COSI-Framework v1.0 Documentation

Document History

James Blanden	13 November 2007	Version 1.0	Initial document.

Table of Contents

Background **2**
Document Scope **2**
Requirements **3**
Overview **4**
Functional Specification **5**
Menus and Activities 5
Contextual Help 6
Authentication 6
Roles and Authorisation 6
User Roles 7
Functional Roles 7
Organisational Roles 8
The Administration Application 8
List Roles 8
Add Role 9
View Role 9
Edit Role 11
Delete Role 12
Brief Technical Overview **13**
System Requirements 13
File Structure 13
Database Structure 15
DHTML Controls 16
The *Please Wait* Control 16
The *Calendar* Control 16

Background

The COSI-Framework is a PHP/PostgreSQL web application framework which has been designed and built to provide shared and consistent functionality across the projects of the Australian Partnership For Sustainable Repositories (APSR) Collection Services and Infrastructures (COSI) initiatives of 2007. More information about these initiatives can be found at the APSR web site http://www.apsr.edu.au/currentprojects/index.htm#cosi.

Document Scope

This document describes the framework only. It does not describe the specifics of software that has been built to utilise it.

Installation instructions are provided in the software download, and so are not repeated here.

This document does not detail the complete Application Programming Interface (API) of the COSI-Framework. The information here is related to the structure and basic concepts behind the framework, it is not intended as a developer's guide. Developers are encouraged to get hands-on with the source code, and that of the COSI project applications, to gain an understanding of how things hang together at that level.

A general knowledge of web application design and technologies is assumed.

Requirements

There was no formal business analysis or requirements gathering undertaken. The architecture and design were driven by informal discussions between developers and managers of the COSI projects. This resulted in the following set of requirements:

- The framework should provide for menuing and navigation to the functionality delivered by the COSI project software.
- The framework should provide for functional roles based restrictions to functionality.
- The framework should provide support for organisational and user roles based restrictions to data.
- The framework should provide a built-in mechanism for authenticating users.
- The framework should provide a mechanism for authenticating users against an external source using Lightweight Directory Access Protocol (LDAP).
- The framework should provide a mechanism for authenticating users against an external source using Shiboleth.
- The framework should provide an Application Programming Interface (API) to enable all project software to be delivered with a consistent user interface.
- The framework should provide an API to simplify interacting via REST (in this context, simple XML over HTTP) interfaces with external applications. This would allow COSI project software to optionally be hosted externally on any platform, but be utilised from within the framework alongside associated groups of functionality.
- The framework should be built using free software, and be made to be as portable and cross-platform as possible.
- Different instances of the framework should be able to be easily branded, while still being recognisable as an instance of the framework.
- The framework should work in all World Wide Web Consortium (W3C) standards compliant web browsers without requiring the download or installation of additional software. Also, the framework should function adequately in less standards compliant but common web browsers such as Internet Explorer 6 for Windows.
- The framework should meet the basic W3C standards for accessibility.
- The framework should provide an application for administering roles and authorisations via a web based user interface.
- The framework should provide developers with useful error and debug information.
- The framework should be as secure and robust as possible including: expiry of sessions after a defined period of inactivity; expire sessions at logout and re-issue at login; bind sessions to IP address and user agent; only use cookies for session handling; provide a production deployment mode that will hide detailed warning and error messages from users; log details of warnings and errors; support SSL for LDAP, REST and client interaction; prevent SQL injection attacks by providing a database access API that utilises parameterised queries; prevent cross site scripting attacks by providing a presentation layer API that escapes unsafe strings for presentation; and more.

Overview

The COSI-Framework is a PHP web application that utilises a PostgreSQL database to store information about roles, activities, and authorisations in order to control access to pages (and therefore the activities or functionality) contained within it.

The framework is designed to house applications that utilise its API to deliver a web-based user interface. These applications can take one of two basic forms: they can be completely written in PHP (and any supporting database languages); or they can exist as the combination of an external (to the framework) application and an internal component (or integration point) written in PHP. In the latter case the integration point exists to present the user interface and to mediate access to the external application.

