signature

```
public String ucSelfRegistrationEmailList ( Connection dbConn,
                                            ArrayList users)
```

Notes

`users` is an `ArrayList` of `SecurityUser` objects. Initially the email recipients is a list of all user who have the `ADMIN` role. This method allows user custom code to add or delete from this list.

If a null list of `users` is returned by this method, no email notification is sent.

ucSetAreaProject

- Add new comment

Purpose

Applies To

Administration Methods

Signature

```
    public void ucSetAreaProject(
                Connection dbconn,
                SesameSession session,
                String areaId,
                String projectId)
                     throws Exception { }
```

Notes

Example

```
public void ucSetAreaProject(
                  Connection con,
                  SesameSession session,
                  String areaId,
                  String projectId) throws Exception {
        int newAreaIdInt = Integer.parseInt(areaId);
        int newProjectIdInt = Integer.parseInt(projectId);
        setAreaProjectRole(con, session, newAreaIdInt, newProjectIdInt);
    }
```

ucStoreAttachment

- Add new comment

Purpose

Applies To

Attachment Methods

Signature

```
public Attachment ucStoreAttachment(
                             Attachment attachment)
```

Notes

Example

```
public Attachment ucStoreAttachment(Attachment attach) { // session object

        String attachId = attach.getAttachmentId();
        String problemId = String.valueOf(attach.getProblemId());

        String zeros = "00000000";

        String zeroId = zeros.substring(problemId.length()) + problemId;

        String millions = zeroId.substring(0, 2);
        String thousands = zeroId.substring(2, 5);
        String rootDir = "";
        rootDir = Z.config.getConfigValue("ATTACHMENT_ROOT_DIR");

        if (rootDir == null) {
            rootDir = System.getProperty("java.io.tmpdir");
        }
        String attachDir = rootDir + "/" + millions + "/" + thousands + "/" +
                              problemId + "/" + attachId;
        String fileName = attach.getFileName();

        try {
            File inputFile = attach.getTempFile();
            File outputDir = null;
            File outputFile = null;
            long filesize = inputFile.length();

            boolean status = false;

            outputDir = new File(attachDir);
            // Create the output directory
            if (!outputDir.exists()) {
                outputDir.mkdirs();
            }
            if (outputDir.exists()) {
                outputFile = new File(attachDir, fileName);
            }
            Z.log.writeToLog(Z.log.INFO, "THE File is : " + outputFile.getAbsolutePath());
            if (inputFile != null && outputFile != null) {
                status = this.copyFile(inputFile, outputFile);
            }
            if (status) {
                attach.setExternalAttachmentIdent(outputFile.getAbsolutePath());
                attach.setStoredInExternalSystem(true);
                attach.setToBeStoredInternal(false);
                attach.setStoredLocal("N");
            }
        }
        catch (Exception e) {
            ErrorWriter.write(e, ErrorWriter.LOG);
        }
        return attach;
    }
```

ucUAGenerateHTML

- Add new comment

Purpose

This display is used on the user account administration screen.

Applies To

User Account Display

Signature

```
public boolean ucUAGenerateHTML(
               String mode,
               StringBuffer html,
               FormParameters fp,
               SecurityUser su,
               HttpServletRequest request,
               HttpServletResponse response,
               Connection dbconn,
               SesameSession session,
               int tabIndex) throws Exception
```

Notes

Example

```
public boolean ucUAGenerateHTML (String mode,
                                     StringBuffer html,
                                     FormParameters fp,
                                     SecurityUser su,
                                     HttpServletRequest request,
                                     HttpServletResponse response,
                                     Connection dbconn,
                                     SesameSession session,
                                     int tabIndex)
    throws Exception {
        Z.log.writeToLog(Z.log.DEBUG, "tabIndex is + " + tabIndex);

        if (tabIndex == 0)
        {

     // getQuickAssignHtml generates the HTML code for displaying the QuickAssign tab:
     // instructions at the top and a list of 10 user fields to select users for.

            String tabHtml = getQuickAssignHtml(mode, fp, dbconn, session, su);
            Z.log.writeToLog(Z.log.DEBUG, "tabHtml: " + tabHtml);
            html.append(tabHtml);
        }
        return true;
    }
```

ucUAGetAttributesDescription

Purpose

This method is used on the user account administration screen. It is used to provide an alternative set of directions to the user on the User Attributes tab.

Applies To

User Account Display

Signature

```
public String ucUAGetAttributesDescription (
                       Connection dbconn,
                       SesameSession session ) throws Exception;
```

Notes

This method simply replaces the text within the directions on the User Attributes tab of the user accounts maintenance screens with a specific set of directions that are geared to the custom implementation of user attributes.

ucUAGetNewTitles

Purpose

This display is used on the user account administration screen. This method can be used to customized the tab titles on the screen.

Applies To

User Account Display

Signature

```
public String[] ucUAGetNewTitles(Connection dbConn, SesameSession session)
   throws Exception
```

Notes

This method returns a one dimension String array. The caller specifies the tab title for the desired tab position and leaves the remainder of the array positions with a null value. The null entry implies there is no replacement of the standard title with a user defined title for that tab position. The base code uses the following titles: [0] - Basic Information [1] - Personal Options [2] - Report Options [3] - Notification Options [4] - Privacy Groups [5] - Attributes

Example

This example changes the last tab title which has a default title of **ATTRIBUTES** to **MY ATTRIBUTES**.

```
    public String[] ucUAGetNewTitles(Connection dbconn, SesameSession session)
      throws Exception {
        String[] newTitles = new String[6];
        newTitles[5] = Z.m.msg(session, "My Attributes");// note the use of Z.m.msg to
                                                         // provide a localizable title
        return newTitles;
    }
```

ucUAGetTabs

- Add new comment

Purpose

This display is used on the user account administration screen. It is called to obtain a list of tabs to be added to the User Account Screen.

