Lance Baker

System Manual

Introduction

The following document comprises of a detailed outline of systematic features included in the final system. The document will consist of design orientated diagramming; such as written use cases, use case diagrams, and the UML class diagrams.

Programmatic Features

ExtJS Foundations

The most sought after question EXTJS… what is it? Well, essentially the whole application is JavaScript based on the foundations called ExtJS; which provides an extremely powerful set of tools, components, and sheer brute force to make the web application more desktop-like. The framework allows for the creation of interfaces (without having to manually tinkle with traditional mark-up languages); composing of controls specifically created for their needed purpose.
Ext.Direct allows for direct invocation of Programmatic objects from JavaScript.

The functionality it provides allows for the Programmatic objects located on the server side, to be directly accessed from the JavaScript code. It behaves as dedicated 'communication' layer in between the GUI and the BOL, which essentially adds a layer to the application called an 'API' which has the list of classes with methods contained on the Server with an Ajax wrapper for sending/receiving.

The catch though, is that the server side must also have something of equivalence to relay the responses once more to the corresponding Business Object. The choice for the server side ExtJS companion ended up being 'DirectJNgine' which is derivative from 'Ext.Direct Java Engine'. Please have a read of the DirectJNgine Guide attached to the appendix for more information relating to the usage.

Batching allows for Ajax requests that are sent within a specific delay period to be grouped together in the same transmission. This will therefore increase the performance of the application (to large proportions) by not having individual requests constantly sent back and forth; which would lag both the browser, and way down the server.

Graphical User Interface

The entire web application is built in such a way which easily supports expandability; using a modular based format (with a window system) it caters the function to spawn modules, create modules on the fly, and to also regulate access to those modules through the assignation of the appropriate group permissions to the user accounts accessing the system.

Written Use Cases

Use Case:	**Validate User Credentials**	
Scenario:	The user desires to confirm their credentials and gain access to the system.	
Triggering Event:	The user enters their credentials into the login area.	
Brief Description	The user logs into the system by specifying their user name and password. The system then checks for a corresponding record based off those given credentials. Once a record has been found of matching specification; the user session will commence, and the panel will get loaded. The site navigation will then be populated once the is successfully logged in.	
Actor/s:	User, Administrator	
Related Use Cases:	Extends: 1. *Generate Site Navigation* 2. *Credentials Invalid* Uses: 1. *Log Session Commencement*	
Stakeholders:	Company Administrator	
Precondition/s:	1. The user is not currently logged into the system.	
Post condition/s:	1. The user has been successfully logged into the system	
Flow of events:	**Actor**	**System**
	1. The user enters their login credentials into the given login area, and proceeds by pressing the corresponding button. 2. The user can now view the menu, and the site navigation system is now triggered to be loaded. 3. The user can now view the menu, and the site navigation system is now triggered to be loaded.	1.1. The system receives the login credentials, and validates whether the details exist in the database. 1.2. The system proceeds to begin the session commencement and records the session into the user log. (*Log Session Commencement* Use Case*).* 1.3. The system returns a success response back to the client; which then invokes for the menu to be loaded. 2.1 The system invokes the <Generate Site Navigation> use case, and it only generates the categories for the top level parents.
Exception Conditions:	1.1 Before it proceeds to login to the system, it will check whether the user's IP address has not exceeded the 'invalid login' count allowed. If they haven't exceeded the count; it will check if their user credentials exist. If their user credentials cannot be found in the system; it will halt and invoke the *Credentials Invalid* Use Case.	

<table>
<tr><td>Use Case:</td><td colspan="2">Logout Action</td></tr>
<tr><td>Scenario:</td><td colspan="2">The user decides to exit the system.</td></tr>
<tr><td>Triggering Event:</td><td colspan="2">The user presses the logout button, located within the side menu.</td></tr>
<tr><td>Brief Description</td><td colspan="2">Once the user triggers the logout operation, it should end the current session with the user, and also ‘finalise’ the session log; by recording the end session date. The logout process will also trigger the site navigation to be cleared, the current windows will need to be closed, and the side panel will be refreshed to the login area.</td></tr>
<tr><td>Actor/s:</td><td colspan="2">User, Administrator</td></tr>
<tr><td>Related Use Cases:</td><td colspan="2">Extends:
1. Invalidate Session
2. Finalise Session Log</td></tr>
<tr><td>Stakeholders:</td><td colspan="2">Company Administrator</td></tr>
<tr><td>Precondition/s:</td><td colspan="2">1. The user is currently logged into the system.</td></tr>
<tr><td>Post condition/s:</td><td colspan="2">2. The user has successfully been logged out of the system</td></tr>
<tr><td>Flow of events:</td><td>Actor
1. The user presses the ‘logout button’.

2. The user will confirm whether they are sure they wish to close the windows, and that they are ready to proceed.

3. The user should be successfully logged out, and be prompted with a notification to confirm that they have logged out of the system.</td><td>System
1.1. The system will proceed to close any active windows.

2.1 The current session will become invalidated <<i>Invalidate Session</i>>, and the session log will become finalised < <i>Finalise Session Log</i>>.

2.2. The client-side scripting, of the system will trigger the navigation to be cleared, and for the side panel to get refreshed; which will reveal the login area once again.</td></tr>
<tr><td>Exception Conditions:</td><td colspan="2">1.1 If there are any currently opened windows, the user will be prompted whether they are ready to close the window. If they press no, it will terminate the logout process.

1.2 If the user is currently logged in, and that the session is active; it will proceed to <<i>Invalidate Session</i>>. If the session is invalidated, it will < <i>Finalise Session Log</i>>.</td></tr>
</table>

Use Case:	**Invoke Section**	
Scenario:	The user desires to spawn a section window.	
Triggering Event:	The user clicks on a section link located within the site navigation.	
Brief Description	The invoking operation will send the spawning request to the system; which will attempt to find the corresponding section (in alignment to the current user's group access permissions). If the section can both successfully be found, and the user is also authorised to view that section; it will continue to find the corresponding module based on the details within the section record. The contents of module will be returned to the client-side system, which will be invoked. The viewing of that section; will be recorded in the access log history after it has successfully been found.	
Actor/s:	User, Administrator	
Related Use Cases:	Extends: 1. *Locate and Retrieve Module* 2. *Add Section Access History*	
Stakeholders:	Company Administrator	
Precondition/s:	1. The user is currently logged into the system.	
Post condition/s:	None.	
Flow of events:	**Actor**	**System**
	1. The user has found a section to invoke from the navigation, and presses on the link.	1.1 The system will send the action to the server, and will *<Locate and Retrieve Module>* based on the access permissions defined in the associated user group. 1.2 The section visitation will be recorded in the access history log. *<Add Section Access History>* 1.3 The module will be returned to the client-side system, and invoked for display.
	2. The window will be displayed for the user, and the current opened state of the window should be reflected in the task bar.	1.4 The window will be added to the 'current opened windows' task bar.
Exception Conditions:	2.1 If the module cannot be found (either due to requested section not existing, or based on insignificant access permissions defined) it will not be invoked for display, and the current use case will be completely halted. 2.2 The section can visitation will only be recorded if the module was successfully found. If the current user session cannot be found in the session log, or if the session has already been ended; the section module will not be invoked.	

Use Case:	**View Records**	
Scenario:	The data in the initial module view will be displayed upon invocation.	
Triggering Event:	The module has been loaded.	
Brief Description	The request will be sent the Server, which will locate the section being viewed based on the primary key received. The section will find the corresponding Section bean, and invoke the method for retrieving a list. The list outputted, will be in the format of JSON, and the module on the client-end will be able to process that received output and display it in the view format as directed.	
Actor/s:	User, Administrator	
Related Use Cases:	Extends: *1. Invoke Section*	
Stakeholders:	Company Administrator	
Precondition/s:	1. The module has been invoked.	
Post condition/s:	None.	
Flow of events:	**Actor**	**System**
	1. The user has invoked the section, and the module is loaded. 2. The user should see the records in a format as reflected by the module itself.	1.1 The system will receive the request to load the initial data. 1.2 The section bean will be located based on the section identifier received. 1.3 Once the bean has been located, it will dynamically invoke the list operation; which will grab all associated records. 1.4 The records will be converted to the JSON format, and returned back to the client system. The records will then be processed, and outputted to the user.
Exception Conditions:	1.2 If the corresponding section bean cannot be located; the current Use Case will halt, and data returned to the client-side system will be empty.	

Use Case:	**View Single Record**	
Scenario:	The user desires to view a record.	
Triggering Event:	The user clicks on the record located in data view.	
Brief Description	Once the record has been pressed, the request will be sent to the system which will find the corresponding record based on the primary identifier received; the system will then instantiate the section bean based on the record received from the database. The section object will then be converted into the JSON format, and returned back to the client module; which will be placed in the corresponding HTML elements.	
Actor/s:	User, Administrator	
Related Use Cases:	Extends: *1. View Records*	
Stakeholders:	Company Administrator	
Precondition/s:	1. The 'list view' of the module was populated successfully.	
Post condition/s:	None.	
Flow of events:	**Actor**	**System**
	1. The user has pressed on the record they desire to be modified.	1.1 The system will receive the request to load the record. 1.2 The section will be located based on the section id received. 1.3 The corresponding section bean will found; which will then fetch the associated record from the database and instantiate that bean based on the results. 1.4 The section object will then be converted to a JSON format, and outputted back to the client module. 1.5 The client module will align the received data to the form elements; therefore displaying the record.
	2. The record is loaded into the form, and the user can view the details/ or make any desired changes.	
Exception Conditions:	1.3 If section cannot be found based on the identifier received, it won't load the record, and instead display a notification to the user telling them that there was a problem encountered.	

Use Case:	**Modify Record**	
Scenario:	The user modifies the current record being viewed, and invokes the save operation.	
Triggering Event:	The user invokes the save operation whilst a record is being viewed.	
Brief Description	After the user has modified the record data within the form, and invokes the save operation; the form will be submitted, along with the section id, and id key of the record currently being viewed. The system checks whether the user has the appropriate security level defined on their access group (which supports updating), and if its support; it will proceed to instantiate the section bean based on the POST parameters received. Once the record has been loaded into its corresponding object, the system will then invoke the update operation; which will modify that record in the database based on the current data within that object. After the update has been successful, the user will be prompted with a notification reflecting that the save has either been successful, or if an error was encountered.	
Actor/s:	User, Administrator	
Related Use Cases:	Extends: *1. View Single Record*	
Stakeholders:	Company Administrator	
Precondition/s:	1. The record has been loaded successfully in the form view.	
Post condition/s:	None.	
Flow of events:	**Actor** 1. The user modifies the form to their choosing, and invokes the save operation.	**System** 1.1 The form data is POST'd to the system along with the corresponding identifier for the section (which will locate the section, and then the corresponding section bean). The record identifier is also submitted (which reflects the record to be modified). 1.2 The security level associated with the user will be checked to ensure that they are allowed to perform the record updating.

	2. The user receives the notification that the data has been successfully modified.	1.3 The submitted POST data is converted to an instance of the section bean, and the update operation is invoked. 1.4 The associated section history visitation is found, and a change record is inserted (which includes the date, and the operation invoked). 1.5 The system outputs a response to be displayed for the user.
Exception Conditions:	1.2 If the user doesn't have the appropriate security level defined in their access group, it will halt this use case entirely. And a notification will be displayed back to the user alerting them they cannot perform the update. The update button should also only be displayed to those with the appropriate security level.	

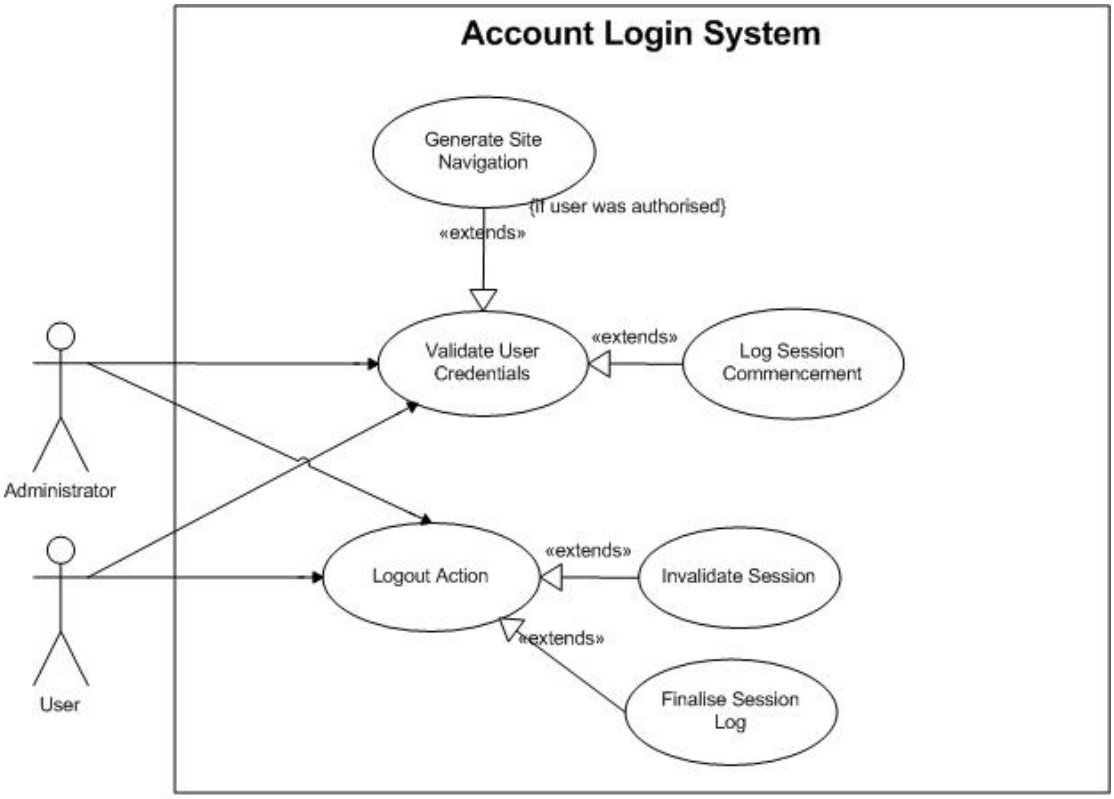
Use Case:	**Delete Record**	
Scenario:	The user desires to delete a record	
Triggering Event:	The user invokes the delete operation whilst a record is being viewed.	
Brief Description	Once the record is currently being viewed; the user can invoke the delete operation. This operation will therefore send a request to the system with the section identifier, and also the record id that is desired to be destroyed. Before, the deletion occurs; the defined security level in the corresponding access group is checked to ensure that the user has the privilege enabled to do such a thing. If the user access group has the appropriate level assigned, it will perform the delete operation. After the record has been removed, the entry is recorded in the corresponding access history log, and the user is then directed back to the record list area with a notification prompted (indicating the success).	
Actor/s:	User, Administrator	
Related Use Cases:	Extends: *1. View Single Record*	
Stakeholders:	Company Administrator	
Precondition/s:	1. The record has been loaded successfully in the form view. 2. The user has the appropriate group security level set, therefore visually seeing the ability to delete a record.	
Post condition/s:	None.	
Flow of events:	**Actor**	**System**
	1. The user is currently viewing a record, and invokes the record to be deleted. 2. The user confirms their intentions, and if they desire to proceed they will continue with the deletion by pressing the 'Proceed' button.	1.1 The system prompts for record deletion confirmation (whether they are sure they wish to proceed). 2.1 The request will be sent to the system; with the corresponding section id and the record id (that they desire to remove). 2.2 The corresponding access group's security level is checked; ensuring that the user has the appropriate level assigned in order to carry out this operation. 2.3 The static delete operation will be invoked on the corresponding section bean, and as a result the record should be removed.

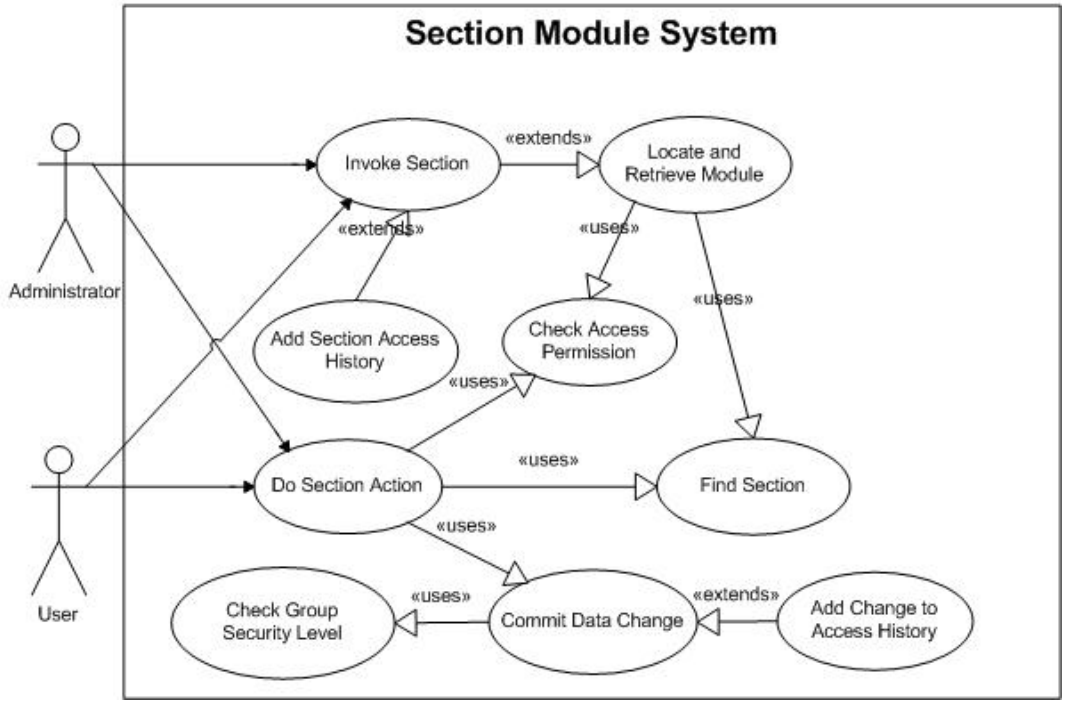
	3. The user should be viewing the main view area of the module, and also have a message prompted.	2.4 The system will redirect the user back to the main view of the module, and display a notification indicating the success of the removal.
Exception Conditions:	3. If the user doesn't agree to continue the delete; it will halt this use case, and the deletion will be cancelled. 1. If the user doesn't have the appropriate security level defined in their access group, it will halt this use case entirely. And a notification will be displayed back to the user alerting them they cannot perform the deletion. The delete button should also only be displayed to those with the appropriate security level, therefore this measurement is only a precaution.	

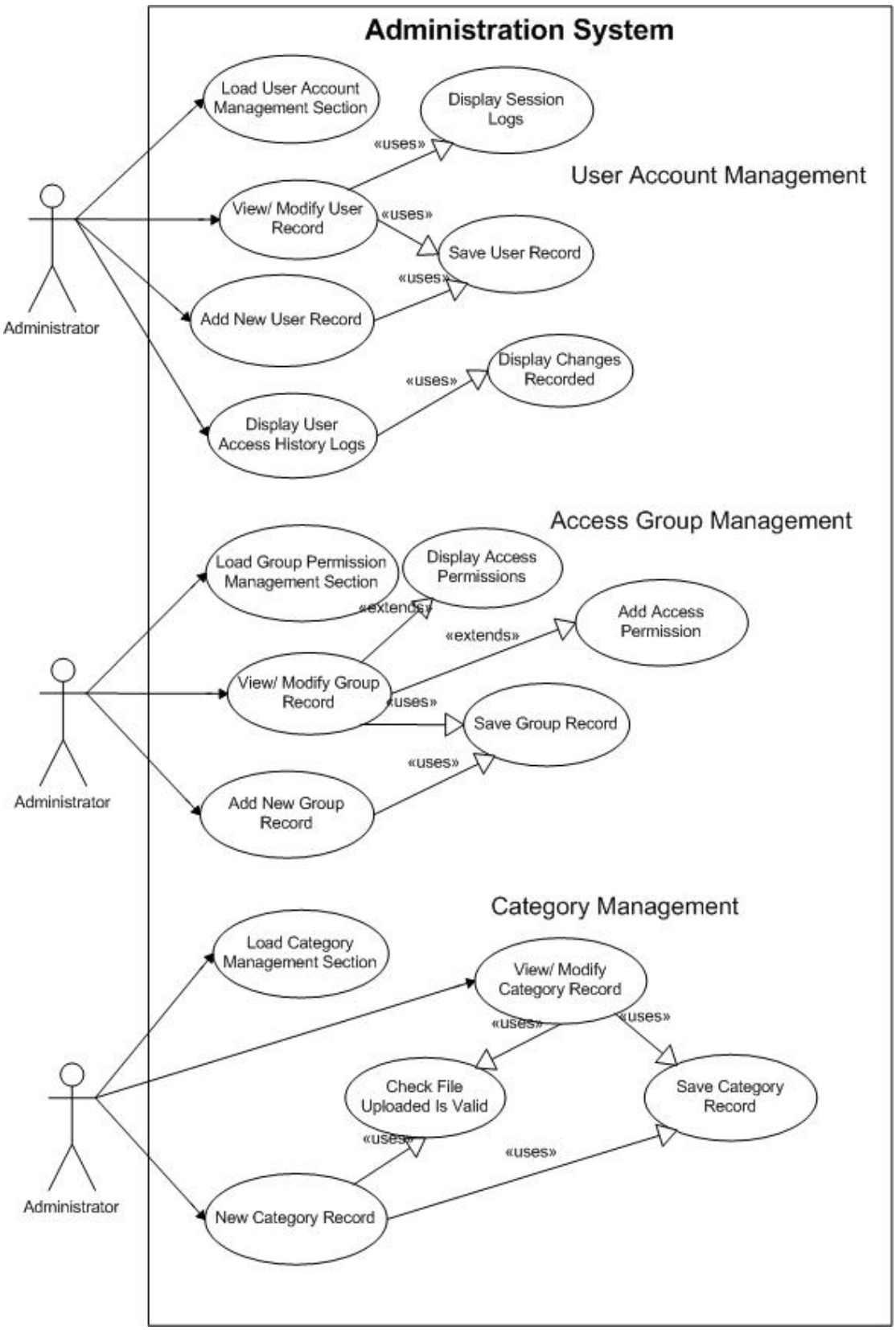
Use Case:	**Insert Record**	
Scenario:	The user desires to insert a new record into the system.	
Triggering Event:	The user invokes the new operation	
Brief Description	Once the module has been loaded, the user can then invoke the 'new' operation. This operation once pressed; will send a request to the system, which will check whether the user has the appropriate security level defined in their corresponding access group. If the user can insert a new record; the form view will be displayed (which will be in an empty state). The user can now fill out the form (which will contain appropriate client-side validation). The form once it has been completed in accordance to the validation set, it can now be saved by invoking the save operation from the corresponding button. Once the save has been pressed; the form will essentially be submitted. The section bean will be found in accordance to the section id received, and an instance of that section bean will be created. The POST data will be dynamically populated into the various properties of that bean, and after it has been create; the insert operation will be invoked. Once it has been inserted, the system will close the form view, and take the user back to the main area (refreshing the display area) thereby showing the newly added record. The user will also be prompted with a notification to reflect the success of the insertion.	
Actor/s:	User, Administrator	
Related Use Cases:	Extends: *1. Invoke Section*	
Stakeholders:	Company Administrator	
Precondition/s:	1. The module has been invoked.	
Post condition/s:	None.	
Flow of events:	**Actor**	**System**
	1. The user is currently in the main module display area, and they press the 'new' record button.	1.1 The system receives the request, and checks wether the user's group has the appropriate security level in order to proceed. 1.2 The system displays the form view of that module (which is empty).
	2. The user completes the form in accordance to the validate criteria set. Once finished, the user can invoke the 'Save' operation.	2.1 The system receives the form data via the POST. The system will find the corresponding section bean, based on the section id received.

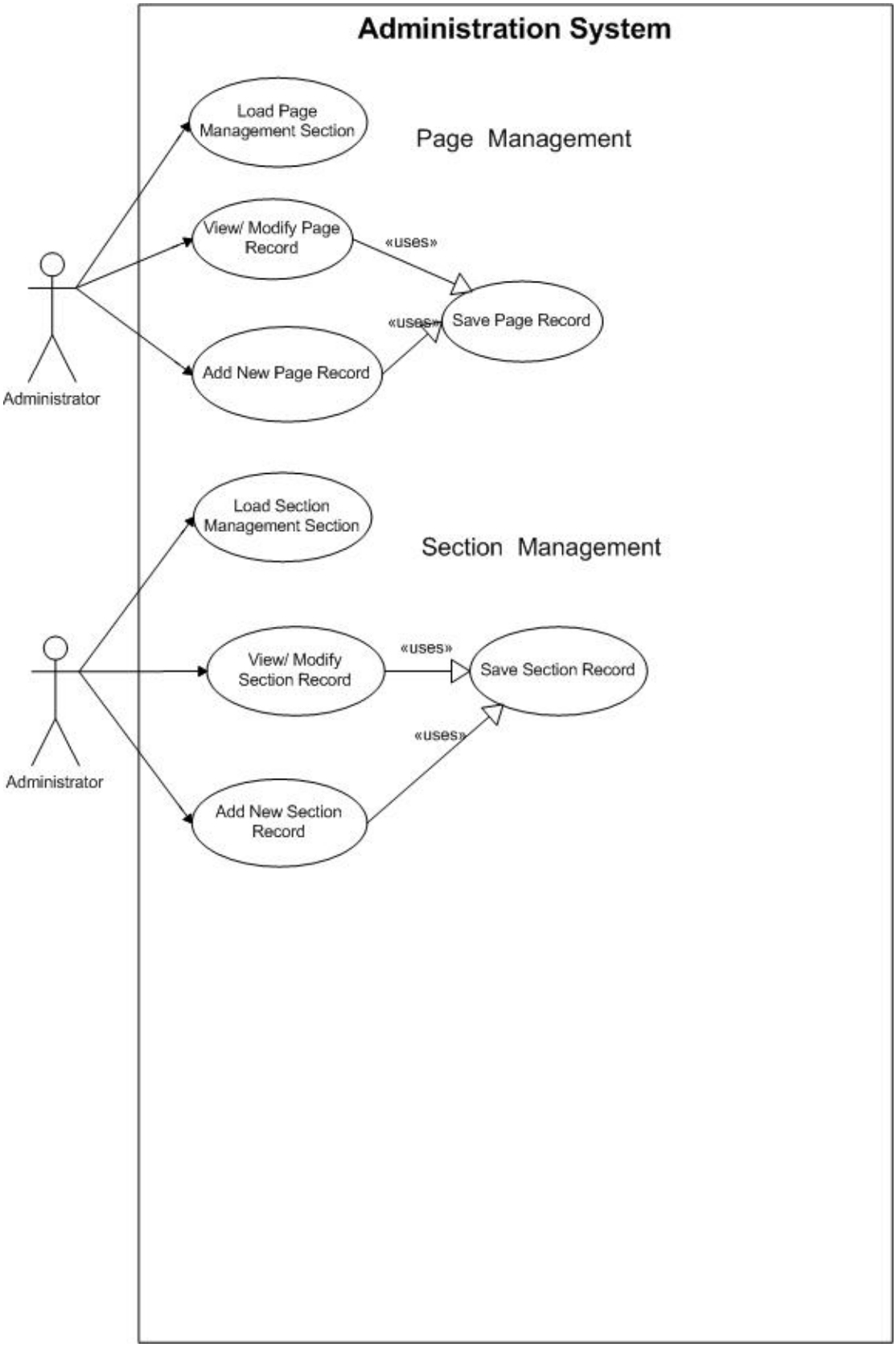
	3. The user will witness the form view closing, and the main view being reloaded; the success notification should also be displayed.	2.2 The form data will be dynamically loaded into an instance of that section bean, which will then enable for the insert operation to be invoked. 2.3 Once the record has been inserted, the form view will be closed, and the system will load the main view in the module; where a notification will be displayed highlighting the success.
Exception Conditions:	1.1 If the user group doesn't have the appropriate access permission defined; the use case will be halted, and a notification will be displayed to the user informing that they cannot carry out this operation. 2.3 If the insert failed, the system will not redirect the user back to the main viewing area. Instead it will prompt a notification indicating that the insert request failed.	

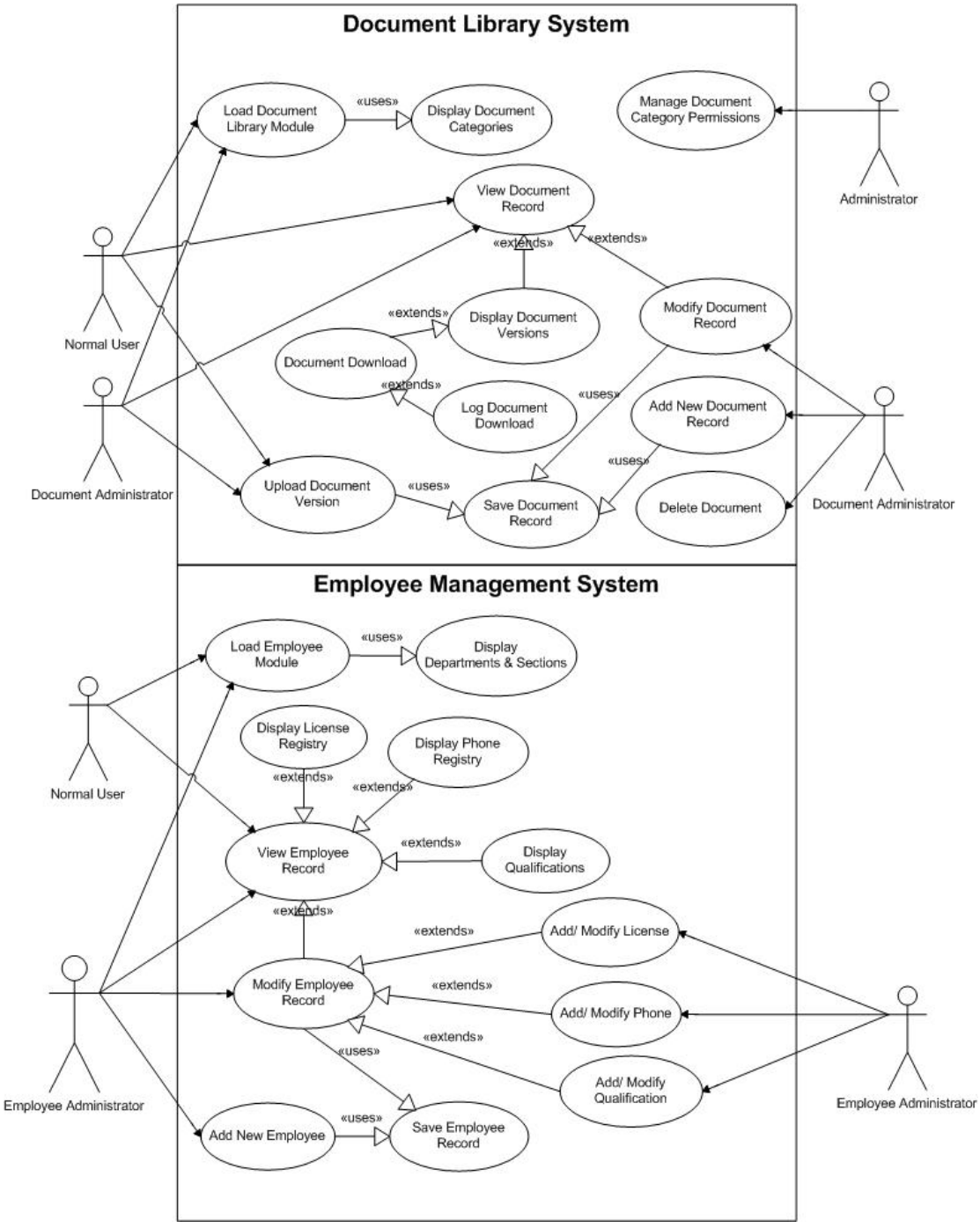
Use Case Diagrams

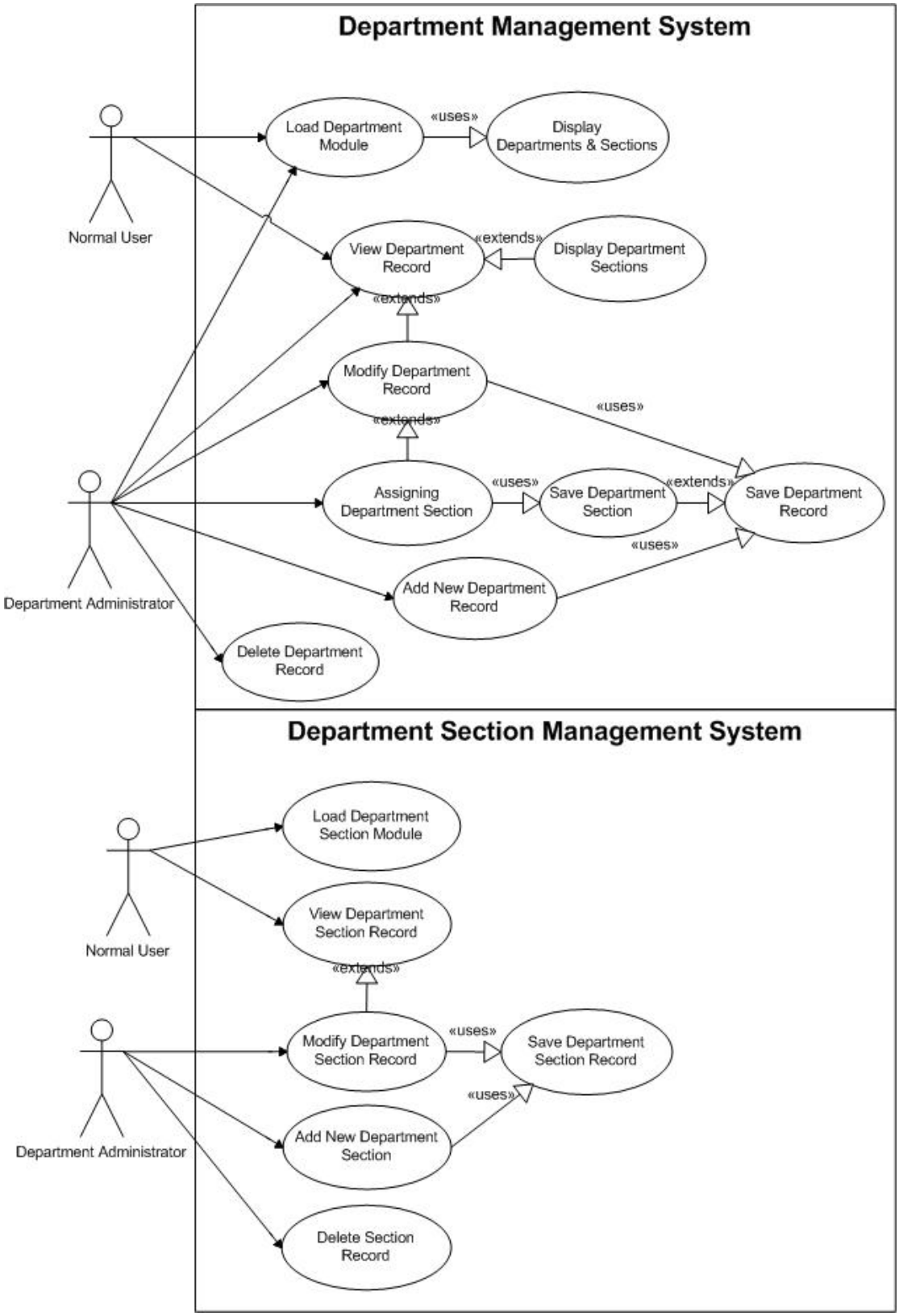


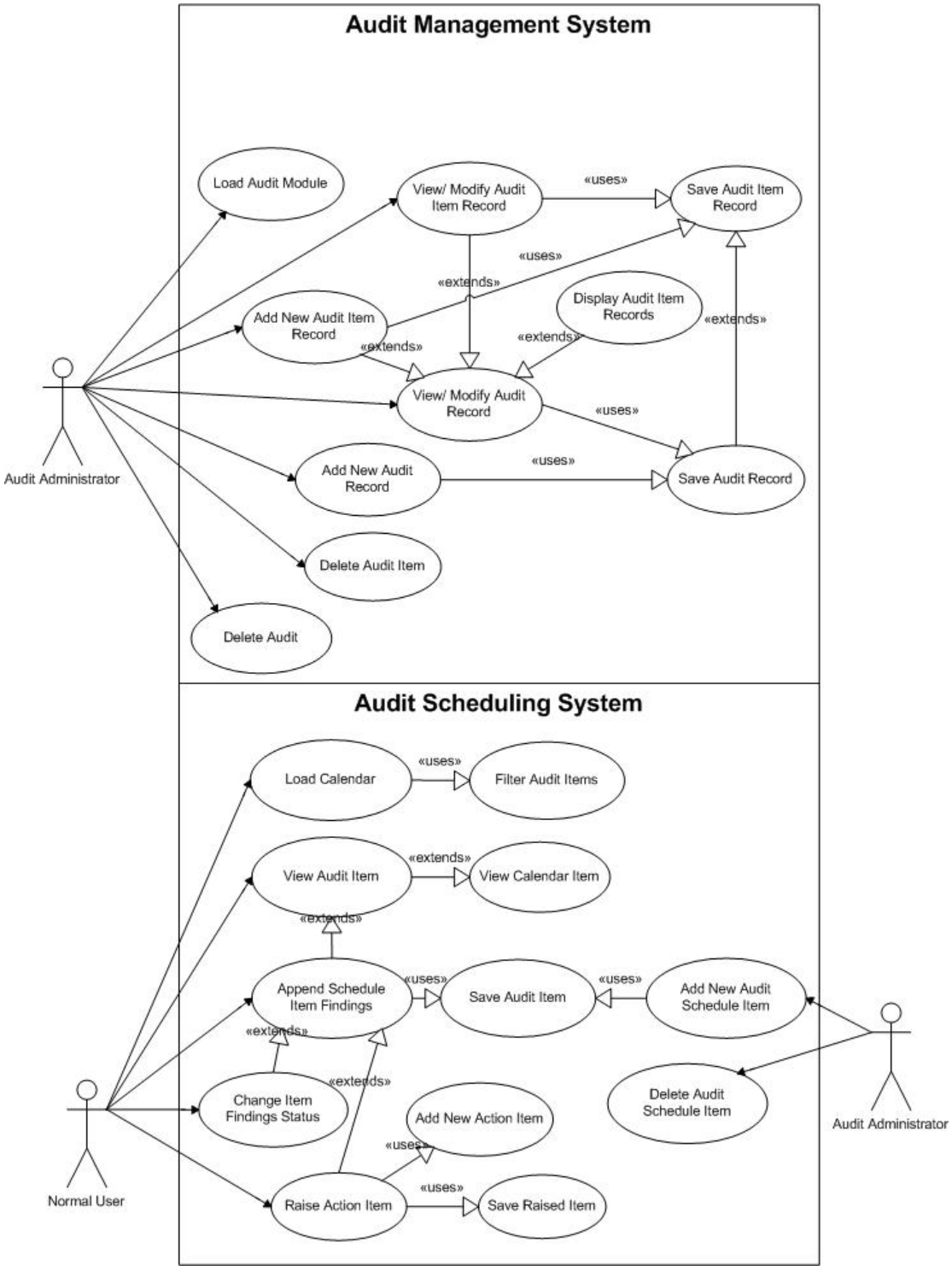


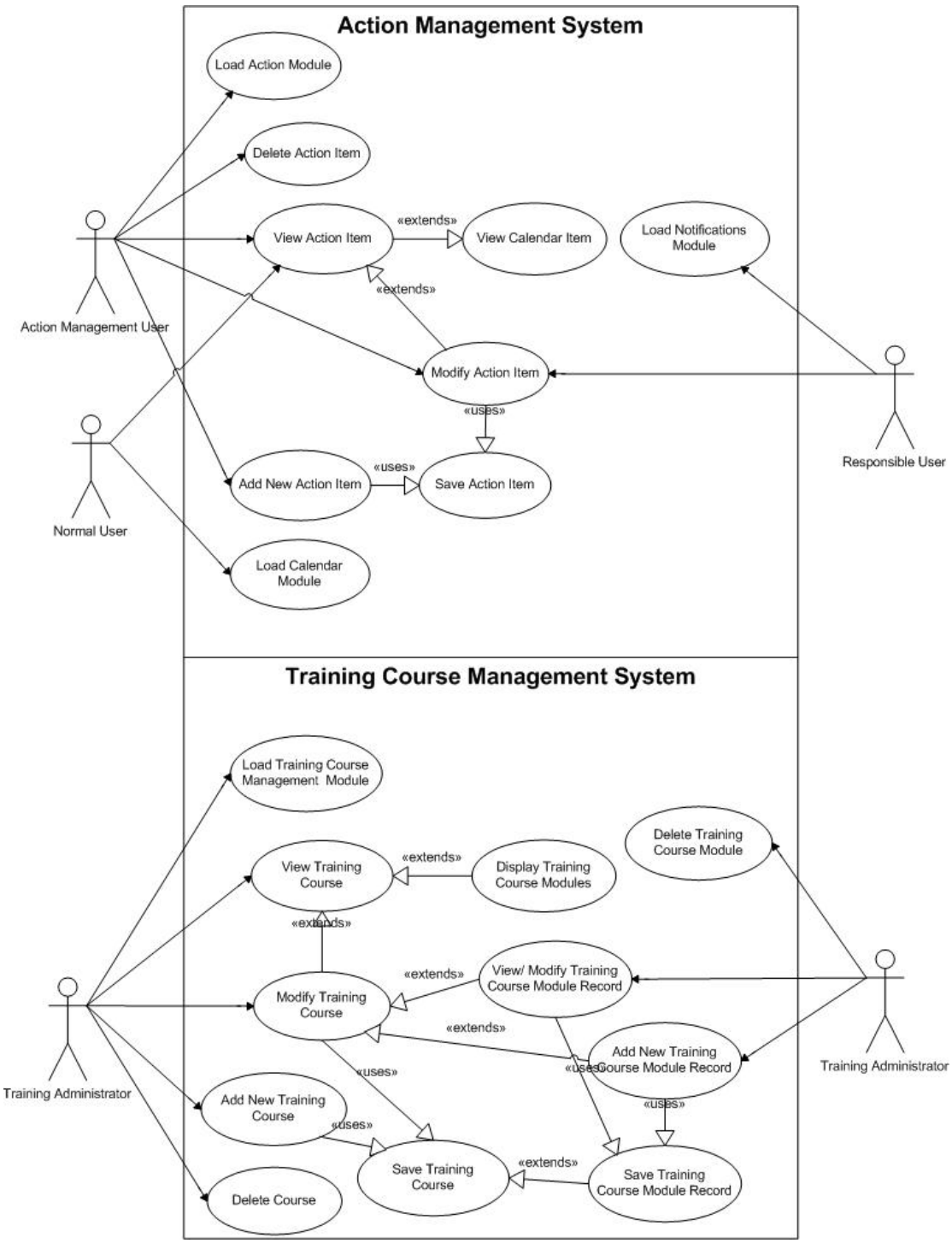

Administration System
Load User Account Management Section
Display Session Logs
«uses»
User Account Management
View/ Modify User Record
«uses»
Save User Record
«uses»
Add New User Record
Display Changes Recorded
«uses»
Display User Access History Logs
Administrator
Access Group Management
Load Group Permission Management Section
Display Access Permissions
«extends»
Add Access Permission
«extends»
View/ Modify Group Record
«uses»
Save Group Record
«uses»
Add New Group Record
Administrator
Category Management
Load Category Management Section
View/ Modify Category Record
«uses»
«uses»
Check File Uploaded Is Valid
Save Category Record
«uses»
«uses»
New Category Record
Administrator