The combination of an external application and PHP integration point, free the developer of the external application to use any technology and platform they wish (it simply needs to expose REST interfaces); to locate the external application wherever they wish; and from the burden of supporting authentication and functional authorisations. There are three important points to note about this scenario. The first, is that the channel for communication of the REST interfaces between the framework and the external application needs be secured with SSL to prevent eavesdropping and to prevent impersonation attacks from accessing the external application directly. The second, is that the external application will need to be configured to only accept requests from trusted sources (the integration point). The third, is that support for organisational or user roles based access to data in the external application will still necessarily need to be provided by the external application. These issues will not apply if no access controls are required, as in the case of simply reading publicly available data. How these requirements are achieved is beyond the scope of this document.

Figure 1 (below) represents the structure described above, and includes a self-contained PHP application (the built-in COSI Administration), a PHP/PostgreSQL application (ORCA-Registry) and some applications comprising a PHP user interface and middleware communicating via REST interfaces to external applications (AONS and RIFF). Of course, any combination of these architectures can be used, such as PHP applications with supporting databases and combining user interface and interaction with any number of external applications via REST or other means—basically forming a web application ‘mashup’. This diagram also indicates how authentication is handled either by communication with an external service or by the built-in service. (Shiboleth support is yet to be implemented.)

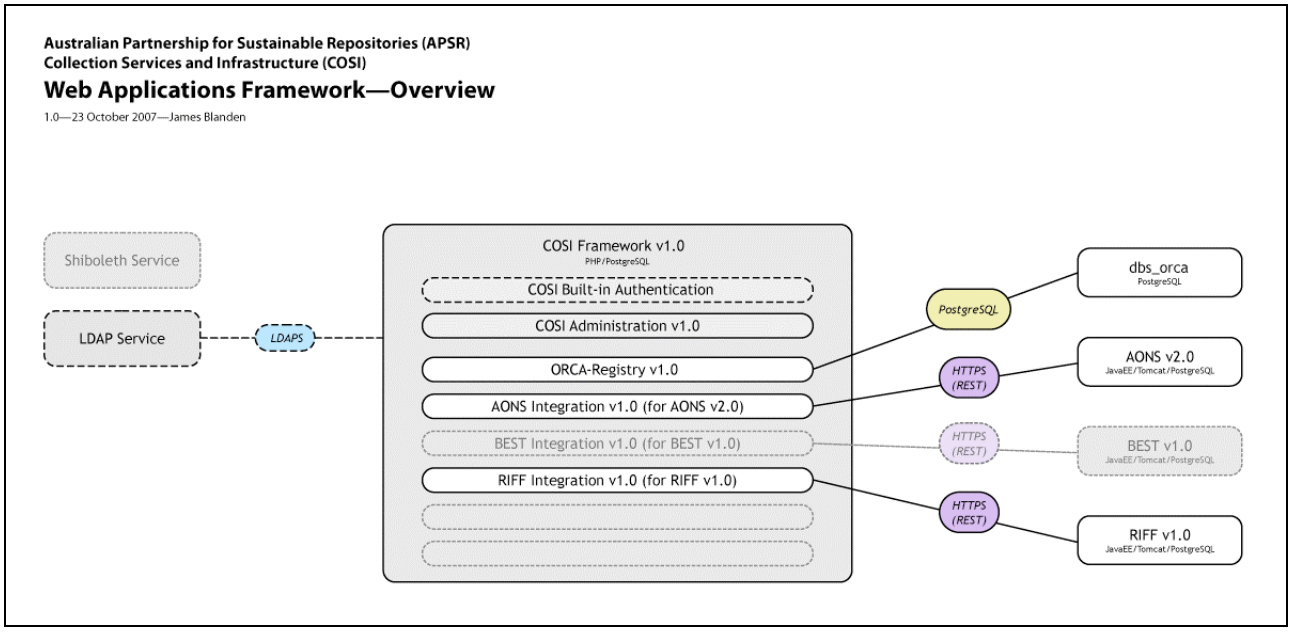

Figure 1: Overview of the COSI-Framework architecture.