Applies To

User Account Display

Signature

```
public String[][] ucUAGetTabs(
                 String mode,
                 Connection dbconn,
                 SesameSession session) throws Exception {
```

Notes

Example

```
public String[][] ucUAGetTabs(String mode,
                                   Connection dbconn,
                                   SesameSession session) throws Exception {

      //check security permissions
      String mUser = session.getUserId();
      int mArea = session.getArea();
      int mProject = session.getProject();

      boolean userAreaRolePerm =SecurityPermission.getObjectAccess(
                                          "UC_USER_AREA_ROLE",
                                           SecurityPermission.WRITE,
                                           mUser, mArea, mProject, session);
    boolean userUserGroupPerm = SecurityPermission.getObjectAccess(
                                          "UC_USER_USER_GROUP",
                                           SecurityPermission.WRITE,
                                           mUser, mArea, mProject, session);
    boolean userSecurityGroupPerm = SecurityPermission.getObjectAccess(
                                           "SE_SECURITY_GROUP",
                                           SecurityPermission.WRITE,
                                           mUser, mArea, mProject, session);

    String[][] tabs = null;
    if (userAreaRolePerm && (userUserGroupPerm && userSecurityGroupPerm)) {
        tabs = new String[][] {
          new String[] {"UserAreaRole", Z.m.msg(session, "Area/Role Mapping"), ""}
          new String[] {"UserUserGroups", Z.m.msg(session, "User Groups"), ""}
        };

    } else if (userAreaRolePerm && userSecurityGroupPerm) {
      tabs = new String[][] {
        new String[] {"UserAreaRole", Z.m.msg(session, "Area/Role Mapping"), ""},
        new String[] {"UserUserGroups", Z.m.msg(session, "User Groups"), "adminOnly"}
      };
    } else if (userUserGroupPerm ) {
      tabs = new String[][] {
        new String[] {"UserAreaRole", Z.m.msg(session, "Area/Role Mapping"), "adminOnly"},
        new String[] {"UserUserGroups", Z.m.msg(session, "User Groups"), ""}
      };
    }
      return tabs;
  }
```

ucUAPostUpdate

- Add new comment

Purpose

Applies To

User Account Display

Signature

```
public void ucUAPostUpdate(
             Connection dbconn,
             SesameSession session,
             String action,
             SecurityUser su,
             SecurityUser oldSu) { }
```

Notes

The action values used to call this method are:

add	when adding users through the GUI

add_api	When adding users through the API
edit	When editing users through the GUI
delete	When editing users through the GUI

Example

```
public void ucUAPostUpdate(Connection con,
                                    SesameSession session,
                                    String action,
                                    SecurityUser su,
                                    SecurityUser oldSu) {
        String u4New =su.getUserField4();
        String u4Old = null;
        if (oldSu != null)
            u4Old = oldSu.getUserField4();
        if (u4New == null && u4Old == null)
            return;

        if (u4New == null)  u4New = "";
        if (u4Old == null)  u4Old = "";

        if (u4New.equals(u4Old))
            return;
        if (!"DENY".equals(u4New) && !"DENY".equals(u4Old))
          return;

        String msg = null;
        if ("DENY".equalsIgnoreCase(u4New))  msg = "Account is on Administrative Hold.";

        PreparedStatement ps = null;
        ResultSet rs = null;
        String udfId = Z.dictionary.getUdfId("ACCOUNT_HOLD_STATUS");
        if (udfId == null)  {
            Z.log.writeToLog (Z.log.ERROR, "Missing UDF for ACCOUNT_HOLD_STATUS");
            return;
        }
        try {
            ps = con.prepareStatement ("select item_id from item where originator = ?
                        and status != 'CLOSED'");
            ps.setString(1, su.getId());
            rs = ps.executeQuery();
            while (rs.next()) {
                String id = rs.getString(1);
                UcUtil.updateIssueUdfs (id, "u", udfId, "v", msg, session);
            }

            rs.close();
            ps.close();
        } catch (Exception e){
            ErrorWriter.write(e, ErrorWriter.LOGERR);
        } finally {
            try { if (rs != null)  rs.close(); } catch (Exception e) {}
            try { if (ps != null)  ps.close(); } catch (Exception e) {}
        }
    }
```

ucUATransactUser

- Add new comment

Purpose

This display is used on the user account administration screen. It is called after all other modifications to the database are complete for UserAccountsDisplay.

Applies To

User Account Display

Signature

```
public boolean ucUATransactUser(
                       String mode,
                       FormParameters fp,
                       SecurityUser su,
                       HttpServletRequest request,
                       HttpServletResponse response,
                       Connection dbconn,
                       SesameSession session) throws Exception {
       return true;
   }
```

Notes

Errors should be communicated to the user by setting the ALERT attribute in the current session.

Example

```
public boolean ucUATransactUser(String mode,
                                    FormParameters fp,
                                    SecurityUser su,
                                    HttpServletRequest request,
                                    HttpServletResponse response,
                                    Connection dbconn,
                                    SesameSession session) throws Exception {
        // Don't perform transaction if the area/role form parameter doesn't exist
        // (in case of non-admin user editing account settings)
        if (! fp.containsKey("p_area_role") )
            return true;

        // Create new area/role list based on form parameters list
        String[] areaRoles = fp.getArray("p_area_role");
        ArrayList areaRoleList = new ArrayList();

        for (int i = 0; i < areaRoles.length; i++) {
            // Get the role id of this area/role mapping
            String value = areaRoles[i]; // formatted as "|"
            int lastIndex = value.lastIndexOf('|');
            String roleId = value.substring(lastIndex + 1);

            if (!roleId.equals("{NULL}")) // ignore none values
                areaRoleList.add(value);
        }

        // Get the id of the user account
        String userId = su.getId();
        String updateUserId = session.getUserId();
        String message = null;
        // Update the database with the area/role mappings
        try {
            if (mode.equals("doAdd"))
                UserAreaRole.addMultiple(dbconn, userId, areaRoleList, updateUserId);
            else
                UserAreaRole.editMultiple(dbconn, userId, areaRoleList, updateUserId);
        }
        catch (Exception e) {
            message = "Error updating Area/Role Mapping.  Please see logged error.";
            Z.log.writeToLog(Z.log.ERROR, "ucUATransactUser Exception");
            ErrorWriter.write(e, ErrorWriter.LOGERR);
        }

        dbconn.commit();

        // Communicate error to user
        if (message != null) {
            session.setAttribute("ALERT", "\n");
            return false;
        }
        return true;
    }
```

ucUAValidateForm

- Add new comment

Purpose

This display is used on the user account administration screen. It is called to validate all of the form parameters returned from the user before database modifications are made.

Applies To

User Account Display

Signature

```
public boolean ucUAValidateForm(
                      String mode,
                      FormParameters fp,
                      SecurityUser su,
                      HttpServletRequest request,
                      HttpServletResponse response,
                      Connection dbconn,
                      SesameSession session) throws Exception
```

Notes

For unsuccessful return, the ALERT attribute in the current session should be set. The form will be refreshed, resulting in an alert to the user.