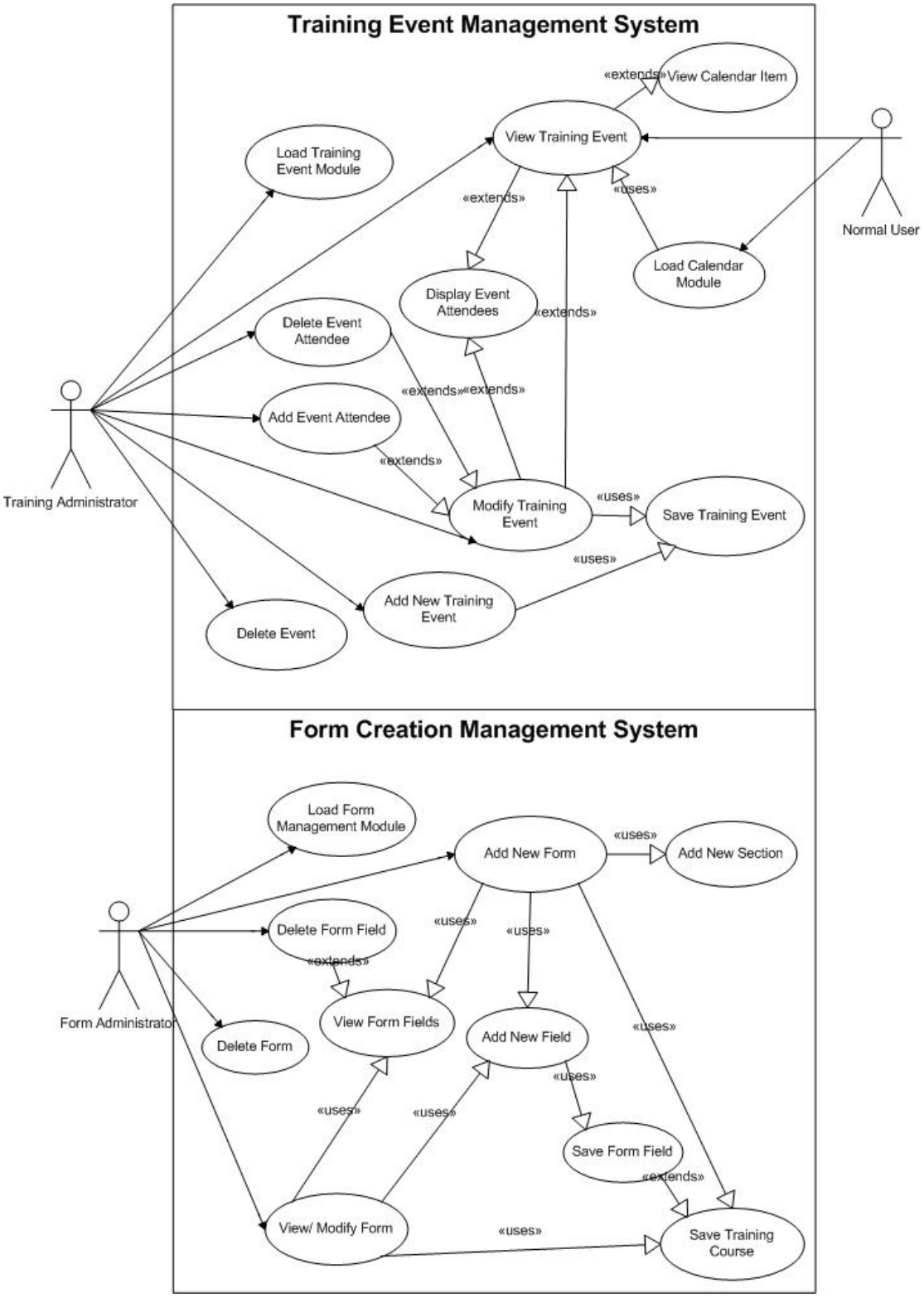

Administration System
Load Page Management Section
Page Management
View/ Modify Page Record
«uses»
Save Page Record
«uses»
Add New Page Record
Administrator
Load Section Management Section
Section Management
View/ Modify Section Record
«uses»
Save Section Record
«uses»
Add New Section Record
Administrator

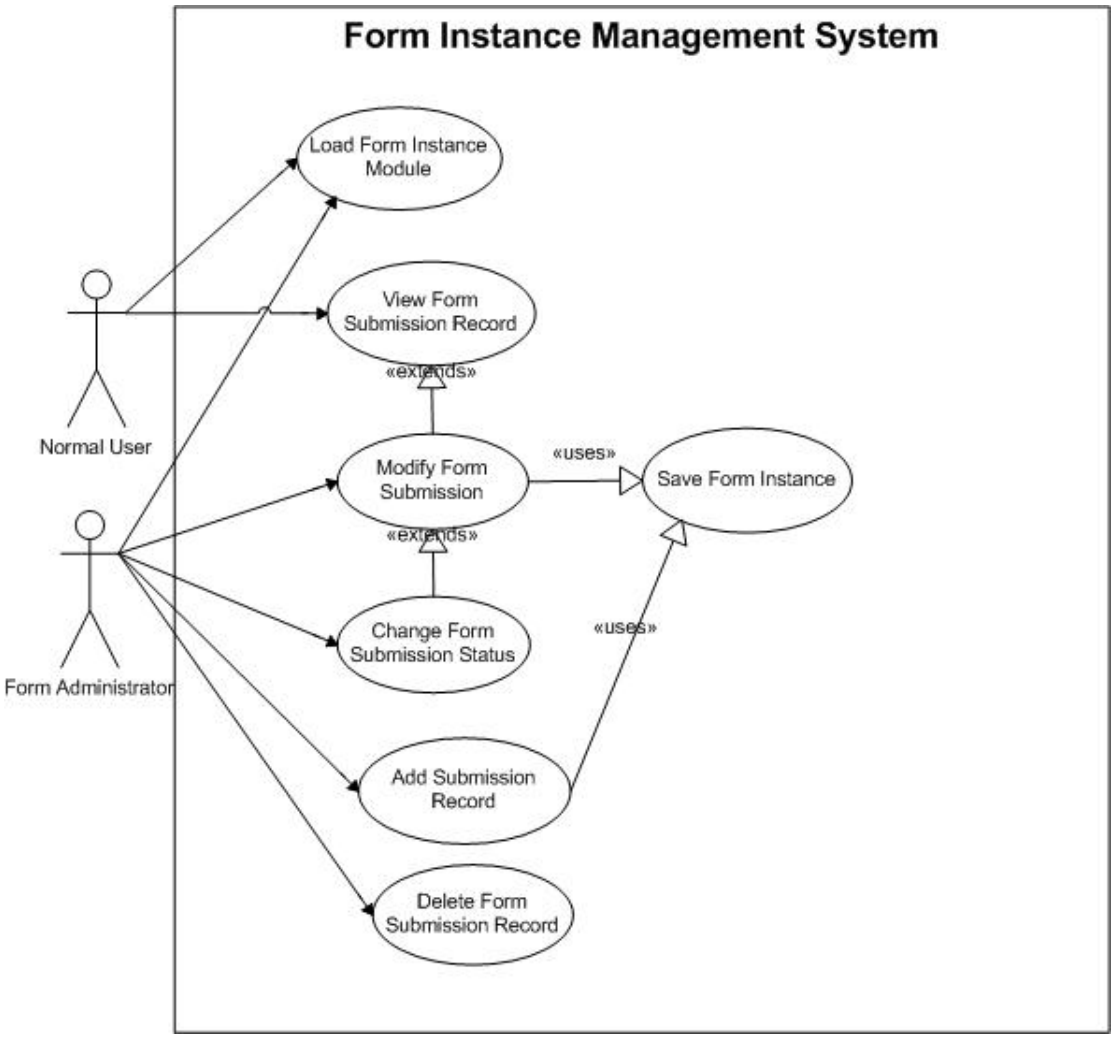

Document Library System
Load Document Library Module
«uses»
Display Document Categories
Manage Document Category Permissions
Administrator
View Document Record
«extends»
«extends»
Normal User
«extends»
Display Document Versions
Modify Document Record
Document Download
«extends»
Log Document Download
«uses»
Add New Document Record
Document Administrator
«uses»
Upload Document Version
«uses»
Save Document Record
Delete Document
Document Administrator
Employee Management System
Load Employee Module
«uses»
Display Departments & Sections
Normal User
Display License Registry
Display Phone Registry
«extends»
«extends»
View Employee Record
«extends»
Display Qualifications
«extends»
«extends»
Add/ Modify License
Modify Employee Record
«extends»
Add/ Modify Phone
«extends»
«uses»
Add/ Modify Qualification
Employee Administrator
Employee Administrator
Add New Employee
«uses»
Save Employee Record


Department Management System
Load Department Module
«uses»
Display Departments & Sections
Normal User
View Department Record
«extends»
Display Department Sections
«extends»
Modify Department Record
«uses»
«extends»
Assigning Department Section
«uses»
Save Department Section
«extends»
Save Department Record
«uses»
Department Administrator
Add New Department Record
Delete Department Record
Department Section Management System
Load Department Section Module
Normal User
View Department Section Record
«extends»
Modify Department Section Record
«uses»
Save Department Section Record
«uses»
Department Administrator
Add New Department Section
Delete Section Record


Audit Management System
Load Audit Module
View/ Modify Audit Item Record
«uses»
Save Audit Item Record
«uses»
«extends»
Add New Audit Item Record
Display Audit Item Records
«extends»
«extends»
«extends»
Audit Administrator
View/ Modify Audit Record
«uses»
Add New Audit Record
«uses»
Save Audit Record
Delete Audit Item
Delete Audit
Audit Scheduling System
Load Calendar
«uses»
Filter Audit Items
View Audit Item
«extends»
View Calendar Item
«extends»
Append Schedule Item Findings
«uses»
Save Audit Item
«uses»
Add New Audit Schedule Item
«extends»
«extends»
Change Item Findings Status
Add New Action Item
Delete Audit Schedule Item
Audit Administrator
«uses»
Normal User
Raise Action Item
«uses»
Save Raised Item

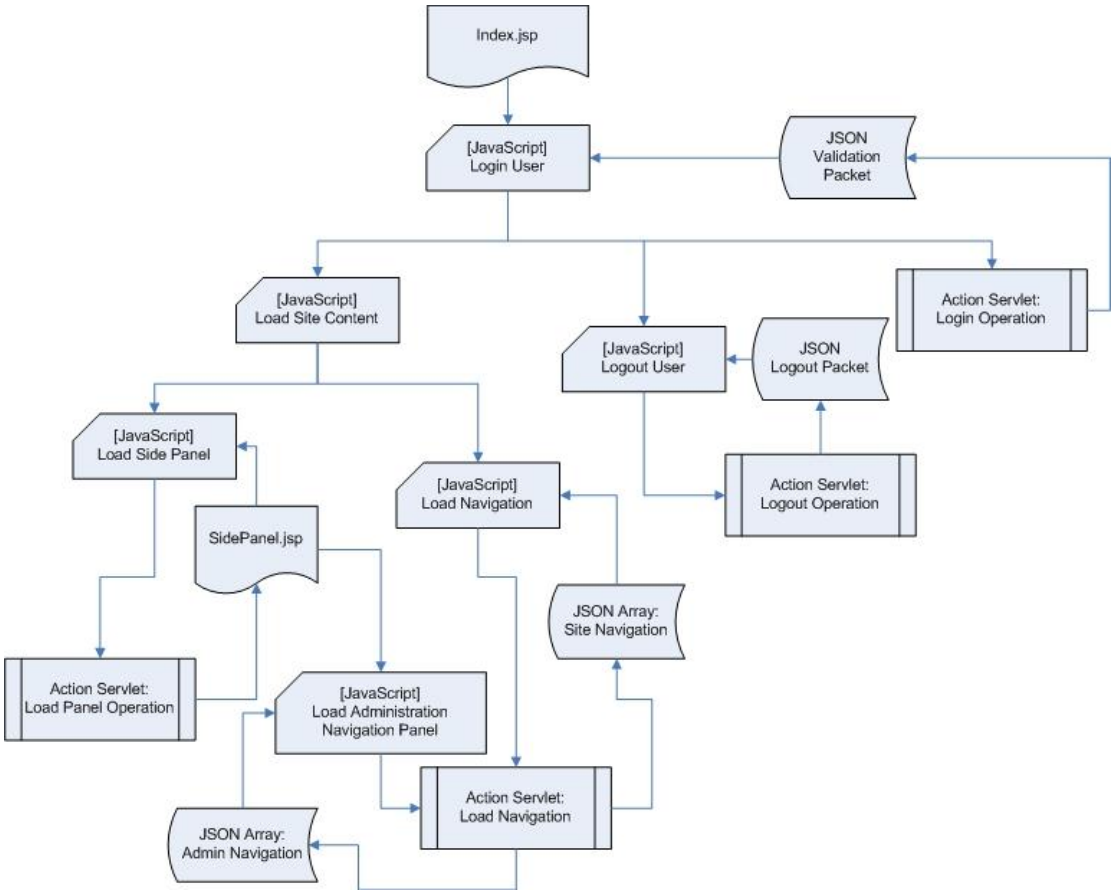

Action Management System
Load Action Module
Delete Action Item
View Action Item
«extends»
View Calendar Item
Load Notifications Module
«extends»
Action Management User
Modify Action Item
«uses»
Responsible User
Add New Action Item
«uses»
Save Action Item
Normal User
Load Calendar Module
Training Course Management System
Load Training Course Management Module
Delete Training Course Module
View Training Course
«extends»
Display Training Course Modules
«extends»
«extends»
View/ Modify Training Course Module Record
Modify Training Course
«extends»
Training Administrator
«extends»
Add New Training Course Module Record
«uses»
«uses»
«uses»
Training Administrator
Add New Training Course
«uses»
Save Training Course
«extends»
Save Training Course Module Record
Delete Course


Training Event Management System
«extends»
View Calendar Item
View Training Event
Load Training Event Module
«extends»
«uses»
Normal User
Load Calendar Module
Display Event Attendees
«extends»
Delete Event Attendee
«extends»
«extends»
Add Event Attendee
«extends»
Training Administrator
Modify Training Event
«uses»
Save Training Event
«uses»
Add New Training Event
Delete Event
Form Creation Management System
Load Form Management Module
Add New Form
«uses»
Add New Section
Delete Form Field
«uses»
«uses»
«extends»
Form Administrator
Delete Form
View Form Fields
Add New Field
«uses»
«uses»
«uses»
«uses»
Save Form Field
«extends»
View/ Modify Form
«uses»
Save Training Course

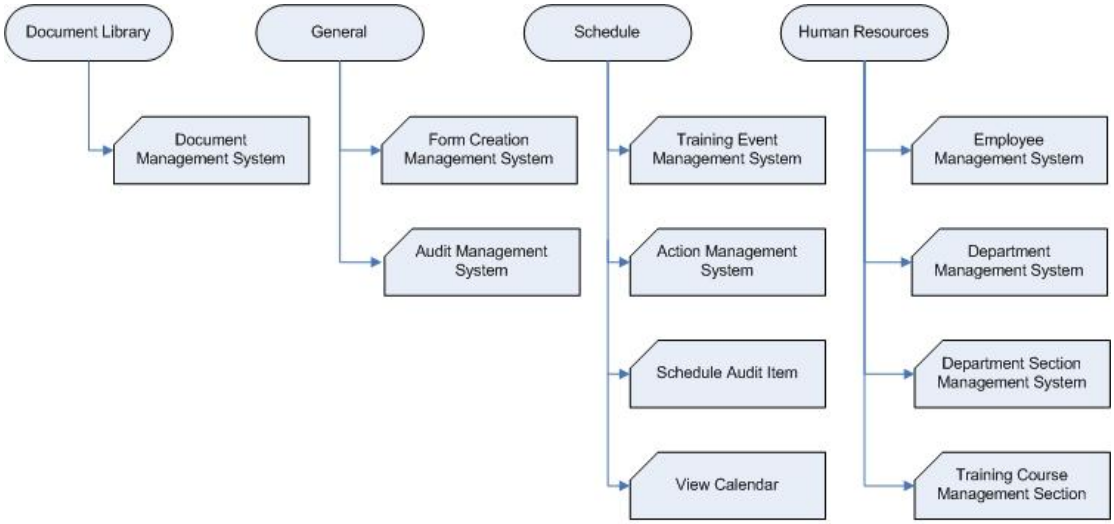

Form Instance Management System
Load Form Instance Module
View Form Submission Record
«extends»
Normal User
Modify Form Submission
«uses»
Save Form Instance
«extends»
Change Form Submission Status
«uses»
Form Administrator
Add Submission Record
Delete Form Submission Record

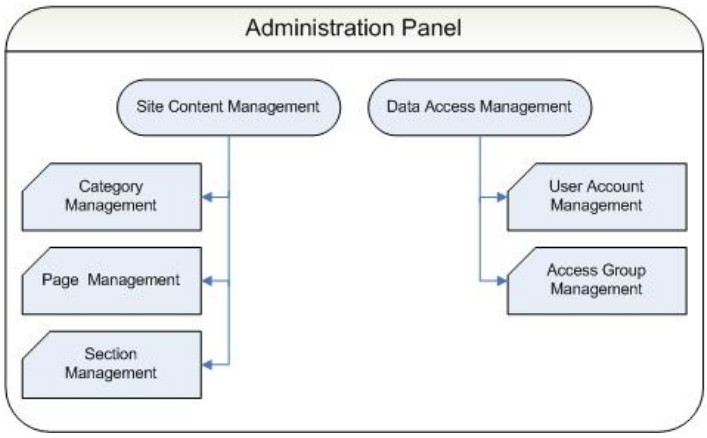
Site Navigation

Client-Side Scripting Request Structure



Navigation Hierarchy



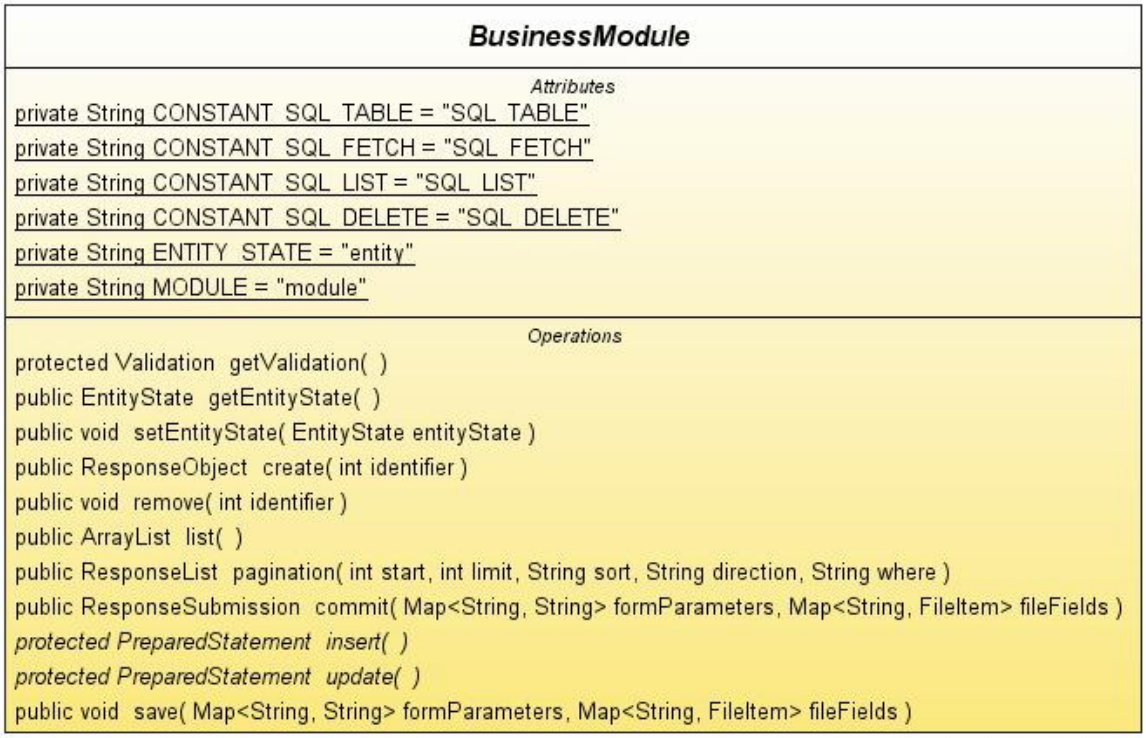


Class Diagrams

The Classes are created with no apparent association intentionally; which is due to the nature of 'everything' in the object being serialised once it is transferred back to the client. Instead, to enable for more swift retrieval in record objects, and to also allow for a more simplistic saving process; the associated records are identified using Integer properties.

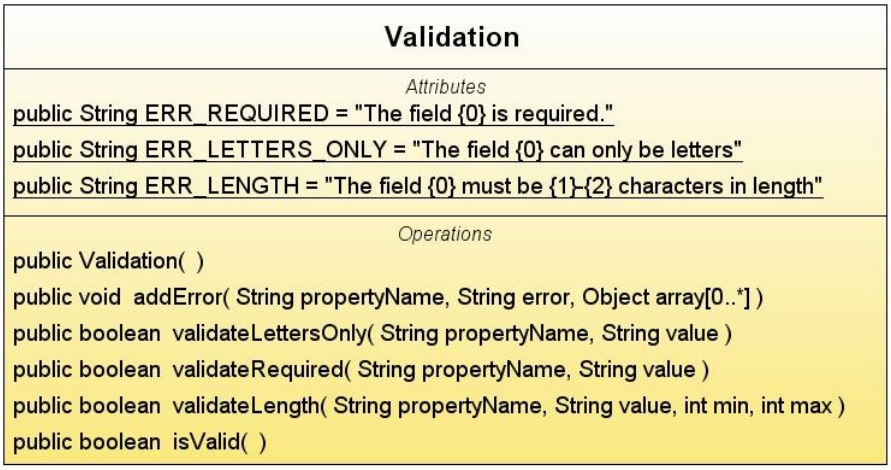
BusinessModule

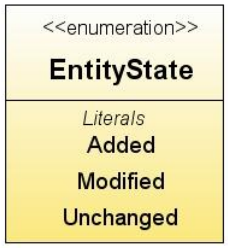
The BusinessModule is the parent of all packaged module children. The parent is mainly used to create generalised actions that can be used for all modules. It contains the methods that are used for the retrieval of objects either through just a regular list, or with Pagination. The single access point for all Form submissions is located in the commit method.



Validation

The BusinessModule also contains an inner class used for property validation (which extends HashMap); enabling the functionality of having Business Object level validation. The validation errors are stored in the BusinessModule, and hence are inherited to all the children; so therefore each record committed will have its own validation map. The validation is normally tested on the setter level within the classes, and should have a bypass Boolean test if the record is 'Unchanged' – thereby allowing records just being loaded from the database to skip all validation.

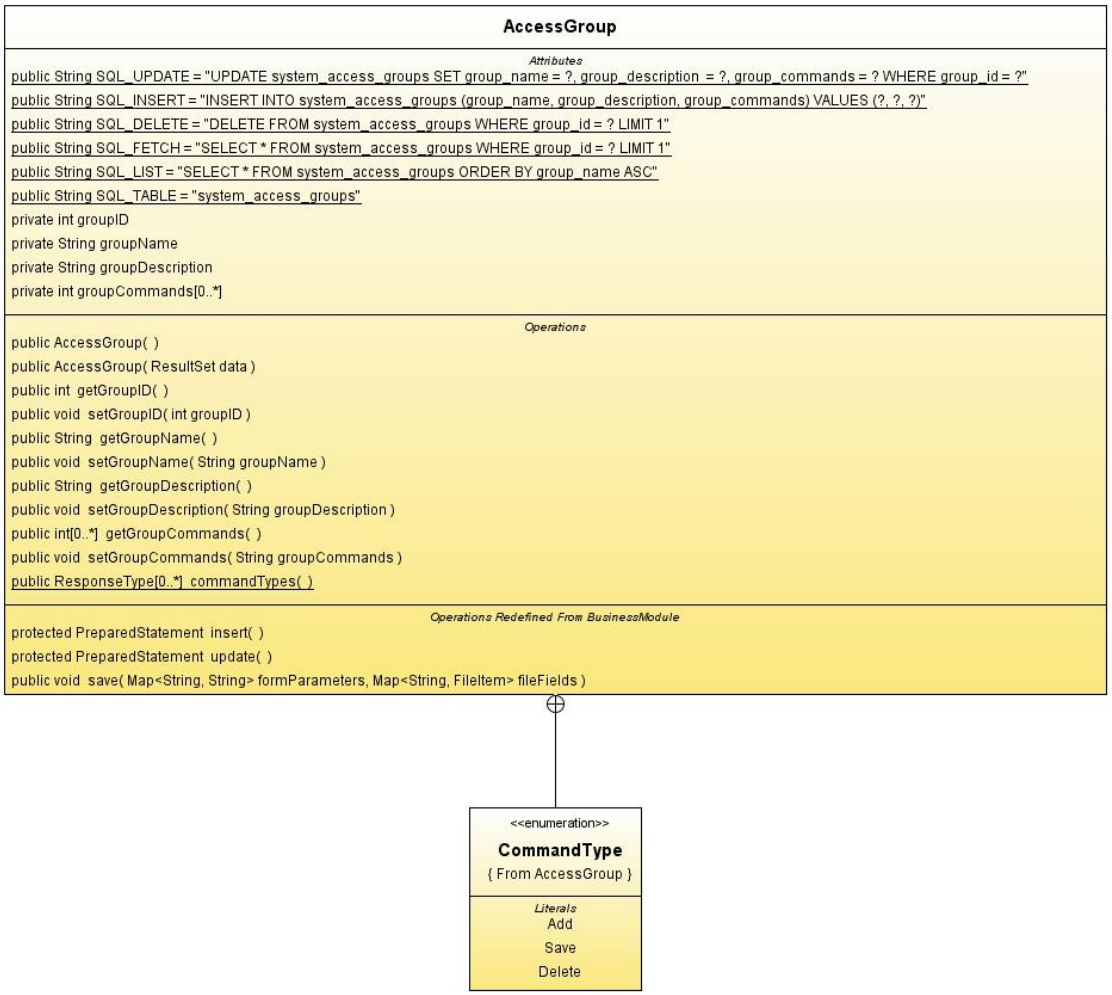




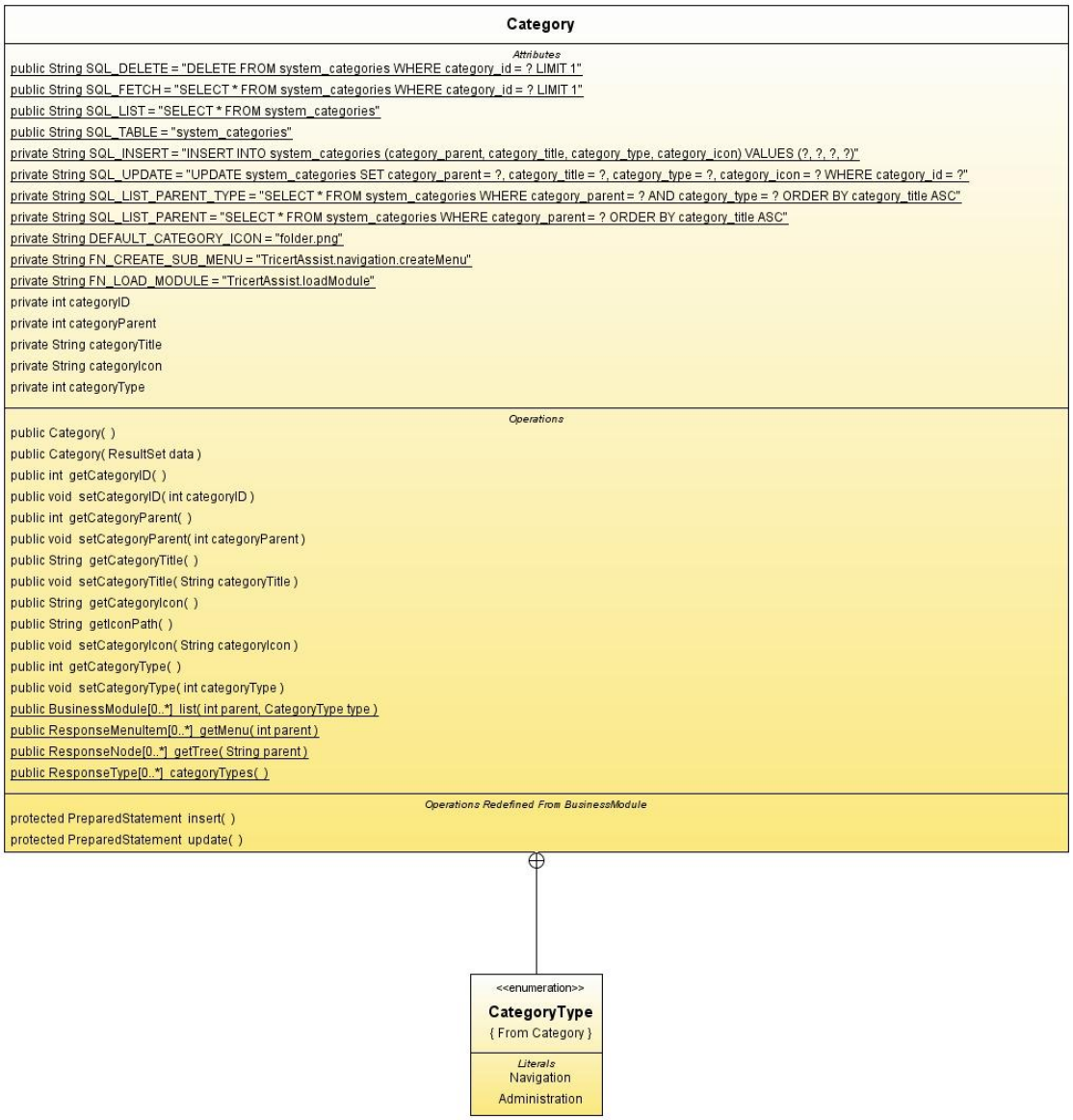
User


User
Attributes
public String SQL_UPDATE = "UPDATE system_user_accounts SET group_id = ?, employee_id = ?, user_name = ?, user_type = ? WHERE user_id = ?"
public String SQL_INSERT = "INSERT INTO system_user_accounts (group_id, employee_id, user_name, user_password, user_type) VALUES (?, ?, ?, SHA1(?), ?)"
public String SQL_DELETE = "DELETE FROM system_user_accounts WHERE user_id = ? LIMIT 1"
public String SQL_FETCH = "SELECT * FROM system_user_accounts WHERE user_id = ? LIMIT 1"
public String SQL_LIST = "SELECT * FROM system_user_accounts"
public String SQL_TABLE = "system_user_accounts"
private String ERR_PASSWORD_REQUIRED = "You must include a password when creating a new record"
private String ERR_USERNAME_TAKEN = "User name is already taken"
private String LOGOUT_MSG = "You have successfully logged out"
private String LOGOUT_TITLE = "Goodbye"
private String SQL_LOGIN_USER = "SELECT * FROM system_user_accounts WHERE user_name = ? AND user_password = SHA1(?)"
private String SQL_CHANGE_PASSWORD = "UPDATE system_user_accounts SET user_password = SHA1(?) WHERE user_id = ?"
private String SQL_CHECK_USERNAME = "SELECT user_id, user_name FROM system_user_accounts WHERE user_name = ? AND user_id != ?"
private String ERROR_INVALID_CREDENTIALS = "You have supplied invalid user credentials"
private String ERROR_LOGIN_FAILED = "Login Failed"
private String MSG_LOGIN_SUCCESS = "Login Success"
private String MSG_NOW_AUTHENTICATED = "You are now authenticated"
private String REGEX_PASSWORD = "^(?=.*[a-zA-Z])(?=.*[0-9])[a-zA-Z0-9_-]{8,15}$"
private String ERR_PASSWORD_DOESNT_MATCH = "Confirmation password does not match"
private String ERR_PASSWORD_STRENGTH = "Password must be 8-15 alphanumeric characters (can contain hyphens and underscores)"
private int userID
private String userName
private String password
private String confirmationPassword
private int groupID
private int employeeID
private int userType
Operations
public User( )
public User(ResultSet data)
public int getUserID( )
public void setUserID(int userID)
public String getUserName( )
public void setUserName(String userName)
public String getPassword( )
public void setPassword(String password)
public String getConfirmationPassword( )
public void setConfirmationPassword(String confirmationPassword)
public int getGroupID( )
public void setGroupID(int groupID)
public int getEmployeeID( )
public void setEmployeeID(int employeeID)
public int getUserType( )
public void setUserType(int userType)
public AccessGroup getAccessGroup( )
public boolean isAdminAccount( )
private boolean isUserNameValid(String userName)
private boolean isPasswordValid(String password)
public ResponseMessage doLogout( )
public ResponseMessage doLogin(Map<String, String> formParameters, Map<String, FileItem> fileFields)
public User getLoggedInUser( )
public ResponseType[0..*] userTypes( )
Operations Redefined From BusinessModule
protected PreparedStatement insert( )
protected PreparedStatement update( )
<<enumeration>>
UserType
{ From User }
Literals
Normal
Admin

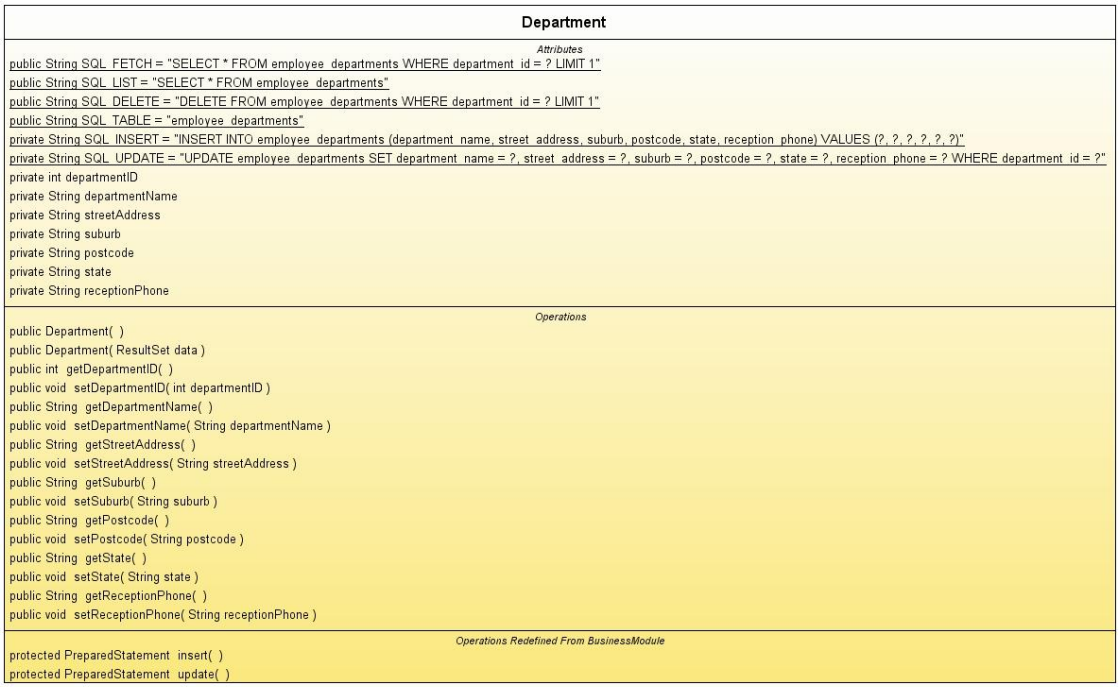
AccessGroup



Category


Category
Attributes
public String SQL_DELETE = "DELETE FROM system_categories WHERE category_id = ? LIMIT 1"
public String SQL_FETCH = "SELECT * FROM system_categories WHERE category_id = ? LIMIT 1"
public String SQL_LIST = "SELECT * FROM system_categories"
public String SQL_TABLE = "system_categories"
private String SQL_INSERT = "INSERT INTO system_categories (category_parent, category_title, category_type, category_icon) VALUES (?, ?, ?, ?)"
private String SQL_UPDATE = "UPDATE system_categories SET category_parent = ?, category_title = ?, category_type = ?, category_icon = ? WHERE category_id = ?"
private String SQL_LIST_PARENT_TYPE = "SELECT * FROM system_categories WHERE category_parent = ? AND category_type = ? ORDER BY category_title ASC"
private String SQL_LIST_PARENT = "SELECT * FROM system_categories WHERE category_parent = ? ORDER BY category_title ASC"
private String DEFAULT_CATEGORY_ICON = "folder.png"
private String FN_CREATE_SUB_MENU = "TricertAssist.navigation.createMenu"
private String FN_LOAD_MODULE = "TricertAssist.loadModule"
private int categoryID
private int categoryParent
private String categoryTitle
private String categoryIcon
private int categoryType
Operations
public Category( )
public Category(ResultSet data)
public int getCategoryID( )
public void setCategoryID(int categoryID)
public int getCategoryParent( )
public void setCategoryParent(int categoryParent)
public String getCategoryTitle( )
public void setCategoryTitle(String categoryTitle)
public String getCategoryIcon( )
public String getIconPath( )
public void setCategoryIcon(String categoryIcon)
public int getCategoryType( )
public void setCategoryType(int categoryType)
public BusinessModule[0..*] list(int parent, CategoryType type)
public ResponseMenuItem[0..*] getMenu(int parent)
public ResponseNode[0..*] getTree(String parent)
public ResponseType[0..*] categoryTypes( )
Operations Redefined From BusinessModule
protected PreparedStatement insert( )
protected PreparedStatement update( )
<<enumeration>>
CategoryType
{ From Category }
Literals
Navigation
Administration

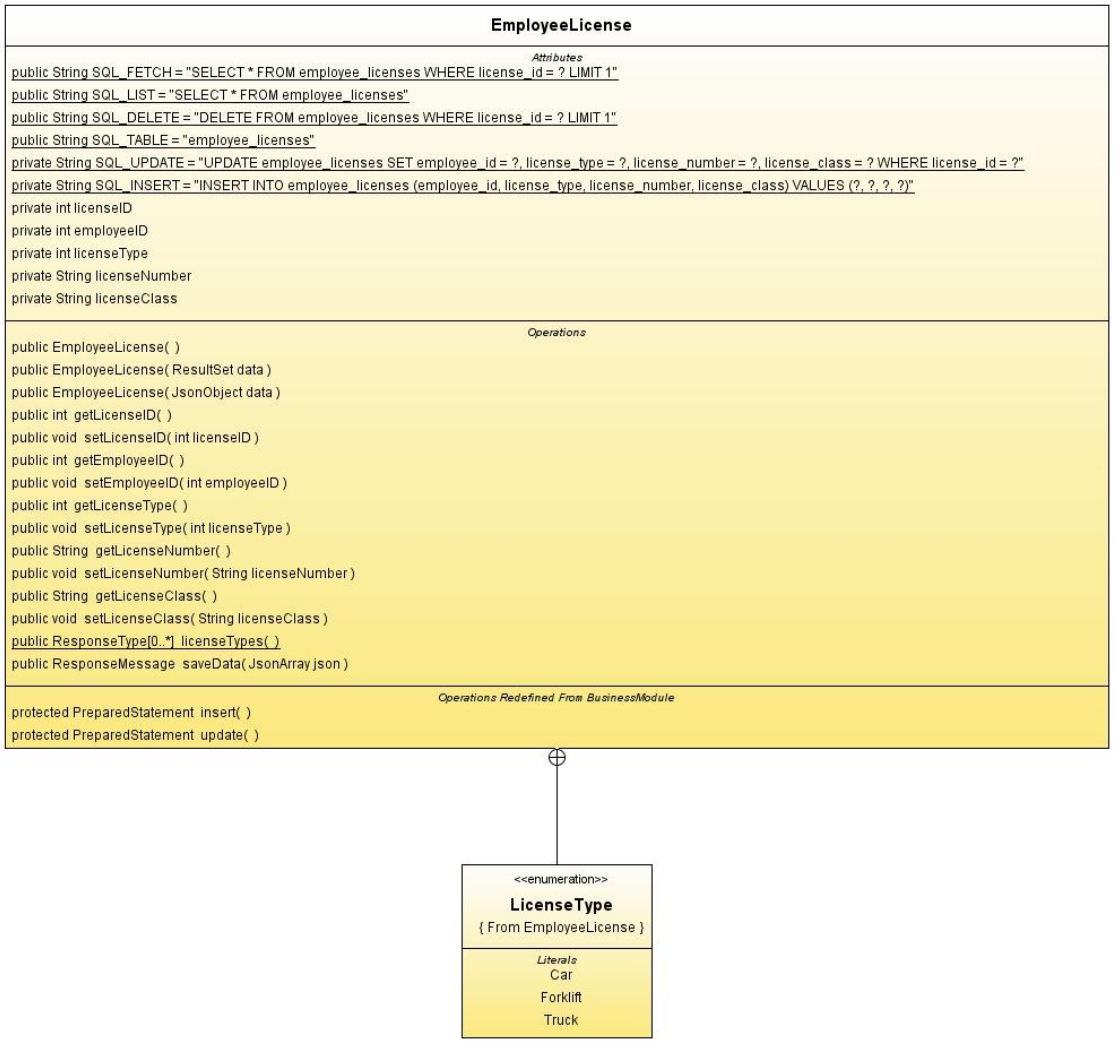
Department


Department
Attributes
public String SQL_FETCH = "SELECT * FROM employee_departments WHERE department_id = ? LIMIT 1"
public String SQL_LIST = "SELECT * FROM employee_departments"
public String SQL_DELETE = "DELETE FROM employee_departments WHERE department_id = ? LIMIT 1"
public String SQL_TABLE = "employee_departments"
private String SQL_INSERT = "INSERT INTO employee_departments (department_name, street_address, suburb, postcode, state, reception_phone) VALUES (?, ?, ?, ?, ?, ?)"
private String SQL_UPDATE = "UPDATE employee_departments SET department_name = ?, street_address = ?, suburb = ?, postcode = ?, state = ?, reception_phone = ? WHERE department_id = ?"
private int departmentID
private String departmentName
private String streetAddress
private String suburb
private String postcode
private String state
private String receptionPhone
Operations
public Department( )
public Department(ResultSet data)
public int getDepartmentID( )
public void setDepartmentID(int departmentID)
public String getDepartmentName( )
public void setDepartmentName(String departmentName)
public String getStreetAddress( )
public void setStreetAddress(String streetAddress)
public String getSuburb( )
public void setSuburb(String suburb)
public String getPostcode( )
public void setPostcode(String postcode)
public String getState( )
public void setState(String state)
public String getReceptionPhone( )
public void setReceptionPhone(String receptionPhone)
Operations Redefined From BusinessModule
protected PreparedStatement insert( )
protected PreparedStatement update( )