Functional Specification

Menus and Activities

Navigation to the functionality contained within an instance of the framework is controlled by the definition of menus and activities. A menu is a container for both sub-menus and activities. Activities correspond to PHP pages within the framework. Menus are identified by a triangular icon that also indicates if the menu is expanded or collapsed, and if the currently selected activity is contained within it (a solid triangle). Activities are identified by a circular icon that also indicates if it's the currently selected activity (a solid circle). Lines connecting menu icons in a path to the currently selected activity icon identify the location of that activity in the menu hierarchy.

Clicking a menu icon will toggle between the current (expanded or collapsed) state of the menu. Activity icons are links, and so clicking them will navigate the web browser to the associated PHP page. It is possible to configure activities that are direct links to locations external to the framework—these are identified by an activity icon with an arrow pointing northeast.

Menus are initially all collapsed. The collapsed or expanded state of menus in the structure is stored in a browser cookie, and so is remembered for a particular browser between sessions.

Figure 2 (below) shows a screenshot of an example instance of the framework. In this example, all of the menus are expanded, and the *Add Data Source* activity is selected.

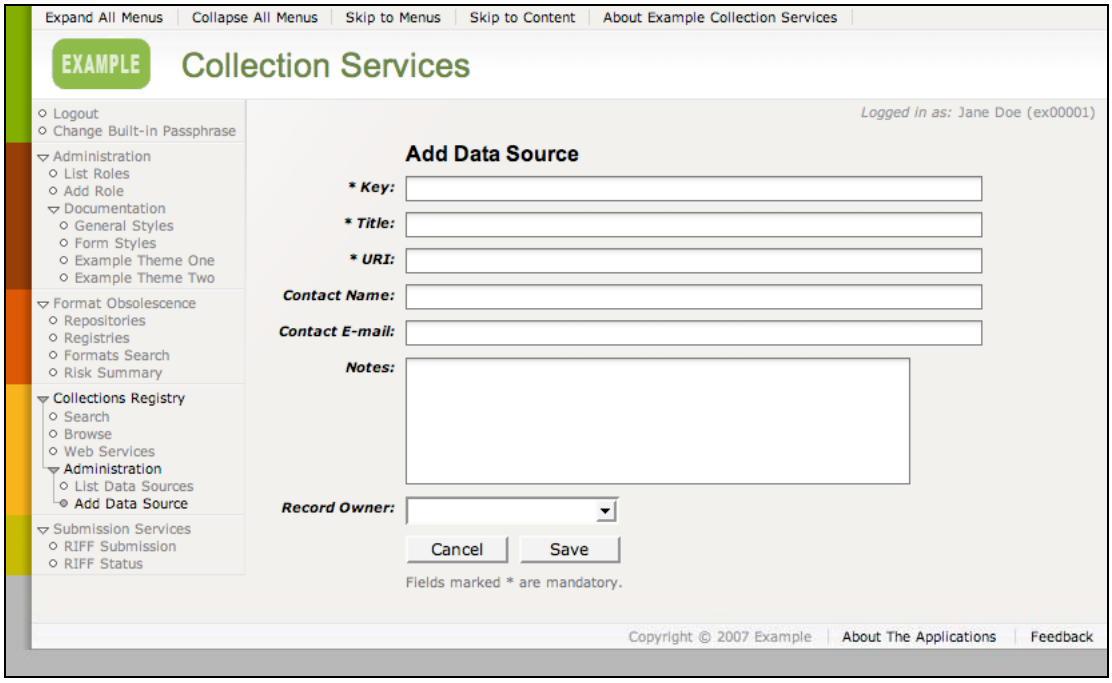

Figure 2: Screenshot from an example instance of the COSI-Framework.