Example

```
public boolean ucUAValidateForm(String mode,
                                   FormParameters fp,
                                   SecurityUser su,
                                   HttpServletRequest request,
                                   HttpServletResponse response,
                                   Connection dbconn,
                                   SesameSession session) throws Exception {

       // Don't validate if the area/role form parameter doesn't exist
       // or if we don't have visible permission on these fields
       // (in case of non-admin user editing account settings)
       //check security permissions

       String mUser = session.getUserId();
       int mArea = session.getArea();
       int mProject = session.getProject();

       boolean userUserGroupPerm =
                  SecurityPermission.getObjectAccess(
                                      "UC_USER_USER_GROUP",
                                      SecurityPermission.WRITE,
                                      mUser, mArea, mProject, session);
       boolean userSecurityGroupPerm =
                  SecurityPermission.getObjectAccess(
                                      "SE_SECURITY_GROUP",
                                       SecurityPermission.WRITE,
                                       mUser, mArea, mProject, session);

       if ((userUserGroupPerm && userSecurityGroupPerm ) &&
            fp.containsKey("p_area_role")  ){


       String[] roles = fp.getArray("p_user_group");

       // Create ArrayList from roles String array, can use ArrayList.contains() method later
       ArrayList rolesList = new ArrayList();

       for (int r = 0; r < roles.length; r++)
           rolesList.add(roles[r]);

       String[] areaRoles = fp.getArray("p_area_role");
       String invalidRoles = null;

       // Check that the area/role mapping selected matches with the list
       // of roles the user has access to
       for (int i = 0; i < areaRoles.length; i++) {

           // Get the role id of this area/role mapping
           String value = areaRoles[i]; // formatted as "|"
           int barIndex = value.indexOf('|');
           String roleId = value.substring(barIndex + 1);

           // Verify that this role is in the list of user's allowed roles,
           // ignoring special none value
           if (!roleId.equals("{NULL}") && !rolesList.contains(roleId)) {
               if (invalidRoles == null)
                   invalidRoles = roleId;
               else
                   invalidRoles += ", " + roleId;
           }
       }

       // If any invalid roles have been found, set the ALERT session attribute
       if (invalidRoles != null) {
           session.setAttribute("ALERT", "\n");
           return false;
       }
```

```
        }
        return true;
    }
```

ucUnsecuredServiceEntry

- Add new comment

Purpose

UnsecuredServiceEntry: called from an unsecured environment via the "unsecuredServiceEntry" invoked in the LoginDisplay class. This allows a pass-through of an unsecured conversation to the user custom code.

Applies To

User Authentication Methods

Signature

```
public void ucUnsecuredServiceEntry(
               HttpServletRequest request,     // the servlet request
               HttpServletResponse response,   // the servlet response
               Connection dbconn,              // the database connection
               SesameSession session)          // current session
                    throws Exception
```

Notes

You must be very careful not to do anything that would violate security concerns in this method, because there is no prior check for passwords, or the User ID, or for session cookies. This is currently used for an anchor call (action = unsecuredServiceEntry, option = LoginDisplay) in a form for modifying the user's password, which may be expired.

Example

```
public void ucUnsecuredServiceEntry(
                HttpServletRequest request,      // the servlet request
                HttpServletResponse response,    // the servlet response
                Connection dbconn,               // the database connection
                SesameSession session)           // current session
                throws Exception {

        String stateVar = "doDisplay";
        if (request.getParameter("stateVar") instanceof String) {
            stateVar = (String) request.getParameter("stateVar");
        }
        if (stateVar != null && "doEdit".equals(stateVar)) RequestPasswordDisplay.doEdit(
                                      request, response, dbconn, session);
        else RequestPasswordDisplay.doDisplay(
                                      request, response, dbconn, session);
    }
```

ucUploadClone

Purpose

When an upload object is contained in an issue that is being cloned, and the upload object is a cloud/repository-based object, this user custom method can be invoked to populate a cloned object. At a minimum, any thumbnails should be copied, and other repository operations may be needed.

Applies To

File attachments

Signature

```
public String ucUploadClone ( Object uploadObject,
                              SesameSession session,
                              Connection dbconn,
                              String uploadObjectType,
                              Object newUploadObject ) throws Exception ;
```

Notes

In the clone operation on the edit screen, contained upload objects must be replicated into the new issue. In the interface, the newUploadObject is the partially populated new upload object (Attachment, ImageItemUDF, or DocumentItemUDF) that can be used to perform the transaction to store it in the database. For more information, please see here.

ucUploadDeleteObject

Purpose

This is used to modify the metadata associated with a file attachment.

Applies To

File attachments

Signature

```
public String ucUploadDeleteObject ( SesameSession session,
                              Connection dbconn,
                              String uploadObjectType,
                              Object uploadObject ) throws Exception ;
```

Notes

The user custom exit ucDeleteAttachment is not invoked with cloud upload objects. The string returned by these methods is an error message that will alert the user in the edit/delete screen; null for successful transaction. For more information, please see here.

ucUploadEditObject

Purpose

This is used to modify the metadata associated with a file attachment.

Applies To

File attachments

Signature

```
public String ucUploadEditObject ( HashMap formParameters,
                              SesameSession session,
                              Connection dbconn,
                              String uploadObjectType,
                              Object uploadObject ) throws Exception ;
```

Notes

For attachments, some parts of the object metadata are editable, e.g. the file description. There is a user custom exit for validation/modification of the edited metadata for cloud/repository upload objects. User modification of upload objects is restricted to the file description, also known as ALT_TEXT, and the charset encoding. A user custom exit is provided to process the modifications before they are made to the object. This exit may validate the parameters, produce an error message, or allow the transaction to proceed. For more information, please see here.

ucUploadFinish

Purpose

After the completion of a file upload operation, there may be resources that were allocated during the upload that need to be recycled.

Applies To

File attachments

Signature

```
public String ucUploadFinish ( Object uploadObject,
                              SesameSession session,
                              Connection dbconn,
                              String uploadObjectType ) throws Exception ;
```

Notes

The ucUploadFinish operation permits cleanup, as it is the last operation performed after the upload is complete. The String return value is an error message or null if successful. For more information, please see here.

ucUploadGetContent

Purpose

This is used to get a copy of the uploaded file content for email inclusion..