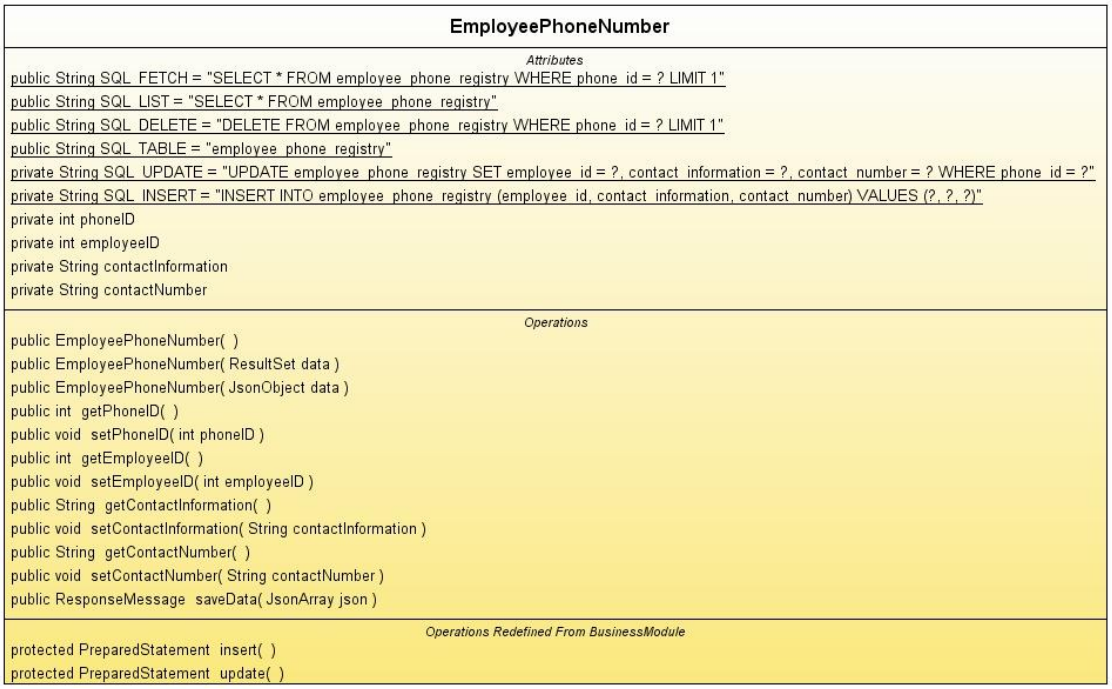
Employee


Employee
Attributes
public String SQL_FETCH = "SELECT * FROM employees WHERE employee_id = ? LIMIT 1"
public String SQL_LIST = "SELECT * FROM employees"
public String SQL_DELETE = "DELETE FROM employees WHERE employee_id = ? LIMIT 1"
public String SQL_TABLE = "employees"
private String SQL_UPDATE = "UPDATE employees SET employee_section_id = ?, department
private String SQL_INSERT = "INSERT INTO employees (employee_section_id, department_id,
private int employeeID
private int employeeSectionID
private int departmentID
private int supervisorID
private int employeeType
private String firstName
private String lastName
private String emailAddress
private String streetAddress
private String suburb
private String postcode
private String state
Operations
public Employee( )
public Employee(ResultSet data)
public int getEmployeeID( )
public void setEmployeeID(int employeeID)
public int getEmployeeSectionID( )
public void setEmployeeSectionID(int employeeSectionID)
public int getDepartmentID( )
public void setDepartmentID(int departmentID)
public int getSupervisorID( )
public void setSupervisorID(int supervisorID)
public int getEmployeeType( )
public void setEmployeeType(int employeeType)
public String getFirstName( )
public void setFirstName(String firstName)
public String getLastName( )
public void setLastName(String lastName)
public String getEmailAddress( )
public void setEmailAddress(String emailAddress)
public String getStreetAddress( )
public void setStreetAddress(String streetAddress)
public String getSuburb( )
public void setSuburb(String suburb)
public String getPostcode( )
public void setPostcode(String postcode)
public String getState( )
public void setState(String state)
public ResponseType[0..*] stateTypes( )
public ResponseType[0..*] employeeTypes( )
Operations Redefined From BusinessModule
protected PreparedStatement insert( )
protected PreparedStatement update( )
<<enumeration>>
EmployeeType
{ From Employee }
Literals
FullTime
PartTime
Casual
<<enumeration>>
State
{ From Employee }
Literals
ACT
NSW
NT
QLD
SA
TAS
VIC
WA

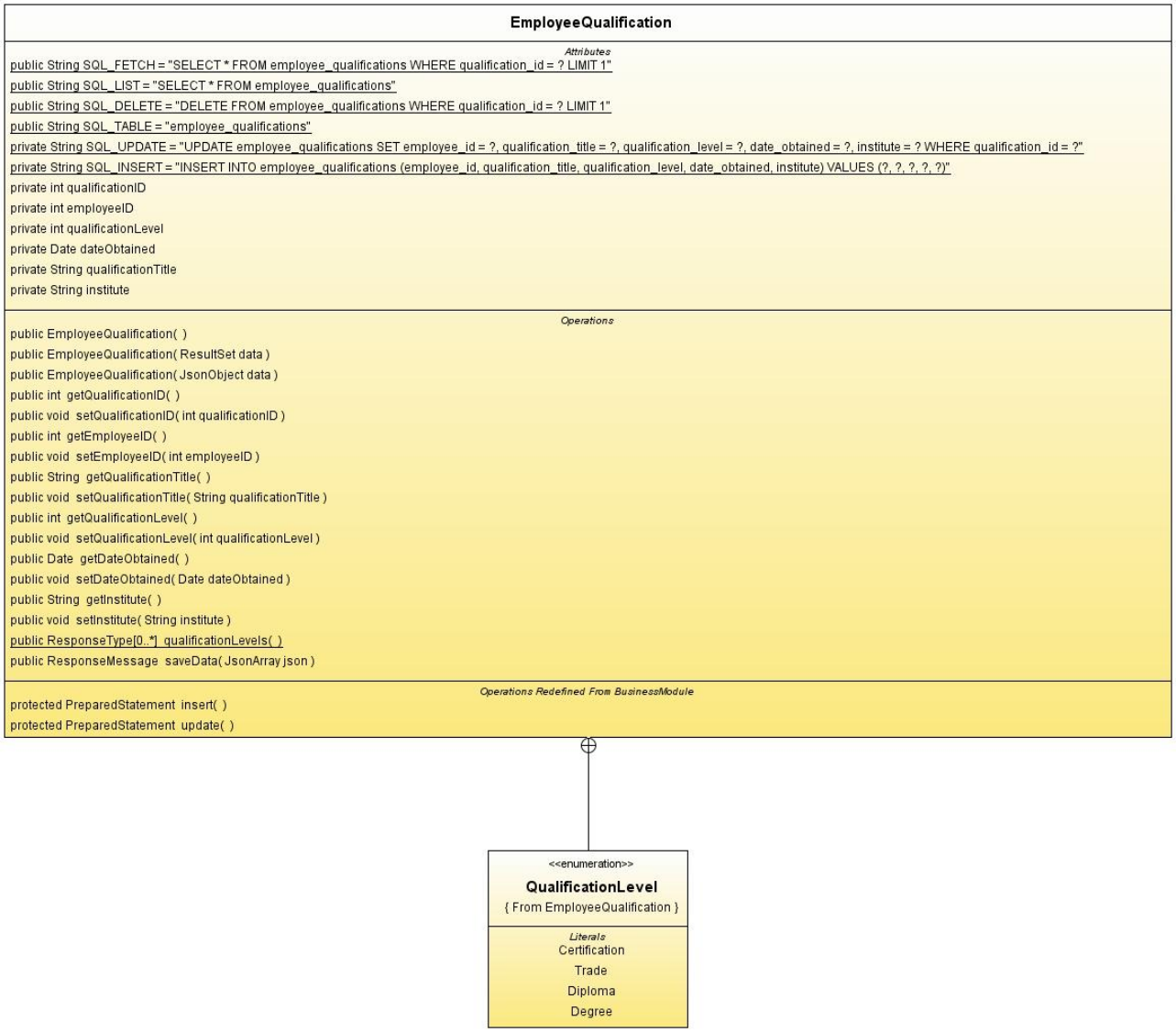
EmployeeLicense



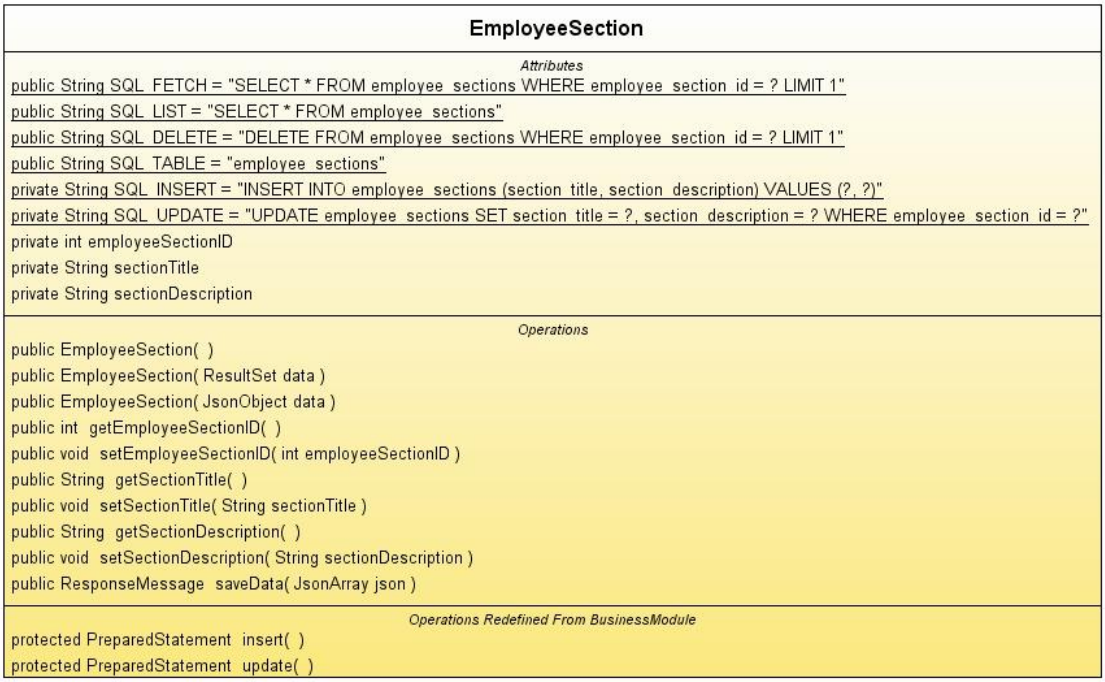
EmployeePhoneNumber



EmployeeQualification



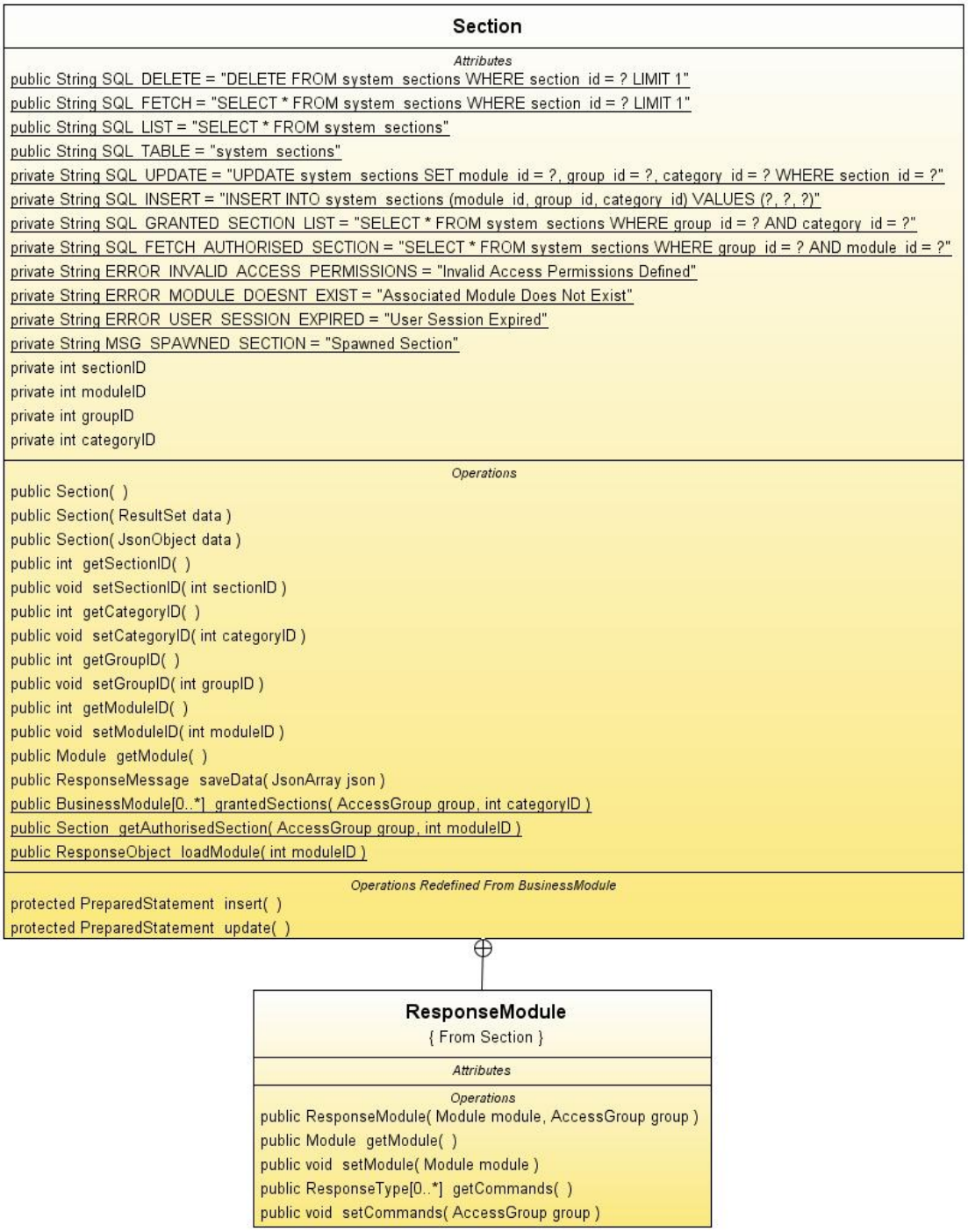
EmployeeSection



Module

Module
Attributes
public String SQL_FETCH = "SELECT * FROM system_modules WHERE module_id = ? LIMIT 1"
public String SQL_DELETE = "DELETE FROM system_modules WHERE module_id = ? LIMIT 1"
public String SQL_LIST = "SELECT * FROM system_modules"
public String SQL_TABLE = "system_modules"
private String SQL_UPDATE = "UPDATE system_modules SET module_title = ?, module_icon = ?, module_source = ?, module_name = ?, module_commands = ? WHERE module_id = ?"
private String SQL_INSERT = "INSERT INTO system_modules (module_title, module_icon, module_source, module_name, module_commands) VALUES (?, ?, ?, ?, ?)"
private String DEFAULT_MODULE_ICON = "application.png"
private int moduleID
private String moduleTitle
private String moduleIcon
private String moduleSource
private String moduleName
private int moduleCommands[0..*]
Operations
public Module( )
public Module(ResultSet data)
public int getModuleID( )
public void setModuleID(int moduleID)
public String getModuleTitle( )
public void setModuleTitle(String moduleTitle)
public String getModuleIcon( )
public String getIconPath( )
public void setModuleIcon(String moduleIcon)
public String getModuleName( )
public void setModuleName(String moduleName)
public int[0..*] getModuleCommands( )
public void setModuleCommands(String moduleCommands)
public String getModuleSource( )
public void setModuleSource(String moduleSource)
public ResponseType[0..*] getSimpleList( )
public ResponseType[0..*] getModuleList( )
Operations Redefined From BusinessModule
protected PreparedStatement insert( )
protected PreparedStatement update( )
public void save(Map<String, String> formParameters, Map<String, FileItem> fileFields)

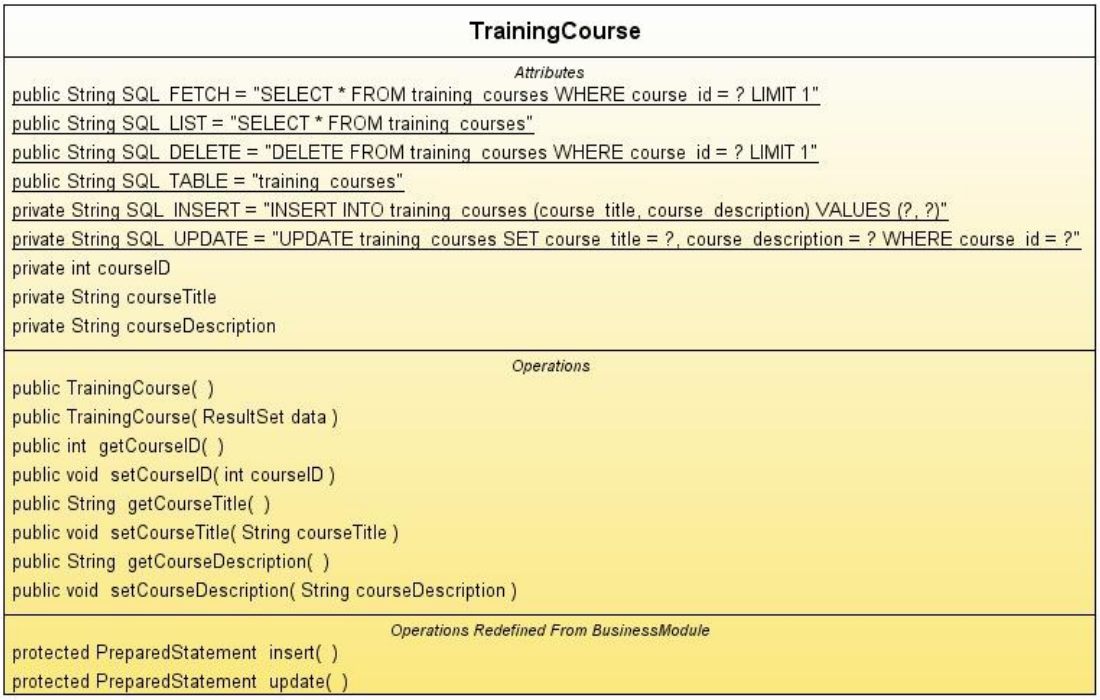
Section


Section
Attributes
public String SQL_DELETE = "DELETE FROM system_sections WHERE section_id = ? LIMIT 1"
public String SQL_FETCH = "SELECT * FROM system_sections WHERE section_id = ? LIMIT 1"
public String SQL_LIST = "SELECT * FROM system_sections"
public String SQL_TABLE = "system_sections"
private String SQL_UPDATE = "UPDATE system_sections SET module_id = ?, group_id = ?, category_id = ? WHERE section_id = ?"
private String SQL_INSERT = "INSERT INTO system_sections (module_id, group_id, category_id) VALUES (?, ?, ?)"
private String SQL_GRANTED_SECTION_LIST = "SELECT * FROM system_sections WHERE group_id = ? AND category_id = ?"
private String SQL_FETCH_AUTHORISED_SECTION = "SELECT * FROM system_sections WHERE group_id = ? AND module_id = ?"
private String ERROR_INVALID_ACCESS_PERMISSIONS = "Invalid Access Permissions Defined"
private String ERROR_MODULE_DOESNT_EXIST = "Associated Module Does Not Exist"
private String ERROR_USER_SESSION_EXPIRED = "User Session Expired"
private String MSG_SPAWNED_SECTION = "Spawned Section"
private int sectionID
private int moduleID
private int groupID
private int categoryID
Operations
public Section( )
public Section(ResultSet data)
public Section(JsonObject data)
public int getSectionID( )
public void setSectionID(int sectionID)
public int getCategoryID( )
public void setCategoryID(int categoryID)
public int getGroupID( )
public void setGroupID(int groupID)
public int getModuleID( )
public void setModuleID(int moduleID)
public Module getModule( )
public ResponseMessage saveData(JsonArray json)
public BusinessModule[0..*] grantedSections(AccessGroup group, int categoryID)
public Section getAuthorisedSection(AccessGroup group, int moduleID)
public ResponseObject loadModule(int moduleID)
Operations Redefined From BusinessModule
protected PreparedStatement insert( )
protected PreparedStatement update( )
ResponseModule
{ From Section }
Attributes
Operations
public ResponseModule(Module module, AccessGroup group)
public Module getModule( )
public void setModule(Module module)
public ResponseType[0..*] getCommands( )
public void setCommands(AccessGroup group)

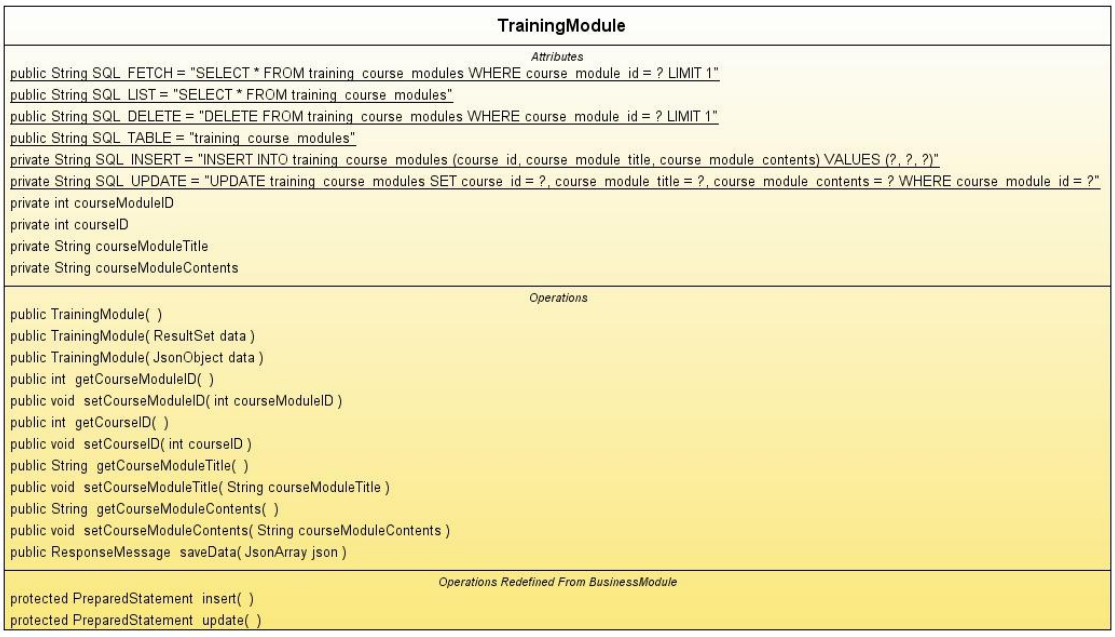
SessionLog


Visit
{ From SessionLog }
Attributes
private String SQL_INSERT = "INSERT INTO system_session_h
private String SQL_LIST = "SELECT h.visit_id, h.session_id, h.s
private int visitID
private int sessionID
private int sectionID
private int moduleID
private String moduleTitle
private Date visitTime
Operations
public Visit( )
public Visit(ResultSet data)
public int getVisitID( )
public void setVisitID(int visitID)
public int getSessionID( )
public void setSessionID(int sessionID)
public int getSectionID( )
public void setSectionID(int sectionID)
public int getModuleID( )
public void setModuleID(int moduleID)
public String getModuleTitle( )
public void setModuleTitle(String moduleTitle)
public Date getVisitTime( )
public void setVisitTime(Date visitTime)
public void save( )
history
0..*
SessionLog
Attributes
private String SQL_INSERT = "INSERT INTO system_sessions (user_id, session_date) VALUES (?, ?)"
private String SQL_TABLE = "system_sessions"
private int sessionID
private int userID
private Date sessionDate
Operations
public SessionLog( )
public SessionLog(ResultSet data)
public int getSessionID( )
public void setSessionID(int logID)
public int getUserID( )
public void setUserID(int userID)
public Date getSessionDate( )
public void setSessionDate(Date date)
public Visit[0..*] getHistory( )
public void setHistory(Visit history[0..*])
public void save( )
public void add(Section section)
public SessionLog getCurrentLog( )
public SessionLog create(User user)
public ResponseList list(int start, int limit, String sort, String direction, int userID)

TrainingCourse



TrainingModule



Website Theme

Tricert.css

```
#aiml_bot {
	background-image: url(images/main_banner_right.png);
	background-repeat: no-repeat;
	float: right;
	height: 205px;
	position: relative;
	width: 410px;
}
#banner_container {
	height: 205px;
	position: relative;
	width: 960px;
}
#body_container {
	position: relative;
	width: 960px;
}
#flash_animation {
	background-image: url(images/main_banner_left.png);
	background-repeat: no-repeat;
	float: left;
	height: 205px;
	position: relative;
	width: 550px;
}
#header_container {
	position: relative;
	width: 960px;
	z-index: 1;
}
#mdi_commands {
	float: left;
	height: 30px;
	margin-bottom: 1px;
	margin-top: 4px;
	width: 200px;
}
#main_container {
	margin-left: auto;
	margin-right: auto;
	margin-top: 20px;
	position: relative;
	text-align: left;
	width: 960px;
}
```

```
#mdi_container {
	overflow: hidden;
}

#navigation {
	height: 50px;
	  margin-bottom: 5px;
	position: relative;
	width: 960px;
	z-index: 1;
}

#side_bar {
	float: left;
	margin-top: 5px;
	position: relative;
	width: 250px;
}

#site_content {
	float: right;
	margin-top: 5px;
	position: relative;
	width: 700px;
}

#mdi_body {
	background-image: url(images/mdi_body.png);
	background-repeat: repeat-y;
	border-left: 1px Solid #5c6671;
	border-right: 1px Solid #5c6671;
	height: 500px;
	overflow: hidden;
	position: relative;
	width: 698px;
}

#mdi_content_container {
	margin-top: 5px;
	overflow: hidden;
	position: relative;
	width: 700px;
}

#mdi_footer {
	background-image: url(images/mdi_footer.png);
	background-repeat: no-repeat;
	height: 20px;
	position: relative;
	width: 700px;
}

#mdi_header {
	position: relative;
	width: inherit;
}
```

```
#mdi_toolbar {
	float: left;
	height: 35px;
	position: relative;
	width: 700px;
}

#mdi_header_left {
	background-image: url(images/mdi_header_left.png);
	background-repeat: no-repeat;
	float: left;
	height: 35px;
	position: relative;
	width: 20px;
}

#mdi_header_repeat {
	background-image: url(images/mdi_header_repeat.png);
	background-repeat: repeat-x;
	float: left;
	height: inherit;
	overflow: hidden;
	position: relative;
	width: 660px;
}

#mdi_header_right {
	background-image: url(images/mdi_header_right.png);
	background-repeat: no-repeat;
	float: right;
	height: 35px;
	position: relative;
	width: 20px;
}

#panel_container {
	margin-top: 5px;
	position: relative;
	width: 250px;
}

#panel_content {
	margin: 10px;
	position: relative;
	width: 240px;
}
```

```
#panel_footer {
	background-image: url(images/panel_footer.png);
	background-repeat: no-repeat;
	height: 20px;
	position: relative;
	width: inherit;
}

#panel_login_area {
	background-image: url(images/panel_login_area.png);
	background-repeat: no-repeat;
	height: 40px;
	position: relative;
	width: inherit;
}

#panel_repeat {
	background-image: url(images/panel_repeat.png);
	background-repeat: repeat-y;
	overflow: hidden;
	position: relative;
	width: inherit;
}
body {
	background-color: #e5eafc;
	color: #789;
	font: 13px/16px "Lucida Grande", Arial, Sans-serif;
	margin-left: 0;
	margin-top: 0;
	text-align: left;
}
```

Index.jsp

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title>Tricert Assist</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <link rel="stylesheet" type="text/css" href="theme/tricert.css" media="screen" />
    <link rel="stylesheet" type="text/css" href="theme/ext-all.css" media="screen" />
</head>
    <body>
      <div id="main_container">
            <div id="header_container">
                  <div id="navigation">

                  </div>
                  <div id="banner_container">
                        <div id="flash_animation">

                        </div>
                        <div id="aiml_bot">

                        </div>
                  </div>
            </div>
            <div id="body_container">
                        <div id="side_bar">
                            <div id="panel_container">
                                  <div id="panel_login_area"></div>
                                  <div id="panel_repeat">
                                      <div id="panel_content">

                                      </div>
                                  </div>
                                  <div id="panel_footer"></div>
                             </div>
                        </div>
                        <div id="site_content">
                            <div id="mdi_content_container">
                                    <div id="mdi_header">
                                            <div id="mdi_toolbar"></div>
                                            <div id="mdi_header_left"></div>
                                            <div id="mdi_header_repeat">
                                                <div id="mdi_commands"></div>
                                                <div id="mdi_taskbar">

                                                </div>
                                            </div>
                                            <div id="mdi_header_right"></div>
                                    </div>
                                    <div id="mdi_body">

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

```
                        </div>
                    </div>
                </div>
            </div>
              <script type="text/javascript" src="js/Ext-Base.js"></script>
              <script type="text/javascript" src="js/Ext-All.js"></script>
              <script type="text/javascript" src="js/Ext-Loader.js"></script>
              <script type="text/javascript" src="js/Api.js"></script>
              <script type="text/javascript" src="js/Tricert.js"></script>
              <!-- GMaps API Key that works for www.extjs.com -->
              <script src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=ABQIAAAA2CKu_qQN-
JHtlfQ5L7BLlRRadLUjZPtnrRT4mXZqcP4UUH-2OxREmPm3GpN_NHsHuvuHd-QKI4YoRg"
type="text/javascript"></script>
              <!-- GMaps API Key that works for localhost -->
              <!--<script
src="http://maps.google.com/maps?file=api&amp;v=2.x&amp;key=ABQIAAAA2CKu_qQN-
JHtlfQ5L7BLlRT2yXp_ZAY8_ufC3CFXhHIE1NvwkxQl3I3p2yrGARYK4f4bkjp9NHpm5w"
type="text/javascript"></script>-->
    </body>
</html>
```

JavaScript Source

Tricert.js

```
Ext.onReady(function () {
    Ext.QuickTips.init();
    Ext.form.Field.prototype.msgTarget = 'side';
    Ext.app.REMOTING_API.enableBuffer = 100;
    Ext.Direct.addProvider(Ext.app.REMOTING_API);
    Ext.ScriptMgr.load({
        scripts: ['js/plugins/Ext.ux.TaskBar.js', 'js/plugins/Ext.ux.Mdi.js', 'js/plugins/Ext.ux.Notification.js',
'js/plugins/Ext.ux.grid.RowEditor.js'],
        callback: function () {
            Navigation = Ext.extend(Ext.Toolbar, {
                renderTo: 'navigation',
                width: 960,
                height: 50,
                style: 'border: 1px solid #000; padding-top: 10px;',
                load: function () {
                    var instance = this;
                    instance.removeAll();
                    Category.getMenu(0, function (result, e) {
                        instance.populate(result, instance);
                        instance.doLayout();
                    });
                },
                populate: function (records, menu) {
                    for (var i = 0; i < records.length; i++) {
                        if (records[i].handler) {
                            eval('records[i].handler = ' + records[i].handler);
                        }
                        if (records[i].menu) {
                            eval('records[i].menu = ' + records[i].menu);
                        }
                        menu.add(records[i]);
                    }
                },
                createMenu: function (parentId) {
                    var menu = new Ext.menu.Menu();
                    var main = this;
                    menu.on('show', function () {
                        var instance = this;
                        instance.menu.removeAll();
                        instance.menu.add('<span class=\'loading-indicator\'>Loading...</span>');
                        Category.getMenu(instance.parentId, function (result, e) {
                            instance.menu.removeAll();
                            if (result.length > 0) {
                                main.populate(result, instance.menu);
                            } else {
                                instance.menu.add({
                                    text: 'empty'
                                });
                            }
                        });
                    },
                    {
                        menu: menu,
                        parentId: parentId()
                    });
                    return menu;
                }
            });
            CommandStrip = Ext.extend(Ext.Toolbar, {
                renderTo: 'mdi_commands',
                height: 30,
                style: 'background-image: none; background-color: transparent;',
```

```
                load: function (commands) {
                    var instance = this;
                    instance.removeAll();
                    Ext.each(commands, function (command, index) {
                        instance.add({
                            xtype: 'tbbutton',
                            cls: 'x-btn-text-icon',
                            icon: 'theme/images/icons/' + command.name + '.png',
                            text: command.name,
                            handler: command.handler()
                        });
                    });
                    instance.doLayout();
                    return true;
                }
            });
            TricertAssist = {
                mdi: new Ext.ux.Mdi(),
                navigation: new Navigation(),
                toolbar: new CommandStrip(),
                showWindow: function (items, config) {
                    var win = this.mdi.createWindow({
                        id: this.generateUniqueID('win'),
                        toolbar: this.toolbar,
                        handlers: config.commands,
                        title: config.text || 'Management Section',
                        width: config.width,
                        height: config.height,
                        iconCls: config.icon || 'icon-window',
                        layout: config.layout || 'fit',
                        defaults: {
                            autoScroll: true
                        },
                        shim: false,
                        animCollapse: false,
                        constrainHeader: true,
                        items: items()
                    });
                    win.show();
                },
                loadModule: function (moduleID) {
                    Section.loadModule(moduleID, function (response, e) {
                        if (response.success) {
                            (new Function('(function(module, commands) { ' + response.data.module.moduleSource + '
})(' + Ext.util.JSON.encode(response.data.module) + ', ' + Ext.util.JSON.encode(response.data.commands) + ');'))();
                        }
                        TricertAssist.showNotification(response);
                    });
                },
                generateUniqueID: function (prefix) {
                    var i = 0;
                    do {
                        i++;
                        var id = prefix + '_' + i;
                    } while (Ext.query('#' + id).length);
                    return id;
                },
                showNotification: function (data) {
                    new Ext.ux.Notification({
                        iconCls: 'x-icon-information',
                        title: data.title,
                        html: data.message,
                        autoDestroy: true,
                        hideDelay: 5000
                    }).show(document);
                }
            };
            MyPanel = Ext.extend(Ext.Panel, {
                renderTo: 'panel_content',
                border: false,
                bodyStyle: 'background-color: transparent;',
```

```
        style: 'margin-right: 10px;',
        load: function () {
            var instance = this;
            User.getLoggedInUser(function (user, e) {
                TricertAssist.navigation.removeAll(true);
                instance.removeAll(true);
                if (user) {
                    instance.add(new Ext.Button({
                        text: 'Logout',
                        width: 100,
                        height: 25,
                        handler: function () {
                            User.doLogout(function (result, e) {
                                TricertAssist.mdi.closeWindows();
                                TricertAssist.toolbar.removeAll();
                                TricertAssist.showNotification(result);
                                instance.load();
                            });
                        }
                    }));
                    TricertAssist.navigation.load();
                } else {
                    var form = new Ext.form.FormPanel({
                        border: false,
                        bodyStyle: 'background-color: transparent;',
                        style: 'margin-right: 10px;',
                        labelWidth: 60,
                        items: [
                        new Ext.form.TextField({
                            name: 'userName',
                            fieldLabel: 'Username',
                            allowBlank: false,
                            blankText: 'Enter your username',
                            width: 130
                        }), new Ext.form.TextField({
                            name: 'password',
                            fieldLabel: 'Password',
                            inputType: 'password',
                            allowBlank: false,
                            blankText: 'Enter your password',
                            width: 130
                        })],
                        api: {
                            submit: User.doLogin()
                        }
                    });
                    instance.add(form);
                    instance.add(new Ext.Button({
                        text: 'Login',
                        width: 100,
                        height: 25,
                        handler: function (button, event) {
                            if (form.getForm().isValid()) {
                                var mask = new Ext.LoadMask(Ext.get('panel_content'), {
                                    msg: 'Loading...'
                                });
                                mask.show();
                                form.api.submit(form.getForm().el, function (result, e) {
                                    TricertAssist.showNotification({
                                        title: result.title,
                                        message: result.message()
                                    });
                                    mask.hide();
                                    if (result.success) instance.load();
                                });
                            }
                        }
                    }));
                }
                instance.doLayout();
            });
```

```
        }
    });
    var myPanel = new MyPanel();
    myPanel.load();
    new Ext.Toolbar({
        renderTo: 'mdi_toolbar',
        style: 'border: 1px solid #000; border-bottom: 0; padding-top: 5px;',
        width: 700,
        height: 35,
        items: [{
            xtype: 'tbbutton',
            cls: 'x-btn-text-icon',
            icon: 'theme/images/icons/arrow_ns.png',
            text: 'Toggle Banner',
            enableToggle: true,
            toggleHandler: function (item, pressed) {
                var banner = Ext.get('banner_container');
                if (pressed) {
                    if (!banner.isVisible()) {
                        banner.slideIn('t', {
                            easing: 'easeOut',
                            duration: .1
                        });
                    }
                } else {
                    if (banner.isVisible()) {
                        banner.slideOut('t', {
                            easing: 'easeOut',
                            duration: .1,
                            remove: false,
                            useDisplay: true
                        });
                    }
                }
            },
            pressed: true
        }]
    });
    FormActions = {
        initFields: function (form, module) {
            form.add({
                xtype: 'hidden',
                name: 'module',
                type: 'int',
                value: module.moduleID
            },
            {
                xtype: 'hidden',
                name: 'entity'
            });
        },
        triggerNew: function (form, recordType, defaultData) {
            form.getForm().reset();
            form.getForm().loadRecord(new recordType(defaultData, 0));
            form.getForm().findField('entity').setRawValue(0);
        },
        triggerSave: function (form, successFn) {
            if (form.getForm().isValid()) {
                form.getForm().submit({
                    waitMsg: 'Saving...',
                    success: function (form, action) {
                        TricertAssist.showNotification(action.result);
                        successFn();
                    },
                    failure: function (form, action) {
                        TricertAssist.showNotification(action.result);
                    }
                });
            }
        },
        triggerLoad: function (form, id) {
```

```
            form.getForm().reset();
            form.getForm().load({
                params: {
                    identifier: id()
                }
            });
            form.getForm().findField('entity').setRawValue(1);
        },
        triggerDelete: function (fn) {
            Ext.Msg.show({
                title: 'Warning',
                msg: 'Are you sure you wish to delete this record?',
                buttons: Ext.Msg.YESNO,
                fn: function (btn) {
                    if (btn == 'yes') {
                        fn();
                    }
                }
            });
        },
        defaultHandler: function (commands, form, recordType, defaultData, newFn, successFn, deleteFn) {
            Ext.each(commands, function (command, index) {
                command.handler = function () {
                    switch (command.id) {
                    case 0:
                        newFn();
                        FormActions.triggerNew(form, recordType, defaultData);
                        break;
                    case 1:
                        FormActions.triggerSave(form, successFn);
                        break;
                    case 2:
                        FormActions.triggerDelete(deleteFn);
                        break;
                    }
                }
            });
        }
    };
    Ext.ux.GridEditor = Ext.extend(Ext.grid.GridPanel, {
        initComponent: function () {
            var instance = this;
            var editor = new Ext.ux.grid.RowEditor({
                saveText: 'Update'
            });

            Ext.apply(this, {
                store: instance.store,
                autoHeight: true,
                autoWidth: true,
                plugins: [editor],
                bodyStyle: 'padding:0px 0px 10px',
                sm: new Ext.grid.RowSelectionModel({
                    singleSelect: true
                }),
                cm: instance.cm,
                tbar: [{
                    text: 'Add',
                    xtype: 'tbbutton',
                    cls: 'x-btn-text-icon',
                    icon: 'theme/images/icons/add.png',
                    handler: function (btn, ev) {
                        editor.stopEditing();
                        instance.store.insert(0, new instance.store.recordType(instance.defaultData()));
                        editor.startEditing(0);
                    }
                },
                '-', {
                    text: 'Delete',
                    xtype: 'tbbutton',
                    cls: 'x-btn-text-icon',
```

```
                            icon: 'theme/images/icons/delete.png',
                            handler: function () {
                                var record = instance.getSelectionModel().getSelected();
                                if (record) {
                                    instance.api.remove(record.get(instance.idProperty), function (result, e) {
                                        instance.store.remove(record);
                                        instance.store.reload();
                                    });
                                }
                            }
                        },
                        '-'],
                        viewConfig: {
                            forceFit: true
                        }
                    });
                    editor.on({
                        scope: this,
                        afteredit: function (roweditor, changes, record, rowIndex) {
                            instance.api.submit(record.data, function (result, e) {
                                TricertAssist.showNotification({
                                    title: result.title,
                                    message: result.message()
                                });
                                if (result.success) {
                                    instance.store.commitChanges();
                                }
                            });
                        },
                        canceledit: function (roweditor) {
                            var record = instance.store.getAt(roweditor.rowIndex);
                            if (record.get(instance.idProperty) === 0) {
                                instance.store.removeAt(roweditor.rowIndex);
                                instance.getView().refresh();
                            }
                            return true;
                        }
                    });
                    Ext.ux.GridEditor.superclass.initComponent.call(this);
                }
            });
        }
    });
});
```

JavaScript Modules

Module Management

```
Ext.onReady(function() {
    Ext.ScriptMgr.load({
        scripts: ['js/plugins/Ext.ux.form.SuperBoxSelect.js', 'js/plugins/Ext.ux.form.IconBrowser.js',
'js/plugins/codepress/Codepress.js', 'js/plugins/Ext.ux.CodePress.js'],
        callback: function() {
            var jsEditorID = TricertAssist.generateUniqueID('textarea');
            var commandComboID = TricertAssist.generateUniqueID('combobox');
            var moduleStore = new Ext.data.DirectStore({
                directFn: Module.getSimpleList,
                paramsAsHash: false,
                idProperty: 'id',
                fields: [{
                    name: 'id'
                },{
                    name: 'name'
                }]
            });
            var recordType = Ext.data.Record.create({
                name: 'moduleID',
                type: 'int'
            },
            {
                name: 'moduleTitle'
            },
            {
                name: 'moduleIcon'
            },
            {
                name: 'moduleSource'
            },
            {
                name: 'moduleName'
            },
            {
                name: 'moduleCommands'
            });
            var commandList = new Ext.data.DirectStore({
                directFn: AccessGroup.commandTypes,
                paramsAsHash: false,
                idProperty: 'id',
                fields: [{
                    name: 'id'
                },{
                    name: 'name'
                }]
            });
            commandList.load();
            var commandComboList = new Ext.data.DirectStore({
                directFn: AccessGroup.commandTypes,
                paramsAsHash: false,
                idProperty: 'id',
                fields: [{
                    name: 'id'
                },{
                    name: 'name'
                }]
            });
            commandComboList.load();



            var moduleList = new Ext.data.DirectStore({
```

```
        directFn: Module.getModuleList,
        paramsAsHash: false,
        idProperty: 'id',
        fields: [{
            name: 'id'
        },{
            name: 'moduleName',
            mapping: 'name'
        }]
    });
    moduleList.load();
    var form = new Ext.FormPanel({
        title: 'Module Management',
        border: true,
        region: 'center',
        defaultType: 'textfield',
        bodyStyle: 'padding:10px 10px 0',
        margins: '3 3 3 3',
        plugins: [new Ext.ux.FormClear()],
        items: [{
            xtype: 'hidden',
            name: 'moduleID'
        },{
            xtype: 'hidden',
            name: 'moduleCommands'
        },{
            xtype: 'fieldset',
            title: 'Module Settings',
            autoHeight: true,
            width: 400,
            items: [{
                layout: 'form',
                labelAlign: 'left',
                border: false,
                items: [new Ext.form.ComboBox({
                    fieldLabel: 'Module',
                    hiddenName: 'moduleName',
                    store: moduleList,
                    valueField: 'moduleName',
                    displayField: 'moduleName',
                    mode: 'remote',
                    editable: false,
                    triggerAction: 'all',
                    emptyText: 'Select a Module ...',
                    allowBlank: false,
                    blankText: 'Please Select a Module',
                    width: 230
                }), {
                    allowBlank: true,
                    id: commandComboID,
                    xtype: 'superboxselect',
                    fieldLabel: 'Commands',
                    resizable: true,
                    name: 'commandSelection',
                    hiddenName: 'commandSelection',
                    value: 'moduleCommands',
                    store: commandComboList,
                    mode: 'remote',
                    displayField: 'name',
                    displayFieldTpl: '{name}',
                    valueField: 'id',
                    triggerAction: 'all',
                    width: 230
                },{
                    xtype: 'textfield',
                    fieldLabel: 'Module Title',
                    name: 'moduleTitle',
                    width: 230
                },new Ext.ux.form.IconBrowserField({
                    fieldLabel: 'Select an icon',
                    name: 'moduleIcon',
```

```
                            width: 230,
                            buttonText: '',
                            buttonCfg: {
                                iconCls: 'upload-icon'
                            }
                        })]
                    }]
                },{
                    xtype: 'fieldset',
                    title: 'JavaScript Source',
                    labelAlign: 'top',
                    width: 400,
                    autoHeight: true,
                    items: [new Ext.ux.CodePress({
                        id: jsEditorID,
                        language: 'javascript',
                        name: 'moduleSource',
                        hiddenName: 'moduleSource',
                        height: 200,
                        width: 375
                    })]
                }],
                api: {
                    load: Module.create,
                    submit: Module.commit
                },
                paramOrder: ['identifier'],
                listeners: {
                    beforeaction: function(form, action) {
                        if (action.type == 'directsubmit') {
                            form.findField('moduleCommands').setRawValue('[' +
Ext.getCmp(commandComboID).getValue().trim() + ']');
                            form.findField('moduleSource').setRawValue(Ext.getCmp(jsEditorID).getMinifiedValue());
                        }
                    },
                    actioncomplete: function(form, action) {
                        if (action.type == 'directload') {
                            Ext.getCmp(commandComboID).setValue(form.findField('moduleCommands').getValue().trim());
                        }
                    }
                }
            });
            FormActions.initFields(form, module);
            var grid = new Ext.grid.GridPanel({
                title: 'System Modules',
                store: moduleStore,
                collapsible: true,
                border: true,
                region: 'west',
                width: 200,
                cm: new Ext.grid.ColumnModel([{
                    header: 'Module Name',
                    sortable: false,
                    dataIndex: 'name'
                }]),
                sm: new Ext.grid.RowSelectionModel({
                    singleSelect: true
                }),
                viewConfig: {
                    forceFit: true
                },
                listeners: {
                    beforerender: function(grid) {
                        moduleStore.load();
                    }
                }
            });
            grid.getSelectionModel().on('rowselect', function(sm, index, r) {
                FormActions.triggerLoad(form, grid.getStore().getAt(index).id);
            });
            grid.getStore().on('load', function() {
```

```
                grid.getSelectionModel().selectFirstRow();
            });
            var defaultData = {
                moduleID: 0,
                moduleName: '',
                moduleTitle: '',
                moduleIcon: '',
                moduleSource: 'alert(\'new module\');',
                commandList: '0,1,2'
            };
            var newFn = function() {};
            var successFn = function() {
                moduleStore.load();
            };
            var deleteFn = function() {
                var sm = grid.getSelectionModel();
                if (sm.hasSelection()) {
                    Module.remove(sm.getSelected().get('moduleID'), function(result, e) {
                        form.getForm().clear();
                        moduleStore.reload();
                    });
                }
            };
            FormActions.defaultHandler(commands, form, recordType, defaultData, newFn, successFn, deleteFn);
            TricertAssist.showWindow([grid, form], {
                text: module.moduleTitle,
                commands: commands,
                layout: 'border',
                width: 650,
                height: 450
            });
        }
    });
});
```

User Management

```
Ext.onReady(function() {
    Ext.ScriptMgr.load({
        scripts: ['js/plugins/Ext.ux.PasswordMeter.js', 'js/plugins/Ext.ux.grid.RowExpander.js'],
        callback: function() {
            Ext.apply(Ext.form.VTypes, {
                password: function(val, field) {
                    if (field.initialPassField) {
                        var pwd = Ext.getCmp(field.initialPassField);
                        return (val == pwd.getValue());
                    }
                    return true;
                },
                passwordText: 'Passwords do not match'
            });
            var passwordID = TricertAssist.generateUniqueID('textinput');
            var tabPanel = new Ext.TabPanel({
                id: TricertAssist.generateUniqueID('body'),
                region: 'center',
                margins: '3 3 3 0',
                activeTab: 0,
                defaults: {
                    autoScroll: true
                },
                items: [{
                    title: 'List View',
                    items: [],
                    listeners: {
                        beforeshow: function() {
                            tabPanel.get(1).setDisabled(true);
                        }
                    }
                },{
                    title: 'Record View',
                    items: []
                }]
            });
            var dataSets = {
                userStore: new Ext.data.GroupingStore({
                    proxy: new Ext.data.DirectProxy({
                        paramsAsHash: false,
                        directFn: User.pagination,
                        paramOrder: 'start|limit|sort|dir|where'
                    }),
                    reader: new Ext.data.JsonReader({
                        idProperty: 'userID',
                        totalProperty: 'totalRows',
                        root: 'data',
                        fields: [{
                            name: 'userID',
                            field: 'user_id',
                            type: 'int'
                        },{
                            name: 'userName',
                            field: 'user_name'
                        },{
                            name: 'groupID',
                            field: 'group_id',
                            type: 'int'
                        },{
                            name: 'employeeID',
                            field: 'employee_id',
                            type: 'int'
                        },{
                            name: 'userType',
                            field: 'user_type',
                            type: 'int'
```

```
                }]
            }),
            groupField: 'userType',
            remoteSort: true,
            baseParams: {
                start: 0,
                limit: 10,
                where: ''
            },
            sortInfo: {
                field: 'user_id',
                direction: 'ASC'
            }
        }),
        groupStore: new Ext.data.DirectStore({
            idProperty: 'groupID',
            directFn: AccessGroup.list,
            paramsAsHash: false,
            fields: [{
                name: 'groupID',
                type: 'int'
            },{
                name: 'groupName'
            }]
        }),
        employeeList: new Ext.data.DirectStore({
            idProperty: 'employeeID',
            directFn: Employee.list,
            paramsAsHash: false,
            fields: [{
                name: 'employeeID',
                type: 'int'
            },{
                name: 'firstName'
            },{
                name: 'lastName'
            }]
        }),
        userTypes: new Ext.data.DirectStore({
            directFn: User.userTypes,
            paramsAsHash: false,
            fields: [{
                name: 'id',
                type: 'int'
            },{
                name: 'name'
            }]
        }),
        load: function() {
            dataSets.userTypes.load();
            dataSets.groupStore.load();
            dataSets.employeeList.load();
        }
    };
    var sessionExpander = new Ext.ux.grid.RowExpander({
        tpl: new Ext.XTemplate('<div class="detailData">', '', '</div>'),
        listeners: {
            expand: function(expander, record, body, index) {
                if (Ext.DomQuery.select("div.x-panel-bwrap", body).length == 0) {
                    new Ext.list.ListView({
                        multiSelect: false,
                        store: new Ext.data.JsonStore({
                            data: record.json.history,
                            fields: [{
                                name: 'visitID',
                                type: 'int'
                            },{
                                name: 'sessionID',
                                type: 'int'
                            },{
                                name: 'moduleID',
```

```
                                type: 'int'
                            },{
                                name: 'moduleTitle'
                            },{
                                name: 'visitTime',
                                type: 'date'
                            }]
                        }),
                        columns: [{
                            dataIndex: 'moduleTitle',
                            header: 'Module',
                            width: .5
                        },{
                            dataIndex: 'visitTime',
                            header: 'Time Accessed',
                            tpl: '{visitTime:date("H:i:s")}',
                            width: .5
                        }],
                        renderTo: Ext.DomQuery.select("div.detailData", body)[0]
                    });
                }
            }
        }
    });

    var sessionList = new Ext.data.DirectStore({
        directFn: SessionLog.list,
        paramsAsHash: false,
        paramOrder: 'start|limit|sort|dir|userID',
        root: 'data',
        idProperty: 'sessionID',
        totalProperty: 'totalRows',
        sortInfo: {
            field: 'session_id',
            direction: 'ASC'
        },
        fields: [{
            name: 'sessionID',
            field: 'session_id'
        },{
            name: 'sessionDate',
            field: 'session_date'
        }],
        remoteSort: true,
        paramNames: {
            start: 'start',
            limit: 'limit',
            sort: 'sort',
            dir: 'dir',
            userID: 'userID'
        },
        baseParams: {
            start: 0,
            limit: 10,
            userID: 0
        }
    });

    var sessionPanel = new Ext.Panel({
        xtype: 'container',
        autoEl: {},
        region: 'center',
        layout: 'fit',
        title: 'Session Log',
        collapsible: true,
        border: true,
        items: [new Ext.grid.GridPanel({
            store: sessionList,
            border: true,
            autoHeight: true,
            disableSelection: true,
```

```
            cm: new Ext.grid.ColumnModel([sessionExpander, {
                header: 'Session Date',
                width: 100,
                sortable: true,
                dataIndex: 'sessionDate'
            }]),
            plugins: sessionExpander,
            viewConfig: {
                forceFit: true
            }
        })],
        bbar: new Ext.PagingToolbar({
            pageSize: 10,
            store: sessionList,
            displayInfo: true
        })
    });


    var renderEmployee = function(value) {
        return dataSets.employeeList.getById(parseInt(value)).get('firstName');
    };


    var renderUserType = function(value) {
        return dataSets.userTypes.getById(parseInt(value)).get('name');
    };



    var userManagement = {
        form: new Ext.FormPanel({
            border: true,
            bodyStyle: 'padding:10px 10px 0',
            title: 'User Management Form',
            collapsible: true,
            autoHeight: true,
            plugins: [new Ext.ux.FormClear()],
            items: [{
                xtype: 'hidden',
                name: 'userID'
            },{
                xtype: 'fieldset',
                title: 'Basic Settings',
                autoHeight: true,
                anchor: '100% -20',
                layout: 'column',
                items: [{
                    columnWidth: '.5',
                    layout: 'form',
                    labelAlign: 'top',
                    border: false,
                    items: [{
                        xtype: 'textfield',
                        fieldLabel: 'User Name',
                        name: 'userName',
                        width: 150
                    },new Ext.ux.PasswordMeter({
                        id: passwordID,
                        fieldLabel: 'Password',
                        name: 'password',
                        inputType: 'password',
                        width: 150
                    }), new Ext.ux.PasswordMeter({
                        fieldLabel: 'Confirm Password',
                        name: 'confirmationPassword',
                        vtype: 'password',
                        initialPassField: passwordID,
```

```
                    inputType: 'password',
                    width: 150
                })]
            },{
                columnWidth: '.5',
                layout: 'form',
                labelAlign: 'top',
                anchor: '100% -20',
                border: false,
                items: [new Ext.form.ComboBox({
                    fieldLabel: 'Access Group',
                    hiddenName: 'groupID',
                    store: dataSets.groupStore,
                    valueField: 'groupID',
                    displayField: 'groupName',
                    mode: 'local',
                    editable: false,
                    triggerAction: 'all',
                    emptyText: 'Select a group...',
                    allowBlank: false,
                    blankText: 'Select a Access Group',
                    width: 150
                }), new Ext.form.ComboBox({
                    fieldLabel: 'Employee',
                    hiddenName: 'employeeID',
                    store: dataSets.employeeList,
                    valueField: 'employeeID',
                    displayField: 'firstName',
                    mode: 'remote',
                    editable: false,
                    triggerAction: 'all',
                    emptyText: 'Select a employee...',
                    allowBlank: false,
                    blankText: 'Select a Employee',
                    width: 150
                }), new Ext.form.ComboBox({
                    fieldLabel: 'User Type',
                    hiddenName: 'userType',
                    store: dataSets.userTypes,
                    valueField: 'id',
                    displayField: 'name',
                    mode: 'remote',
                    editable: false,
                    triggerAction: 'all',
                    emptyText: 'Select a type ...',
                    allowBlank: false,
                    blankText: 'Select a User Type',
                    width: 150
                })]
            }]
        }],
        api: {
            load: User.create,
            submit: User.commit
        },
        paramOrder: ['identifier'],
        listeners: {
            beforeaction: function(form, action) {
                if (action.type == 'directsubmit') {}
            },
            actioncomplete: function(form, action) {
                if (action.type == 'directload') {
                    sessionList.setBaseParam('userID', parseInt(form.findField('userID').getValue()));
                    sessionList.load();
                }
            }
        }
    }),
```

```
        grid: new Ext.grid.GridPanel({
            store: dataSets.userStore,
            border: false,
            autoHeight: true,
            cm: new Ext.grid.ColumnModel([new Ext.grid.RowNumberer(), {
                header: 'User Name',
                width: 100,
                sortable: true,
                dataIndex: 'userName'
            },{
                header: 'Employee File',
                width: 100,
                sortable: true,
                renderer: renderEmployee,
                dataIndex: 'employeeID'
            },{
                header: 'User Type',
                width: 100,
                sortable: true,
                renderer: renderUserType,
                dataIndex: 'userType',
                hidden: true
            }]),
            listeners: {
                rowdblclick: function(grid, index, e) {
                    tabPanel.get(1).setDisabled(false).show();
                    FormActions.triggerLoad(userManagement.form, grid.getStore().getAt(index).id);
                }
            },
            view: new Ext.grid.GroupingView({
                forceFit: true,
                groupTextTpl: '{text}'
            }),
            tbar: new Ext.PagingToolbar({
                pageSize: 10,
                store: dataSets.userStore,
                displayInfo: true
            })
        }),
        defaultData: {
            userID: 0,
            userName: '',
            groupID: '',
            employeeID: '',
            userType: ''
        }
    };

    var groupGrid = new Ext.grid.GridPanel({
        title: 'Access Groups',
        store: dataSets.groupStore,
        border: true,
        region: 'west',
        width: 200,
        cm: new Ext.grid.ColumnModel([{
            header: 'Group Name',
            sortable: false,
            dataIndex: 'groupName'
        }]),
        sm: new Ext.grid.RowSelectionModel({
            singleSelect: true
        }),
        viewConfig: {
            forceFit: true
        },
        listeners: {
            beforerender: function(grid) {
                dataSets.load();
            }
        }
    });
```

```
            groupGrid.getSelectionModel().on('rowselect', function(sm, index, r) {
                tabPanel.get(0).show();
               dataSets.userStore.setBaseParam('where', ' WHERE group_id = ' +
               groupGrid.getStore().getAt(index).id);
                dataSets.userStore.load();
            });
            groupGrid.getStore().on('load', function() {
                groupGrid.getSelectionModel().selectFirstRow();
            });
            tabPanel.get(0).add(userManagement.grid);
            tabPanel.get(1).add([userManagement.form, sessionPanel]);
            tabPanel.doLayout();
            var newFn = function() {
                tabPanel.get(1).setDisabled(false).show();
            };
            var successFn = function() {
                dataSets.userStore.load();
            };
            var deleteFn = function() {
                var sm = userManagement.grid.getSelectionModel();
                if (sm.hasSelection()) {
                    User.remove(sm.getSelected().get('userID'), function(result, e) {
                        dataSets.userStore.load();
                    });
                }
            };
            FormActions.initFields(userManagement.form, module);
            FormActions.defaultHandler(commands, userManagement.form,
               dataSets.userStore.recordType, userManagement.defaultData, newFn, successFn, deleteFn);
            TricertAssist.showWindow([groupGrid, tabPanel], {
                text: module.moduleTitle,
                commands: commands,
                layout: 'border',
                width: 625,
                height: 390
            });
        }
    });
});
```

Manage Categories

```
Ext.onReady(function() {
    Ext.ScriptMgr.load({
        scripts: ['js/plugins/Ext.ux.grid.RowEditor.js', 'js/plugins/Ext.ux.form.IconBrowser.js'],
        callback: function() {
            var categoryList = new Ext.data.DirectStore({
                directFn: Category.list,
                paramsAsHash: false,
                idProperty: 'categoryID',
                fields: [{
                    name: 'categoryID'
                },{
                    name: 'categoryTitle'
                }]
            });

            categoryList.on('load', function() {
                categoryList.add(new categoryList.recordType({
                    categoryID: 0,
                    categoryTitle: 'Main Category',
                },
                0));
            });
            categoryList.load();

            var categoryTypes = new Ext.data.DirectStore({
                directFn: Category.categoryTypes,
                paramsAsHash: false,
                fields: [{
                    name: 'id'
                },{
                    name: 'name'
                }]
            });
            categoryTypes.load();



            var form = new Ext.FormPanel({
                border: true,
                region: 'center',
                title: 'Category Management',
                bodyStyle: 'padding:10px 10px 10px',
                plugins: [new Ext.ux.FormClear()],
                fileUpload: true,
                items: [{
                    xtype: 'hidden',
                    name: 'categoryID'
                },{
                    xtype: 'fieldset',
                    title: 'Category Settings',
                    autoHeight: true,
                    width: 350,
                    items: [{
                        xtype: 'textfield',
                        fieldLabel: 'Category Title',
                        name: 'categoryTitle',
                        width: 200
                    },new Ext.ux.form.IconBrowserField({
                        fieldLabel: 'Select an icon',
                        name: 'categoryIcon',
                        width: 200,
                        buttonText: '',
                        buttonCfg: {
                            iconCls: 'upload-icon'
                        }
```

```
            }), new Ext.form.ComboBox({
                fieldLabel: 'Category Parent',
                hiddenName: 'categoryParent',
                store: categoryList,
                valueField: 'categoryID',
                displayField: 'categoryTitle',
                mode: 'remote',
                editable: false,
                triggerAction: 'all',
                allowBlank: false,
                width: 200
            }), new Ext.form.ComboBox({
                fieldLabel: 'Category Type',
                hiddenName: 'categoryType',
                store: categoryTypes,
                valueField: 'id',
                displayField: 'name',
                mode: 'remote',
                editable: false,
                triggerAction: 'all',
                emptyText: 'Select a type ...',
                allowBlank: false,
                blankText: 'Select a Category Type',
                width: 200
            })]
        }],
        api: {
            load: Category.create,
            submit: Category.commit
        },
        paramOrder: ['identifier'],
        listeners: {
            beforeaction: function(form, action) {
                if (action.type == 'directsubmit') {}
            },
            actioncomplete: function(form, action) {
                if (action.type == 'directload') {}
            }
        }
    });



    FormActions.initFields(form, module);
    var recordType = Ext.data.Record.create({
        name: 'categoryID',
        type: 'int'
    },
    {
        name: 'categoryParent',
        type: 'int'
    },
    {
        name: 'categoryTitle'
    },
    {
        name: 'categoryIcon'
    },
    {
        name: 'categoryType'
    });
    var defaultData = {
        categoryID: 0,
        categoryTitle: '',
        categoryParent: 0,
        categoryIcon: '',
        categoryType: ''
    };
    var rootNode = new Ext.tree.AsyncTreeNode({
        id: '0'
    });
```

```
            var tree = new Ext.tree.TreePanel({
                title: 'Category List',
                border: true,
                region: 'west',
                width: 200,
                split: true,
                collapsible: true,
                loader: new Ext.tree.TreeLoader({
                    directFn: Category.getTree
                }),
                root: rootNode,
                rootVisible: false,
                listeners: {
                    click: function(node, e) {
                        FormActions.triggerLoad(form, parseInt(node.id));
                    }
                }
            });
            rootNode.expand();
            var newFn = function() {};
            var reloadCategories = function() {
                tree.getLoader().load(rootNode);
                TricertAssist.navigation.load();
            };
            var successFn = function() {
                reloadCategories();
            };
            var deleteFn = function() {
                Category.remove(parseInt(tree.getSelectionModel().getSelectedNode().id), function(result, e) {
                    form.getForm().clear();
                    reloadCategories();
                });
            };
            FormActions.defaultHandler(commands, form, recordType, defaultData, newFn, successFn, deleteFn);
            tree.on('load', function() {
                var node = rootNode.item(0);
                tree.selectPath(node.getPath());
                node.fireEvent('click', node);
            });
            TricertAssist.showWindow([tree, form], {
                text: module.moduleTitle,
                commands: commands,
                layout: 'border',
                width: 600,
                height: 400
            });
        }
    });
});
```

Group Management

```
Ext.onReady(function() {
    Ext.ScriptMgr.load({
        scripts: ['js/plugins/Ext.ux.form.SuperBoxSelect.js'],
        callback: function() {
            var commandComboID = TricertAssist.generateUniqueID('combobox');
            var groupStore = new Ext.data.DirectStore({
                idProperty: 'groupID',
                directFn: AccessGroup.list,
                paramsAsHash: false,
                fields: [{
                    name: 'groupID',
                    type: 'int'
                },{
                    name: 'groupName'
                }]
            });
            var recordType = Ext.data.Record.create({
                name: 'groupID',
                type: 'int'
            },
            {
                name: 'groupName'
            },
            {
                name: 'groupCommands'
            },
            {
                name: 'groupDescription'
            });
            var commandList = new Ext.data.DirectStore({
                directFn: AccessGroup.commandTypes,
                paramsAsHash: false,
                idProperty: 'id',
                fields: [{
                    name: 'id'
                },{
                    name: 'name'
                }]
            });
            commandList.load();
            var commandComboList = new Ext.data.DirectStore({
                directFn: AccessGroup.commandTypes,
                paramsAsHash: false,
                idProperty: 'id',
                fields: [{
                    name: 'id'
                },{
                    name: 'name'
                }]
            });
            commandComboList.load();
            var sectionList = new Ext.data.DirectStore({
                paramsAsHash: false,
                directFn: Section.pagination,
                paramOrder: 'start|limit|sort|dir|where',
                root: 'data',
                idProperty: 'sectionID',
                totalProperty: 'totalRows',
                sortInfo: {
                    field: 'section_id',
                    direction: 'ASC'
                },
                fields: [{
                    name: 'sectionID',
                    field: 'section_id',
                    type: 'int'
                },{
```

```
            name: 'categoryID',
            field: 'category_id',
            type: 'int'
        },{
            name: 'moduleID',
            field: 'module_id',
            type: 'int'
        },{
            name: 'groupID',
            field: 'group_id',
            type: 'int'
        }],
        remoteSort: true,
        paramNames: {
            start: 'start',
            limit: 'limit',
            sort: 'sort',
            dir: 'dir',
            where: 'where'
        },
        baseParams: {
            start: 0,
            limit: 10,
            where: ''
        }
    });
    var form = new Ext.FormPanel({
        border: true,
        bodyStyle: 'padding:10px 10px 0',
        title: 'Group Management Form',
        collapsible: true,
        autoHeight: true,
        autoWidth: true,
        region: 'center',
        layout: 'fit',
        plugins: [new Ext.ux.FormClear()],
        items: [{
            xtype: 'hidden',
            name: 'groupID'
        },{
            xtype: 'hidden',
            name: 'groupCommands'
        },{
            xtype: 'fieldset',
            title: 'Group Settings',
            autoHeight: true,
            autoWidth: true,
            items: [{
                xtype: 'textfield',
                fieldLabel: 'Group Name',
                name: 'groupName',
                width: 230
            },{
                allowBlank: true,
                id: commandComboID,
                xtype: 'superboxselect',
                fieldLabel: 'Commands',
                resizable: true,
                name: 'commandSelection',
                hiddenName: 'commandSelection',
                value: 'groupCommands',
                store: commandComboList,
                mode: 'remote',
                displayField: 'name',
                displayFieldTpl: '{name}',
                valueField: 'id',
                triggerAction: 'all',
                width: 230
```

```
            },new Ext.form.TextArea({
                fieldLabel: 'Description',
                width: 230,
                name: 'groupDescription'
            })]
        }],
        api: {
            load: AccessGroup.create,
            submit: AccessGroup.commit
        },
        paramOrder: ['identifier'],
        listeners: {
            beforeaction: function(form, action) {
                if (action.type == 'directsubmit') {
                form.findField('groupCommands').setRawValue('[' +
               Ext.getCmp(commandComboID).getValue().trim() + ']');
                }
            },
            actioncomplete: function(form, action) {
                if (action.type == 'directload') {
                    Ext.getCmp(commandComboID).setValue(form.findField('groupCommands').getValue().trim());
                      sectionList.setBaseParam('where', ' WHERE group_id = ' +
                      form.findField('groupID').getValue());
                    sectionList.load();
                }
            }
        }
    });

    var moduleList = new Ext.data.DirectStore({
        directFn: Module.getSimpleList,
        paramsAsHash: false,
        idProperty: 'id',
        fields: [{
            name: 'id',
            type: 'int'
        },{
            name: 'name'
        }]
    });

    moduleList.load();
    var groupList = new Ext.data.DirectStore({
        idProperty: 'groupID',
        directFn: AccessGroup.list,
        paramsAsHash: false,
        fields: [{
            name: 'groupID',
            type: 'int'
        },{
            name: 'groupName'
        }]
    });

    var categoryList = new Ext.data.DirectStore({
        directFn: Category.list,
        paramsAsHash: false,
        idProperty: 'categoryID',
        fields: [{
            name: 'categoryID'
        },{
            name: 'categoryTitle'
        }]
    });
    categoryList.load();
```

```
var moduleComboBox = new Ext.form.ComboBox({
    fieldLabel: 'Module',
    hiddenName: 'moduleID',
    store: moduleList,
    valueField: 'id',
    displayField: 'name',
    mode: 'remote',
    editable: false,
    triggerAction: 'all',
    emptyText: 'Select a module ...',
    allowBlank: false,
    blankText: 'Select a Module',
    width: 200
});


var categoryComboBox = new Ext.form.ComboBox({
    fieldLabel: 'Category',
    hiddenName: 'categoryID',
    store: categoryList,
    valueField: 'categoryID',
    displayField: 'categoryTitle',
    mode: 'remote',
    editable: false,
    triggerAction: 'all',
    emptyText: 'Select a category ...',
    allowBlank: false,
    blankText: 'Select a Category',
    width: 200
});
var groupComboBox = new Ext.form.ComboBox({
    fieldLabel: 'Group',
    hiddenName: 'groupID',
    store: groupList,
    valueField: 'groupID',
    displayField: 'groupName',
    mode: 'local',
    editable: false,
    triggerAction: 'all',
    emptyText: 'Select a category ...',
    allowBlank: false,
    blankText: 'Select a Category',
    width: 200
});
var sectionGrid = new Ext.ux.GridEditor({
    store: sectionList,
    idProperty: 'sectionID',
    defaultData: function() {
        return {
            sectionID: 0,
            categoryID: '',
            groupID: form.getForm().findField('groupID').getValue(),
            moduleID: ''
        };
    },
    api: {
        submit: Section.saveData,
        remove: Section.remove
    },
    cm: new Ext.grid.ColumnModel([{
        header: 'Module',
        width: 100,
        sortable: true,
        dataIndex: 'moduleID',
        renderer: Ext.ux.renderer.Combo(moduleComboBox),
        editor: moduleComboBox
    },{
        header: 'Category',
        width: 100,
        sortable: true,
        dataIndex: 'categoryID',
```

```
            renderer: Ext.ux.renderer.Combo(categoryComboBox),
            editor: categoryComboBox
        },{
            header: 'Group',
            width: 100,
            sortable: true,
            dataIndex: 'groupID',
            renderer: Ext.ux.renderer.Combo(groupComboBox),
            editor: groupComboBox
        }])
    });


    var sections = new Ext.Panel({
        layout: 'fit',
        title: 'Assigned Sections',
        collapsible: true,
        border: true,
        items: [sectionGrid],
        bbar: new Ext.PagingToolbar({
            pageSize: 10,
            store: sectionList,
            displayInfo: true
        })
    });


    FormActions.initFields(form, module);
    var grid = new Ext.grid.GridPanel({
        title: 'Access Groups',
        store: groupStore,
        collapsible: true,
        border: true,
        region: 'west',
        width: 200,
        cm: new Ext.grid.ColumnModel([{
            header: 'Access Group',
            sortable: false,
            dataIndex: 'groupName'
        }]),
        sm: new Ext.grid.RowSelectionModel({
            singleSelect: true
        }),
        viewConfig: {
            forceFit: true
        },
        listeners: {
            beforerender: function(grid) {
                groupStore.load();
            }
        }
    });
    grid.getSelectionModel().on('rowselect', function(sm, index, r) {
        FormActions.triggerLoad(form, grid.getStore().getAt(index).id);
    });
    groupStore.on('load', function() {
        grid.getSelectionModel().selectFirstRow();
        groupList.add(groupStore.getRange(0, groupStore.getCount()));
    });
    var defaultData = {
        groupID: 0,
        groupName: '',
        groupDescription: '',
        commandList: '0,1,2'
    };


    var newFn = function() {};
    var successFn = function() {
        groupStore.load();
    };
```

```
            var deleteFn = function() {
                var sm = grid.getSelectionModel();
                if (sm.hasSelection()) {
                    AccessGroup.remove(sm.getSelected().get('groupID'), function(result, e) {
                        form.getForm().clear();
                        groupStore.reload();
                    });
                }
            };


            FormActions.defaultHandler(commands, form, recordType, defaultData, newFn, successFn, deleteFn);
            TricertAssist.showWindow([grid, new Ext.Panel({
                margins: '3 3 3 3',
                region: 'center',
                layout: 'fit',
                items: [form, sections]
            })], {
                text: module.moduleTitle,
                commands: commands,
                layout: 'border',
                width: 650,
                height: 450
            });
        }
    });
});
```

Employee Departments

```
Ext.onReady(function () {
    Ext.ScriptMgr.load({
        scripts: ['js/plugins/Ext.ux.GMapPanel.js'],
        callback: function () {
            var departmentStore = new Ext.data.DirectStore({
                idProperty: 'departmentID',
                directFn: Department.list,
                paramsAsHash: false,
                fields: [{
                    name: 'departmentID',
                    type: 'int'
                },
                {
                    name: 'departmentName'
                }]
            });
            var recordType = Ext.data.Record.create({
                name: 'departmentID',
                type: 'int'
            },
            {
                name: 'departmentName'
            },
            {
                name: 'streetAddress'
            },
            {
                name: 'suburb'
            },
            {
                name: 'postcode'
            },
```

```
        {
            name: 'state'
        },
        {
            name: 'receptionPhone'
        });



        var stateList = new Ext.data.DirectStore({
            directFn: Employee.stateTypes,
            paramsAsHash: false,
            idProperty: 'id',
            fields: [{
                name: 'id'
            },
            {
                name: 'name'
            }]
        });
        stateList.load();
        var map = new Ext.ux.GMapPanel({
            zoomLevel: 14,
            title: 'Google Map',
            gmapType: 'map',
            border: true,
            autoWidth: true,
            height: 400,
            collapsible: true,
            mapConfOpts: ['enableScrollWheelZoom', 'enableDoubleClickZoom', 'enableDragging'],
            mapControls: ['GSmallMapControl', 'GMapTypeControl', 'NonExistantControl']
        });



        var form = new Ext.FormPanel({
            border: true,
            bodyStyle: 'padding:10px 10px 0',
            title: 'Employee Departments',
            collapsible: true,
            autoHeight: true,
            autoWidth: true,
            region: 'center',
            layout: 'fit',
            plugins: [new Ext.ux.FormClear()],
            items: [{
                xtype: 'hidden',
                name: 'departmentID'
            },
            {
                xtype: 'fieldset',
                title: 'Department Information',
                autoHeight: true,
                anchor: '100% -20',
                layout: 'column',
                items: [{
                    columnWidth: '.5',
                    layout: 'form',
                    labelAlign: 'top',
                    border: false,
                    items: [{
                        xtype: 'textfield',
                        fieldLabel: 'Department Name',
                        name: 'departmentName',
                        width: 150
                    },
                    {
                        xtype: 'textfield',
                        fieldLabel: 'Street Address',
                        name: 'streetAddress',
                        width: 150
```

```
                },
                new Ext.form.ComboBox({
                    fieldLabel: 'State',
                    hiddenName: 'state',
                    store: stateList,
                    valueField: 'name',
                    displayField: 'name',
                    mode: 'remote',
                    editable: false,
                    triggerAction: 'all',
                    emptyText: 'Select a state ...',
                    allowBlank: false,
                    blankText: 'Select a state',
                    width: 150
                })]
            },
            {
                columnWidth: '.5',
                layout: 'form',
                labelAlign: 'top',
                anchor: '100% -20',
                border: false,
                items: [{
                    xtype: 'textfield',
                    fieldLabel: 'Reception Phone',
                    name: 'receptionPhone',
                    width: 150
                },
                {
                    xtype: 'textfield',
                    fieldLabel: 'Suburb',
                    name: 'suburb',
                    width: 150
                },
                {
                    xtype: 'textfield',
                    fieldLabel: 'Postcode',
                    name: 'postcode',
                    width: 150
                }]
            }]
        }],
        api: {
            load: Department.create,
            submit: Department.commit
        },
        paramOrder: ['identifier'],
        listeners: {
            beforeaction: function (form, action) {
                if (action.type == 'directsubmit') {}
            },
            actioncomplete: function (form, action) {
                if (action.type == 'directload') {
                    map.setCenter({
                       geoCodeAddr: (form.findField('streetAddress').getValue() + ' ' +
                      form.findField('suburb').getValue() + ', ' + form.findField('state').getValue() + ' '
                      + form.findField('postcode').getValue()),
                        marker: {
                            title: form.findField('departmentName').getValue()
                        }
                    });
                }
            }
        }
    });
    var bodyPanel = new Ext.Panel({
        margins: '3 3 3 3',
        region: 'center',
        layout: 'fit',
        items: [form, map]
    });
```

```
            FormActions.initFields(form, module);
            var grid = new Ext.grid.GridPanel({
                title: 'Department',
                store: departmentStore,
                collapsible: true,
                border: true,
                region: 'west',
                width: 200,
                cm: new Ext.grid.ColumnModel([{
                    header: 'Department',
                    sortable: false,
                    dataIndex: 'departmentName'
                }]),
                sm: new Ext.grid.RowSelectionModel({
                    singleSelect: true
                }),
                viewConfig: {
                    forceFit: true
                },
                listeners: {
                    beforerender: function (grid) {
                        departmentStore.load();
                    }
                }
            });
            grid.getSelectionModel().on('rowselect', function (sm, index, r) {
                FormActions.triggerLoad(form, grid.getStore().getAt(index).id);
            });
            departmentStore.on('load', function () {
                grid.getSelectionModel().selectFirstRow();
            });
            var defaultData = {
                departmentID: 0,
                departmentName: '',
                streetAddress: '',
                suburb: '',
                postcode: '',
                state: '',
                receptionPhone: ''
            };
            var newFn = function () {};
            var successFn = function () {
                departmentStore.load();
            };
            var deleteFn = function () {
                var sm = grid.getSelectionModel();
                if (sm.hasSelection()) {
                    Department.remove(sm.getSelected().get('departmentID'), function (result, e) {
                        form.getForm().clear();
                        departmentStore.reload();
                    });
                }
            };
            FormActions.defaultHandler(commands, form, recordType, defaultData, newFn, successFn, deleteFn);
            TricertAssist.showWindow([grid, bodyPanel], {
                text: module.moduleTitle,
                commands: commands,
                layout: 'border',
                width: 650,
                height: 450
            });
        }
    });
});
```

Employee Sections

```
Ext.onReady(function() {
    Ext.ScriptMgr.load({
        scripts: ['js/plugins/Ext.ux.grid.RowEditor.js'],
        callback: function() {
            var store = new Ext.data.DirectStore({
                paramsAsHash: false,
                directFn: EmployeeSection.pagination,
                paramOrder: 'start|limit|sort|dir|where',
                root: 'data',
                idProperty: 'employeeSectionID',
                totalProperty: 'totalRows',
                sortInfo: {
                    field: 'employee_section_id',
                    direction: 'ASC'
                },
                fields: [{
                    name: 'employeeSectionID',
                    field: 'employee_section_id',
                    type: 'int'
                },{
                    name: 'sectionTitle',
                    field: 'section_title'
                },{
                    name: 'sectionDescription',
                    field: 'section_description'
                }],
                remoteSort: true,
                paramNames: {
                    start: 'start',
                    limit: 'limit',
                    sort: 'sort',
                    dir: 'dir',
                    where: 'where'
                },



                baseParams: {
                    start: 0,
                    limit: 10,
                    where: ''
                }
            });
            var editor = new Ext.ux.grid.RowEditor({
                saveText: 'Update'
            });
            editor.on({
                scope: this,
                afteredit: function(roweditor, changes, record, rowIndex) {
                    EmployeeSection.saveData(record.data, function(result, e) {
                        TricertAssist.showNotification({
                            title: result.title,
                            message: result.message
                        });
                        if (result.success) {
                            store.commitChanges();
                        }
                    });
                },
                canceledit: function(roweditor) {
                    var record = store.getAt(roweditor.rowIndex);
                    if (record.get('employeeSectionID') === 0) {
                        store.removeAt(roweditor.rowIndex);
                        grid.getView().refresh();
                    }
                    return true;
                }
```

```
});
var grid = new Ext.grid.GridPanel({
    store: store,
    autoHeight: true,
    autoWidth: true,
    disableSelection: true,
    plugins: [editor],
    bodyStyle: 'padding:0px 0px 10px',
    sm: new Ext.grid.RowSelectionModel({
        singleSelect: true
    }),
    cm: new Ext.grid.ColumnModel([{
        header: 'Title',
        width: 100,
        sortable: true,
        dataIndex: 'sectionTitle',
        editor: {
            xtype: 'textfield',
            allowBlank: false
        }
    },{
        header: 'Description',
        width: 100,
        sortable: true,
        dataIndex: 'sectionDescription',
        editor: {
            xtype: 'textfield',
            allowBlank: true
        }
    }]),
    listeners: {
        beforerender: function(grid) {
            store.load();
        }
    },
    viewConfig: {
        forceFit: true
    }
});


var panel = new Ext.Panel({
    xtype: 'container',
    autoEl: {},
    layout: 'fit',
    title: 'Employee Sections',
    region: 'center',
    collapsible: true,
    border: true,
    items: [grid],
    bbar: new Ext.PagingToolbar({
        pageSize: 10,
        store: store,
        displayInfo: true
    })
});
Ext.each(commands, function(command, index) {
    command.handler = function() {
        switch (command.id) {
        case 0:
            var defaultData = new store.recordType({
                employeeSectionID: 0,
                sectionTitle: '',
                sectionDescription: ''
            });
            editor.stopEditing();
            store.insert(0, defaultData);
            editor.startEditing(0);
            break;
        case 2:
            var record = grid.getSelectionModel().getSelected();
```

```
                        if (record) {
                            EmployeeSection.remove(record.get('employeeSectionID'), function(result, e) {
                                store.remove(record);
                                store.reload();
                            });
                        }
                        break;
                    }
                }
            });
            TricertAssist.showWindow([panel], {
                text: module.moduleTitle,
                commands: commands,
                layout: 'fit',
                width: 350,
                height: 300
            });
        }
    });
});
```