Applies To

File attachments

Signature

```
public InputStream ucUploadGetContent ( Object uploadObject,
                              SesameSession session,
                              Connection dbconn,
                              String uploadObjectType ) throws Exception ;
```

Notes

The binary content of the uploaded object must be made available for email inclusion. Emails that include attachments, images, or documents generally have a thumbnail of the object in addition to the binary object in its entirety, not just a link. The ucUploadGetContent method is invoked to populate the email message with the content of the object. The thumbnail is obtained directly from the object in the database. The InputStream returned value must stream the binary contents of the upload object for inclusion in emails. For more information, please see here.

ucUploadProcessForm

Purpose

This is used to process a file upload form as part of a trnsaction to interface with an external, cloud-based file repository.

Applies To

File attachments

Signature

```
public String ucUploadProcessForm ( HashMap formParameters,
                              SesameSession session,
                              Connection dbconn,
                              String uploadObjectType,
                              Object uploadObject ) throws Exception ;
```

Notes

While it is possible for the repository DIV to generate a form submission that bypasses all existing base ExtraView code, there would be substantial work involved in the new user custom service that simply replicated existing code. Therefore, it is strongly recommended that the upload HTML designer use the existing action/option and HTML FORM for submission to the server. A user custom exit is made available to process the form parameters and request object received from the upload page. This user custom exit is responsible for:

- Parameter validation
- Upload object creation (attachment, image_item_udf, or document_item_udf) – this includes thumbnail generation
- Upload object insertion to the database

This custom exit is not responsible for:

- Providing the service action/option method
- Collecting the parameters from the multipart form
- Closing the upload popup

Once the user has entered the necessary information to the upload screen, the form is submitted as a multipart form, and the ExtraView base code parses the request. The result consists of form parameters in a HashMap, representing the values entered, or pre-seeded on the upload page. If the page is submitted in the Repository mode, the form parameters and the request are passed to the user custom exit ucUploadProcessForm defined here.

A new, partially constructed uploadObject is passed as a parameter. This will be either an Attachment, a DocumentItemUDF, or an ImageItemUDF object that can be used in creating the upload object in the database. In this object the required fields that are populated are: ITEM_ID, UDF_ID (if any), CREATED_BY_USER and UPDATED_BY_USER. Other system-generated fields such as LAST_DATE_UPDATED are populated during transactions on the upload object. Aside from the pre-populated fields and the system-generated fields, the user custom exit is responsible for setting the object values using its setter methods. The form parameters here do not support multi-valued parameters. This is a legacy-based restriction due to the way the parameters are handled internally in ExtraView. If multiple values are needed for a specific parameter, it is recommended that the parameter name be uniquefied with numbers, e.g., file1, file2, file3, …. in order to work around this restriction.

The string returned by this method is an error message that will alert the user in the upload screen; null for successful upload. For more information, please see here.

books:
ExtraView 7.0

ucUploadSetForm

Purpose

This is used to interact with the file upload form as part of a transaction to interface with an external, cloud-based file repository.

Applies To

File attachments

Signature

```
public void ucUploadSetForm ( HashMap tags,
                              SesameSession session,
                              Connection dbconn,
                              String uploadObjectType ) throws Exception ;
```

Notes

Each of these operations requires specific processing by user custom exits when the upload object is in the cloud or a non-ExtraView repository. The user custom exits defined in this section are the only ones used for cloud/repository upload objects. There are other, specific exits that deal directly with file Attachments, for example, ucDeleteAttachment, but these should not be used since they may or may not be invoked for cloud/repository upload objects. The only legacy interfaces used for these upload types are ucViewAttachment and ucViewDocument. The upload process consists of the following high-level, user-initiated, operations:

Upload Screen Interaction - ucUploadSetForm

A user custom exit is made available to populate an HTML DIV on the upload page with HTML created within the user custom exit. The DIV will become visible based on the MODE of the upload screen. This is selectable by the user from one of three modes: Drag and Drop, Standard and Repository. All messages and button titles visible by the user are customizable. The user custom exit modifies a tags HashMap that is used with the inbuilt addAttachment.html template to produce the upload page. There are two mandatory tags of interest to the user custom callout:

- NO_REPOSITORY – should be set to “false” to include the repository DIV
- REPOSITORY_DIV_HTML – should be set to the HTML to include inside the DIV used for repository upload

Other tags of interest are:

- REPOSITORY_TOOLTIP – is the tooltip shown in a balloon for mouseover event on the repository anchor image
- USE_REPOSITORY_UPLOAD_MSG – is the text shown for the repository mode switch anchor

The submit button used for standard mode form submission is in the tags HashMap with the key **SUBMIT_BUTTON**. This can be reused for inclusion in the repository html. The parameter uploadObjectType is either ATTACHMENT, IMAGE or DOCUMENT. For more information, please see here.

books:
ExtraView 7.0

ucUpsertUserRecord

- Add new comment

Purpose

Applies To

User Account Display

Signature

```
public ArrayList ucUpsertUserRecord(
                Connection dbconn,
                SesameSession session,
                SecurityUser su,
                HashMap suHm,
                ArrayList UFIColumnDope )
```

Notes

Example

```
public ArrayList ucUpsertUserRecord(
                   Connection dbconn,
                   SesameSession session,
                   SecurityUser su,
                   HashMap suHm,
                   ArrayList UFIColumnDope )
                       throws Exception {

      //Loop through the UFIDope array list, for each table, pick up the column name
      //then grab the values, if any, from the suHm and do that insert
      ArrayList uarSecGroupIdList = new ArrayList();
      ArrayList uarAreaIdList = new ArrayList();
      ArrayList ugUserGroupIdList = new ArrayList();
      String tempVal = new String();
      for (int x=0; x< UFIColumnDope.size(); x++) {
        tempVal = "";
        UFIColumnDope ufi = (UFIColumnDope)UFIColumnDope.get(x);
        String columnName = ufi.getColumnName();

        //UC_USER_AREA_ROLE.SECURITY_GROUP_ID is a column name
        //ADMIN is the value OP_SPECIALE

        //UC_USER_AREA_ROLE.AREA_ID is a column name
        //Fault Tracking is the value
        //Conformance is the value
        if (suHm.containsKey(columnName)) {

          if (suHm.get(columnName) instanceof ArrayList){
             //grab each arrayList and keep it until the end of the columnDope loop
             if ("UC_USER_AREA_ROLE.SECURITY_GROUP_ID".equalsIgnoreCase(columnName)) {
               uarSecGroupIdList = (ArrayList)suHm.get(columnName);
             } else if ("UC_USER_AREA_ROLE.AREA_ID".equalsIgnoreCase(columnName)) {
               uarAreaIdList = (ArrayList) suHm.get(columnName);
             } else if ("USER_GROUP_ID".equalsIgnoreCase(columnName)) {
               ugUserGroupIdList = (ArrayList) suHm.get(columnName);
             }
          } else {
            if ("UC_USER_AREA_ROLE.SECURITY_GROUP_ID".equalsIgnoreCase(columnName)){
              tempVal = (String)suHm.get(columnName);
              uarSecGroupIdList.add(tempVal);
            } else if ("UC_USER_AREA_ROLE.AREA_ID".equalsIgnoreCase(columnName)) {
              tempVal = (String)suHm.get(columnName);
              uarAreaIdList.add(tempVal);
            } else if ("USER_GROUP_ID".equalsIgnoreCase(columnName)){
              tempVal = (String)suHm.get(columnName);
              ugUserGroupIdList.add(tempVal);
            }
          }
        }
      }

      //now that we've looped through all of the columns for this user, we should
      //have a handful of ArrayLists that we can pull data out of and insert appropriate
      //values into the system for the current user.

      //first we'll work on the UC_USER_AREA_ROLE table
      ArrayList areaRoleList = new ArrayList();

      for (int y=0; y<uarAreaIdList.size(); y++) {
        String areaVal = new String();
        String roleVal = new String();

        areaVal = (String)uarAreaIdList.get(y);

        if (uarSecGroupIdList.size()>0 && uarSecGroupIdList.size()<=y) {
          roleVal = (String)uarSecGroupIdList.get(y);
        }
        if (TextManager.isStringVisible(areaVal) && TextManager.isStringVisible(roleVal) ) {
          areaRoleList.add(areaVal + "|" + roleVal);
        }
      }

      //when inserting users into user area roles, we'll also use the standard
      //methods so that we're consistent with the GUI methodology
      if (areaRoleList.size()>0){
        UserAreaRole.addMultiple(dbconn, su.getId(), areaRoleList, session.getUserId());
      }

      //now let's deal with the USER_GROUP stuff
      //when inserting users into user groups, we want to use the custom methods
      //so that the user is also added to each project, privacy group etc etc
      for (int y=0; y<ugUserGroupIdList.size(); y++){
        String userGroupId = (String)ugUserGroupIdList.get(y);
        if (TextManager.isStringVisible(userGroupId)) {
          Z.log.writeToLog(Z.log.DEBUG, "adding " + userGroupId + " for user " + su.getId());
          UserGroup.addUserToUserGroup(dbconn, userGroupId, su.getId());
        }
      }
      return null;
    }
```

ucValidateUserRecord

Purpose

Applies To

User Account Display

Signature

```
public ArrayList ucValidateUserRecord(
                Connection dbconn,
                SesameSession session,
                SecurityUser su,
                HashMap suHm,
                ArrayList UFIColumnDope )
                      throws Exception
```

Notes

Example

```
public ArrayList ucValidateUserRecord(
                   Connection dbconn,
                   SesameSession session,
                   SecurityUser su,
                 HashMap suHm,
                 ArrayList UFIColumnDope ) throws Exception {
  ArrayList validationErrors = new ArrayList();
  SesameMessageFormat smf = new SesameMessageFormat(dbconn, session);

  for (int x=0; x 0) {
          String myProjectString = (String) myProjectId.get(0);
          if (!myParentProjects.add(myProjectString)) {
            err = true;
          }
        }
        if (err) {
          smf.clear();
          smf.append("The user cannot be added to the group ");
          smf.appendString(myUserGroupId);
          smf.append(" because the user already belongs to a User Group in the same Project");
          validationErrors.add(smf.getFormattedMessage());
        }
      }
    }
  }
     return validationErrors;
   }
```

ucViewAttachment

- Add new comment

Purpose

This exit is used to view the contents of file attachments stored within the ExtraView database, on the filesystem, or within an external repository linked to ExtraView

Applies To

Attachment Methods

Signature

```
public boolean ucViewAttachment(
                      ServletOutputStream out,
                      Attachment attachment)
```

Notes

There is a ucViewDocument exit which is used to view the contents of fields with a display type of *document* and *image*.

Example

```
    public boolean ucViewAttachment(ServletOutputStream out, Attachment attach) {
        Z.log.writeToLog(Z.log.DEBUG1, "View Attachment  : ");

        boolean status = false;
        int count = 0;
        long bytesRead = 0l;
        FileInputStream is = null;
        BufferedInputStream bis = null;

        BufferedOutputStream bos = null;
        File input = new File(attach.getExternalAttachmentIdent());
        long fileSize =  input.length();
        try {
            // Get the input stream of the Blob
            is = new FileInputStream(input);
            bis = new BufferedInputStream(is);

            bos = new BufferedOutputStream(out);

            byte[] bytes = new byte[1024];

            // Read the blob 1024 bytes at a time
            while ( (count = bis.read(bytes)) > 0) {
                // Write the bytes out to the response
                if (bos != null)
                    bos.write(bytes, 0, count);
                bytesRead = bytesRead + count;
            }
            if (fileSize == bytesRead)
                status = true;
        }
        catch (Exception e) {
            ErrorWriter.write(e, ErrorWriter.LOGERR);
        }
        finally {
            try {
                if (bis != null)
                    bis.close();
            }
            catch (Exception e) {}
            try {
                if (bos != null)
                    bos.close();
            }
            catch (Exception e) {}
        }
        return true;
    }
```

ucViewDocument

Purpose

This exit is used to view fields with display types of *document* and *image*.

Applies To

Attachment Methods

Signature

```
public boolean ucViewDocument (
                   OutputStream out,
                   ImageItemUDF iiu)
```

Notes

This exit may be used to view a document or image field which resides internally within the ExtraView database, stored on the filesystem, or stored in an external repository such as Dropbox, which is integrated with ExtraView. Also see ucViewAttachment which is used to view file attachments, as opposed to the contents of document and image fields. For more information, please see here.

Session Variables

- Add new comment

ExtraView supports a concept known as session variables. These are defined in the data dictionary and supported by code internal to ExtraView, or by user custom code. For example, there is an in-built session variable named **USER**, which is used to provide the user's session with the name of the current user. This has a title of *** Current User Name ***.