Employee Management

```
Ext.onReady(function () {
    var dataSets = {
        departmentStore: new Ext.data.DirectStore({
            idProperty: 'departmentID',
            directFn: Department.list,
            paramsAsHash: false,
            fields: [{
                name: 'departmentID',
                type: 'int'
            },
            {
                name: 'departmentName'
            }]
        }),
        employeeStore: new Ext.data.GroupingStore({
            proxy: new Ext.data.DirectProxy({
                paramsAsHash: false,
                directFn: Employee.pagination,
                paramOrder: 'start|limit|sort|dir|where'
            }),
            reader: new Ext.data.JsonReader({
                idProperty: 'employeeID',
                totalProperty: 'totalRows',
                root: 'data',
                fields: [{
                    name: 'employeeID',
                    field: 'employee_id',
                    type: 'int'
                },
                {
                    name: 'employeeSectionID',
                    field: 'employee_section_id',
                    type: 'int'
                },
                {
                    name: 'departmentID',
                    field: 'department_id',
                    type: 'int'
                },
                {
                    name: 'supervisorID',
                    field: 'supervisor_id',
                    type: 'int'
                },
                {
```

```
                name: 'employeeType',
                field: 'employee_type',
                type: 'int'
            },
            {
                name: 'firstName',
                field: 'first_name'
            },
            {
                name: 'lastName',
                field: 'last_name'
            },
            {
                name: 'emailAddress',
                field: 'email_address'
            },
            {
                name: 'streetAddress',
                field: 'street_address'
            },
            {
                name: 'suburb',
                field: 'suburb'
            },
            {
                name: 'postcode',
                field: 'postcode'
            },
            {
                name: 'state',
                field: 'state'
            }]
        }),
        groupField: 'employeeSectionID',
        remoteSort: true,
        baseParams: {
            start: 0,
            limit: 10,
            where: ''
        },
        sortInfo: {
            field: 'employee_id',
            direction: 'ASC'
        }
    }),
    employeeList: new Ext.data.DirectStore({
        idProperty: 'employeeID',
        directFn: Employee.list,
        paramsAsHash: false,
        fields: [{
            name: 'employeeID',
            type: 'int'
        },
        {
            name: 'firstName'
        },
        {
            name: 'lastName'
        }]
    }),
    employeeSectionList: new Ext.data.DirectStore({
        idProperty: 'employeeSectionID',
        directFn: EmployeeSection.list,
        paramsAsHash: false,
        fields: [{
            name: 'employeeSectionID',
            type: 'int'
        },
        {
            name: 'sectionTitle'
        }]
```

```
        }),
        employeeTypes: new Ext.data.DirectStore({
            directFn: Employee.employeeTypes,
            paramsAsHash: false,
            idProperty: 'id',
            fields: [{
                name: 'id',
                type: 'int'
            },
            {
                name: 'name'
            }]
        }),
        stateList: new Ext.data.DirectStore({
            directFn: Employee.stateTypes,
            paramsAsHash: false,
            idProperty: 'id',
            fields: [{
                name: 'id'
            },
            {
                name: 'name'
            }]
        })
    };
    var licenseTypes = new Ext.data.DirectStore({
        directFn: EmployeeLicense.licenseTypes,
        paramsAsHash: false,
        idProperty: 'id',
        fields: [{
            name: 'id'
        },
        {
            name: 'name'
        }]
    });
    licenseTypes.load();
    var qualificationLevels = new Ext.data.DirectStore({
        directFn: EmployeeQualification.qualificationLevels,
        paramsAsHash: false,
        idProperty: 'id',
        fields: [{
            name: 'id'
        },
        {
            name: 'name'
        }]
    });
    qualificationLevels.load();
    var licenseStore = new Ext.data.DirectStore({
        paramsAsHash: false,
        directFn: EmployeeLicense.pagination,
        paramOrder: 'start|limit|sort|dir|where',
        root: 'data',
        idProperty: 'licenseID',
        totalProperty: 'totalRows',
        sortInfo: {
            field: 'license_id',
            direction: 'ASC'
        },
        fields: [{
            name: 'licenseID',
            field: 'license_id',
            type: 'int'
        },
        {
            name: 'employeeID',
            field: 'employee_id',
            type: 'int'
        },
        {
```

```
            name: 'licenseType',
            field: 'license_type',
            type: 'int'
        },
        {
            name: 'licenseNumber',
            field: 'license_number'
        },
        {
            name: 'licenseClass',
            field: 'license_class'
        }],
        remoteSort: true,
        paramNames: {
            start: 'start',
            limit: 'limit',
            sort: 'sort',
            dir: 'dir',
            where: 'where'
        },
        baseParams: {
            start: 0,
            limit: 10,
            where: ''
        }
    });
    var qualificationStore = new Ext.data.DirectStore({
        paramsAsHash: false,
        directFn: EmployeeQualification.pagination,
        paramOrder: 'start|limit|sort|dir|where',
        root: 'data',
        idProperty: 'qualificationID',
        totalProperty: 'totalRows',
        sortInfo: {
            field: 'qualification_id',
            direction: 'ASC'
        },
        fields: [{
            name: 'qualificationID',
            field: 'qualification_id',
            type: 'int'
        },
        {
            name: 'employeeID',
            field: 'employee_id',
            type: 'int'
        },
        {
            name: 'qualificationLevel',
            field: 'qualification_level',
            type: 'int'
        },
        {
            name: 'qualificationTitle',
            field: 'qualification_title'
        },
        {
            name: 'dateObtained',
            field: 'date_obtained'
        },
        {
            name: 'institute',
            field: 'institute'
        }],
        remoteSort: true,
        paramNames: {
            start: 'start',
            limit: 'limit',
            sort: 'sort',
            dir: 'dir',
            where: 'where'
```

```
        },
        baseParams: {
            start: 0,
            limit: 10,
            where: ''
        }
    });
    var phoneStore = new Ext.data.DirectStore({
        paramsAsHash: false,
        directFn: EmployeePhoneNumber.pagination,
        paramOrder: 'start|limit|sort|dir|where',
        root: 'data',
        idProperty: 'phoneID',
        totalProperty: 'totalRows',
        sortInfo: {
            field: 'phone_id',
            direction: 'ASC'
        },
        fields: [{
            name: 'phoneID',
            field: 'phone_id',
            type: 'int'
        },
        {
            name: 'employeeID',
            field: 'employee_id',
            type: 'int'
        },
        {
            name: 'contactInformation',
            field: 'contact_information'
        },
        {
            name: 'contactNumber',
            field: 'contact_number'
        }],
        remoteSort: true,
        paramNames: {
            start: 'start',
            limit: 'limit',
            sort: 'sort',
            dir: 'dir',
            where: 'where'
        },
        baseParams: {
            start: 0,
            limit: 10,
            where: ''
        }
    });
    var licenseTypeComboBox = new Ext.form.ComboBox({
        fieldLabel: 'License Type',
        hiddenName: 'licenseType',
        store: licenseTypes,
        valueField: 'id',
        displayField: 'name',
        mode: 'remote',
        editable: false,
        triggerAction: 'all',
        emptyText: 'Select a type ...',
        allowBlank: false,
        blankText: 'Select a type',
        width: 200
    });
    var qualificationLevelComboBox = new Ext.form.ComboBox({
        fieldLabel: 'Qualification',
        hiddenName: 'qualificationLevel',
        store: qualificationLevels,
        valueField: 'id',
        displayField: 'name',
        mode: 'remote',
```

```
        editable: false,
        triggerAction: 'all',
        emptyText: 'Select a level ...',
        allowBlank: false,
        blankText: 'Select a level',
        width: 200
    });
    var employeeManagement = {
        form: new Ext.FormPanel({
            border: true,
            bodyStyle: 'padding:10px 10px 0',
            title: 'Employee Details',
            collapsible: true,
            autoWidth: true,
            autoHeight: true,
            layout: 'fit',
            plugins: [new Ext.ux.FormClear()],
            items: [{
                xtype: 'hidden',
                name: 'employeeID'
            },
            {
                xtype: 'fieldset',
                autoHeight: true,
                anchor: '100% -20',
                layout: 'column',
                items: [{
                    columnWidth: '.5',
                    layout: 'form',
                    labelAlign: 'top',
                    border: false,
                    items: [{
                        xtype: 'textfield',
                        fieldLabel: 'First Name',
                        name: 'firstName',
                        tabIndex: 1,
                        width: 150
                    },
                    new Ext.form.ComboBox({
                        fieldLabel: 'Department',
                        hiddenName: 'departmentID',
                        store: dataSets.departmentStore,
                        valueField: 'departmentID',
                        displayField: 'departmentName',
                        mode: 'local',
                        editable: false,
                        triggerAction: 'all',
                        emptyText: 'Select a department ...',
                        allowBlank: false,
                        blankText: 'Select a department',
                        tabIndex: 3,
                        width: 150
                    }), {
                        xtype: 'textfield',
                        fieldLabel: 'Street Address',
                        name: 'streetAddress',
                        tabIndex: 5,
                        width: 150
                    },
                    {
                        xtype: 'textfield',
                        fieldLabel: 'Postcode',
                        name: 'postcode',
                        tabIndex: 7,
                        width: 150
                    },
                    {
                        xtype: 'textfield',
                        fieldLabel: 'Email Address',
                        name: 'emailAddress',
                        tabIndex: 9,
```

```
                width: 150
            }]
        },
        {
            columnWidth: '.5',
            layout: 'form',
            labelAlign: 'top',
            border: false,
            items: [{
                xtype: 'textfield',
                fieldLabel: 'Last Name',
                name: 'lastName',
                tabIndex: 2,
                width: 150
            },
            new Ext.form.ComboBox({
                fieldLabel: 'Section',
                hiddenName: 'employeeSectionID',
                store: dataSets.employeeSectionList,
                valueField: 'employeeSectionID',
                displayField: 'sectionTitle',
                mode: 'local',
                editable: false,
                triggerAction: 'all',
                emptyText: 'Select a section ...',
                allowBlank: false,
                blankText: 'Select a section',
                tabIndex: 4,
                width: 150
            }), {
                xtype: 'textfield',
                fieldLabel: 'Suburb',
                name: 'suburb',
                tabIndex: 6,
                width: 150
            },
            new Ext.form.ComboBox({
                fieldLabel: 'State',
                hiddenName: 'state',
                store: dataSets.stateList,
                valueField: 'name',
                displayField: 'name',
                mode: 'remote',
                editable: false,
                triggerAction: 'all',
                emptyText: 'Select a state ...',
                allowBlank: false,
                blankText: 'Select a state',
                tabIndex: 8,
                width: 150
            }), new Ext.form.ComboBox({
                fieldLabel: 'Employee Type',
                hiddenName: 'employeeType',
                store: dataSets.employeeTypes,
                valueField: 'id',
                displayField: 'name',
                mode: 'remote',
                editable: false,
                triggerAction: 'all',
                emptyText: 'Select a type ...',
                allowBlank: false,
                blankText: 'Select a type',
                tabIndex: 10,
                width: 150
            })]
        }]
    }],
    api: {
        load: Employee.create,
        submit: Employee.commit
    },
```

```
            paramOrder: ['identifier'],
            listeners: {
                beforeaction: function (form, action) {
                    if (action.type == 'directsubmit') {}
                },
                actioncomplete: function (form, action) {
                    if (action.type == 'directload') {
                        var where = ' WHERE employee_id = ' + form.findField('employeeID').getValue();
                        licenseStore.setBaseParam('where', where);
                        qualificationStore.setBaseParam('where', where);
                        phoneStore.setBaseParam('where', where);
                        licenseStore.load();
                        qualificationStore.load();
                        phoneStore.load();
                        recordPanel.show();
                    }
                }
            }
        }),
        grid: new Ext.grid.GridPanel({
            store: dataSets.employeeStore,
            border: false,
            autoHeight: true,
            autoWidth: true,
            cm: new Ext.grid.ColumnModel([new Ext.grid.RowNumberer(), {
                header: 'First Name',
                width: 100,
                sortable: true,
                dataIndex: 'firstName'
            },
            {
                header: 'Last Name',
                width: 100,
                sortable: true,
                dataIndex: 'lastName'
            },
            {
                header: 'Section',
                width: 100,
                sortable: true,
                renderer: function (value) {
                    return dataSets.employeeSectionList.getById(parseInt(value)).get('sectionTitle');
                },
                dataIndex: 'employeeSectionID',
                hidden: true
            }]),
            view: new Ext.grid.GroupingView({
                forceFit: true,
                groupTextTpl: '{text}'
            }),
            tbar: new Ext.PagingToolbar({
                pageSize: 10,
                store: dataSets.employeeStore,
                displayInfo: true
            }),
            listeners: {
                rowdblclick: function (grid, index, e) {
                    tabPanel.get(1).setDisabled(false).show();
                    FormActions.triggerLoad(employeeManagement.form, grid.getStore().getAt(index).id);
                }
            }
        }),
        defaultData: {
            employeeID: 0,
            sectionalisedID: '',
            supervisorID: '',
            employeeType: 0,
            firstName: '',
            lastName: '',
            streetAddress: '',
            suburb: '',
```

```
            postcode: '',
            state: '',
            emailAddress: ''
        }
    };
    var licenseGrid = new Ext.ux.GridEditor({
        store: licenseStore,
        idProperty: 'licenseID',
        autoHeight: true,
        autoWidth: true,
        defaultData: function () {
            return {
                licenseID: 0,
                employeeID: employeeManagement.form.getForm().findField('employeeID').getValue(),
                licenseType: 0,
                licenseNumber: '',
                licenseClass: ''
            };
        },
        api: {
            submit: EmployeeLicense.saveData,
            remove: EmployeeLicense.remove
        },
        cm: new Ext.grid.ColumnModel([{
            header: 'License Number',
            width: 100,
            sortable: true,
            dataIndex: 'licenseNumber',
            editor: {
                xtype: 'textfield',
                allowBlank: false
            }
        },
        {
            header: 'License Class',
            width: 100,
            sortable: true,
            dataIndex: 'licenseClass',
            editor: {
                xtype: 'textfield',
                allowBlank: false
            }
        },
        {
            header: 'License Type',
            width: 100,
            sortable: true,
            dataIndex: 'licenseType',
            renderer: Ext.ux.renderer.Combo(licenseTypeComboBox),
            editor: licenseTypeComboBox
        }])
    });
    var qualificationGrid = new Ext.ux.GridEditor({
        store: qualificationStore,
        idProperty: 'qualificationID',
        autoHeight: true,
        autoWidth: true,
        defaultData: function () {
            return {
                qualificationID: 0,
                employeeID: employeeManagement.form.getForm().findField('employeeID').getValue(),
                qualificationLevel: 0,
                dateObtained: '',
                qualificationTitle: '',
                institute: ''
            };
        },
        api: {
            submit: EmployeeQualification.saveData,
            remove: EmployeeQualification.remove
        },
```

```
        cm: new Ext.grid.ColumnModel([{
            header: 'Qualification Title',
            width: 100,
            sortable: true,
            dataIndex: 'qualificationTitle',
            editor: {
                xtype: 'textfield',
                allowBlank: false
            }
        },
        {
            header: 'Institute',
            width: 100,
            sortable: true,
            dataIndex: 'institute',
            editor: {
                xtype: 'textfield',
                allowBlank: false
            }
        },
        {
            header: 'Date Obtained',
            width: 100,
            sortable: true,
            dataIndex: 'dateObtained',
            editor: {
                xtype: 'datefield',
                allowBlank: false
            },
            renderer: Ext.util.Format.dateRenderer('d-m-Y')
        },
        {
            header: 'Level',
            width: 100,
            sortable: true,
            dataIndex: 'qualificationLevel',
            renderer: Ext.ux.renderer.Combo(qualificationLevelComboBox),
            editor: qualificationLevelComboBox
        }])
    });
    var phoneRegistryGrid = new Ext.ux.GridEditor({
        store: phoneStore,
        idProperty: 'phoneID',
        autoHeight: true,
        autoWidth: true,
        defaultData: function () {
            return {
                phoneID: 0,
                employeeID: employeeManagement.form.getForm().findField('employeeID').getValue(),
                contactInformation: '',
                contactNumber: ''
            };
        },
        api: {
            submit: EmployeePhoneNumber.saveData,
            remove: EmployeePhoneNumber.remove
        },
        cm: new Ext.grid.ColumnModel([{
            header: 'Contact Information',
            width: 100,
            sortable: true,
            dataIndex: 'contactInformation',
            editor: {
                xtype: 'textfield',
                allowBlank: false
            }
        },
        {
            header: 'Phone Number',
            width: 100,
            sortable: true,
```

```
            dataIndex: 'contactNumber',
            editor: {
                xtype: 'textfield',
                allowBlank: false
            }
        }])
    });
    var gridPanel = new Ext.grid.GridPanel({
        title: 'Department',
        store: dataSets.departmentStore,
        collapsible: true,
        border: true,
        region: 'west',
        width: 200,
        cm: new Ext.grid.ColumnModel([{
            header: 'Department',
            sortable: false,
            dataIndex: 'departmentName'
        }]),
        sm: new Ext.grid.RowSelectionModel({
            singleSelect: true
        }),
        viewConfig: {
            forceFit: true
        },
        listeners: {
            beforerender: function (grid) {
                dataSets.employeeSectionList.load();
                dataSets.employeeTypes.load();
                dataSets.departmentStore.load();
            }
        }
    });
    gridPanel.getSelectionModel().on('rowselect', function (sm, index, r) {
        tabPanel.get(0).show();
        dataSets.employeeStore.setBaseParam('where', ' WHERE department_id = ' +
gridPanel.getStore().getAt(index).id);
        dataSets.employeeStore.load();
    });
    dataSets.departmentStore.on('load', function () {
        gridPanel.getSelectionModel().selectFirstRow();
    });
    var recordPanel = new Ext.TabPanel({
        region: 'center',
        margins: '3 3 3 3',
        activeTab: 0,
        autoHeight: true,
        items: [{
            title: 'Licenses',
            layout: 'fit',
            items: [new Ext.Panel({
                layout: 'fit',
                border: false,
                autoHeight: true,
                autoWidth: true,
                items: [licenseGrid],
                bbar: new Ext.PagingToolbar({
                    pageSize: 10,
                    store: licenseStore,
                    displayInfo: true
                })
            })]
        },
        {
            title: 'Qualifications',
            layout: 'fit',
            items: [new Ext.Panel({
                layout: 'fit',
                border: false,
                autoHeight: true,
                autoWidth: true,
```

```
                items: [qualificationGrid],
                bbar: new Ext.PagingToolbar({
                    pageSize: 10,
                    store: qualificationStore,
                    displayInfo: true
                })
            })]
        },
        {
            title: 'Phone Registry',
            layout: 'fit',
            items: [new Ext.Panel({
                layout: 'fit',
                border: false,
                autoHeight: true,
                autoWidth: true,
                items: [phoneRegistryGrid],
                bbar: new Ext.PagingToolbar({
                    pageSize: 10,
                    store: phoneStore,
                    displayInfo: true
                })
            })]
        }]
    });
    recordPanel.doLayout();
    var tabPanel = new Ext.TabPanel({
        region: 'center',
        margins: '3 3 3 3',
        activeTab: 0,
        defaults: {
            autoScroll: true
        },
        items: [{
            title: 'List View',
            layout: 'fit',
            items: [employeeManagement.grid],
            listeners: {
                beforeshow: function () {
                    tabPanel.get(1).setDisabled(true);
                }
            }
        },
        {
            title: 'Record View',
            layout: 'fit',
            items: [employeeManagement.form, recordPanel]
        }]
    });
    tabPanel.doLayout();
    var newFn = function () {
        tabPanel.get(1).setDisabled(false).show();
        recordPanel.hide();
    };
    var successFn = function () {
        dataSets.employeeStore.load();
    };
    var deleteFn = function () {
        var sm = employeeManagement.grid.getSelectionModel();
        if (sm.hasSelection()) {
            Employee.remove(sm.getSelected().get('employeeID'), function (result, e) {
                dataSets.employeeStore.load();
            });
        }
    };
    FormActions.initFields(employeeManagement.form, module);
    FormActions.defaultHandler(commands, employeeManagement.form, dataSets.employeeStore.recordType,
employeeManagement.defaultData, newFn, successFn, deleteFn);
    TricertAssist.showWindow([gridPanel, tabPanel], {
        text: module.moduleTitle,
        commands: commands,
```

```
        layout: 'border',
        width: 625,
        height: 390,
    });
});
```

Training Courses

```
Ext.onReady(function () {
    var courseStore = new Ext.data.DirectStore({
        idProperty: 'courseID',
        directFn: TrainingCourse.list,
        paramsAsHash: false,
        fields: [{
            name: 'courseID',
            type: 'int'
        },
        {
            name: 'courseTitle'
        }]
    });
    var moduleStore = new Ext.data.DirectStore({
        paramsAsHash: false,
        directFn: TrainingModule.pagination,
        paramOrder: 'start|limit|sort|dir|where',
        root: 'data',
        idProperty: 'courseModuleID',
        totalProperty: 'totalRows',
        sortInfo: {
            field: 'course_module_id',
            direction: 'ASC'
        },
        fields: [{
            name: 'courseModuleID',
            field: 'course_module_id',
            type: 'int'
        },
        {
            name: 'courseID',
            field: 'course_id',
            type: 'int'
        },
        {
            name: 'courseModuleTitle',
            field: 'course_module_title'
        },
        {
            name: 'courseModuleContents',
            field: 'course_module_contents'
        }],
        remoteSort: true,
        paramNames: {
            start: 'start',
            limit: 'limit',
            sort: 'sort',
            dir: 'dir',
            where: 'where'
        },
        baseParams: {
            start: 0,
            limit: 10,
            where: ''
        }
    });
    var recordType = Ext.data.Record.create({
        name: 'courseID',
```

```
        type: 'int'
    },
    {
        name: 'courseTitle'
    },
    {
        name: 'courseDescription'
    });
    var form = new Ext.FormPanel({
        border: true,
        bodyStyle: 'padding:10px 10px 0',
        title: 'Training Course',
        collapsible: true,
        autoHeight: true,
        autoWidth: true,
        region: 'center',
        layout: 'fit',
        plugins: [new Ext.ux.FormClear()],
        items: [{
            xtype: 'hidden',
            name: 'courseID'
        },
        {
            xtype: 'fieldset',
            autoHeight: true,
            autoWidth: true,
            items: [{
                xtype: 'textfield',
                fieldLabel: 'Course Title',
                name: 'courseTitle',
                width: 230,
                allowBlank: false,
                blankText: 'Enter a Course Title'
            },
            new Ext.form.TextArea({
                fieldLabel: 'Description',
                width: 230,
                name: 'courseDescription',
                allowBlank: false,
                blankText: 'Enter a Course Description'
            })]
        }],
        api: {
            load: TrainingCourse.create,
            submit: TrainingCourse.commit
        },
        paramOrder: ['identifier'],
        listeners: {
            beforeaction: function (form, action) {
                if (action.type == 'directsubmit') {}
            },
            actioncomplete: function (form, action) {
                if (action.type == 'directload') {
                    moduleStore.setBaseParam('where', ' WHERE course_id = ' +
form.findField('courseID').getValue());
                    moduleStore.load();
                    modulePanel.show();
                }
            }
        }
    });
    var moduleGrid = new Ext.ux.GridEditor({
        store: moduleStore,
        idProperty: 'courseModuleID',
        autoHeight: true,
        autoWidth: true,
        defaultData: function () {
            return {
                courseModuleID: 0,
                courseID: form.getForm().findField('courseID').getValue(),
                courseModuleTitle: '',
```

```
            courseModuleContents: ''
        };
    },
    api: {
        submit: TrainingModule.saveData,
        remove: TrainingModule.remove
    },
    cm: new Ext.grid.ColumnModel([{
        header: 'Module Title',
        sortable: false,
        width: 100,
        dataIndex: 'courseModuleTitle',
        editor: {
            xtype: 'textfield',
            allowBlank: false
        }
    },
    {
        header: 'Contents',
        sortable: false,
        width: 200,
        dataIndex: 'courseModuleContents',
        editor: {
            xtype: 'textfield',
            allowBlank: false
        }
    }])
});
var modulePanel = new Ext.Panel({
    title: 'Course Modules',
    collapsible: true,
    border: true,
    layout: 'fit',
    items: [moduleGrid],
    bbar: new Ext.PagingToolbar({
        pageSize: 10,
        store: moduleStore,
        displayInfo: true
    })
});
FormActions.initFields(form, module);
var grid = new Ext.grid.GridPanel({
    title: 'Training Courses',
    store: courseStore,
    border: true,
    region: 'west',
    width: 200,
    cm: new Ext.grid.ColumnModel([{
        header: 'Course Title',
        sortable: false,
        dataIndex: 'courseTitle'
    }]),
    sm: new Ext.grid.RowSelectionModel({
        singleSelect: true
    }),
    viewConfig: {
        forceFit: true
    },
    listeners: {
        beforerender: function (grid) {
            courseStore.load();
        }
    }
});
grid.getSelectionModel().on('rowselect', function (sm, index, r) {
    FormActions.triggerLoad(form, grid.getStore().getAt(index).id);
});
courseStore.on('load', function () {
    grid.getSelectionModel().selectFirstRow();
});
var defaultData = {
```

```
        courseID: 0,
        courseTitle: '',
        courseDescription: ''
    };
    var newFn = function () {
        modulePanel.hide();
    };
    var successFn = function () {
        courseStore.load();
    };
    var deleteFn = function () {
        var sm = grid.getSelectionModel();
        if (sm.hasSelection()) {
            TrainingCourse.remove(sm.getSelected().get('courseID'), function (result, e) {
                form.getForm().clear();
                courseStore.load();
            });
        }
    };
    FormActions.defaultHandler(commands, form, recordType, defaultData, newFn, successFn, deleteFn);
    TricertAssist.showWindow([grid, new Ext.Panel({
        margins: '3 3 3 3',
        region: 'center',
        layout: 'fit',
        autoHeight: true,
        items: [form, modulePanel],
        defaults: {
            autoScroll: true
        }
    })], {
        text: module.moduleTitle,
        commands: commands,
        layout: 'border',
        width: 650,
        height: 450
    });
})
```

Java Backend Source

Business Object Layer

BusinessModule.java

```
package bol;

import bol.modules.AccessGroup;
import bol.modules.Section;
import bol.modules.User;
import bol.util.Response.ResponseSubmission;
import java.util.Map;
import org.apache.commons.beanutils.BeanUtils;
import org.apache.commons.fileupload.FileItem;
import bol.util.Response.ResponseList;
import bol.util.Response.ResponseObject;
import com.jgoodies.validation.util.ValidationUtils;
import djn.config.annotations.DirectFormPostMethod;
import djn.config.annotations.DirectMethod;
import dal.Dal;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.logging.Level;
import java.util.logging.Logger;

public abstract class BusinessModule {

    private static final String CONSTANT_SQL_TABLE = "SQL_TABLE";
    private static final String CONSTANT_SQL_FETCH = "SQL_FETCH";
    private static final String CONSTANT_SQL_LIST = "SQL_LIST";
    private static final String CONSTANT_SQL_DELETE = "SQL_DELETE";
    private static final String ENTITY_STATE = "entity";
    private static final String MODULE = "module";


    private EntityState entityState;
    private Validation validation = new Validation();

    public enum EntityState {

        Added, Modified, Unchanged
    };

    protected Validation getValidation() {
        return this.validation;
    }

    public EntityState getEntityState() {
        return this.entityState;
    }

    public void setEntityState(EntityState entityState) {
        this.entityState = entityState;
    }

    @DirectMethod
    public ResponseObject create(int identifier) throws Exception {
        ResponseObject response = new ResponseObject();
```

```
            response.setObject(Socket.create(new Object[]{identifier},
        this.getClass().getField(BusinessModule.CONSTANT_SQL_FETCH).get(null).toString(), this.getClass(),
        Dal.StatmentType.preparedStatment));
         response.setSuccess(true);
         return response;
    }

    @DirectMethod
    public void remove(int identifier) throws Exception {
             Dal.doMutation(new Object[]{identifier},
        this.getClass().getField(BusinessModule.CONSTANT_SQL_DELETE).get(null).toString());
    }

    @DirectMethod
    public ArrayList list() throws Exception {
             return Socket.list(new Object[]{},
        this.getClass().getField(BusinessModule.CONSTANT_SQL_LIST).get(null).toString(), this.getClass(),
        Dal.StatmentType.preparedStatment);
    }

    @DirectMethod
    public ResponseList pagination(int start, int limit, String sort, String direction, String where)
throws  Exception {
         return Socket.pagination(this.getClass().getField(BusinessModule.CONSTANT_SQL_TABLE).get(null).toString(),
        this.getClass(), start, limit, sort, direction, where);
    }

    @DirectFormPostMethod
    public ResponseSubmission commit(Map<String, String> formParameters, Map<String, FileItem> fileFields) {
         ResponseSubmission response = new ResponseSubmission();
         try {
             User user = User.getLoggedInUser();
             if (user != null) {
                 AccessGroup group = user.getAccessGroup();
                         Section section = Section.getAuthorisedSection(group,
                    Integer.valueOf(formParameters.get(BusinessModule.MODULE)));
                 if (section != null) {
                     BusinessModule instance = this.getClass().newInstance();
                     instance.setEntityState(EntityState.values()[
                        Integer.valueOf(formParameters.get(BusinessModule.ENTITY_STATE))]);
                     BeanUtils.populate(instance, formParameters);
                     if (instance.getValidation().isValid()) {
                         instance.save(formParameters, fileFields);
                     }
                     response.setErrors(instance.getValidation());
                     response.setSuccess(instance.getValidation().isValid());
                        response.setMessage(response.isSuccess() ? Constants.MSG_SAVE_SUCCESS :
                     Constants.MSG_SAVE_FAILED);
                     response.setTitle(response.isSuccess() ? Constants.TITLE_SAVE_SUCCESS :
                     Constants.TITLE_SAVE_FAILED);
                 }
             }
         } catch (Exception ex) {
             Logger.getLogger(this.getClass().getName()).log(Level.SEVERE, null, ex);
             response.setMessage(ex.getMessage());
             response.setTitle(Constants.TITLE_SAVE_FAILED);
             response.setSuccess(false);
         }
         return response;
    }

    abstract protected PreparedStatement insert() throws SQLException;

    abstract protected PreparedStatement update() throws SQLException;
```

```
    /**
     * The default save method. It is overriden in the sub classes where manipulation
     * using the formParameters, or fileFields are required. The method executes the abstract insert or update
     * methods depending on the state of the object. This method is only used when the commit DirectFormPostMethod
has been triggered.
     * @param formParameters
     * @param fileFields
     */
    public void save(Map<String, String> formParameters, Map<String, FileItem> fileFields) throws SQLException {
        switch (this.getEntityState()) {
            case Added:
                this.insert();
                break;
            case Modified:
                this.update();
                break;
        }
    }

    public static class Validation extends HashMap<String, String> {

        public static final String ERR_REQUIRED = "The field {0} is required.";
        public static final String ERR_LETTERS_ONLY = "The field {0} can only be letters";
        public static final String ERR_LENGTH = "The field {0} must be {1}-{2} characters in length";

        public Validation() {
            // This is required in order for an empty HashMap to be serialised properly.
            this.put(Constants.EMPTY_STRING, Constants.EMPTY_STRING);
        }

        public void addError(String propertyName, String error, Object[] array) {
            MessageFormat message = new MessageFormat(error);
            this.put(propertyName, message.format(array));
        }

        public boolean validateLettersOnly(String propertyName, String value) {
            if (!ValidationUtils.isAlpha(value)) {
                this.addError(propertyName, Validation.ERR_LETTERS_ONLY, new Object[]{propertyName});
                return false;
            }
            return true;
        }

        public boolean validateRequired(String propertyName, String value) {
            if (ValidationUtils.isBlank(value)) {
                this.addError(propertyName, Validation.ERR_REQUIRED, new Object[]{propertyName});
                return false;
            }
            return true;
        }

        public boolean validateLength(String propertyName, String value, int min, int max) {
            if (!ValidationUtils.hasBoundedLength(value, min, max)) {
                this.addError(propertyName, Validation.ERR_LENGTH, new Object[]{propertyName, min, max});
                return false;
            }
            return true;
        }

        public boolean isValid() {
            return (this.size() == 1);
        }
    }
}
```

Constants.java

```
package bol;

public class Constants {

    public static final String SESSION_USER_PROPERTY = "User";
    public static final String SESSION_LOG_PROPERTY = "SessionLog";
    public static final String EMPTY_STRING = "";
    public static final String DOT_CLASS = ".class";
    public static final String DOT = ".";
    public static final String ICON_IMG_PATH = "theme/images/icons/";
    public static final String ICON_EXT = ".png";
    public static final String MSG_SAVE_SUCCESS = "The data has been saved successfully.";
    public static final String MSG_SAVE_FAILED = "The data entered does not meet the validation criteria";
    public static final String TITLE_SAVE_SUCCESS = "Data Saved Successfully";
    public static final String TITLE_SAVE_FAILED = "Save Failed";
    // Users
    public static final String PROPERTY_USER_ID = "userID";
    public static final String PROPERTY_USER_NAME = "userName";
    public static final String PROPERTY_PASSWORD = "password";
    public static final String PROPERTY_CONFIRMATION_PASSWORD = "confirmationPassword";
    public static final String PROPERTY_USER_TYPE = "userType";
    public static final String DB_FIELD_USER_ID = "user_id";
    public static final String DB_FIELD_USER_NAME = "user_name";
    public static final String DB_FIELD_PASSWORD = "user_password";
    public static final String DB_FIELD_USER_TYPE = "user_type";
    // Access Groups
    public static final String PROPERTY_GROUP_ID = "groupID";
    public static final String PROPERTY_GROUP_NAME = "groupName";
    public static final String PROPERTY_GROUP_DESCRIPTION = "groupDescription";
    public static final String PROPERTY_GROUP_COMMANDS = "groupCommands";
    public static final String DB_FIELD_GROUP_ID = "group_id";
    public static final String DB_FIELD_GROUP_NAME = "group_name";
    public static final String DB_FIELD_GROUP_DESCRIPTION = "group_description";
    public static final String DB_FIELD_GROUP_COMMANDS = "group_commands";
    // Category
    public static final String PROPERTY_CATEGORY_ID = "categoryID";
    public static final String PROPERTY_CATEGORY_PARENT = "categoryParent";
    public static final String PROPERTY_CATEGORY_TITLE = "categoryTitle";
    public static final String PROPERTY_CATEGORY_ICON = "categoryIcon";
    public static final String PROPERTY_CATEGORY_TYPE = "categoryType";
    public static final String DB_FIELD_CATEGORY_ID = "category_id";
    public static final String DB_FIELD_CATEGORY_PARENT = "category_parent";
    public static final String DB_FIELD_CATEGORY_TITLE = "category_title";
    public static final String DB_FIELD_CATEGORY_TYPE = "category_type";
    public static final String DB_FIELD_CATEGORY_ICON = "category_icon";
    // Session Logs
    public static final String PROPERTY_SESSION_ID = "sessionID";
    public static final String PROPERTY_SESSION_DATE = "sessionDate";
    public static final String PROPERTY_VISIT_ID = "visitID";
    public static final String PROPERTY_VISIT_TIME = "visitTime";
    public static final String DB_FIELD_SESSION_ID = "session_id";
    public static final String DB_FIELD_SESSION_DATE = "session_date";
    public static final String DB_FIELD_VISIT_ID = "visit_id";
    public static final String DB_FIELD_VISIT_TIME = "visit_time";
    // Section
    public static final String PROPERTY_SECTION_ID = "sectionID";
    public static final String DB_FIELD_SECTION_ID = "section_id";
    // Module
    public static final String PROPERTY_MODULE_ID = "moduleID";
    public static final String PROPERTY_MODULE_TITLE = "moduleTitle";
    public static final String PROPERTY_MODULE_ICON = "moduleIcon";
    public static final String PROPERTY_MODULE_NAME = "moduleName";
    public static final String PROPERTY_MODULE_COMMANDS = "moduleCommands";
    public static final String PROPERTY_MODULE_SOURCE = "moduleSource";
    public static final String DB_FIELD_MODULE_ID = "module_id";
    public static final String DB_FIELD_MODULE_TITLE = "module_title";
```

```
    public static final String DB_FIELD_MODULE_ICON = "module_icon";
    public static final String DB_FIELD_MODULE_NAME = "module_name";
    public static final String DB_FIELD_MODULE_COMMANDS = "module_commands";
    public static final String DB_FIELD_MODULE_SOURCE = "module_source";
    // Permission
    public static final String PROPERTY_PERMISSION_ID = "permissionID";
    public static final String PROPERTY_LINK_ID = "linkID";
    public static final String PROPERTY_PERMISSION_TYPE = "permissionType";
    public static final String DB_FIELD_PERMISSION_ID = "permission_id";
    public static final String DB_FIELD_LINK_ID = "link_id";
    public static final String DB_FIELD_PERMISSION_TYPE = "permission_type";
    // Department
    public static String DB_FIELD_DEPARTMENT_ID = "department_id";
    public static String DB_FIELD_DEPARTMENT_NAME = "department_name";
    public static String DB_FIELD_RECEPTION_PHONE = "reception_phone";
    public static String DB_FIELD_EMPLOYEE_SECTION_ID = "employee_section_id";
    public static String DB_FIELD_MANAGER_ID = "manager_id";
    public static String PROPERTY_DEPARTMENT_ID = "departmentID";
    public static String PROPERTY_EMPLOYEE_SECTION_ID = "employeeSectionID";
    public static String PROPERTY_MANAGER_ID = "managerID";
    public static String PROPERTY_SECTION_TITLE = "sectionTitle";
    public static String PROPERTY_SECTION_DESCRIPTION = "sectionDescription";
    public static String DB_FIELD_SECTION_TITLE = "section_title";
    public static String DB_FIELD_SECTION_DESCRIPTION = "section_description";
    // Employee
    public static final String PROPERTY_EMPLOYEE_ID = "employeeID";
    public static final String PROPERTY_SUPERVISOR_ID = "supervisorID";
    public static final String PROPERTY_EMPLOYEE_TYPE = "employeeType";
    public static final String PROPERTY_FIRST_NAME = "firstName";
    public static final String PROPERTY_LAST_NAME = "lastName";
    public static final String PROPERTY_EMAIL_ADDRESS = "emailAddress";
    public static final String PROPERTY_STREET_ADDRESS = "streetAddress";
    public static final String PROPERTY_SUBURB = "suburb";
    public static final String PROPERTY_POSTCODE = "postcode";
    public static final String PROPERTY_STATE = "state";
    public static final String DB_FIELD_EMPLOYEE_ID = "employee_id";
    public static final String DB_FIELD_SECTIONALISED_ID = "sectionalised_id";
    public static final String DB_FIELD_SUPERVISOR_ID = "supervisor_id";
    public static final String DB_FIELD_EMPLOYEE_TYPE = "employee_type";
    public static final String DB_FIELD_FIRST_NAME = "first_name";
    public static final String DB_FIELD_LAST_NAME = "last_name";
    public static final String DB_FIELD_EMAIL_ADDRESS = "email_address";
    public static final String DB_FIELD_STREET_ADDRESS = "street_address";
    public static final String DB_FIELD_SUBURB = "suburb";
    public static final String DB_FIELD_POSTCODE = "postcode";
    public static final String DB_FIELD_STATE = "state";
    //Employee License
    public static final String DB_FIELD_LICENSE_ID = "license_id";
    public static final String DB_FIELD_LICENSE_TYPE = "license_type";
    public static final String DB_FIELD_LICENSE_NUMBER = "license_number";
    public static final String DB_FIELD_LICENSE_CLASS = "license_class";
    public static final String PROPERTY_LICENSE_ID = "licenseID";
    public static final String PROPERTY_LICENSE_TYPE = "licenseType";
    public static final String PROPERTY_LICENSE_NUMBER = "licenseNumber";
    public static final String PROPERTY_LICENSE_CLASS = "licenseClass";
    //Employee Qualifications
    public static final String DB_FIELD_QUALIFICATION_ID = "qualification_id";
    public static final String DB_FIELD_QUALIFICATION_LEVEL = "qualification_level";
    public static final String DB_FIELD_QUALIFICATION_TITLE = "qualification_title";
    public static final String DB_FIELD_DATE_OBTAINED = "date_obtained";
    public static final String DB_FIELD_INSTITUTE = "institute";
    public static final String PROPERTY_QUALIFICATION_ID = "qualificationID";
    public static final String PROPERTY_QUALIFICATION_LEVEL = "qualificationLevel";
    public static final String PROPERTY_QUALIFICATION_TITLE = "qualificationTitle";
    public static final String PROPERTY_DATE_OBTAINED = "dateObtained";
    public static final String PROPERTY_INSTITUTE = "institute";
    //Employee Phone Number
    public static final String DB_FIELD_PHONE_ID = "phone_id";
    public static final String DB_FIELD_CONTACT_INFORMATION = "contact_information";
    public static final String DB_FIELD_CONTACT_NUMBER = "contact_number";
    public static final String PROPERTY_PHONE_ID = "phoneID";
```

```
    public static final String PROPERTY_CONTACT_INFORMATION = "contactInformation";
    public static final String PROPERTY_CONTACT_NUMBER = "contactNumber";
    // Training Courses
    public static final String DB_FIELD_COURSE_ID = "course_id";
    public static final String DB_FIELD_COURSE_TITLE = "course_title";
    public static final String DB_FIELD_COURSE_DESCRIPTION = "course_description";
    public static final String PROPERTY_COURSE_ID = "courseID";
    public static final String PROPERTY_COURSE_TITLE = "courseTitle";
    public static final String PROPERTY_COURSE_DESCRIPTION = "courseDescription";
    // Training Course Modules
    public static final String DB_FIELD_COURSE_MODULE_ID = "course_module_id";
    public static final String DB_FIELD_COURSE_MODULE_TITLE = "course_module_title";
    public static final String DB_FIELD_COURSE_MODULE_CONTENTS = "course_module_contents";
    public static final String PROPERTY_COURSE_MODULE_ID = "courseModuleID";
    public static final String PROPERTY_COURSE_MODULE_TITLE = "courseModuleTitle";
    public static final String PROPERTY_COURSE_MODULE_CONTENTS = "courseModuleContents";
}
```

Socket.java

```
package bol;

import bol.util.Response.ResponseList;
import bol.util.Response.ResponseMessage;
import com.google.gson.JsonObject;
import dal.Dal;
import dal.Dal.StatmentType;
import java.lang.reflect.Constructor;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Socket {

    private static <E> void processResult(ArrayList<E> list, ResultSet records, Class clazz) throws Exception {
        Constructor constructor = clazz.getConstructor(ResultSet.class);
        while (records.next()) {
            list.add((E) constructor.newInstance(records));
        }
    }

    public static <E> E create(Object[] parameters, String statement, Class clazz, StatmentType type) {
        try {
            Constructor constructor = clazz.getConstructor(ResultSet.class);
            ResultSet records = Dal.doQuery(parameters, statement, type);
            return (Dal.hasRows(records) ? (E) constructor.newInstance(records) : null);
        } catch (Exception ex) {
            Logger.getLogger(Socket.class.getName()).log(Level.SEVERE, null, ex);
        }
        return null;
    }

    public static <E> ResponseList pagination(String table, Class clazz, int start, int limit, String sort, String
direction, String where) {
        ResponseList response = new ResponseList();
        try {
            response.setData(new ArrayList<E>());
            response.setTotalRows(Dal.getCount(table, where));
            Socket.<E>processResult((ArrayList<E>) response.getData(), Dal.doQuery(String.format("SELECT * FROM %s%s
ORDER BY %s %s LIMIT %d, %d", table, where, sort, direction, start, limit)), clazz);
            response.setSuccess(true);
        } catch (Exception ex) {
            Logger.getLogger(BusinessModule.class.getName()).log(Level.SEVERE, null, ex);
            response.setSuccess(false);
            response.setMessage(ex.getMessage());
        }
        return response;
```

```
    }

    public static <E> ArrayList<E> list(Object[] parameters, String statement, Class clazz, StatmentType type) {
        ArrayList<E> list = new ArrayList<E>();
        try {
            ResultSet records = Dal.doQuery(parameters, statement, type);
            Socket.<E>processResult(list, records, clazz);
        } catch (Exception ex) {
            Logger.getLogger(BusinessModule.class.getName()).log(Level.SEVERE, null, ex);
        }
        return list;
    }

    public static ResponseMessage saveData(JsonObject data, String propertyID, Class clazz) {
        ResponseMessage response = new ResponseMessage();
        try {
            Constructor constructor = clazz.getConstructor(JsonObject.class);
            BusinessModule object = (BusinessModule) constructor.newInstance(data);
            if (data.get(propertyID).getAsInt() == 0) object.insert(); else object.update();
            response.setTitle(Constants.TITLE_SAVE_SUCCESS);
            response.setMessage(Constants.MSG_SAVE_SUCCESS);
            response.setSuccess(true);
        } catch (Exception ex) {
            Logger.getLogger(clazz.getName()).log(Level.SEVERE, null, ex);
            response.setTitle(Constants.TITLE_SAVE_FAILED);
            response.setMessage(ex.getMessage());
            response.setSuccess(false);
        }
        return response;
    }
}
```