You will see this being used in several places in ExtraView, most notably within user field display type lists on search layouts. This allows the user to define a report filter, which when run, will always point to the current user. Session variables can be defined in the following way:

- Create a new field in the data dictionary, under the tab named **Session Variables**. For example, create a field named *MY_VAL*
- Provide a title to this new variable, for example * *My Value* *
- Use a display type of *CUSTOM to complete the entry, then save the new field.*

Now we want to use this field, we need to support it with code that provides a runtime value. For example, we may want to place this new runtime variable onto a list. Within the method ucRenderListValues you can add the following code:

```
if (ddName.equals("OWNER_GROUP")){
              // Get the DDE for the variable to add to list
              dde = Z.dictionary.getDDEntry(dbConn,"USR_GRP");
              selList.put("$$USR_GRP$$",dde.getTitle(loc));
              return;
            }
```

You must also make sure your variable is placed into the Session object at runtime with your user custom code.

Customizing File Uploads

This section describes, in a single place, the collection of user custom exits which facilitate the handling of cloud-resident attachments, image field values, and document field values.

The basic requirement described in this section is to make cloud-based or repository-based files behave like uploaded file objects in ExtraView. Generally, ExtraView stores upload objects in the file system or the database; some user custom exits permit storage of upload objects in user-defined external storage.

User custom exits support cloud-based file systems due to the wide variety of cloud storage types and the assumptions of the current upload process. No two cloud-based systems have the same interface.

Upload Process Operations

Each of these operations requires specific processing by user custom exits when the upload object is in the cloud or a non-ExtraView repository. The user custom exits defined in this section are the only ones used for cloud/repository upload objects. There are other, specific exits that deal directly with file Attachments, for example, ucDeleteAttachment, but these should not be used since they may or may not be invoked for cloud/repository upload objects. The only legacy interfaces used for these upload types are ucViewAttachment and ucViewDocument.

The upload process consists of the following high-level, user-initiated, operations:

Upload Screen Interaction - ucUploadSetForm

A user custom exit is made available to populate an HTML DIV on the upload page with HTML created within the user custom exit. The DIV will become visible based on the MODE of the upload screen. This is selectable by the user from one of three modes: Drag and Drop, Standard and Repository. All messages and button titles visible by the user are customizable.

The user custom exit modifies a tags HashMap that is used with the inbuilt addAttachment.html template to produce the upload page. There are two mandatory tags of interest to the user custom callout:

- NO_REPOSITORY – should be set to "false" to include the repository DIV
- REPOSITORY_DIV_HTML – should be set to the HTML to include inside the DIV used for repository upload

Other tags of interest are:

- REPOSITORY_TOOLTIP – is the tooltip shown in a balloon for mouseover event on the repository anchor image
- USE_REPOSITORY_UPLOAD_MSG – is the text shown for the repository mode switch anchor

The submit button used for standard mode form submission is in the tags HashMap with the key **SUBMIT_BUTTON**. This can be reused for inclusion in the repository html.

The parameter uploadObjectType is either ATTACHMENT, IMAGE or DOCUMENT. Interface:

```
public void ucUploadSetForm ( HashMap tags,
                              SesameSession session,
                              Connection dbconn,
                              String uploadObjectType ) throws Exception ;
```

Submitted Upload Form Actions - ucUploadProcessForm

While it is possible for the repository DIV to generate a form submission that bypasses all existing base ExtraView code, there would be substantial work involved in the new user custom service that simply replicated existing code. Therefore, it is strongly recommended that the upload HTML designer use the existing action/option and HTML FORM for submission to the server.

A user custom exit is made available to process the form parameters and request object received from the upload page. This user custom exit is responsible for:

- Parameter validation
- Upload object creation (attachment, image_item_udf, or document_item_udf) – this includes thumbnail generation
- Upload object insertion to the database

This custom exit is not responsible for:

- Providing the service action/option method
- Collecting the parameters from the multipart form
- Closing the upload popup

Once the user has entered the necessary information to the upload screen, the form is submitted as a multipart form, and the ExtraView base code parses the request. The result consists of form parameters in a HashMap, representing the values entered, or pre-seeded on the upload page. If the page is submitted in the Repository mode, the form parameters and the request are passed to the user custom exit ucUploadProcessForm defined here. A new, partially constructed uploadObject is passed as a parameter. This will be either an Attachment, a DocumentItemUDF, or an ImageItemUDF object that can be used in creating the upload object in the database. In this object the required fields that are populated are: ITEM_ID, UDF_ID (if any), CREATED_BY_USER and UPDATED_BY_USER. Other system-generated fields such as LAST_DATE_UPDATED are populated during transactions on the upload object. Aside from the pre-populated fields and the system-generated fields, the user custom exit is responsible for setting the object values using its setter methods.

The form parameters here do not support multi-valued parameters. This is a legacy-based restriction due to the way the parameters are handled internally in ExtraView. If multiple values are needed for a specific parameter, it is recommended that the parameter name be uniquefied with numbers, e.g., file1, file2, file3, …. in order to work around this restriction.

The string returned by this method is an error message that will alert the user in the upload screen; null for successful upload. Interface:

```
public String ucUploadProcessForm ( HashMap formParameters,
                                    SesameSession session,
                                    Connection dbconn,
                                    String uploadObjectType,
                                    Object uploadObject ) throws Exception
```

There are special considerations for the ADD screen because the ITEM_ID is **0** as the issue has yet to be added to the database and assigned its ITEM_ID. For image and document data types, when the ITEM_ID is **0**, the executeTransactionUncommitted should be used to store the imageItemUDF; otherwise, use the executeTransaction method.

For attachments, when the ITEM_ID is 0, the attachment is stored in an attachment list in the current session. When the issue is later committed to the database, the proper fields will be filled in and the attachment will be stored in the database.

Render Upload Object in Browser - ucViewDocument & ucViewAttachment

Two user custom exits are available to view the upload object in the browser according to its MIME type. This stage also includes the rendering of a thumbnail image of the object.