Business Object Layer Modules

Module.java

```
package bol.modules;

import bol.BusinessModule;
import bol.Constants;
import bol.Socket;
import bol.util.Response.ResponseType;
import com.google.gson.Gson;
import djn.config.annotations.DirectMethod;
import dal.Dal;
import java.io.File;
import java.io.Serializable;
import java.net.URLDecoder;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.commons.fileupload.FileItem;

public class Module extends BusinessModule implements Serializable {

    public static final String SQL_FETCH = "SELECT * FROM system_modules WHERE module_id = ? LIMIT 1";
    public static final String SQL_DELETE = "DELETE FROM system_modules WHERE module_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM system_modules";
    public static final String SQL_TABLE = "system_modules";
    private static final String SQL_UPDATE = "UPDATE system_modules SET module_title = ?, module_icon = ?,
    module_source = ?, module_name = ?, module_commands = ? WHERE module_id = ?";
    private static final String SQL_INSERT = "INSERT INTO system_modules (module_title, module_icon, module_source,
    module_name, module_commands) VALUES (?, ?, ?, ?, ?)";
    private static final String DEFAULT_MODULE_ICON = "application.png";
    private int moduleID;
    private String moduleTitle;
    private String moduleIcon;
    private String moduleSource;
    private String moduleName;
    private int[] moduleCommands;

    public Module() {
    }

    public Module(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setModuleID(data.getInt(Constants.DB_FIELD_MODULE_ID));
            this.setModuleTitle(data.getString(Constants.DB_FIELD_MODULE_TITLE));
            this.setModuleIcon(data.getString(Constants.DB_FIELD_MODULE_ICON));
            this.setModuleSource(data.getString(Constants.DB_FIELD_MODULE_SOURCE));
            this.setModuleName(data.getString(Constants.DB_FIELD_MODULE_NAME));
            this.setModuleCommands(data.getString(Constants.DB_FIELD_MODULE_COMMANDS));
        } catch (SQLException ex) {
            Logger.getLogger(Module.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public int getModuleID() {
        return this.moduleID;
    }



    public void setModuleID(int moduleID) {
```

```
        this.moduleID = moduleID;
    }

    public String getModuleTitle() {
        return moduleTitle;
    }

    public void setModuleTitle(String moduleTitle) {
        this.moduleTitle = moduleTitle;
    }

    public String getModuleIcon() {
        return moduleIcon;
    }

    public String getIconPath() {
                return Constants.ICON_IMG_PATH + ((!this.getModuleIcon().equals(Constants.EMPTY_STRING)) ?
        this.getModuleIcon() + Constants.ICON_EXT : Module.DEFAULT_MODULE_ICON);
    }

    public void setModuleIcon(String moduleIcon) {
        this.moduleIcon = moduleIcon;
    }

    public String getModuleName() {
        return this.moduleName;
    }

    public void setModuleName(String moduleName) {
        this.moduleName = moduleName;
    }

    public int[] getModuleCommands() {
        return this.moduleCommands;
    }

    public void setModuleCommands(String moduleCommands) {
        Gson gson = new Gson();
        this.moduleCommands = gson.fromJson(moduleCommands, int[].class);
    }

    public String getModuleSource() {
        return this.moduleSource;
    }

    public void setModuleSource(String moduleSource) {
        this.moduleSource = moduleSource;
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        Gson gson = new Gson();
                return Dal.doMutation(new Object[]{this.getModuleTitle(), this.getModuleIcon(),
        this.getModuleSource(), this.getModuleName(), gson.toJson(this.getModuleCommands(), int[].class)},
        Module.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        Gson gson = new Gson();
        return Dal.doMutation(new Object[]{this.getModuleTitle(), this.getModuleIcon(), this.getModuleSource(),
        this.getModuleName(), gson.toJson(this.getModuleCommands(), int[].class), this.getModuleID()},
        Module.SQL_UPDATE);
    }
```

```
    @Override
    public void save(Map<String, String> formParameters, Map<String, FileItem> fileFields) throws SQLException {
        this.setModuleCommands(formParameters.get(Constants.PROPERTY_MODULE_COMMANDS));
        switch (this.getEntityState()) {
            case Added:
                this.insert();
                break;
            case Modified:
                this.update();
                break;
        }
    }

    @DirectMethod
    public static ArrayList<ResponseType> getSimpleList() {
        ArrayList<ResponseType> list = new ArrayList<ResponseType>();
        for (Module module : Socket.<Module>list(new Object[]{}, Module.SQL_LIST, Module.class,
        Dal.StatmentType.preparedStatment)) {
            list.add(new ResponseType(module.getModuleID(), module.getModuleTitle()));
        }
        return list;
    }

    @DirectMethod
    public static ArrayList<ResponseType> getModuleList() {
        ArrayList<ResponseType> list = new ArrayList<ResponseType>();
        try {
                            File directory = new
               File(URLDecoder.decode(Thread.currentThread().getContextClassLoader().getResource(Module.class.getPac
               kage().getName().replace('.', '/')).getFile(), "UTF-8"));
            if (directory.isDirectory()) {
                int index = 0;
                for (File file : directory.listFiles()) {
                    if (file.getName().endsWith(Constants.DOT_CLASS)) {
                                        Class clazz = Class.forName(Module.class.getPackage().getName() +
               Constants.DOT + file.getName().replace(Constants.DOT_CLASS, Constants.EMPTY_STRING));
                        if (clazz.getSuperclass().equals(BusinessModule.class)) {
                            list.add(new ResponseType(index, clazz.getSimpleName()));
                        }
                    }
                    index++;
                }
            }
        } catch (Exception ex) {
            Logger.getLogger(Module.class.getName()).log(Level.SEVERE, null, ex);
        }
        return list;
    }
}
```

Section.java

```
package bol.modules;

import dal.Dal;
import bol.BusinessModule;
import bol.Constants;
import bol.Socket;
import bol.util.Response.ResponseMessage;
import bol.util.Response.ResponseObject;
import bol.util.Response.ResponseType;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import com.mysql.jdbc.PreparedStatement;
import djn.config.annotations.DirectMethod;
import java.io.Serializable;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Section extends BusinessModule implements Serializable {

    public static final String SQL_DELETE = "DELETE FROM system_sections WHERE section_id = ? LIMIT 1";
    public static final String SQL_FETCH = "SELECT * FROM system_sections WHERE section_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM system_sections";
    public static final String SQL_TABLE = "system_sections";
    private static final String SQL_UPDATE = "UPDATE system_sections SET module_id = ?, group_id = ?, category_id =
? WHERE section_id = ?";
    private static final String SQL_INSERT = "INSERT INTO system_sections (module_id, group_id, category_id) VALUES
(?, ?, ?)";
    private static final String SQL_GRANTED_SECTION_LIST = "SELECT * FROM system_sections WHERE group_id = ? AND
category_id = ?";
    private static final String SQL_FETCH_AUTHORISED_SECTION = "SELECT * FROM system_sections WHERE group_id = ? AND
module_id = ?";
    private static final String ERROR_INVALID_ACCESS_PERMISSIONS = "Invalid Access Permissions Defined";
    private static final String ERROR_MODULE_DOESNT_EXIST = "Associated Module Does Not Exist";
    private static final String ERROR_USER_SESSION_EXPIRED = "User Session Expired";
    private static final String MSG_SPAWNED_SECTION = "Spawned Section";
    private int sectionID;
    private int moduleID;
    private int groupID;
    private int categoryID;

    public Section() {
    }

    public Section(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setSectionID(data.getInt(Constants.DB_FIELD_SECTION_ID));
            this.setModuleID(data.getInt(Constants.DB_FIELD_MODULE_ID));
            this.setGroupID(data.getInt(Constants.DB_FIELD_GROUP_ID));
            this.setCategoryID(data.getInt(Constants.DB_FIELD_CATEGORY_ID));
        } catch (SQLException ex) {
            Logger.getLogger(Section.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public Section(JsonObject data) {
        this.setSectionID(data.get(Constants.PROPERTY_SECTION_ID).getAsInt());
        this.setModuleID(data.get(Constants.PROPERTY_MODULE_ID).getAsInt());
        this.setGroupID(data.get(Constants.PROPERTY_GROUP_ID).getAsInt());
        this.setCategoryID(data.get(Constants.PROPERTY_CATEGORY_ID).getAsInt());
    }

    public int getSectionID() {
```

```
        return this.sectionID;
    }

    public void setSectionID(int sectionID) {
        this.sectionID = sectionID;
    }

    public int getCategoryID() {
        return this.categoryID;
    }

    public void setCategoryID(int categoryID) {
        this.categoryID = categoryID;
    }

    public int getGroupID() {
        return this.groupID;
    }

    public void setGroupID(int groupID) {
        this.groupID = groupID;
    }

    public int getModuleID() {
        return this.moduleID;
    }

    public void setModuleID(int moduleID) {
        this.moduleID = moduleID;
    }

    public Module getModule() {
        return Socket.<Module>create(new Object[]{this.getModuleID()}, Module.SQL_FETCH, Module.class,
Dal.StatmentType.preparedStatment);
    }

    @DirectMethod
    public ResponseMessage saveData(JsonArray json) {
        return Socket.saveData(json.get(0).getAsJsonObject(), Constants.PROPERTY_SECTION_ID, this.getClass());
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getModuleID(), this.getGroupID(), this.getCategoryID()},
Section.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        return Dal.doMutation(new Object[]{this.getModuleID(), this.getGroupID(), this.getCategoryID(),
this.getSectionID()}, Section.SQL_UPDATE);
    }

    public static ArrayList<BusinessModule> grantedSections(AccessGroup group, int categoryID) {
        return (ArrayList<BusinessModule>) Socket.<BusinessModule>list(new Object[]{group.getGroupID(), categoryID},
Section.SQL_GRANTED_SECTION_LIST, Section.class, Dal.StatmentType.preparedStatment);
    }

    public static Section getAuthorisedSection(AccessGroup group, int moduleID) {
        return Socket.<Section>create(new Object[]{group.getGroupID(), moduleID},
Section.SQL_FETCH_AUTHORISED_SECTION, Section.class, Dal.StatmentType.preparedStatment);
    }

    @DirectMethod
    public static ResponseObject loadModule(int moduleID) {
        ResponseObject response = new ResponseObject();
        try {
            User user = User.getLoggedInUser();
            if (user != null) {
                AccessGroup group = user.getAccessGroup();
                Section section = Section.getAuthorisedSection(group, moduleID);
```

```
                if (section != null) {
                    SessionLog.getCurrentLog().add(section);
                    Module module = section.getModule();
                    if (module != null) {
                        response.setObject(new ResponseModule(module, group));
                        response.setSuccess(true);
                        response.setTitle(Section.MSG_SPAWNED_SECTION);
                        response.setMessage(module.getModuleTitle());
                    } else {
                        throw new Exception(Section.ERROR_MODULE_DOESNT_EXIST);
                    }
                } else {
                    throw new Exception(Section.ERROR_INVALID_ACCESS_PERMISSIONS);
                }
            } else {
                throw new Exception(Section.ERROR_USER_SESSION_EXPIRED);
            }
        } catch (Exception ex) {
            response.setSuccess(false);
            response.setMessage(ex.getMessage());
        }
        return response;
    }

    public static class ResponseModule {

        private Module module;
        private List<ResponseType> commands = new ArrayList<ResponseType>();

        public ResponseModule(Module module, AccessGroup group) {
            this.setModule(module);
            this.setCommands(group);
        }

        public Module getModule() {
            return this.module;
        }

        public void setModule(Module module) {
            this.module = module;
        }

        public List<ResponseType> getCommands() {
            return this.commands;
        }

        public void setCommands(AccessGroup group) {
            int[] allowed = group.getGroupCommands();
            Arrays.sort(allowed);
            for (int ordinal : this.getModule().getModuleCommands()) {
                if (Arrays.binarySearch(allowed, ordinal) >= 0) {
                    AccessGroup.CommandType type = AccessGroup.CommandType.values()[ordinal];
                    this.getCommands().add(new ResponseType(type.ordinal(), type.toString().toLowerCase()));
                }
            }
        }
    }
}
```

User.java

```
package bol.modules;

import dal.Dal;
import bol.BusinessModule;
import bol.Constants;
import bol.Socket;
import bol.util.Response.ResponseType;
import bol.util.Response.ResponseMessage;
import com.jgoodies.validation.util.ValidationUtils;
import djn.config.annotations.DirectFormPostMethod;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import djn.config.annotations.DirectMethod;
import djn.servlet.ssm.WebContextManager;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Map;
import javax.servlet.http.HttpSession;
import org.apache.commons.fileupload.FileItem;

public class User extends BusinessModule implements Serializable {

    public static final String SQL_UPDATE = "UPDATE system_user_accounts SET group_id = ?, employee_id = ?,
    user_name = ?, user_type = ? WHERE user_id = ?";
    public static final String SQL_INSERT = "INSERT INTO system_user_accounts (group_id, employee_id, user_name,
user_password, user_type) VALUES (?, ?, ?, SHA1(?), ?)";
    public static final String SQL_DELETE = "DELETE FROM system_user_accounts WHERE user_id = ? LIMIT 1";
    public static final String SQL_FETCH = "SELECT * FROM system_user_accounts WHERE user_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM system_user_accounts";
    public static final String SQL_TABLE = "system_user_accounts";
    private static final String ERR_PASSWORD_REQUIRED = "You must include a password when creating a new record";
    private static final String ERR_USERNAME_TAKEN = "User name is already taken";
    private static final String LOGOUT_MSG = "You have successfully logged out";
    private static final String LOGOUT_TITLE = "Goodbye";
    private static final String SQL_LOGIN_USER = "SELECT * FROM system_user_accounts WHERE user_name = ? AND
user_password = SHA1(?)";
    private static final String SQL_CHANGE_PASSWORD = "UPDATE system_user_accounts SET user_password = SHA1(?) WHERE
user_id = ?";
    private static final String SQL_CHECK_USERNAME = "SELECT user_id, user_name FROM system_user_accounts WHERE
user_name = ? AND user_id != ?";
    private static final String ERROR_INVALID_CREDENTIALS = "You have supplied invalid user credentials";
    private static final String ERROR_LOGIN_FAILED = "Login Failed";
    private static final String MSG_LOGIN_SUCCESS = "Login Success";
    private static final String MSG_NOW_AUTHENTICATED = "You are now authenticated";
    private static final String REGEX_PASSWORD = "^(?=.*[a-zA-Z])(?=.*[0-9])[a-zA-Z0-9_-]{8,15}$";
    private static final String ERR_PASSWORD_DOESNT_MATCH = "Confirmation password does not match";
    private static final String ERR_PASSWORD_STRENGTH = "Password must be 8-15 alphanumeric characters (can contain
hyphens and underscores)";

    public enum UserType {

        Normal, Admin
    };
    private int userID;
    private String userName;
    private String password;
    private String confirmationPassword;
    private int groupID;
    private int employeeID;
    private int userType;

    public User() {
    }
```

```
    public User(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setUserID(data.getInt(Constants.DB_FIELD_USER_ID));
            this.setGroupID(data.getInt(Constants.DB_FIELD_GROUP_ID));
            this.setEmployeeID(data.getInt(Constants.DB_FIELD_EMPLOYEE_ID));
            this.setUserName(data.getString(Constants.DB_FIELD_USER_NAME));
            this.setUserType(data.getInt(Constants.DB_FIELD_USER_TYPE));
        } catch (SQLException ex) {
            Logger.getLogger(User.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public int getUserID() {
        return this.userID;
    }

    public void setUserID(int userID) {
        this.userID = userID;
    }

    public String getUserName() {
        return this.userName;
    }

    public void setUserName(String userName) {
        if (this.isUserNameValid(userName)) {
            this.userName = userName;
        }
    }

    public String getPassword() {
        return this.password;
    }

    public void setPassword(String password) {
        if (this.isPasswordValid(password)) {
            this.password = password;
        }
    }

    public String getConfirmationPassword() {
        return confirmationPassword;
    }

    public void setConfirmationPassword(String confirmationPassword) {
        this.confirmationPassword = confirmationPassword;
    }

    public int getGroupID() {
        return this.groupID;
    }

    public void setGroupID(int groupID) {
        this.groupID = groupID;
    }

    public int getEmployeeID() {
        return this.employeeID;
    }

    public void setEmployeeID(int employeeID) {
        this.employeeID = employeeID;
    }

    public int getUserType() {
        return this.userType;
    }

    public void setUserType(int userType) {
```

```
        this.userType = userType;
    }

    public AccessGroup getAccessGroup() {
        return Socket.<AccessGroup>create(new Object[]{this.getGroupID()}, AccessGroup.SQL_FETCH, AccessGroup.class,
Dal.StatmentType.preparedStatment);
    }

    public boolean isAdminAccount() {
        return (User.UserType.values()[this.getUserType()] == User.UserType.Admin);
    }

    private boolean isUserNameValid(String userName) {
        if (this.getEntityState() != EntityState.Unchanged) {
            if (this.getValidation().validateRequired(Constants.PROPERTY_USER_NAME, userName) &&
this.getValidation().validateLength(Constants.PROPERTY_USER_NAME, userName, 5, 15)) {
                try {
                    ResultSet records = Dal.doQuery(new Object[]{userName, this.getUserID()}, SQL_CHECK_USERNAME,
Dal.StatmentType.preparedStatment);
                    if (Dal.hasRows(records)) {
                        this.getValidation().put(Constants.PROPERTY_USER_NAME, ERR_USERNAME_TAKEN);
                        return false;
                    }
                } catch (SQLException ex) {
                    Logger.getLogger(User.class.getName()).log(Level.SEVERE, null, ex);
                    return false;
                }
            } else {
                return false;
            }
        }
        return true;
    }

    private boolean isPasswordValid(String password) {
        if (this.getEntityState() != EntityState.Unchanged) {
            if (ValidationUtils.isNotBlank(password)) {
                if (password.equals(this.getConfirmationPassword())) {
                    if (!password.matches(REGEX_PASSWORD)) {
                        this.getValidation().put(Constants.PROPERTY_PASSWORD, ERR_PASSWORD_STRENGTH);
                        return false;
                    }
                } else {
                    this.getValidation().put(Constants.PROPERTY_PASSWORD, ERR_PASSWORD_DOESNT_MATCH);
                    return false;
                }
            } else {
                if (this.getEntityState() == EntityState.Added) {
                    this.getValidation().put(Constants.PROPERTY_PASSWORD, ERR_PASSWORD_REQUIRED);
                    return false;
                }
            }
        }
        return true;
    }

    @DirectMethod
    public static ResponseMessage doLogout() {
        ResponseMessage response = new ResponseMessage();
        WebContextManager.get().getSession().invalidate();
        response.setTitle(LOGOUT_TITLE);
        response.setMessage(LOGOUT_MSG);
        return response;
    }
```

```
    @DirectFormPostMethod
    public static ResponseMessage doLogin(Map<String, String> formParameters, Map<String, FileItem> fileFields)
throws IOException {
        User user = (User) Socket.<User>create(new Object[]{formParameters.get(Constants.PROPERTY_USER_NAME),
formParameters.get(Constants.PROPERTY_PASSWORD)}, User.SQL_LOGIN_USER, User.class,
Dal.StatmentType.preparedStatment);
        ResponseMessage response = new ResponseMessage();
        HttpSession session = WebContextManager.get().getSession();
        if (user != null) {
            response.setSuccess(true);
            response.setTitle(MSG_LOGIN_SUCCESS);
            response.setMessage(MSG_NOW_AUTHENTICATED);
            session.setAttribute(Constants.SESSION_USER_PROPERTY, user);
            session.setAttribute(Constants.SESSION_LOG_PROPERTY, SessionLog.create(user));
        } else {
            response.setSuccess(false);
            response.setTitle(ERROR_LOGIN_FAILED);
            response.setMessage(ERROR_INVALID_CREDENTIALS);
        }
        return response;
    }

    @DirectMethod
    public static User getLoggedInUser() {
        HttpSession session = WebContextManager.get().getSession();
        return (User)session.getAttribute(Constants.SESSION_USER_PROPERTY);
    }

    @DirectMethod
    public static ArrayList<ResponseType> userTypes() {
        return ResponseType.<UserType>create(UserType.values());
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getGroupID(), this.getEmployeeID(), this.getUserName(),
this.getPassword(), this.getUserType()}, User.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        PreparedStatement update = Dal.doMutation(new Object[]{this.getGroupID(), this.getEmployeeID(),
this.getUserName(), this.getUserType(), this.getUserID()}, User.SQL_UPDATE);
        if (!this.getPassword().isEmpty()) {
            Dal.doMutation(new Object[]{this.getPassword(), this.getUserID()}, User.SQL_CHANGE_PASSWORD);
        }
        return update;
    }
}
```

SessionLog.java

```
package bol.modules;

import bol.Constants;
import bol.Socket;
import bol.util.Response.ResponseList;
import dal.Dal;
import djn.config.annotations.DirectMethod;
import djn.servlet.ssm.WebContextManager;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.http.HttpSession;

public class SessionLog implements Serializable {

    private static final String SQL_INSERT = "INSERT INTO system_sessions (user_id, session_date) VALUES (?, ?)";
    private static final String SQL_TABLE = "system_sessions";
    private int sessionID;
    private int userID;
    private Date sessionDate;
    private ArrayList<Visit> history;

    public SessionLog() {
        this.setHistory(new ArrayList<Visit>());
    }

    public SessionLog(ResultSet data) {
        try {
            this.setSessionID(data.getInt(Constants.DB_FIELD_SESSION_ID));
            this.setUserID(data.getInt(Constants.DB_FIELD_USER_ID));
            this.setSessionDate(data.getTimestamp(Constants.DB_FIELD_SESSION_DATE));
            this.setHistory(Socket.<Visit>list(new Object[]{data.getInt(Constants.DB_FIELD_SESSION_ID)},
Visit.SQL_LIST, Visit.class, Dal.StatmentType.preparedStatment));
        } catch (SQLException ex) {
            Logger.getLogger(SessionLog.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public int getSessionID() {
        return this.sessionID;
    }

    public void setSessionID(int logID) {
        this.sessionID = logID;
    }

    public int getUserID() {
        return this.userID;
    }

    public void setUserID(int userID) {
        this.userID = userID;
    }

    public Date getSessionDate() {
        return this.sessionDate;
    }

    public void setSessionDate(Date date) {
        this.sessionDate = date;
    }
```

```
    public ArrayList<Visit> getHistory() {
        return this.history;
    }

    public void setHistory(ArrayList<Visit> history) {
        this.history = history;
    }

    public void save() {
        try {
            PreparedStatement statement = Dal.doMutation(new Object[]{this.getUserID(), this.getSessionDate()},
SessionLog.SQL_INSERT);
            ResultSet result = statement.getGeneratedKeys();
            if (result != null && result.next()) {
                this.setSessionID(result.getInt(1));
            }
        } catch (SQLException ex) {
            Logger.getLogger(SessionLog.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void add(Section section) throws SQLException {
        Visit visit = new Visit();
        this.getHistory().add(visit);
        visit.setSessionID(this.getSessionID());
        visit.setSectionID(section.getSectionID());
        visit.setVisitTime(Calendar.getInstance().getTime());
        visit.save();
    }

    @DirectMethod
    public static SessionLog getCurrentLog() {
        HttpSession session = WebContextManager.get().getSession();
        return (SessionLog) session.getAttribute(Constants.SESSION_LOG_PROPERTY);
    }

    public static SessionLog create(User user) {
        SessionLog session = new SessionLog();
        session.setUserID(user.getUserID());
        session.setSessionDate(Calendar.getInstance().getTime());
        session.save();
        return session;
    }

    @DirectMethod
    public static ResponseList list(int start, int limit, String sort, String direction, int userID) {
        return Socket.<SessionLog>pagination(SQL_TABLE, SessionLog.class, start, limit, sort, direction, " WHERE
user_id = '" + userID + "'");
    }

    public static class Visit implements Serializable {

        private static final String SQL_INSERT = "INSERT INTO system_session_history (session_id, section_id,
visit_time) VALUES (?, ?, ?)";
        private static final String SQL_LIST = "SELECT h.visit_id, h.session_id, h.section_id, s.module_id,
m.module_title, h.visit_time FROM system_session_history As h INNER JOIN system_sections As s ON h.section_id =
s.section_id INNER JOIN system_modules As m ON s.module_id = m.module_id WHERE h.session_id = ? ORDER BY
h.visit_time DESC";
        private int visitID;
        private int sessionID;
        private int sectionID;
        private int moduleID;
        private String moduleTitle;
        private Date visitTime;

        public Visit() {
        }

        public Visit(ResultSet data) {
            try {
```

```
                this.setVisitID(data.getInt(Constants.DB_FIELD_VISIT_ID));
                this.setSessionID(data.getInt(Constants.DB_FIELD_SESSION_ID));
                this.setSectionID(data.getInt(Constants.DB_FIELD_SECTION_ID));
                this.setModuleID(data.getInt(Constants.DB_FIELD_MODULE_ID));
                this.setModuleTitle(data.getString(Constants.DB_FIELD_MODULE_TITLE));
                this.setVisitTime(data.getTimestamp(Constants.DB_FIELD_VISIT_TIME));
            } catch (SQLException ex) {
                Logger.getLogger(Visit.class.getName()).log(Level.SEVERE, null, ex);
            }
        }

        public int getVisitID() {
            return this.visitID;
        }

        public void setVisitID(int visitID) {
            this.visitID = visitID;
        }

        public int getSessionID() {
            return this.sessionID;
        }

        public void setSessionID(int sessionID) {
            this.sessionID = sessionID;
        }

        public int getSectionID() {
            return sectionID;
        }

        public void setSectionID(int sectionID) {
            this.sectionID = sectionID;
        }

        public int getModuleID() {
            return this.moduleID;
        }

        public void setModuleID(int moduleID) {
            this.moduleID = moduleID;
        }

        public String getModuleTitle() {
            return moduleTitle;
        }

        public void setModuleTitle(String moduleTitle) {
            this.moduleTitle = moduleTitle;
        }

        public Date getVisitTime() {
            return this.visitTime;
        }

        public void setVisitTime(Date visitTime) {
            this.visitTime = visitTime;
        }

        public void save() throws SQLException {
            Dal.doMutation(new Object[]{this.getSessionID(), this.getSectionID(), this.getVisitTime()},
Visit.SQL_INSERT);
        }
    }
}
```

Category.java

```
package bol.modules;

import dal.Dal;
import bol.BusinessModule;
import bol.Constants;
import bol.Socket;
import bol.util.Response.ResponseMenuItem;
import bol.util.Response.ResponseNode;
import bol.util.Response.ResponseType;
import djn.config.annotations.DirectMethod;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Category extends BusinessModule implements Serializable {

    public static final String SQL_DELETE = "DELETE FROM system_categories WHERE category_id = ? LIMIT 1";
    public static final String SQL_FETCH = "SELECT * FROM system_categories WHERE category_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM system_categories";
    public static final String SQL_TABLE = "system_categories";
    private static final String SQL_INSERT = "INSERT INTO system_categories (category_parent, category_title,
category_type, category_icon) VALUES (?, ?, ?, ?)";
    private static final String SQL_UPDATE = "UPDATE system_categories SET category_parent = ?, category_title = ?,
category_type = ?, category_icon = ? WHERE category_id = ?";
    private static final String SQL_LIST_PARENT_TYPE = "SELECT * FROM system_categories WHERE category_parent = ?
AND category_type = ? ORDER BY category_title ASC";
    private static final String SQL_LIST_PARENT = "SELECT * FROM system_categories WHERE category_parent = ? ORDER
BY category_title ASC";
    private static final String DEFAULT_CATEGORY_ICON = "folder.png";
    private static final String FN_CREATE_SUB_MENU = "TricertAssist.navigation.createMenu";
    private static final String FN_LOAD_MODULE = "TricertAssist.loadModule";

    public enum CategoryType {

        Navigation, Administration
    };
    private int categoryID;
    private int categoryParent;
    private String categoryTitle;
    private String categoryIcon;
    private int categoryType;

    public Category() {
    }

    public Category(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setCategoryID(data.getInt(Constants.DB_FIELD_CATEGORY_ID));
            this.setCategoryParent(data.getInt(Constants.DB_FIELD_CATEGORY_PARENT));
            this.setCategoryTitle(data.getString(Constants.DB_FIELD_CATEGORY_TITLE));
            this.setCategoryType(data.getInt(Constants.DB_FIELD_CATEGORY_TYPE));
            this.setCategoryIcon(data.getString(Constants.DB_FIELD_CATEGORY_ICON));
        } catch (SQLException ex) {
            Logger.getLogger(this.getClass().getName()).log(Level.SEVERE, null, ex);
        }
    }

    public int getCategoryID() {
        return this.categoryID;
    }
```

```
    public void setCategoryID(int categoryID) {
        this.categoryID = categoryID;
    }


    public int getCategoryParent() {
        return this.categoryParent;
    }

    public void setCategoryParent(int categoryParent) {
        this.categoryParent = categoryParent;
    }

    public String getCategoryTitle() {
        return this.categoryTitle;
    }

    public void setCategoryTitle(String categoryTitle) {
        if (this.getEntityState() == EntityState.Unchanged ||
                this.getValidation().validateLength(Constants.PROPERTY_CATEGORY_TITLE, categoryTitle, 5, 20)) {
            this.categoryTitle = categoryTitle;
        }
    }

    public String getCategoryIcon() {
        return this.categoryIcon;
    }

    public String getIconPath() {
        return Constants.ICON_IMG_PATH + ((!this.getCategoryIcon().equals(Constants.EMPTY_STRING)) ?
this.getCategoryIcon() + Constants.ICON_EXT : Category.DEFAULT_CATEGORY_ICON);
    }

    public void setCategoryIcon(String categoryIcon) {
        this.categoryIcon = categoryIcon;
    }

    public int getCategoryType() {
        return this.categoryType;
    }

    public void setCategoryType(int categoryType) {
        this.categoryType = categoryType;
    }

    public static ArrayList<BusinessModule> list(int parent, CategoryType type) throws SQLException {
        return (ArrayList<BusinessModule>) Socket.<BusinessModule>list(new Object[]{parent, type.ordinal()},
Category.SQL_LIST_PARENT_TYPE, Category.class, Dal.StatmentType.preparedStatment);
    }

    @DirectMethod
    public static ArrayList<ResponseMenuItem> getMenu(int parent) {
        ArrayList<ResponseMenuItem> menu = new ArrayList<ResponseMenuItem>();
        try {
            ArrayList<BusinessModule> items = new ArrayList<BusinessModule>();
            User user = User.getLoggedInUser();
            if (user != null) {
                items.addAll(Category.list(parent, CategoryType.Navigation));
                if (user.isAdminAccount()) {
                    items.addAll(Category.list(parent, CategoryType.Administration));
                }
                items.addAll(Section.grantedSections(user.getAccessGroup(), parent));
                for (BusinessModule object : items) {
                    ResponseMenuItem item = new ResponseMenuItem();
                    if (object instanceof Category) {
                        Category category = (Category) object;
                        item.setText(category.getCategoryTitle());
                        item.setIcon(category.getIconPath());
                        item.setMenu(Category.FN_CREATE_SUB_MENU + "(" + category.getCategoryID() + ")");
                    } else if (object instanceof Section) {
                        Module module = ((Section) object).getModule();
```

```
                        item.setText(module.getModuleTitle());
                        item.setHandler("function () { " + Category.FN_LOAD_MODULE + "(" + module.getModuleID() +
"); }");
                        item.setIcon(module.getIconPath());
                    }
                    item.setScale(ResponseMenuItem.SCALE);
                    item.setCls(ResponseMenuItem.CLS);

                    menu.add(item);
                }
            }
        } catch (Exception ex) {
            Logger.getLogger(Category.class.getName()).log(Level.SEVERE, null, ex);
        }
        return menu;
    }

    @DirectMethod
    public static ArrayList<ResponseNode> getTree(String parent) throws SQLException {
        ArrayList<ResponseNode> tree = new ArrayList<ResponseNode>();
        for (Category c : Socket.<Category>list(new Object[]{Integer.valueOf(parent)}, Category.SQL_LIST_PARENT,
Category.class, Dal.StatmentType.preparedStatment)) {
            tree.add(new ResponseNode(Integer.toString(c.getCategoryID()), c.getCategoryTitle(),
(Dal.getCount(Category.SQL_TABLE, " WHERE category_parent = " + c.getCategoryID()) == 0)));
        }
        return tree;
    }

    @DirectMethod
    public static ArrayList<ResponseType> categoryTypes() {
        return ResponseType.<CategoryType>create(CategoryType.values());
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getCategoryParent(), this.getCategoryTitle(),
this.getCategoryType(), this.getCategoryIcon()}, Category.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        return Dal.doMutation(new Object[]{this.getCategoryParent(), this.getCategoryTitle(),
this.getCategoryType(), this.getCategoryIcon(), this.getCategoryID()}, Category.SQL_UPDATE);
    }
}
```

AccessGroup.java

```
package bol.modules;

import bol.BusinessModule;
import bol.Constants;
import bol.util.Response.ResponseType;
import com.google.gson.Gson;
import dal.Dal;
import djn.config.annotations.DirectMethod;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.commons.fileupload.FileItem;

public class AccessGroup extends BusinessModule implements Serializable {

    public static final String SQL_UPDATE = "UPDATE system_access_groups SET group_name = ?, group_description  = ?,
group_commands = ? WHERE group_id = ?";
    public static final String SQL_INSERT = "INSERT INTO system_access_groups (group_name, group_description,
group_commands) VALUES (?, ?, ?)";
    public static final String SQL_DELETE = "DELETE FROM system_access_groups WHERE group_id = ? LIMIT 1";
    public static final String SQL_FETCH = "SELECT * FROM system_access_groups WHERE group_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM system_access_groups ORDER BY group_name ASC";
    public static final String SQL_TABLE = "system_access_groups";

    public enum CommandType {

        Add, Save, Delete
    };
    private int groupID;
    private String groupName;
    private String groupDescription;
    private int[] groupCommands;

    public AccessGroup() {
    }

    public AccessGroup(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setGroupID(data.getInt(Constants.DB_FIELD_GROUP_ID));
            this.setGroupName(data.getString(Constants.DB_FIELD_GROUP_NAME));
            this.setGroupDescription(data.getString(Constants.DB_FIELD_GROUP_DESCRIPTION));
            this.setGroupCommands(data.getString(Constants.DB_FIELD_GROUP_COMMANDS));
        } catch (SQLException ex) {
            Logger.getLogger(AccessGroup.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public int getGroupID() {
        return this.groupID;
    }

    public void setGroupID(int groupID) {
        this.groupID = groupID;
    }

    public String getGroupName() {
        return this.groupName;
    }
```

```
    public void setGroupName(String groupName) {
        this.groupName = groupName;
    }

    public String getGroupDescription() {
        return this.groupDescription;
    }

    public void setGroupDescription(String groupDescription) {
        this.groupDescription = groupDescription;
    }

    public int[] getGroupCommands() {
        return this.groupCommands;
    }

    public void setGroupCommands(String groupCommands) {
        Gson gson = new Gson();
        this.groupCommands = gson.fromJson(groupCommands, int[].class);
    }

    @DirectMethod
    public static List<ResponseType> commandTypes() {
        return ResponseType.<CommandType>create(CommandType.values());
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        Gson gson = new Gson();
        return Dal.doMutation(new Object[]{this.getGroupName(), this.getGroupDescription(),
gson.toJson(this.getGroupCommands(), int[].class)}, AccessGroup.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        Gson gson = new Gson();
        return Dal.doMutation(new Object[]{this.getGroupName(), this.getGroupDescription(),
gson.toJson(this.getGroupCommands(), int[].class), this.getGroupID()}, AccessGroup.SQL_UPDATE);
    }

    @Override
    public void save(Map<String, String> formParameters, Map<String, FileItem> fileFields) throws SQLException {
        this.setGroupCommands(formParameters.get(Constants.PROPERTY_GROUP_COMMANDS));
        switch (this.getEntityState()) {
            case Added:
                this.insert();
                break;
            case Modified:
                this.update();
                break;
        }
    }
}
```

Department.java

```
package bol.modules;

import bol.BusinessModule;
import bol.BusinessModule.EntityState;
import bol.Constants;
import dal.Dal;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Department extends BusinessModule implements Serializable {

    public static final String SQL_FETCH = "SELECT * FROM employee_departments WHERE department_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM employee_departments";
    public static final String SQL_DELETE = "DELETE FROM employee_departments WHERE department_id = ? LIMIT 1";
    public static final String SQL_TABLE = "employee_departments";
    private static final String SQL_INSERT = "INSERT INTO employee_departments (department_name, street_address,
suburb, postcode, state, reception_phone) VALUES (?, ?, ?, ?, ?, ?)";
    private static final String SQL_UPDATE = "UPDATE employee_departments SET department_name = ?, street_address =
?, suburb = ?, postcode = ?, state = ?, reception_phone = ? WHERE department_id = ?";
    private int departmentID;
    private String departmentName;
    private String streetAddress;
    private String suburb;
    private String postcode;
    private String state;
    private String receptionPhone;

    public Department() {
    }

    public Department(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setDepartmentID(data.getInt(Constants.DB_FIELD_DEPARTMENT_ID));
            this.setDepartmentName(data.getString(Constants.DB_FIELD_DEPARTMENT_NAME));
            this.setStreetAddress(data.getString(Constants.DB_FIELD_STREET_ADDRESS));
            this.setSuburb(data.getString(Constants.DB_FIELD_SUBURB));
            this.setPostcode(data.getString(Constants.DB_FIELD_POSTCODE));
            this.setState(data.getString(Constants.DB_FIELD_STATE));
            this.setReceptionPhone(data.getString(Constants.DB_FIELD_RECEPTION_PHONE));
        } catch (SQLException ex) {
            Logger.getLogger(Department.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public int getDepartmentID() {
        return this.departmentID;
    }

    public void setDepartmentID(int departmentID) {
        this.departmentID = departmentID;
    }

    public String getDepartmentName() {
        return this.departmentName;
    }

    public void setDepartmentName(String departmentName) {
        this.departmentName = departmentName;
    }

    public String getStreetAddress() {
```

```
        return this.streetAddress;
    }

    public void setStreetAddress(String streetAddress) {
        this.streetAddress = streetAddress;
    }

    public String getSuburb() {
        return this.suburb;
    }

    public void setSuburb(String suburb) {
        this.suburb = suburb;
    }

    public String getPostcode() {
        return this.postcode;
    }

    public void setPostcode(String postcode) {
        this.postcode = postcode;
    }

    public String getState() {
        return this.state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getReceptionPhone() {
        return this.receptionPhone;
    }

    public void setReceptionPhone(String receptionPhone) {
        this.receptionPhone = receptionPhone;
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getDepartmentName(), this.getStreetAddress(), this.getSuburb(),
this.getPostcode(), this.getState(), this.getReceptionPhone()}, Department.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        return Dal.doMutation(new Object[]{this.getDepartmentName(), this.getStreetAddress(), this.getSuburb(),
this.getPostcode(), this.getState(), this.getReceptionPhone(), this.getDepartmentID()}, Department.SQL_UPDATE);
    }
}
```

Employee.java

```
package bol.modules;

import bol.*;
import bol.util.Response.ResponseType;
import dal.Dal;
import djn.config.annotations.DirectMethod;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Employee extends BusinessModule implements Serializable {

    public static final String SQL_FETCH = "SELECT * FROM employees WHERE employee_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM employees";
    public static final String SQL_DELETE = "DELETE FROM employees WHERE employee_id = ? LIMIT 1";
    public static final String SQL_TABLE = "employees";
    private static final String SQL_UPDATE = "UPDATE employees SET employee_section_id = ?, department_id = ?,
supervisor_id = ?, employee_type = ?, first_name = ?, last_name = ?, email_address = ?, street_address = ?, suburb =
?, postcode = ?, state = ? WHERE employee_id = ?";
    private static final String SQL_INSERT = "INSERT INTO employees (employee_section_id, department_id,
supervisor_id, employee_type, first_name, last_name, email_address, street_address, suburb, postcode, state) VALUES
(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

    public enum EmployeeType {

        FullTime, PartTime, Casual
    };

    public enum State {

        ACT, NSW, NT, QLD, SA, TAS, VIC, WA
    };
    private int employeeID;
    private int employeeSectionID;
    private int departmentID;
    private int supervisorID;
    private int employeeType;
    private String firstName;
    private String lastName;
    private String emailAddress;
    private String streetAddress;
    private String suburb;
    private String postcode;
    private String state;

    public Employee() {
    }

    public Employee(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setEmployeeID(data.getInt(Constants.DB_FIELD_EMPLOYEE_ID));
            this.setEmployeeSectionID(data.getInt(Constants.DB_FIELD_EMPLOYEE_SECTION_ID));
            this.setDepartmentID(data.getInt(Constants.DB_FIELD_DEPARTMENT_ID));
            this.setSupervisorID(data.getInt(Constants.DB_FIELD_SUPERVISOR_ID));
            this.setEmployeeType(data.getInt(Constants.DB_FIELD_EMPLOYEE_TYPE));
            this.setFirstName(data.getString(Constants.DB_FIELD_FIRST_NAME));
            this.setLastName(data.getString(Constants.DB_FIELD_LAST_NAME));
            this.setEmailAddress(data.getString(Constants.DB_FIELD_EMAIL_ADDRESS));
            this.setStreetAddress(data.getString(Constants.DB_FIELD_STREET_ADDRESS));
            this.setSuburb(data.getString(Constants.DB_FIELD_SUBURB));
            this.setPostcode(data.getString(Constants.DB_FIELD_POSTCODE));
            this.setState(data.getString(Constants.DB_FIELD_STATE));
```

```
        } catch (SQLException ex) {
            Logger.getLogger(Employee.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public int getEmployeeID() {
        return this.employeeID;
    }

    public void setEmployeeID(int employeeID) {
        this.employeeID = employeeID;
    }

    public int getEmployeeSectionID() {
        return employeeSectionID;
    }

    public void setEmployeeSectionID(int employeeSectionID) {
        this.employeeSectionID = employeeSectionID;
    }

    public int getDepartmentID() {
        return departmentID;
    }

    public void setDepartmentID(int departmentID) {
        this.departmentID = departmentID;
    }

    public int getSupervisorID() {
        return this.supervisorID;
    }

    public void setSupervisorID(int supervisorID) {
        this.supervisorID = supervisorID;
    }

    public int getEmployeeType() {
        return employeeType;
    }

    public void setEmployeeType(int employeeType) {
        this.employeeType = employeeType;
    }

    public String getFirstName() {
        return this.firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return this.lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmailAddress() {
        return this.emailAddress;
    }

    public void setEmailAddress(String emailAddress) {
        this.emailAddress = emailAddress;
    }

    public String getStreetAddress() {
        return this.streetAddress;
```

```
    }

    public void setStreetAddress(String streetAddress) {
        this.streetAddress = streetAddress;
    }

    public String getSuburb() {
        return this.suburb;
    }

    public void setSuburb(String suburb) {
        this.suburb = suburb;
    }

    public String getPostcode() {
        return postcode;
    }

    public void setPostcode(String postcode) {
        this.postcode = postcode;
    }

    public String getState() {
        return this.state;
    }

    public void setState(String state) {
        this.state = state;
    }

    @DirectMethod
    public static ArrayList<ResponseType> stateTypes() {
        return ResponseType.<State>create(State.values());
    }

    @DirectMethod
    public static ArrayList<ResponseType> employeeTypes() {
        return ResponseType.<EmployeeType>create(EmployeeType.values());
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getEmployeeSectionID(), this.getDepartmentID(),
this.getSupervisorID(), this.getEmployeeType(), this.getFirstName(), this.getLastName(), this.getEmailAddress(),
this.getStreetAddress(), this.getSuburb(), this.getPostcode(), this.getState()}, Employee.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        return Dal.doMutation(new Object[]{this.getEmployeeSectionID(), this.getDepartmentID(),
this.getSupervisorID(), this.getEmployeeType(), this.getFirstName(), this.getLastName(), this.getEmailAddress(),
this.getStreetAddress(), this.getSuburb(), this.getPostcode(), this.getState(), this.getEmployeeID()},
Employee.SQL_UPDATE);
    }
}
```

EmployeeLicense.java

```
package bol.modules;

import bol.BusinessModule;
import bol.Constants;
import bol.Socket;
import bol.util.Response.ResponseMessage;
import bol.util.Response.ResponseType;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import dal.Dal;
import djn.config.annotations.DirectMethod;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class EmployeeLicense extends BusinessModule implements Serializable {

    public static final String SQL_FETCH = "SELECT * FROM employee_licenses WHERE license_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM employee_licenses";
    public static final String SQL_DELETE = "DELETE FROM employee_licenses WHERE license_id = ? LIMIT 1";
    public static final String SQL_TABLE = "employee_licenses";
    private static final String SQL_UPDATE = "UPDATE employee_licenses SET employee_id = ?, license_type = ?,
license_number = ?, license_class = ? WHERE license_id = ?";
    private static final String SQL_INSERT = "INSERT INTO employee_licenses (employee_id, license_type,
license_number, license_class) VALUES (?, ?, ?, ?)";

    public enum LicenseType {Car, Forklift, Truck}
    private int licenseID;
    private int employeeID;
    private int licenseType;
    private String licenseNumber;
    private String licenseClass;

    public EmployeeLicense() {
    }

    public EmployeeLicense(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setLicenseID(data.getInt(Constants.DB_FIELD_LICENSE_ID));
            this.setEmployeeID(data.getInt(Constants.DB_FIELD_EMPLOYEE_ID));
            this.setLicenseType(data.getInt(Constants.DB_FIELD_LICENSE_TYPE));
            this.setLicenseNumber(data.getString(Constants.DB_FIELD_LICENSE_NUMBER));
            this.setLicenseClass(data.getString(Constants.DB_FIELD_LICENSE_CLASS));
        } catch (SQLException ex) {
            Logger.getLogger(EmployeeLicense.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public EmployeeLicense(JsonObject data) {
        this.setLicenseID(data.get(Constants.PROPERTY_LICENSE_ID).getAsInt());
        this.setEmployeeID(data.get(Constants.PROPERTY_EMPLOYEE_ID).getAsInt());
        this.setLicenseType(data.get(Constants.PROPERTY_LICENSE_TYPE).getAsInt());
        this.setLicenseNumber(data.get(Constants.PROPERTY_LICENSE_NUMBER).getAsString());
        this.setLicenseClass(data.get(Constants.PROPERTY_LICENSE_CLASS).getAsString());
    }

    public int getLicenseID() {
        return this.licenseID;
    }
```

```
    public void setLicenseID(int licenseID) {
        this.licenseID = licenseID;
    }

    public int getEmployeeID() {
        return this.employeeID;
    }

    public void setEmployeeID(int employeeID) {
        this.employeeID = employeeID;
    }

    public int getLicenseType() {
        return this.licenseType;
    }

    public void setLicenseType(int licenseType) {
        this.licenseType = licenseType;
    }

    public String getLicenseNumber() {
        return this.licenseNumber;
    }

    public void setLicenseNumber(String licenseNumber) {
        this.licenseNumber = licenseNumber;
    }

    public String getLicenseClass() {
        return this.licenseClass;
    }

    public void setLicenseClass(String licenseClass) {
        this.licenseClass = licenseClass;
    }

    @DirectMethod
    public static ArrayList<ResponseType> licenseTypes() {
        return ResponseType.<LicenseType>create(LicenseType.values());
    }

    @DirectMethod
    public ResponseMessage saveData(JsonArray json) {
        return Socket.saveData(json.get(0).getAsJsonObject(), Constants.PROPERTY_LICENSE_ID, this.getClass());
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getEmployeeID(), this.getLicenseType(), this.getLicenseNumber(),
this.getLicenseClass()}, EmployeeLicense.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        return Dal.doMutation(new Object[]{this.getEmployeeID(), this.getLicenseType(), this.getLicenseNumber(),
this.getLicenseClass(), this.getLicenseID()}, EmployeeLicense.SQL_UPDATE);
    }
}
```

EmployeePhoneNumber.java

```
package bol.modules;

import bol.BusinessModule;
import bol.Constants;
import bol.Socket;
import bol.util.Response.ResponseMessage;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import dal.Dal;
import djn.config.annotations.DirectMethod;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class EmployeePhoneNumber extends BusinessModule implements Serializable {

    public static final String SQL_FETCH = "SELECT * FROM employee_phone_registry WHERE phone_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM employee_phone_registry";
    public static final String SQL_DELETE = "DELETE FROM employee_phone_registry WHERE phone_id = ? LIMIT 1";
    public static final String SQL_TABLE = "employee_phone_registry";
    private static final String SQL_UPDATE = "UPDATE employee_phone_registry SET employee_id = ?,
contact_information = ?, contact_number = ? WHERE phone_id = ?";
    private static final String SQL_INSERT = "INSERT INTO employee_phone_registry (employee_id, contact_information,
contact_number) VALUES (?, ?, ?)";
    private int phoneID;
    private int employeeID;
    private String contactInformation;
    private String contactNumber;

    public EmployeePhoneNumber() {
    }

    public EmployeePhoneNumber(ResultSet data) {
        try {
            this.setPhoneID(data.getInt(Constants.DB_FIELD_PHONE_ID));
            this.setEmployeeID(data.getInt(Constants.DB_FIELD_EMPLOYEE_ID));
            this.setContactInformation(data.getString(Constants.DB_FIELD_CONTACT_INFORMATION));
            this.setContactNumber(data.getString(Constants.DB_FIELD_CONTACT_NUMBER));
        } catch (SQLException ex) {
            Logger.getLogger(EmployeePhoneNumber.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public EmployeePhoneNumber(JsonObject data) {
        this.setPhoneID(data.get(Constants.PROPERTY_PHONE_ID).getAsInt());
        this.setEmployeeID(data.get(Constants.PROPERTY_EMPLOYEE_ID).getAsInt());
        this.setContactInformation(data.get(Constants.PROPERTY_CONTACT_INFORMATION).getAsString());
        this.setContactNumber(data.get(Constants.PROPERTY_CONTACT_NUMBER).getAsString());
    }

    public int getPhoneID() {
        return this.phoneID;
    }

    public void setPhoneID(int phoneID) {
        this.phoneID = phoneID;
    }

    public int getEmployeeID() {
        return this.employeeID;
    }
```

```
    public void setEmployeeID(int employeeID) {
        this.employeeID = employeeID;
    }

    public String getContactInformation() {
        return this.contactInformation;
    }

    public void setContactInformation(String contactInformation) {
        this.contactInformation = contactInformation;
    }

    public String getContactNumber() {
        return this.contactNumber;
    }

    public void setContactNumber(String contactNumber) {
        this.contactNumber = contactNumber;
    }

    @DirectMethod
    public ResponseMessage saveData(JsonArray json) {
        return Socket.saveData(json.get(0).getAsJsonObject(), Constants.PROPERTY_PHONE_ID, this.getClass());
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getEmployeeID(), this.getContactInformation(),
this.getContactNumber()}, EmployeePhoneNumber.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        return Dal.doMutation(new Object[]{this.getEmployeeID(), this.getContactInformation(),
this.getContactNumber(), this.getPhoneID()}, EmployeePhoneNumber.SQL_UPDATE);
    }
}
```

EmployeeQualification.java

```
package bol.modules;
import bol.BusinessModule;
import bol.Constants;
import bol.Socket;
import bol.util.Response.ResponseMessage;
import bol.util.Response.ResponseType;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import dal.Dal;
import djn.config.annotations.DirectMethod;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;

public class EmployeeQualification extends BusinessModule implements Serializable {

    public static final String SQL_FETCH = "SELECT * FROM employee_qualifications WHERE qualification_id = ? LIMIT
1";
    public static final String SQL_LIST = "SELECT * FROM employee_qualifications";
    public static final String SQL_DELETE = "DELETE FROM employee_qualifications WHERE qualification_id = ? LIMIT
1";
    public static final String SQL_TABLE = "employee_qualifications";
    private static final String SQL_UPDATE = "UPDATE employee_qualifications SET employee_id = ?,
qualification_title = ?, qualification_level = ?, date_obtained = ?, institute = ? WHERE qualification_id = ?";
    private static final String SQL_INSERT = "INSERT INTO employee_qualifications (employee_id, qualification_title,
qualification_level, date_obtained, institute) VALUES (?, ?, ?, ?, ?)";

    public enum QualificationLevel {

        Certification, Trade, Diploma, Degree
    }
    private int qualificationID;
    private int employeeID;
    private int qualificationLevel;
    private Date dateObtained;
    private String qualificationTitle;
    private String institute;

    public EmployeeQualification() {
    }

    public EmployeeQualification(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setQualificationID(data.getInt(Constants.DB_FIELD_QUALIFICATION_ID));
            this.setEmployeeID(data.getInt(Constants.DB_FIELD_EMPLOYEE_ID));
            this.setQualificationLevel(data.getInt(Constants.DB_FIELD_QUALIFICATION_LEVEL));
            this.setDateObtained(data.getDate(Constants.DB_FIELD_DATE_OBTAINED));
            this.setQualificationTitle(data.getString(Constants.DB_FIELD_QUALIFICATION_TITLE));
            this.setInstitute(data.getString(Constants.DB_FIELD_INSTITUTE));
        } catch (SQLException ex) {
            Logger.getLogger(EmployeeQualification.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public EmployeeQualification(JsonObject data) {
        try {
            this.setQualificationID(data.get(Constants.PROPERTY_QUALIFICATION_ID).getAsInt());
            this.setEmployeeID(data.get(Constants.PROPERTY_EMPLOYEE_ID).getAsInt());
            this.setQualificationLevel(data.get(Constants.PROPERTY_QUALIFICATION_LEVEL).getAsInt());
```

```
            this.setDateObtained(new SimpleDateFormat("yyyy-MM-
dd").parse(data.get(Constants.PROPERTY_DATE_OBTAINED).getAsString()));
            this.setQualificationTitle(data.get(Constants.PROPERTY_QUALIFICATION_TITLE).getAsString());
            this.setInstitute(data.get(Constants.PROPERTY_INSTITUTE).getAsString());
        } catch (Exception ex) {
            Logger.getLogger(EmployeeQualification.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public int getQualificationID() {
        return this.qualificationID;
    }

    public void setQualificationID(int qualificationID) {
        this.qualificationID = qualificationID;
    }

    public int getEmployeeID() {
        return employeeID;
    }

    public void setEmployeeID(int employeeID) {
        this.employeeID = employeeID;
    }

    public String getQualificationTitle() {
        return this.qualificationTitle;
    }

    public void setQualificationTitle(String qualificationTitle) {
        this.qualificationTitle = qualificationTitle;
    }

    public int getQualificationLevel() {
        return this.qualificationLevel;
    }

    public void setQualificationLevel(int qualificationLevel) {
        this.qualificationLevel = qualificationLevel;
    }

    public Date getDateObtained() {
        return this.dateObtained;
    }

    public void setDateObtained(Date dateObtained) {
        this.dateObtained = dateObtained;
    }

    public String getInstitute() {
        return this.institute;
    }

    public void setInstitute(String institute) {
        this.institute = institute;
    }

    @DirectMethod
    public static ArrayList<ResponseType> qualificationLevels() {
        return ResponseType.<QualificationLevel>create(QualificationLevel.values());
    }

    @DirectMethod
    public ResponseMessage saveData(JsonArray json) {
        return Socket.saveData(json.get(0).getAsJsonObject(), Constants.PROPERTY_QUALIFICATION_ID, this.getClass());
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getEmployeeID(), this.getQualificationTitle(),
this.getQualificationLevel(), this.getDateObtained(), this.getInstitute()}, EmployeeQualification.SQL_INSERT);
```

```
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        return Dal.doMutation(new Object[]{this.getEmployeeID(), this.getQualificationTitle(),
this.getQualificationLevel(), this.getDateObtained(), this.getInstitute(), this.getQualificationID()},
EmployeeQualification.SQL_UPDATE);
    }
}
```

EmployeeSection.java

```
package bol.modules;

import bol.BusinessModule;
import bol.Constants;
import bol.Socket;
import bol.util.Response.ResponseMessage;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import dal.Dal;
import djn.config.annotations.DirectMethod;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class EmployeeSection extends BusinessModule implements Serializable {

    public static final String SQL_FETCH = "SELECT * FROM employee_sections WHERE employee_section_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM employee_sections";
    public static final String SQL_DELETE = "DELETE FROM employee_sections WHERE employee_section_id = ? LIMIT 1";
    public static final String SQL_TABLE = "employee_sections";
    private static final String SQL_INSERT = "INSERT INTO employee_sections (section_title, section_description)
VALUES (?, ?)";
    private static final String SQL_UPDATE = "UPDATE employee_sections SET section_title = ?, section_description =
? WHERE employee_section_id = ?";
    private int employeeSectionID;
    private String sectionTitle;
    private String sectionDescription;

    public EmployeeSection() {
    }

    public EmployeeSection(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setEmployeeSectionID(data.getInt(Constants.DB_FIELD_EMPLOYEE_SECTION_ID));
            this.setSectionTitle(data.getString(Constants.DB_FIELD_SECTION_TITLE));
            this.setSectionDescription(data.getString(Constants.DB_FIELD_SECTION_DESCRIPTION));
        } catch (SQLException ex) {
            Logger.getLogger(EmployeeSection.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public EmployeeSection(JsonObject data) {
        this.setEmployeeSectionID(data.get(Constants.PROPERTY_EMPLOYEE_SECTION_ID).getAsInt());
        this.setSectionTitle(data.get(Constants.PROPERTY_SECTION_TITLE).getAsString());
        this.setSectionDescription(data.get(Constants.PROPERTY_SECTION_DESCRIPTION).getAsString());
    }

    public int getEmployeeSectionID() {
        return this.employeeSectionID;
    }
```

```
    public void setEmployeeSectionID(int employeeSectionID) {
        this.employeeSectionID = employeeSectionID;
    }

    public String getSectionTitle() {
        return this.sectionTitle;
    }

    public void setSectionTitle(String sectionTitle) {
        this.sectionTitle = sectionTitle;
    }

    public String getSectionDescription() {
        return this.sectionDescription;
    }

    public void setSectionDescription(String sectionDescription) {
        this.sectionDescription = sectionDescription;
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getSectionTitle(), this.getSectionDescription()},
EmployeeSection.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        return Dal.doMutation(new Object[]{this.getSectionTitle(), this.getSectionDescription(),
this.getEmployeeSectionID()}, EmployeeSection.SQL_UPDATE);
    }

    @DirectMethod
    public ResponseMessage saveData(JsonArray json) {
        return Socket.saveData(json.get(0).getAsJsonObject(), Constants.PROPERTY_EMPLOYEE_SECTION_ID,
this.getClass());
    }
}
```

TrainingCourse.java

```
package bol.modules;

import bol.BusinessModule;
import bol.Constants;
import dal.Dal;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class TrainingCourse extends BusinessModule implements Serializable {
    public static final String SQL_FETCH = "SELECT * FROM training_courses WHERE course_id = ? LIMIT 1";
    public static final String SQL_LIST = "SELECT * FROM training_courses";
    public static final String SQL_DELETE = "DELETE FROM training_courses WHERE course_id = ? LIMIT 1";
    public static final String SQL_TABLE = "training_courses";
    private static final String SQL_INSERT = "INSERT INTO training_courses (course_title, course_description) VALUES
(?, ?)";
    private static final String SQL_UPDATE = "UPDATE training_courses SET course_title = ?, course_description = ?
WHERE course_id = ?";

    private int courseID;
    private String courseTitle;
    private String courseDescription;

    public TrainingCourse() {

    }

    public TrainingCourse(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setCourseID(data.getInt(Constants.DB_FIELD_COURSE_ID));
            this.setCourseTitle(data.getString(Constants.DB_FIELD_COURSE_TITLE));
            this.setCourseDescription(data.getString(Constants.DB_FIELD_COURSE_DESCRIPTION));
        } catch (SQLException ex) {
            Logger.getLogger(TrainingCourse.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public int getCourseID() {
        return this.courseID;
    }

    public void setCourseID(int courseID) {
        this.courseID = courseID;
    }

    public String getCourseTitle() {
        return this.courseTitle;
    }

    public void setCourseTitle(String courseTitle) {
        this.courseTitle = courseTitle;
    }

    public String getCourseDescription() {
        return this.courseDescription;
    }

    public void setCourseDescription(String courseDescription) {
        this.courseDescription = courseDescription;
    }
```

```
    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getCourseTitle(), this.getCourseDescription()},
TrainingCourse.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        return Dal.doMutation(new Object[]{this.getCourseTitle(), this.getCourseDescription(), this.getCourseID()},
TrainingCourse.SQL_UPDATE);
    }
}
```

TrainingModule.java

```
package bol.modules;

import bol.BusinessModule;
import bol.Constants;
import bol.Socket;
import bol.util.Response.ResponseMessage;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import dal.Dal;
import djn.config.annotations.DirectMethod;
import java.io.Serializable;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class TrainingModule extends BusinessModule implements Serializable {

    public static final String SQL_FETCH = "SELECT * FROM training_course_modules WHERE course_module_id = ? LIMIT
1";
    public static final String SQL_LIST = "SELECT * FROM training_course_modules";
    public static final String SQL_DELETE = "DELETE FROM training_course_modules WHERE course_module_id = ? LIMIT
1";
    public static final String SQL_TABLE = "training_course_modules";
    private static final String SQL_INSERT = "INSERT INTO training_course_modules (course_id, course_module_title,
course_module_contents) VALUES (?, ?, ?)";
    private static final String SQL_UPDATE = "UPDATE training_course_modules SET course_id = ?, course_module_title
= ?, course_module_contents = ? WHERE course_module_id = ?";
    private int courseModuleID;
    private int courseID;
    private String courseModuleTitle;
    private String courseModuleContents;

    public TrainingModule() {
    }

    public TrainingModule(ResultSet data) {
        try {
            super.setEntityState(EntityState.Unchanged);
            this.setCourseModuleID(data.getInt(Constants.DB_FIELD_COURSE_MODULE_ID));
            this.setCourseID(data.getInt(Constants.DB_FIELD_COURSE_ID));
            this.setCourseModuleTitle(data.getString(Constants.DB_FIELD_COURSE_MODULE_TITLE));
            this.setCourseModuleContents(data.getString(Constants.DB_FIELD_COURSE_MODULE_CONTENTS));
        } catch (SQLException ex) {
            Logger.getLogger(TrainingModule.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public TrainingModule(JsonObject data) {
```

```
        this.setCourseModuleID(data.get(Constants.PROPERTY_COURSE_MODULE_ID).getAsInt());
        this.setCourseID(data.get(Constants.PROPERTY_COURSE_ID).getAsInt());
        this.setCourseModuleTitle(data.get(Constants.PROPERTY_COURSE_MODULE_TITLE).getAsString());
        this.setCourseModuleContents(data.get(Constants.PROPERTY_COURSE_MODULE_CONTENTS).getAsString());
    }

    public int getCourseModuleID() {
        return this.courseModuleID;
    }

    public void setCourseModuleID(int courseModuleID) {
        this.courseModuleID = courseModuleID;
    }

    public int getCourseID() {
        return this.courseID;
    }

    public void setCourseID(int courseID) {
        this.courseID = courseID;
    }

    public String getCourseModuleTitle() {
        return this.courseModuleTitle;
    }

    public void setCourseModuleTitle(String courseModuleTitle) {
        this.courseModuleTitle = courseModuleTitle;
    }

    public String getCourseModuleContents() {
        return this.courseModuleContents;
    }

    public void setCourseModuleContents(String courseModuleContents) {
        this.courseModuleContents = courseModuleContents;
    }

    @DirectMethod
    public ResponseMessage saveData(JsonArray json) {
        return Socket.saveData(json.get(0).getAsJsonObject(), Constants.PROPERTY_COURSE_MODULE_ID, this.getClass());
    }

    @Override
    protected PreparedStatement insert() throws SQLException {
        return Dal.doMutation(new Object[]{this.getCourseID(), this.getCourseModuleTitle(),
this.getCourseModuleContents()}, TrainingModule.SQL_INSERT);
    }

    @Override
    protected PreparedStatement update() throws SQLException {
        return Dal.doMutation(new Object[]{this.getCourseID(), this.getCourseModuleTitle(),
this.getCourseModuleContents(), this.getCourseModuleID()}, TrainingModule.SQL_UPDATE);
    }
}
```

Business Object Layer Utils

Response.java

```
package bol.util;

import java.util.ArrayList;
import java.util.Map;

public class Response {

    public static class ResponseList {

        private int totalRows;
        private boolean success;
        private ArrayList data;
        private String message;

        public int getTotalRows() {
            return totalRows;
        }

        public void setTotalRows(int totalRows) {
            this.totalRows = totalRows;
        }

        public boolean isSuccess() {
            return success;
        }

        public void setSuccess(boolean success) {
            this.success = success;
        }

        public ArrayList getData() {
            return this.data;
        }

        public void setData(ArrayList data) {
            this.data = data;
        }

        public String getMessage() {
            return this.message;
        }

        public void setMessage(String message) {
            this.message = message;
        }
    }

    public static class ResponseObject {

        private Object data;
        private boolean success;
        private String title;
        private String message;

        public Object getObject() {
            return data;
        }

        public void setObject(Object object) {
            this.data = object;
        }

        public boolean isSuccess() {
```

```
            return success;
        }

        public void setSuccess(boolean success) {
            this.success = success;
        }

        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        public String getMessage() {
            return message;
        }

        public void setMessage(String message) {
            this.message = message;
        }
    }

    public static class ResponseType {

        private int id;
        private String name;

        public ResponseType() {
        }

        public ResponseType(int id, String name) {
            this.setId(id);
            this.setName(name);
        }

        public int getId() {
            return this.id;
        }

        public void setId(int id) {
            this.id = id;
        }

        public String getName() {
            return this.name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public static <E extends Enum> ArrayList<ResponseType> create(E[] types) {
            ArrayList<ResponseType> data = new ArrayList<ResponseType>();
            for (E type : types) {
                data.add(new ResponseType(type.ordinal(), type.toString()));
            }
            return data;
        }
    }

    public static class ResponseMessage {

        private boolean success;
        private String title;
        private String message;

        public boolean isSuccess() {
            return this.success;
        }
```

```
        public void setSuccess(boolean success) {
            this.success = success;
        }

        public String getTitle() {
            return this.title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        public String getMessage() {
            return this.message;
        }

        public void setMessage(String message) {
            this.message = message;
        }
    }

    public static class ResponseSubmission {

        private boolean success;
        private Map<String, String> errors;
        private String message;
        private String title;

        public boolean isSuccess() {
            return this.success;
        }

        public void setSuccess(boolean success) {
            this.success = success;
        }

        public String getMessage() {
            return this.message;
        }

        public void setMessage(String message) {
            this.message = message;
        }

        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        public Map<String, String> getErrors() {
            return this.errors;
        }

        public void setErrors(Map<String, String> errors) {
            this.errors = errors;
        }
    }

    public static class ResponseNode {

        private String id;
        private String text;
        private boolean leaf;

        public ResponseNode(String id, String text, boolean leaf) {
            this.setId(id);
            this.setText(text);
```

```
            this.setLeaf(leaf);
        }

        public String getId() {
            return this.id;
        }

        public void setId(String id) {
            this.id = id;
        }

        public String getText() {
            return this.text;
        }

        public void setText(String text) {
            this.text = text;
        }

        public boolean isLeaf() {
            return this.leaf;
        }

        public void setLeaf(boolean leaf) {
            this.leaf = leaf;
        }
    }

    public static class ResponseMenuItem {

        public static final String CLS = "x-btn-text-icon";
        public static final String SCALE = "medium";
        private String scale;
        private String text;
        private String cls;
        private String icon;
        private String menu;
        private String handler;

        public ResponseMenuItem() {
        }

        public void setScale(String scale) {
            this.scale = scale;
        }

        public void setText(String text) {
            this.text = text;
        }

        public void setCls(String cls) {
            this.cls = cls;
        }

        public void setIcon(String icon) {
            this.icon = icon;
        }

        public void setMenu(String menu) {
            this.menu = menu;
        }

        public void setHandler(String handler) {
            this.handler = handler;
        }
    }
}
```

Data Access Layer

Dal.java

```
package dal;

import com.mysql.jdbc.CallableStatement;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.PreparedStatement;
import com.mysql.jdbc.Statement;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Dal {

    private static final String CALL_POSTFIX = " }";
    private static final String CALL_PREFIX = "{ call ";
    private static final String SQL_COUNT_QUERY = "SELECT COUNT(*) FROM ";
    private static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    private static final String JDBC_URL_LOCATION = "jdbc:mysql://localhost/tricert_system";
    private static final String DB_USER_NAME = "root";
    private static final String DB_PASSWORD = "";
    private static Connection connection;

    public enum StatmentType {

        preparedStatment, callableStatement
    };

    private static Statement getStatement() throws SQLException {
        return (Statement) Dal.getConnection().createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
    }

    private static void addParameters(PreparedStatement statement, Object[] parameters) throws SQLException {
        if (parameters != null) {
            for (int i = 0; i < (parameters.length); i++) {
                statement.setObject((i + 1), parameters[i]);
            }
        }
    }

    public static ResultSet doQuery(Object[] parameters, String sql, Dal.StatmentType type) throws SQLException {
        PreparedStatement statement = ((type == StatmentType.callableStatement) ? (CallableStatement)
Dal.getConnection().prepareCall(Dal.CALL_PREFIX + sql + Dal.CALL_POSTFIX)
                : (PreparedStatement) Dal.getConnection().prepareStatement(sql));
        Dal.addParameters(statement, parameters);
        return statement.executeQuery();
    }

    public static ResultSet doQuery(String sql) throws SQLException {
        return Dal.getStatement().executeQuery(sql);
    }

    public static PreparedStatement doMutation(Object[] parameters, String sql) throws SQLException {
        PreparedStatement statement = (PreparedStatement) Dal.getConnection().prepareStatement(sql);
        Dal.addParameters(statement, parameters);
        statement.executeUpdate();
        return statement;
    }

    public static int getCount(String table, String where) throws SQLException {
        ResultSet records = Dal.doQuery(Dal.SQL_COUNT_QUERY + table + where);
        records.next();
```

```
        return records.getInt(1);
    }

    public static boolean hasRows(ResultSet results) throws SQLException {
        results.first();
        return ((results != null) ? (results.getRow() > 0) : false);
    }

    private static Connection getConnection() {
        try {
            if (Dal.connection == null) {
                Class.forName(JDBC_DRIVER);
                Dal.connection = (Connection) DriverManager.getConnection(JDBC_URL_LOCATION, DB_USER_NAME,
DB_PASSWORD);
            }
        } catch (Exception ex) {
            Logger.getLogger(Dal.class.getName()).log(Level.SEVERE, null, ex);
        }
        return Dal.connection;
    }
}
```

Database Structure

Overview

The database will be running in a relational enforced integrity database storage engine called 'InnoDB'; which specifies the relationships between the tables. Therefore, the data within the system will be more related, and updated accordingly throughout the database once a change has been made.

The data throughout the system can contain child data, this child data will be enforced to be removed once the parent has been deleted, therefore 'cascading' the deletion throughout the related tables. Confirmation for the deletion; will be taken place on the user's end, and they will be warned also about the related data being removed before the delete is proceeded.

The naming conventions used throughout this system, is a standard prefix of the section name across all related tables; therefore it will provide a clear overview of the grouped tables. The names across the table names, and also the fields are kept as simple as possible, but also providing a detailed enough description of what the data contains. Multiple word usages are also separated using the underscore.

The relationships between the tables are enforced using the constraints, therefore mapping the foreign keys to their related table. The foreign keys used throughout the system, have indexes accordingly placed on the most frequent queried areas; therefore keeping the system in a fast retrievable state.

Security concerns: The password for the users account will be encrypted using SHA1 hashing. Further encryption methodologies for other details will be investigated later on down the track. The table data-types in this system will be kept as though they will contain the correct unencrypted data.

Initial System Base

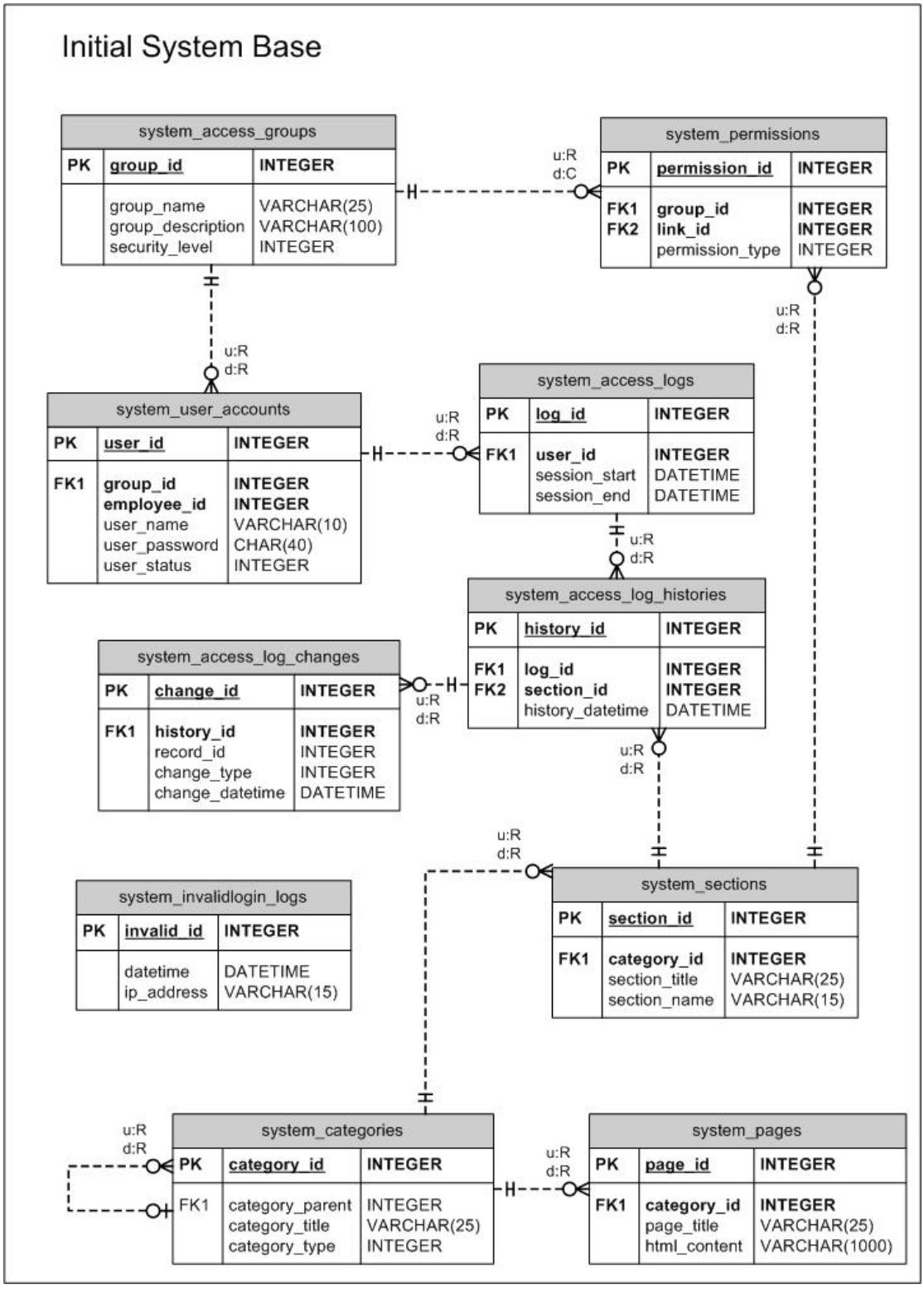


Table structure for table `system_access_groups`

```
CREATE TABLE IF NOT EXISTS `system_access_groups` (
  `group_id` int(11) NOT NULL AUTO_INCREMENT,
  `group_name` varchar(25) NOT NULL,
  `group_description` varchar(100) NOT NULL,
  `group_security_level` int(11) NOT NULL,
  PRIMARY KEY (`group_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Field Name	Description
goup_id	The group Id is a primary identification for the access group table.
goup_name	The brief name of the security permissions group
goup_description	A further in-depth detail of the group permissions *optional*
goup_security_level	A security level which is an integer value that represents what the group can actually do. Either supporting just a read-only view, or if they are indeed allowed to make modifications.

Table structure for table `system_access_logs`

```
CREATE TABLE IF NOT EXISTS `system_access_logs` (
  `log_id` int(11) NOT NULL AUTO_INCREMENT,
  `user_id` int(11) NOT NULL,
  `session_start` datetime NOT NULL,
  `session_end` datetime NOT NULL,
  `ip_address` varchar(15) NOT NULL,
  PRIMARY KEY (`log_id`),
  KEY `user_id` (`user_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Field Name	Description
log_id	The log Id is a primary identification for the access logs table.
user_id	The user_id for which the log applies to. Each user can have multiple logs for each session that gets created.
session_start	Records the start datetime of the session.
session_end	Records the time for which the user session has expired.
ip_address	Records the ip address of the authenticated session

Constraints for table `system_access_logs`

```
ALTER TABLE `system_access_logs`
  ADD CONSTRAINT `FKaccess_logs_user_id` FOREIGN KEY (`user_id`)
        REFERENCES `system_user_accounts` (`user_id`) ON DELETE CASCADE;
```

Table structure for table `system_access_log_histories`

```
CREATE TABLE IF NOT EXISTS `system_access_log_histories` (
  `history_id` int(11) NOT NULL AUTO_INCREMENT,
  `log_id` int(11) NOT NULL,
  `section_id` int(11) NOT NULL,
  `history_datetime` datetime NOT NULL,
  PRIMARY KEY (`history_id`),
  KEY `log_id` (`log_id`),
  KEY `section_id` (`section_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `system_access_log_histories`

```
ALTER TABLE `system_access_log_histories`
  ADD CONSTRAINT `FKaccess_log_histories_section_id` FOREIGN KEY (`section_id`)
        REFERENCES `system_sections` (`section_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKaccess_log_histories_log_id` FOREIGN KEY (`log_id`)
REFERENCES `system_access_logs` (`log_id`) ON DELETE CASCADE;
```

Field Name	Description
history_id	The primary identification for viewed section.
log_id	The session log which the history corresponds to.
section_id	The section that was actually viewed.
history_datetime	The date-time the section was viewed.

Table structure for table `system_access_log_changes`

```
CREATE TABLE IF NOT EXISTS `system_access_log_changes` (
  `change_id` int(11) NOT NULL AUTO_INCREMENT,
  `history_id` int(11) NOT NULL,
  `record_id` int(11) NOT NULL,
  `change_type` int(11) NOT NULL,
  `change_datetime` datetime NOT NULL,
  PRIMARY KEY (`change_id`),
  KEY `history_id` (`history_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Field Name	Description
change_id	The primary identification for recorded changes table.
history_id	The history_id of the section viewed.
record_id	The record for which the change was applied to.
change_type	The type of data change made, either if it was an update, deletion, or even an insertion.
change_datetime	The date-time the change was made.

Constraints for table `system_access_log_changes`

```
ALTER TABLE `system_access_log_changes`
  ADD CONSTRAINT `FKaccess_log_changes_history_id` FOREIGN KEY (`history_id`)
        REFERENCES `system_access_log_histories` (`history_id`) ON DELETE CASCADE;
```

Table structure for table `system_categories`

```
CREATE TABLE IF NOT EXISTS `system_categories` (
  `category_id` int(11) NOT NULL AUTO_INCREMENT,
  `category_parent` int(11) NOT NULL,
  `category_title` varchar(25) NOT NULL,
  `category_type` int(11) NOT NULL,
  PRIMARY KEY (`category_id`),
  KEY `category_parent` (`category_parent`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `system_categories`

```
ALTER TABLE `system_categories`
  ADD CONSTRAINT `FKcategories_category_parent` FOREIGN KEY (`category_parent`)
        REFERENCES `system_categories` (`category_id`) ON DELETE CASCADE;
```

Field Name	Description
category_id	The primary identification for category table.
category_parent	The category can be a parent or a child of another category; therefore having a relation to its own table.
category_title	The title of the category, this will be displayed on the main navigation bar.
category_type	The category type; whether if it's a main navigation, document library category, or even an administration category.

Notes: The icon for the main category might be another field, or it could be stored in another way. Further research will be undertaken once development on this section commences.

Table structure for table `system_pages`

```
CREATE TABLE IF NOT EXISTS `system_pages` (
  `page_id` int(11) NOT NULL AUTO_INCREMENT,
  `category_id` int(11) NOT NULL,
  `page_title` varchar(25) NOT NULL,
  `html_content` varchar(1000) NOT NULL,
  PRIMARY KEY (`page_id`),
  KEY `category_id` (`category_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `system_pages`

```
ALTER TABLE `system_pages`
  ADD CONSTRAINT `FKpages_category_id` FOREIGN KEY (`category_id`)
REFERENCES `system_categories` (`category_id`) ON DELETE CASCADE;
```

Field Name	Description
page_id	The primary identification for the pages table.
category_id	The page is a child of a category selected.
page_title	The title of the page will be displayed on the text for the hyperlink.
html_content	The page content will be stored and read in HTML. The system will also provide the user with a method to edit this html, through the use of a web mark-up language editor tool; such as TinyMCE.

Table structure for table `system_permissions`

```
CREATE TABLE IF NOT EXISTS `system_permissions` (
  `permission_id` int(11) NOT NULL AUTO_INCREMENT,
  `group_id` int(11) NOT NULL,
  `link_id` int(11) NOT NULL,
  `permission_type` int(11) NOT NULL,
  PRIMARY KEY (`permission_id`),
  KEY `group_id` (`group_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `system_permissions`

```
ALTER TABLE `system_permissions`
  ADD CONSTRAINT `FKpermissions_group_id` FOREIGN KEY (`group_id`)
        REFERENCES `system_access_groups` (`group_id`) ON DELETE CASCADE;
```

Field Name	Description
permission_id	Primary identification for the permissions table.
group_id	Group for which the permission corresponds to.
link_id	The permission could be for either a section, or a document category; so therefore there will not be any referential integrity enforced onto this field, and it will be kept as generic as possible.
permission_type	Either a navigation category or a document category. It will also provide the ability to be further expanded by keeping the permission system generically designed.

Table structure for table `system_sections`

```
CREATE TABLE IF NOT EXISTS `system_sections` (
  `section_id` int(11) NOT NULL AUTO_INCREMENT,
  `category_id` int(11) NOT NULL,
  `section_title` varchar(25) NOT NULL,
  `section_name` varchar(15) NOT NULL,
  PRIMARY KEY (`section_id`),
  KEY `category_id` (`category_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=12 ;
```

Constraints for table `system_sections`

```
ALTER TABLE `system_sections`
  ADD CONSTRAINT `FKsections_category_id` FOREIGN KEY (`category_id`)
        REFERENCES `system_categories` (`category_id`) ON DELETE CASCADE;
```

Field Name	Description
section _id	Primary section identification for the sections table.
category_id	Category for which the section will correspond to.
section_title	Title of the section.
section_name	Section filename, which will be used to load the section.

Table structure for table `system_invalidlogin_logs`

```
CREATE TABLE IF NOT EXISTS `system_invalidlogin_logs` (
  `invalid_id` int(11) NOT NULL AUTO_INCREMENT,
  `datetime` datetime NOT NULL,
  `ip_address` varchar(15) NOT NULL,
  PRIMARY KEY (`log_id`),
  KEY `user_id` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `system_security_logs`

```
ALTER TABLE `system_security_logs`
  ADD CONSTRAINT `FKsecurity_logs_user_id` FOREIGN KEY (`user_id`)
        REFERENCES `system_user_accounts` (`user_id`) ON DELETE CASCADE;
```

Field Name	Description
invalid_id	Primary identification for the invalid login attempts table.
datetime	Date time for which the login attempt was tried.
ip_address	IP address of the user.

Table structure for table `system_user_accounts`

```
CREATE TABLE IF NOT EXISTS `system_user_accounts` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT,
  `group_id` int(11) NOT NULL,
  `employee_id` int(11) NOT NULL,
  `user_name` varchar(10) NOT NULL,
  `user_password` char(40) NOT NULL,
  `user_status` int(11) NOT NULL,
  PRIMARY KEY (`user_id`),
  KEY `group_id` (`group_id`),
  KEY `employee_id` (`employee_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `system_user_accounts`

```
ALTER TABLE `system_user_accounts`
  ADD CONSTRAINT `FKuser_accounts_employee_id` FOREIGN KEY (`employee_id`)
        REFERENCES `employees` (`employee_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKuser_accounts_group_id` FOREIGN KEY (`group_id`)
        REFERENCES `system_access_groups` (`group_id`) ON DELETE CASCADE;
```

Field Name	Description
user_id	Primary identification for the user accounts table.
group_id	Permissions group for which the user belongs to.
employee_id	Linked employee record.
user_name	User name for the user account.
user_password	Password for the user account. It will be using SHA1 encryption
user_status	User account can either be enabled, or disabled for access.

Document Library Management System

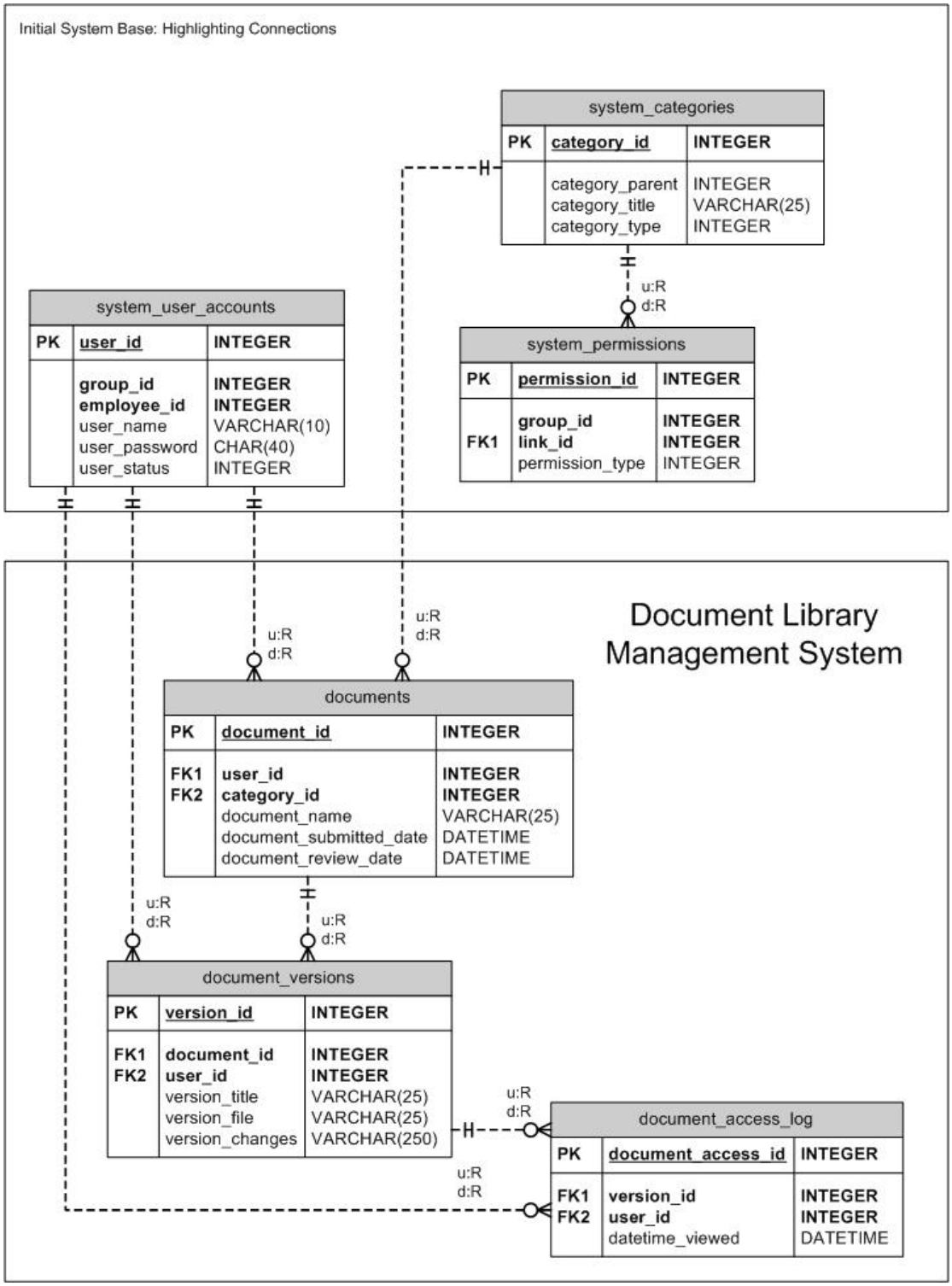


Table structure for table `documents`

```
CREATE TABLE IF NOT EXISTS `documents` (
  `document_id` int(11) NOT NULL AUTO_INCREMENT,
  `user_id` int(11) NOT NULL,
  `category_id` int(11) NOT NULL,
  `document_name` varchar(25) NOT NULL,
  `document_submitted_date` datetime NOT NULL,
  `document_review_date` datetime NOT NULL,
  PRIMARY KEY (`document_id`),
  KEY `user_id` (`user_id`),
  KEY `category_id` (`category_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `documents`

```
ALTER TABLE `documents`
  ADD CONSTRAINT `FKdocuments_category_id` FOREIGN KEY (`category_id`)
        REFERENCES `system_categories` (`category_id`),
  ADD CONSTRAINT `FKdocuments_user_id` FOREIGN KEY (`user_id`)
        REFERENCES `system_user_accounts` (`user_id`);
```

Field Name	Description
document_id	Primary identification for the documents table.
user_id	The user for which originally uploaded the file.
category_id	Category that the document belongs to.
document_name	The display title for the document.
document_submitted_date	The submission date of the document uploaded.
document_review_date	The review date set, which is a mandatory date for which the document needs to have, that requires for the document to be checked/ or updated with a new version before the review date expires.