There are two methods for rendering an upload object from a repository:

- ucViewDocument – used for viewing image and document data types
- ucViewAttachment – used for viewing attachments

Interface:

```
public boolean ucViewDocument (OutputStream out, ImageItemUDF iiu) ;

public boolean ucViewAttachment (OutputStream out, Attachment attachment);
```

Edit Upload Object Metadata - ucUploadEditObject

For attachments, some parts of the object metadata are editable, e.g. the file description. There is a user custom exit for validation/modification of the edited metadata for cloud/repository upload objects. User modification of upload objects is restricted to the file description, also known as ALT_TEXT, and the charset encoding. A user custom exit is provided to process the modifications before they are made to the object. This exit may validate the parameters, produce an error message, or allow the transaction to proceed. Interface:

```
public String ucUploadEditObject ( HashMap formParameters,
                              SesameSession session,
                              Connection dbconn,
                              String uploadObjectType,
                              Object uploadObject ) throws Exception
```

Delete Upload Object - ucUploadDeleteObject

All upload objects may be deleted based on actions by the user, e.g. the deletion of containing issue. There is a user custom exit for cloud/repository object deletion to permit recycling of resources owned by the upload object.

```
public String ucUploadDeleteObject ( SesameSession session,
                              Connection dbconn,
                              String uploadObjectType,
                              Object uploadObject ) throws Exception
```

The user custom exit ucDeleteAttachment is not invoked with cloud upload objects. The string returned by these methods is an error message that will alert the user in the edit/delete screen; null for successful transaction.

Clone Upload Object - ucUploadClone

When an upload object is contained in an issue that is being cloned, and the upload object is a cloud/repository-based object, a user custom method can be invoked to populate a cloned object. At a minimum, any thumbnails should be copied, and other repository operations may be needed.

In the clone operation on the *edit* screen, contained upload objects must be replicated into the new issue. In the interface, the newUploadObject is the partially populated new upload object (Attachment, ImageItemUDF, or DocumentItemUDF) that can be used to perform the transaction to store it in the database. Interface:

```
public String ucUploadClone ( Object uploadObject,
                              SesameSession session,
                              Connection dbconn,
                              String uploadObjectType,
                              Object newUploadObject ) throws Exception
```

Thumbnail images are considered to be contained in the upload object; therefore, they must not be shared between objects, and in a clone operation, the thumbnail must be created as a new object in the database.

Finish Upload Operation - ucUploadFinish

After the completion of an upload operation, there may be resources that were allocated during the upload that need to be recycled. The Finish Upload operation permits cleanup, as it is the last operation performed after the upload is complete. Interface:

```
public String ucUploadFinish ( Object uploadObject,
                              SesameSession session,
                              Connection dbconn,
                              String uploadObjectType ) throws Exception
```

The String return value is an error message or null if successful.

Get Upload Object Content - ucUploadGetContent

The binary content of the uploaded object must be made available for email inclusion. Emails that include attachments, images, or documents generally have a thumbnail of the object in addition to the binary object in its entirety, not just a link. The ucUploadGetContent method is invoked to populate the email message with the content of the object. The thumbnail is obtained directly from the object in the database. Interface:

```
public InputStream ucUploadGetContent ( Object uploadObject,
                                        SesameSession session,
                                        Connection dbconn,
                                        String uploadObjectType ) throws Exception
```

The InputStream returned value must stream the binary contents of the upload object for inclusion in emails.

Implementation Considerations

The metadata associated with document or image field object consists of the following (those preceded by "*" are values that should be defined by user custom code):

- IMAGE_ITEM_UDF_ID – auto-generated primary key for document and image data types
- UDF_ID – the UDF id of the containing field
- DATE_CREATED – auto-generated timestamp
- LAST_DATE_UPDATED – auto-generated timestamp
- LAST_UPDATED_BY_USER – auto-generated user id
- CREATED_BY_USER – auto-generated user id
- ITEM_ID – the item id of the containing item
- *MIME_TYPE – the MIME type of the object
- *STORED_INTERNAL – Y for database-resident objects; N for file system objects; R for repository-based objects
- *EXTERNAL_IDENT – string containing enough key information to identify the object externally; this is opaque to ExtraView base code
- *ALT_TEXT – also known as File Description, this is text that is important for accessibility guideline compliance
- VALUE_BLOB – when stored in the database, the blob identifier
- *THUMBNAIL_BLOB – when thumbnail is present, the blob identifier of the thumbnail image
- *FILE_NAME – a file name, usually the one specified in the upload screen; must not be null or blank
- *CHARSET – the charset name for the character set in which the file is stored. This is used ONLY for text files (usually with an extension of ".txt", but may be other files based on the MIME type)
- *FORMAT – IGNORE, TEXT or BINARY: used by quick search to generate an index for this object
- *EXTENSION – a file extension, e.g., ".txt" to identify the file type for quick search

The metadata associated with an attachment object consists of the following (those preceded by "*" are values that should be defined by user custom code):

- ATTACHMENT_ID– auto-generated primary key for attachments
- *FILE_DESC – also known as ALT TEXT, this is text that is important for accessibility guideline compliance
- *FILE_NAME– a file name, usually the one specified in the upload screen; must not be null or blank
- *PATH– a file directory path, usually the one specified in the upload screen
- *CONTENT_TYPE– the MIME type of the object
- *FILE_SIZE – the size in bytes of the object
- DATE_CREATED– auto-generated timestamp
- CREATED_BY_USER– auto-generated user id

- ITEM_ID – the item id of the containing item
- *CHARSET – the charset name for the character set in which the file is stored. This is used ONLY for text files (usually with an extension of ".txt", but may be other files based on the MIME type)
- *STORED_INTERNAL – Y for database-resident objects; N for file system objects; R for repository-based objects
- *EXTERNAL_IDENT – string containing enough key information to identify the object externally; this is opaque to ExtraView base code
- LAST_DATE_UPDATED– auto-generated timestamp
- LAST_UPDATED_BY_USER– auto-generated user id
- *TYPE_ID – 0 for attachment object, 1 for thumbnail
- *THUMBNAIL_ID – ID of the thumbnail attachment object associated with this attachment (non-null only in attachment objects, null for thumbnail objects)

Implementation Notes

1. The user custom code should not modify any database tables directly. The developershould use the internal Attachment, DocumentItemUDF and ImageItemUDF objects to set values and transact with the database
2. The CHARSET for documents and attachments may be entered by the user at the time the object is uploaded. If the cloud/repository cannot determine the correct character set of the object, then the form should include the CHARSET drop-down list as is done for the Standard upload mode.

QuickFind for Cloud/Repository Upload Objects

QuickFind is not capable of building indexes based on the contents of cloud/repository objects. A future release will cover this functionality.

Code Example - Thumbnail Creation for ImageItemUDF

This is a Java source code example of how to create a thumbnail and store it into an ImageItemUDF object in the database. In this example, the image file is assumed to be readable (iiu.getTempFile()).

```
int maxPixelsPerDimension = 4000;
String appDefaultMax = ApplicationDefault.getAttribute("MAX_IMAGE_DIMENSION_PIXELS");
if (isANumber(appDefaultMax)) {
  maxPixelsPerDimension = Integer.parseInt(appDefaultMax);
}
// check against max dimensions...
int[] dimensions = ImageProcess.getDimensions( iiu.getTempFile(), "");
if (dimensions[0] > maxPixelsPerDimension || dimensions[1] > maxPixelsPerDimension)
       throw new Exception(Z.m.msg(session, "Image too large"));
  // make the thumbnail...
  thumbFile = File.createTempFile("thumb", iiu.getFileName(), new File(Z.getTempDir()));
  status = (new ImageProcess()).makeThumbnail(iiu.getTempFile(), thumbFile,
                                          iiu.getFileName());
  if (status) {
    iiu.setThumbnailSize( (int) thumbFile.length());
    thumbIs = new FileInputStream(thumbFile);
    iiu.setThumbnailBlobIS(thumbIs);
    is = new FileInputStream(iiu.getTempFile());
    iiu.setValueBlobIS(is);
    String userId = session.getUserId();
    iiu.executeTransactionUncommitted(userId);
    is.close();
    is = null;
    // If we got here with no exceptions commit and return;
    dbconn.commit();
    status = true;
  } else throw new Exception(Z.m.msg(session,
              "The most probable reason is that it is not an image file."));
```

Code Example - Creating a Thumbnail for Attachments

Thumbnails for Attachments are separate Attachment objects with a type_id = 1. To create a thumbnail, the following example may be used:

```
public boolean generateThumbnail(Attachment orig, Connection dbConn) {
  boolean status = false;
  String origFilename = orig.getFileName();
  String outFilename = null;
  File strtsFile = orig.getTempFile();
  String strtsFilename  = orig.getOrigFileName();
  if (origFilename != null)
          outFilename = origFilename.substring(0,origFilename.lastIndexOf(".")) +
          "_thm" + origFilename.substring(origFilename.lastIndexOf("."));
  File thumbnailFile = new File(strtsFile.getParent(),outFilename);
  String test = thumbnailFile.getAbsolutePath();

  if (ImageProcess.canMakeThumbnail(origFilename)){
    try {
      DbTime dbt = new DbTime(dbConn);
      ImageProcess ip = new ImageProcess();
      status = ip.makeThumbnail(strtsFile,thumbnailFile,origFilename);
      this.tempFile = thumbnailFile;
      if (this.tempFile != null) this.tempFilePath = thumbnailFile.getAbsolutePath();
      this.origFileName = outFilename;
      this.fileName = outFilename;
      this.fileSize = thumbnailFile.length();
      this.problemId = orig.getProblemId();
      this.dateCreated = dbt.getDbNowTimestamp();
      if (!status) {
        this.errorMsg = Z.m.msg(session,
                "Cannot generate thumbnail. The file is not the right type.");
      }
    } catch (Exception e) {
      //ErrorWriter.write(e, ErrorWriter.LOGERR);
      status = false;
    }
  }
  // Log the failure here, success will be logged later
  return status;
}
```

CSS Interface

Their are occasions when it is useful to apply your own styles to specific elements within a form on the *add* or *edit* screen. An understanding of Cascading Style Sheets is essential to use this feature. You can use most browsers "inspect" mode to determine the element names to which you want to apply a style.

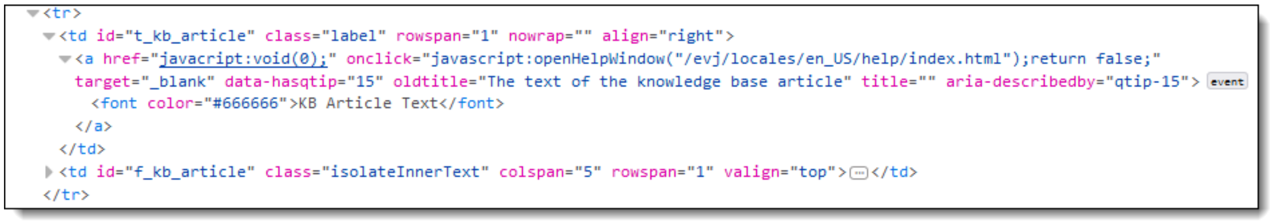
To accommodate this, there is a file within the directory structure `evj\stylesheets\user_stylesheets` named `user_stylesheet.css`. evj is the location of your installation within your server.

You may place any valid CSS within this file to augment the styles on your *add* and *edit* screens. Not every style may be overridden, but it is likely that those you want to alter can be adjusted within this file.

Example

You have an HTML Area field named `kb_article` on an *edit* screen. This is read-only but you want to restrict its width to fit neatly on the screen. Layout cell attributes such as SIZE and STYLE only work with writable fields so are not a solution.

An inspection of the browser's DOM shows the field structure as this:



```
▼<tr>
  ▼<td id="t_kb_article" class="label" rowspan="1" nowrap="" align="right">
    ▼<a href="javacript:void(0);" onclick="javascript:openHelpWindow("/evj/locales/en_US/help/index.html");return false;"
      target="_blank" data-hasqtip="15" oldtitle="The text of the knowledge base article" title="" aria-describedby="qtip-15"> event
        <font color="#666666">KB Article Text</font>
      </a>
    </td>
  ▶<td id="f_kb_article" class="isolateInnerText" colspan="5" rowspan="1" valign="top">…</td>
  </tr>
```

The field is rendered within the row with the ID of `f_kb_article`. To limit the width of this field, but not alter the CSS class named `isolateInnerText` which may be used elsewhere on the same form, create an entry within the `user_stylesheet.css` file like this:

```
#f_kb_article .isolateInnerText {
    max-width:500px;
}
```

This will limit the field to have a maximum width of 500 pixels without altering any other field rendered on the form.

Hints

- If you want to replace a style within a form, you can use the `!important` modifier, unless this is already used on style. This is usually going to work within *add* and *edit* screens
- The top level identifier within both *add* and *edit* screens is a table with the ID of `top_lvl_table`. You can refer to this within CSS as `#top_lvl_table`
- The labels of fields are preceded with an ID of `t_`. For example the ID of the field label in the above example is `t_kb_article`
- The values of fields are preceded with an ID of `f_`. For example, the ID of the field value in the above example is `f_kb_article`.