Table structure for table `document_access_log`

```
CREATE TABLE IF NOT EXISTS `document_access_log` (
  `document_log_id` int(11) NOT NULL AUTO_INCREMENT,
  `version_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `datetime_viewed` datetime NOT NULL,
  PRIMARY KEY (`document_log_id`),
  KEY `version_id` (`version_id`),
  KEY `user_id` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `document_access_log`

```
ALTER TABLE `document_access_log`
  ADD CONSTRAINT `FKdocument_access_log_user_id` FOREIGN KEY (`user_id`)
        REFERENCES `system_user_accounts` (`user_id`),
  ADD CONSTRAINT `FKdocument_access_log_version_id` FOREIGN KEY (`version_id`)
        REFERENCES `document_versions` (`version_id`) ON DELETE CASCADE;
```

Field Name	Description
document_log_id	Primary identification for the document access log table.
version_id	The version id of the document viewed
user_id	User who viewed the document version
datetime_viewed	Datetime the viewing occurred.

Table structure for table `document_versions`

```
CREATE TABLE IF NOT EXISTS `document_versions` (
  `version_id` int(11) NOT NULL AUTO_INCREMENT,
  `document_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `version_title` varchar(25) NOT NULL,
  `version_file` varchar(25) NOT NULL,
  `version_changes` varchar(250) NOT NULL,
  PRIMARY KEY (`version_id`),
  KEY `user_id` (`user_id`),
  KEY `document_id` (`document_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `document_versions`

```
ALTER TABLE `document_versions`
  ADD CONSTRAINT `FKdocument_versions_document_id` FOREIGN KEY (`document_id`)
        REFERENCES `documents` (`document_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKdocument_versions_user_id` FOREIGN KEY (`user_id`)
        REFERENCES `system_user_accounts` (`user_id`);
```

Field Name	Description
version_id	The primary identification of the document version.
document_id	The document id for which the version corresponds to.
user_id	The user in who uploaded the latest document version.
version_title	Title of the version uploaded.
version_file	The filename of the version.
version_changes	A brief description of the changes applied to the document.

Employee Management System

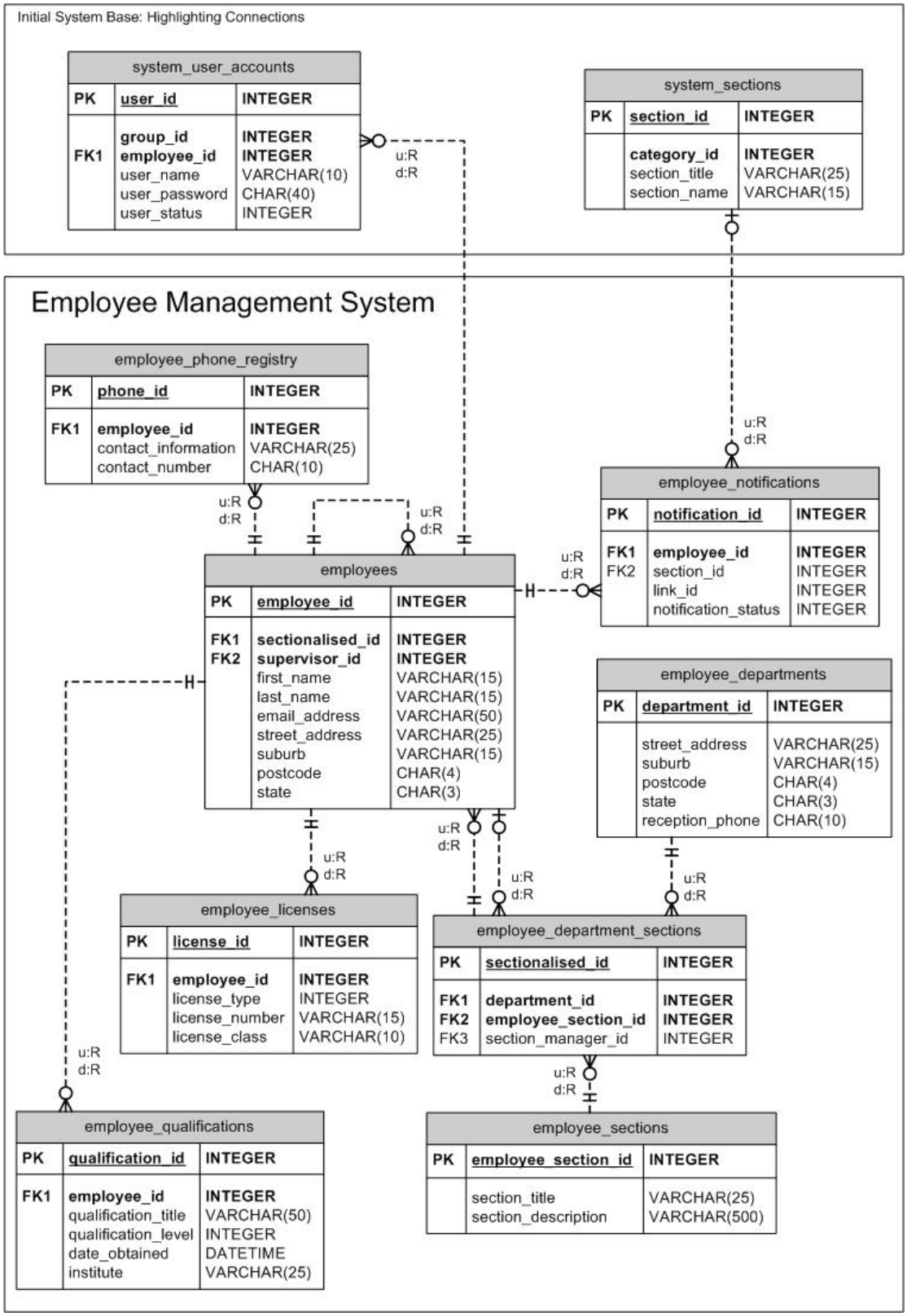


Table structure for table `employees`

```
CREATE TABLE IF NOT EXISTS `employees` (
  `employee_id` int(11) NOT NULL AUTO_INCREMENT,
  `sectionalised_id` int(11) NOT NULL,
  `supervisor_id` int(11) NOT NULL,
  `first_name` varchar(15) NOT NULL,
  `last_name` varchar(15) NOT NULL,
  `email_address` varchar(50) NOT NULL,
  `street_address` varchar(25) NOT NULL,
  `suburb` varchar(15) NOT NULL,
  `postcode` char(4) NOT NULL,
  `state` char(3) NOT NULL,
  PRIMARY KEY (`employee_id`),
  KEY `sectionalised_id` (`sectionalised_id`),
  KEY `supervisor_id` (`supervisor_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `employees`

```
ALTER TABLE `employees`
  ADD CONSTRAINT `FKemployees_sectionalised_id` FOREIGN KEY (`sectionalised_id`)
          REFERENCES `employee_department_sections` (`sectionalised_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKemployees_supervisor_id` FOREIGN KEY (`supervisor_id`)
          REFERENCES `employees` (`employee_id`) ON DELETE CASCADE;
```

Field Name	Description
employee_id	The primary identification of the employee table
sectionalised_id	The section in the department for which the employee is assigned to.
supervisor_id	A relation to the same table, which allows for an employee to be assigned a supervisor. Therefore, it creates a employee hierarchy.
first_name	First name of the employee.
last_name	Last name of the employee.
email_address	Email address of the employee.
street_address	Street address of the employee.
suburb	Suburb of the employee residence.
postcode	Postcode of the employee suburb.
state	The state of residence.

Table structure for table `employee_departments`

```
CREATE TABLE IF NOT EXISTS `employee_departments` (
  `department_id` int(11) NOT NULL AUTO_INCREMENT,
  `department_name` varchar(20) NOT NULL,
  `street_address` varchar(25) NOT NULL,
  `suburb` varchar(15) NOT NULL,
  `postcode` char(4) NOT NULL,
  `state` char(3) NOT NULL,
  `reception_phone` char(10) NOT NULL,
  PRIMARY KEY (`department_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Field Name	Description
department_id	The primary identification of the department's table
department_name	A brief title of the department name
street_address	Location for the department
suburb	Suburb of the department
postcode	Postcode
state	State
reception_phone	The phone number of the department's reception desk.

Table structure for table `employee_department_sections`

```
CREATE TABLE IF NOT EXISTS `employee_department_sections` (
  `sectionalised_id` int(11) NOT NULL AUTO_INCREMENT,
  `department_id` int(11) NOT NULL,
  `employee_section_id` int(11) NOT NULL,
  `section_manager_id` int(11) NOT NULL,
  PRIMARY KEY (`sectionalised_id`),
  KEY `department_id` (`department_id`),
  KEY `employee_section_id` (`employee_section_id`),
  KEY `section_manager_id` (`section_manager_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `employee_department_sections`

```
ALTER TABLE `employee_department_sections`
  ADD CONSTRAINT `FKdepartment_sections_department_id` FOREIGN KEY (`department_id`)
        REFERENCES `employee_departments` (`department_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKdepartment_sections_employee_section_id` FOREIGN KEY (`employee_section_id`)
        REFERENCES `employee_sections` (`employee_section_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKdepartment_sections_section_manager_id` FOREIGN KEY (`section_manager_id`)
        REFERENCES `employees` (`employee_id`) ON DELETE CASCADE;
```

Field Name	Description
sectionalised_id	The primary identification of the sectionalised department's table
department_id	The department for which the section belongs to.
employee_section_id	The section which is within this particular department.
section_manager_id	The manager who is in-charge of that section.

Table structure for table `employee_licenses`

```
CREATE TABLE IF NOT EXISTS `employee_licenses` (
  `license_id` int(11) NOT NULL AUTO_INCREMENT,
  `employee_id` int(11) NOT NULL,
  `license_type` int(11) NOT NULL,
  `license_number` varchar(15) NOT NULL,
  `license_class` varchar(10) NOT NULL,
  PRIMARY KEY (`license_id`),
  KEY `employee_id` (`employee_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ROW_FORMAT=DYNAMIC
AUTO_INCREMENT=1 ;
```

Constraints for table `employee_licenses`

```
ALTER TABLE `employee_licenses`
  ADD CONSTRAINT `FKlicenses_employee_id` FOREIGN KEY (`employee_id`)
        REFERENCES `employees` (`employee_id`) ON DELETE CASCADE;
```

Field Name	Description
license_id	The primary identification of the license table
employee_id	The employee for which the license belongs to.
license_type	The type of license, It could be a drivers license, green card etc..
license_number	The number of the license
license_class	The license class.

*notes: more fields might be additionally added or changed once development on this section is undertaken. The license_type might be removed, and replaced with a simple text data string; therefore allowing the user to input anything they want and it would not be limited to the constraints of the enumeration.

Table structure for table `employee_notifications`

```
CREATE TABLE IF NOT EXISTS `employee_notifications` (
  `notification_id` int(11) NOT NULL AUTO_INCREMENT,
  `employee_id` int(11) NOT NULL,
  `section_id` int(11) NOT NULL,
  `record_id` int(11) NOT NULL,
  `notification_status` int(11) NOT NULL,
  PRIMARY KEY (`notification_id`),
  KEY `employee_id` (`employee_id`),
  KEY `section_id` (`section_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `employee_notifications`

```
ALTER TABLE `employee_notifications`
  ADD CONSTRAINT `FKnotifications_employee_id` FOREIGN KEY (`employee_id`)
        REFERENCES `employees` (`employee_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKnotifications_section_id` FOREIGN KEY (`section_id`)
        REFERENCES `system_sections` (`section_id`) ON DELETE CASCADE;
```

Field Name	Description
notification_id	The primary identification of the notifications table
employee_id	The employee for which will be notified.
section_id	The section ID which the user will be informed about.
record_id	The corresponding record in that particular section.
notification_status	The notification status, either if the user is contacted, or still pending (for a administrator to make a physical contact if there isn't a user account for that particular employee).

Table structure for table `employee_phone_registry`

```
CREATE TABLE IF NOT EXISTS `employee_phone_registry` (
  `phone_id` int(11) NOT NULL AUTO_INCREMENT,
  `employee_id` int(11) NOT NULL,
  `contact_information` varchar(25) NOT NULL,
  `contact_number` char(10) NOT NULL,
  PRIMARY KEY (`phone_id`),
  KEY `employee_id` (`employee_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ROW_FORMAT=DYNAMIC
AUTO_INCREMENT=1 ;
```

Constraints for table `employee_phone_registry`

```
ALTER TABLE `employee_phone_registry`
  ADD CONSTRAINT `FKphone_registry_employee_id` FOREIGN KEY (`employee_id`)
          REFERENCES `employees` (`employee_id`) ON DELETE CASCADE;
```

Field Name	Description
phone_id	The primary identification of the phone registry table.
employee_id	The employee for which the phone record corresponds to.
contact_information	The contact information string: either it could have 'Home Phone', or some other type entered, even if it's just a 'Related Friend'. The string will be decided upon by the user entering the phone number just to keep it simple.
contact_number	The phone contact number.

Table structure for table `employee_qualifications`

```
CREATE TABLE IF NOT EXISTS `employee_qualifications` (
  `qualification_id` int(11) NOT NULL AUTO_INCREMENT,
  `employee_id` int(11) NOT NULL,
  `qualification_title` varchar(50) NOT NULL,
  `qualification_level` int(11) NOT NULL,
  `date_obtained` datetime NOT NULL,
  `institute` varchar(25) NOT NULL,
  PRIMARY KEY (`qualification_id`),
  KEY `employee_id` (`employee_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `employee_qualifications`

```
ALTER TABLE `employee_qualifications`
  ADD CONSTRAINT `FKqualifications_employee_id` FOREIGN KEY (`employee_id`)
        REFERENCES `employees` (`employee_id`) ON DELETE CASCADE;
```

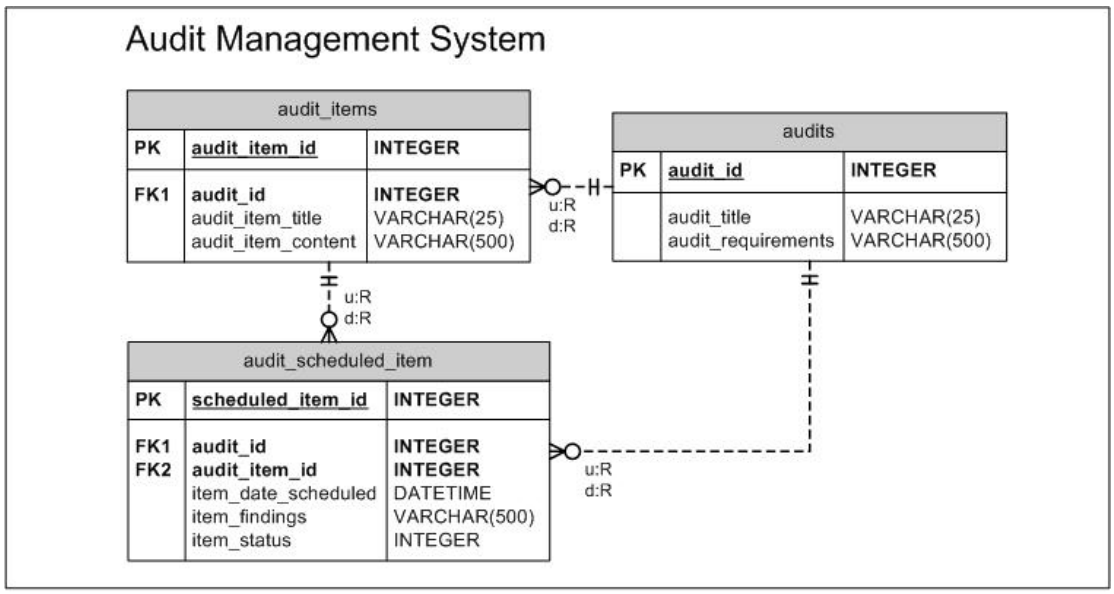
Field Name	Description
qualification_id	The primary identification of the qualifications table
employee_id	The employee for which the qualification belongs to.
qualification_title	The title of the qualification achieved.
qualification_level	The education level of the qualification.
date_obtained	The date the qualification was obtained.
institute	The name of the institute where it was studied.

Table structure for table `employee_sections`

```
CREATE TABLE IF NOT EXISTS `employee_sections` (
  `employee_section_id` int(11) NOT NULL AUTO_INCREMENT,
  `section_title` varchar(25) NOT NULL,
  `section_description` varchar(500) NOT NULL,
  PRIMARY KEY (`employee_section_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ROW_FORMAT=DYNAMIC
AUTO_INCREMENT=1 ;
```

Field Name	Description
employee_section_id	The primary identification of the sections table
section_title	The title of the section, it could either be Human Resources, Engineering and etc…
section_description	A description of what the section does.

Audit Management System



The audit creation system is structured in a way which is both simple, and allows for the further addition of new audits. The audit created, has a main 'audits' table which is the parent of the audit. The audit table can have multiple audit items, but these items can have the ability to be scheduled multiple times; therefore it brings another table called 'audit scheduled item', which contains the scheduled date, the findings of the audit, and also the status.

Each scheduled audit item can have the ability to either be in an open or closed state, with the option of allowing for the raising of action items against that particular audit item. So therefore, the ability to view, and add action items should appear once the audit scheduled item is being viewed.

Table structure for table `audits`

```
CREATE TABLE IF NOT EXISTS `audits` (
  `audit_id` int(11) NOT NULL AUTO_INCREMENT,
  `audit_title` varchar(25) NOT NULL,
  `audit_requirements` varchar(500) NOT NULL,
  PRIMARY KEY (`audit_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

Field Name	Description
audit_id	The primary identification of the audits table
audit_title	The title of the audit.
audit_requirements	A description of the requirements which the audit must comply to

Table structure for table `audit_items`

```
CREATE TABLE IF NOT EXISTS `audit_items` (
  `audit_item_id` int(11) NOT NULL AUTO_INCREMENT,
  `audit_id` int(11) NOT NULL,
  `audit_item_title` varchar(25) NOT NULL,
  `audit_item_content` varchar(500) NOT NULL,
  PRIMARY KEY (`audit_item_id`),
  KEY `audit_id` (`audit_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `audit_items`

```
ALTER TABLE `audit_items`
  ADD CONSTRAINT `FKitems_audit_id` FOREIGN KEY (`audit_id`)
        REFERENCES `audits` (`audit_id`) ON DELETE CASCADE;
```

Field Name	Description
audit_item_id	The primary identification of the audits item table
audit_id	The audit for which the item belongs to.
audit_item_title	The title of the audit item.
audit_item_content	The requirements/ content information for the audit item.

Table structure for table `audit_scheduled_item`

```
CREATE TABLE IF NOT EXISTS `audit_scheduled_item` (
  `scheduled_item_id` int(11) NOT NULL AUTO_INCREMENT,
  `audit_id` int(11) NOT NULL,
  `audit_item_id` int(11) NOT NULL,
  `item_date_scheduled` datetime NOT NULL,
  `item_findings` varchar(500) NOT NULL,
  `item_status` int(11) NOT NULL,
  PRIMARY KEY (`scheduled_item_id`),
  KEY `audit_id` (`audit_id`),
  KEY `audit_item_id` (`audit_item_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `audit_scheduled_item`

```
ALTER TABLE `audit_scheduled_item`
  ADD CONSTRAINT `FKscheduled_item_audit_id` FOREIGN KEY (`audit_id`)
        REFERENCES `audits` (`audit_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKscheduled_item_audit_item_id` FOREIGN KEY (`audit_item_id`)
        REFERENCES `audit_items` (`audit_item_id`) ON DELETE CASCADE;
```

Field Name	Description
scheduled_item_id	The primary identification of the scheduled audit item table
audit_id	The audit which the scheduled item belongs to.
audit_item_id	The audit item that has just been scheduled for compliance.
item_date_scheduled	The scheduled date.
item_findings	The findings from the audit item scheduled.
item_status	The status of the item (whether its open, or closed).

Action Management System

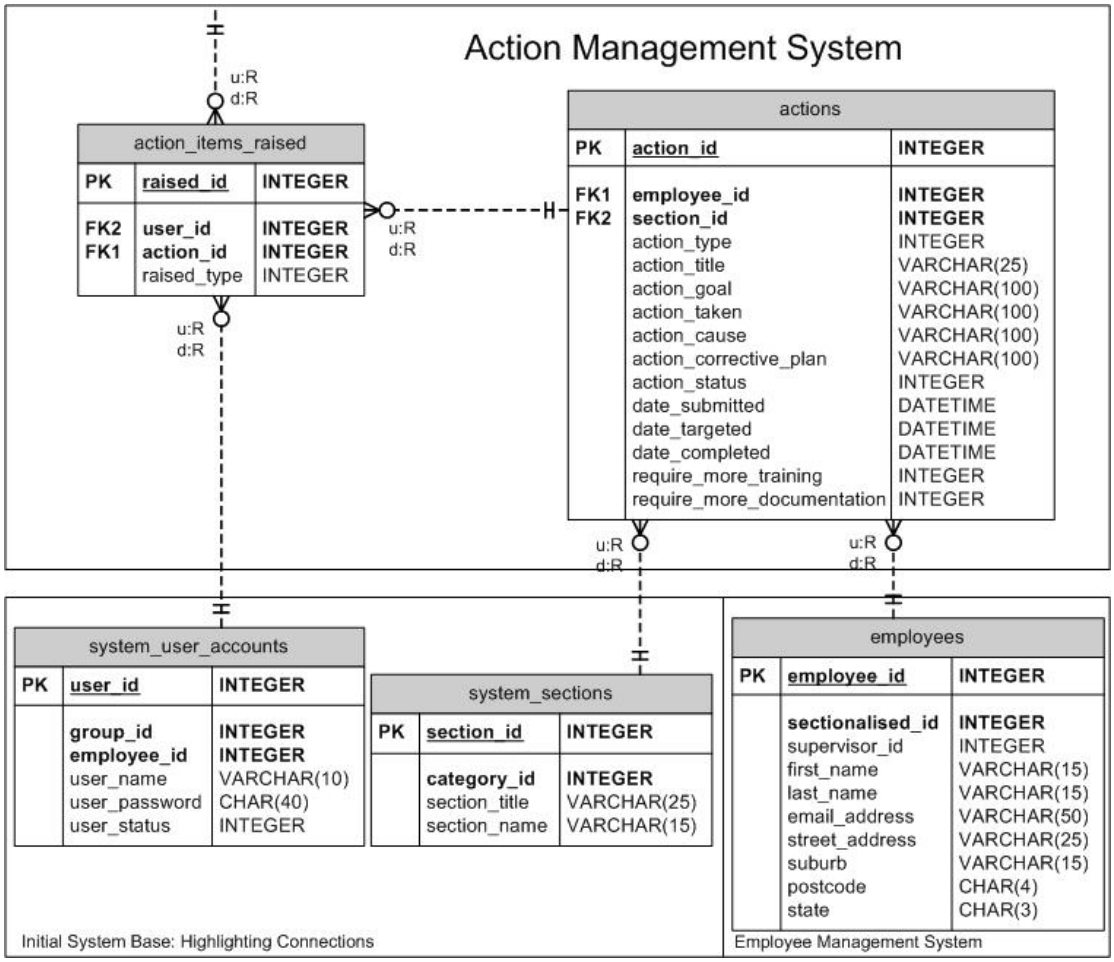


Table structure for table `actions`

```
CREATE TABLE IF NOT EXISTS `actions` (
  `action_id` int(11) NOT NULL AUTO_INCREMENT,
  `employee_id` int(11) NOT NULL,
  `section_id` int(11) NOT NULL,
  `action_type` int(11) NOT NULL,
  `action_title` varchar(25) NOT NULL,
  `action_goal` varchar(100) NOT NULL,
  `action_taken` varchar(100) NOT NULL,
  `action_cause` varchar(100) NOT NULL,
  `action_corrective_plan` varchar(100) NOT NULL,
  `action_status` int(11) NOT NULL,
  `date_submitted` year(4) NOT NULL,
  `date_targeted` datetime NOT NULL,
  `date_completed` datetime NOT NULL,
  `require_more_training` int(11) NOT NULL,
  `require_more_documentation` int(11) NOT NULL,
  PRIMARY KEY (`action_id`),
  KEY `employee_id` (`employee_id`),
  KEY `section_id` (`section_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

Constraints for table `actions`

```
ALTER TABLE `actions`
  ADD CONSTRAINT `FKactions_employee_id` FOREIGN KEY (`employee_id`)
          REFERENCES `employees` (`employee_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKactions_section_id` FOREIGN KEY (`section_id`)
          REFERENCES  `system_sections` (`section_id`) ON DELETE CASCADE;
```

Field Name	Description
action_id	The primary identification of the actions table
employee_id	The assigned responsible employee.
section_id	The section, which the action item was raised against.
action_type	The action type (either To Do Action, or a Corrective Action)
action_title	The title of the action
action_goal	The completion goal of the action
action_taken	The action to be taken to achieve this goal.
action_cause	The cause of the action (if any).
action_corrective_plan	The corrective plan (only visible if it's a corrective action).
action_status	Status of the action (whether it's open, pending, or closed).
date_submitted	The date the action was submitted.
date_targeted	The date which the action is targeted to be resolved by.
date_completed	The actual date the action was completed.
require_more_training	An indication whether more training is required for the action.
require_more_documentation	An indication whether more documentation is required.

Table structure for table `action_items_raised`

```
CREATE TABLE IF NOT EXISTS `action_items_raised` (
  `raised_id` int(11) NOT NULL AUTO_INCREMENT,
  `user_id` int(11) NOT NULL,
  `action_id` int(11) NOT NULL,
  `link_id` int(11) NOT NULL,
  `raised_type` int(11) NOT NULL,
  PRIMARY KEY (`raised_id`),
  KEY `user_id` (`user_id`),
  KEY `action_id` (`action_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

Constraints for table `action_items_raised`

```
ALTER TABLE `action_items_raised`
  ADD CONSTRAINT `FKitems_raised_action_id` FOREIGN KEY (`action_id`)
        REFERENCES `actions` (`action_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKitems_raised_user_id` FOREIGN KEY (`user_id`)
        REFERENCES `system_user_accounts` (`user_id`) ON DELETE CASCADE;
```

Field Name	Description
raised_id	The primary identification of the raised action.
user_id	The user who raised the action against existing data.
action_id	The action id, for the action that was rose.
link_id	The link id of the record which the action is raised against.
raised_type	The type of the raised action, it will also be used for the determination of which table to query to get the record of the link id.

Training Management System

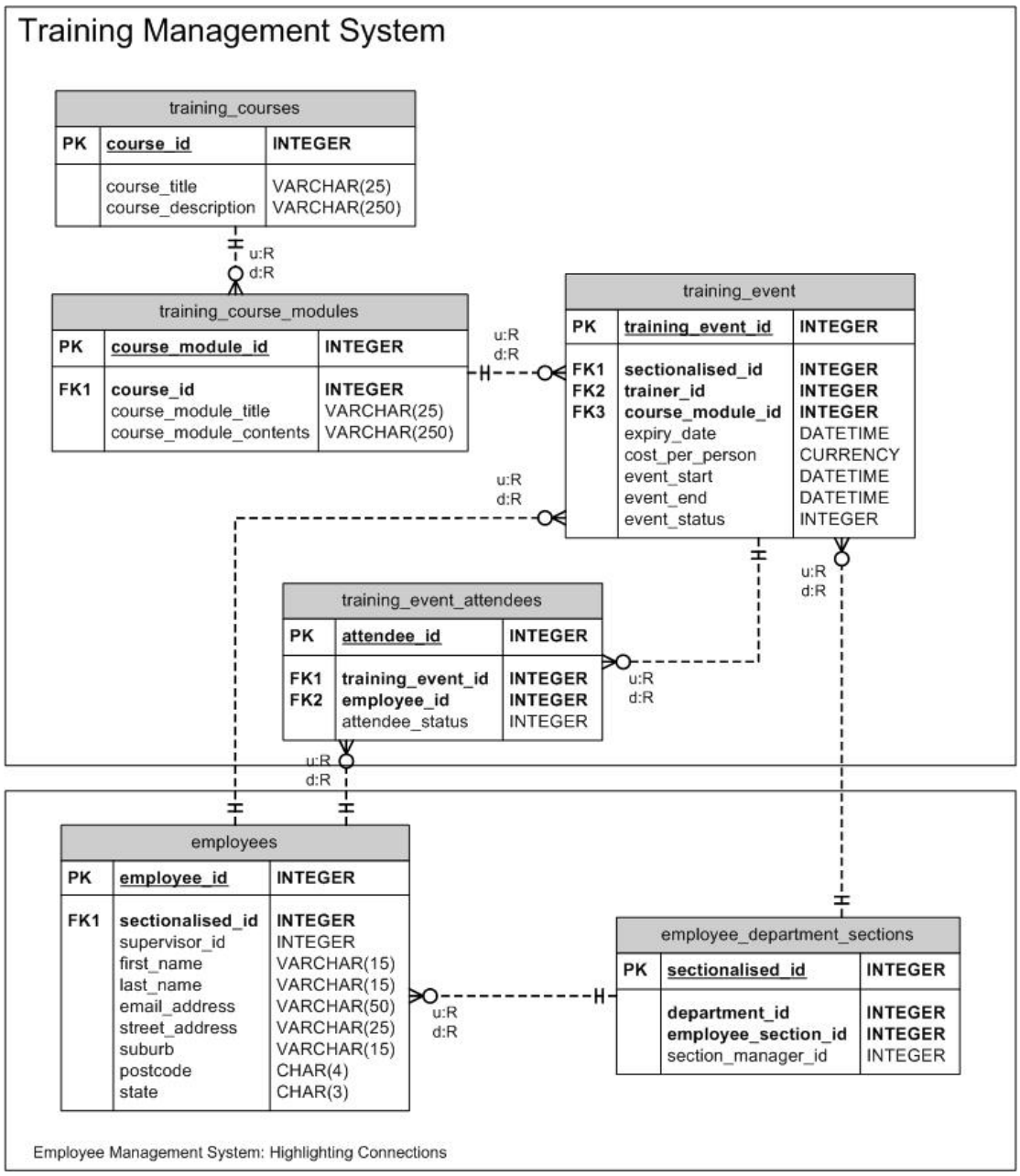


Table structure for table `training_courses`

```
CREATE TABLE IF NOT EXISTS `training_courses` (
  `course_id` int(11) NOT NULL AUTO_INCREMENT,
  `course_title` varchar(25) NOT NULL,
  `course_description` varchar(250) NOT NULL,
  PRIMARY KEY (`course_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Field Name	Description
course_id	The primary identification of the course.
course_title	The course title.
course_description	The description of the course contents.

Table structure for table `training_course_modules`

```
CREATE TABLE IF NOT EXISTS `training_course_modules` (
  `course_module_id` int(11) NOT NULL AUTO_INCREMENT,
  `course_id` int(11) NOT NULL,
  `course_module_title` varchar(25) NOT NULL,
  `course_module_contents` varchar(250) NOT NULL,
  PRIMARY KEY (`course_module_id`),
  KEY `course_id` (`course_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `training_course_modules`

```
ALTER TABLE `training_course_modules`
  ADD CONSTRAINT `FKcourse_modules_course_id` FOREIGN KEY (`course_id`)
        REFERENCES `training_courses` (`course_id`) ON DELETE CASCADE;
```

Field Name	Description
course_module_id	The primary identification of the course module.
course_id	The course which the module belongs to.
course_module_title	The module title.
course_module_contents	The description of the contents of the training module.

Table structure for table `training_events`

```
CREATE TABLE IF NOT EXISTS `training_events` (
  `training_event_id` int(11) NOT NULL AUTO_INCREMENT,
  `sectionalised_id` int(11) NOT NULL,
  `trainer_id` int(11) NOT NULL,
  `course_module_id` int(11) NOT NULL,
  `expiry_date` datetime NOT NULL,
  `cost_per_person` decimal(10,2) NOT NULL,
  `event_start` datetime NOT NULL,
  `event_end` datetime NOT NULL,
  `event_status` int(11) NOT NULL,
  PRIMARY KEY (`training_event_id`),
  KEY `sectionalised_id` (`sectionalised_id`),
  KEY `trainer_id` (`trainer_id`),
  KEY `course_module_id` (`course_module_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ROW_FORMAT=COMPACT
AUTO_INCREMENT=1 ;
```

Constraints for table `training_events`

```
ALTER TABLE `training_events`
  ADD CONSTRAINT `FKevent_course_module_id` FOREIGN KEY (`course_module_id`)
        REFERENCES `training_course_modules` (`course_module_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKevent_sectionalised_id` FOREIGN KEY (`sectionalised_id`)
        REFERENCES `employee_department_sections` (`sectionalised_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKevent_trainer_id` FOREIGN KEY (`trainer_id`)
        REFERENCES `employees` (`employee_id`) ON DELETE CASCADE;
```

Field Name	Description
training_event_id	The primary identification of the training event.
sectionalised_id	The department section of the training course.
trainer_id	The trainer employee file id.
course_module_id	The course module which will be trained against.
expiry_date	The expiry date of the training which the employee has received.
cost_per_person	The cost per person, which will be overall calculated based on the attendees which the course receives.
event_start	The start duration of the course.
event_end	The end duration of the course.
event_status	The current status of the event; whether it is open, or completed.

Table structure for table `training_event_attendees`

```
CREATE TABLE IF NOT EXISTS `training_event_attendees` (
  `attendee_id` int(11) NOT NULL AUTO_INCREMENT,
  `training_event_id` int(11) NOT NULL,
  `employee_id` int(11) NOT NULL,
  `attendee_status` int(11) NOT NULL,
  PRIMARY KEY (`attendee_id`),
  KEY `training_event_id` (`training_event_id`),
  KEY `employee_id` (`employee_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `training_event_attendees`

```
ALTER TABLE `training_event_attendees`
  ADD CONSTRAINT `FKevent_attendees_employee_id` FOREIGN KEY (`employee_id`)
          REFERENCES `employees` (`employee_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKevent_attendees_training_event_id` FOREIGN KEY (`training_event_id`)
          REFERENCES `training_events` (`training_event_id`) ON DELETE CASCADE;
```

Field Name	Description
attendee_id	The primary identification of the attendee who is taking part of the training event scheduled.
training_event_id	Training event for which the attendee is assigned to participate in.
employee_id	The attendee's employee file.
attendee_status	The status of the attendee; whether it has been confirmed of their attendance. Either non-confirmed, or confirmed.

Dynamic Form Creation System

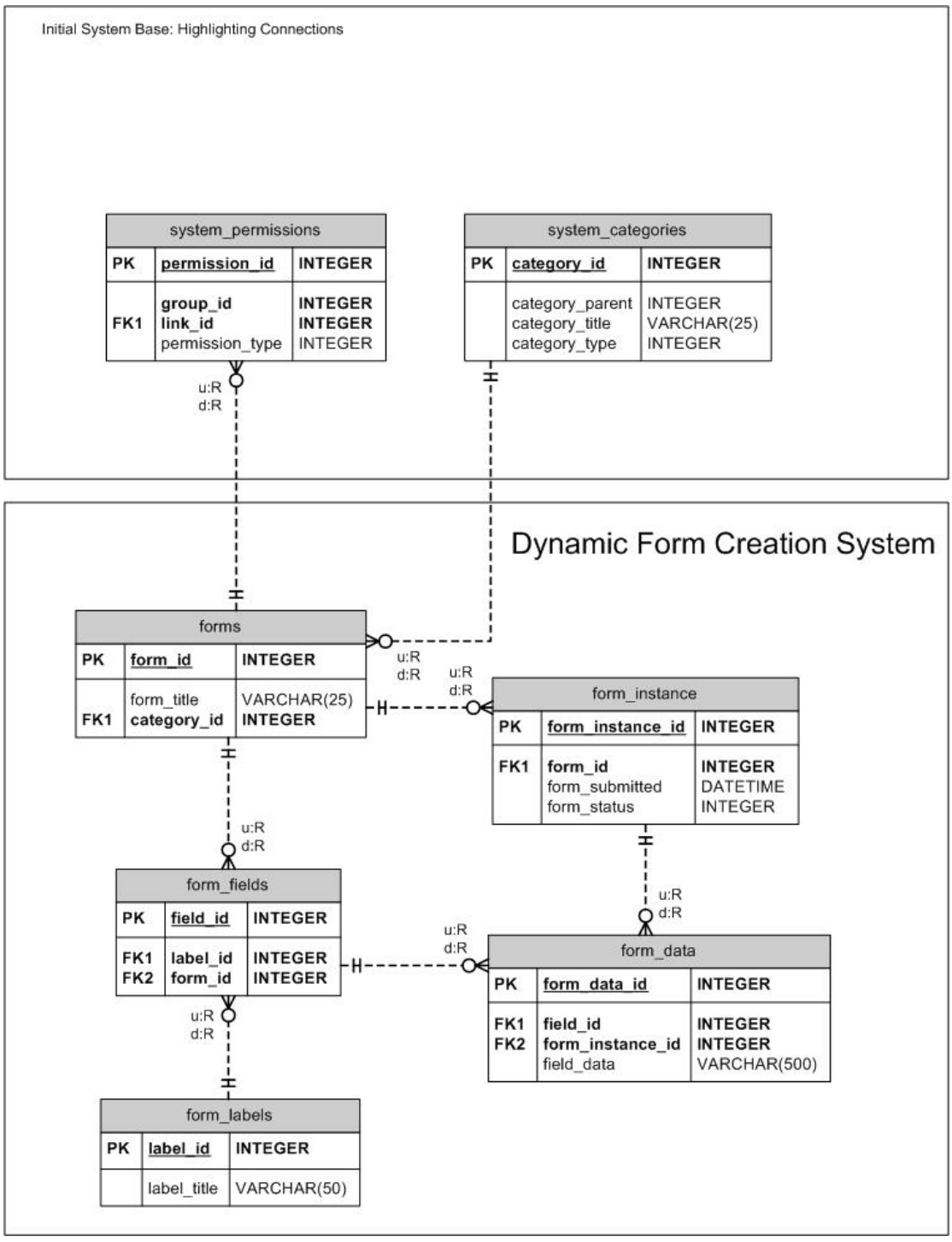


Table structure for table `forms`

```
CREATE TABLE IF NOT EXISTS `forms` (
  `form_id` int(11) NOT NULL AUTO_INCREMENT,
  `category_id` int(11) NOT NULL,
  `form_title` varchar(25) NOT NULL,
  PRIMARY KEY (`form_id`),
  KEY `category_id` (`category_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `forms`

```
ALTER TABLE `forms`
  ADD CONSTRAINT `FKforms_category_id` FOREIGN KEY (`category_id`)
        REFERENCES `system_categories` (`category_id`) ON DELETE CASCADE;
```

Field Name	Description
form_id	The primary identification of the form.
category_id	The category which the form belongs to.
form_title	The title of the form.

Table structure for table `form_fields`

```
CREATE TABLE IF NOT EXISTS `form_fields` (
  `field_id` int(11) NOT NULL AUTO_INCREMENT,
  `label_id` int(11) NOT NULL,
  `form_id` int(11) NOT NULL,
  PRIMARY KEY (`field_id`),
  KEY `FKfields_label_id` (`label_id`),
  KEY `FKfields_form_id` (`form_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `form_fields`

```
ALTER TABLE `form_fields`
  ADD CONSTRAINT `FKfields_label_id` FOREIGN KEY (`label_id`)
        REFERENCES `form_labels` (`label_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKfields_form_id` FOREIGN KEY (`form_id`)
REFERENCES `forms` (`form_id`) ON DELETE CASCADE;
```

Field Name	Description
field_id	The primary identification of the form field.
label_id	The label that the field is assigned.
form_id	The form that the field belongs to.

Table structure for table `form_data`

```
CREATE TABLE IF NOT EXISTS `form_data` (
  `form_data_id` int(11) NOT NULL AUTO_INCREMENT,
  `field_id` int(11) NOT NULL,
  `form_instance_id` int(11) NOT NULL,
  `field_data` varchar(500) NOT NULL,
  PRIMARY KEY (`form_data_id`),
  KEY `field_id` (`field_id`),
  KEY `form_instance_id` (`form_instance_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Constraints for table `form_data`

```
ALTER TABLE `form_data`
  ADD CONSTRAINT `FKdata_field_id` FOREIGN KEY (`field_id`)
        REFERENCES `form_fields` (`field_id`) ON DELETE CASCADE,
  ADD CONSTRAINT `FKdata_form_instance_id` FOREIGN KEY (`form_instance_id`)
        REFERENCES `form_instances` (`form_instance_id`) ON DELETE CASCADE;
```

Field Name	Description
form_data_id	The primary identification of the form submission data.
field_id	The field that corresponds to the data recorded.
form_instance_id	The instance of the submission that the data belongs to.
field_data	The field data.

Table structure for table `form_instances`

```
CREATE TABLE IF NOT EXISTS `form_instances` (
  `form_instance_id` int(11) NOT NULL AUTO_INCREMENT,
  `form_id` int(11) NOT NULL,
  `form_submission_date` datetime NOT NULL,
  `form_status` int(11) NOT NULL,
  PRIMARY KEY (`form_instance_id`),
  KEY `form_id` (`form_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ROW_FORMAT=COMPACT
AUTO_INCREMENT=1 ;
```

Constraints for table `form_instances`

```
ALTER TABLE `form_instances`
  ADD CONSTRAINT `FKforms_form_id` FOREIGN KEY (`form_id`)
        REFERENCES `forms` (`form_id`) ON DELETE CASCADE;
```

Field Name	Description
form_instance_id	The primary identification of the form instance.
form_id	The form that the instance of the submission corresponds with.
form_submission_date	The date the form is submitted.
form_status	The status of the submission; whether it is open, pending, or closed.

Table structure for table `form_labels`

```
CREATE TABLE IF NOT EXISTS `form_labels` (
  `label_id` int(11) NOT NULL AUTO_INCREMENT,
  `label_title` varchar(50) NOT NULL,
  PRIMARY KEY (`label_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Field Name	Description
label_id	The primary identification of the form label.
label_title	The title/ description of the field.