Activities can optionally be configured to not be displayed in the navigation, or to be displayed only when they are active. This is to allow for activities that can't have a direct link to them from the navigation—usually because they are returning a data record and need an identifier for the record to be passed them by an intermediate page (like a list of records).

Which activities are displayed in the navigation will also depend on the authorisations required to access them. Menus will always (and only) be displayed if they contain anywhere within them an activity that is currently accessible to the user.

Contextual Help

The framework provides support for the linking of a help page to an activity. Any activities that are configured as having a help page available will display a link (*help*) to this page towards the top right of the user interface. Pages for which no help is available will not display the link.

Access to the help for an activity requires the same authorisations as the activity.

Authentication

Authentication in this context is the process of identifying a user by having them provide credentials, and then verifying those credentials against an authoritative source (either the source that issued and maintains them, or a trusted affiliated source).

Names and identifiers for users of an instance of the framework are stored in the framework database. Credentials take the form of the *User ID* and *Passphrase* supplied by the user via the login form. The framework database also stores information about the authoritative source to use for the verification of a particular user's credentials. The framework provides built-in services for holding and verifying credentials as well as providing support for the use of external verification via LDAP (over SSL). Future versions may add support for the use of Shiboleth for verification.

Users who have their identity verified via the login process will then be granted the appropriate authorisations for the duration of their session. Their session will expire either when they logout, or after a fixed period of inactivity (i.e. no page requests). The default session timeout is 120 minutes.

Logged in users will see their *Name* and *User ID* towards the top right of the user interface, and can logout by clicking on the *Logout* activity (which changed from *Login* when they logged in) towards the top left of the user interface.

Roles and Authorisation

Access to every activity within the framework is controlled by the required authorisations for that activity. No activity is accessible without a defined authorisation for it existing in the database and the requesting user meeting the requirements of that authorisation—i.e. being the member (either directly or indirectly) of a functional role that is defined as having access to that activity.

All users, including anonymous (unverified) users, have a default level of *Public* access.

Activities for which a user does not have access will not appear in the navigation. Menus that contain no activities for which a user has access will not appear in the navigation. A request for a page for which a user does not have access, will result in the termination of their session and the redirection of their browser to the login page.

If the login page was reached via redirection from a page that was not accessible (as in the case of a restricted page being requested via a bookmark before the user has logged in), it will remember the page that redirected to it. Then, if the subsequent login is successful and the user is authorised to access that page, the login page will redirect back to the originally requested page.

Authorisations are controlled by the membership of roles, and the association of roles with activities (and with records within applications). The framework utilises three types of role.

Roles that aren't flagged as being enabled in the database will be treated as though they don't exist for the purpose of determining authorisations. Note that this means that the child roles of disabled roles will effectively be orphaned in that branch of the hierarchy, and also treated as though they don't exist (in that branch of the hierarchy).

Authorisations for an activity can be checked and used to provide granular access or visibility to components within another activity. For example to not display the *Edit* button on a *View* record page for users that don't have access to the *Edit* activity (otherwise, the clicking of which would just result in the logging out of that user).

User Roles

A user role represents an individual user. Applications within the framework can optionally store user role information and use it to restrict access to data. For example, to only let a user modify records flagged as being owned by that user.

User roles are the only roles that can login/authenticate.

The *Name* of a user role will be the name of the person that it identifies. The *ID* of the user must be the identifier used by the authentication source selected for that user role. It is recommended to use an e-mail address as the *ID* for users of the built-in authentication, as this is unique and descriptive.

A user role can be a member of functional roles and organisational roles. Roles inherit the authorisations of every role that they are a member of.

Functional Roles

A functional role represents authorisations for a collection of activities. Activities are defined in the framework application configuration file and represented in the framework database. *Role-Activity* records in the database define functional authorisations.

The *Name* of a functional role should describe the role that a person with that collection of authorisations (i.e. a member of the role) would be undertaking. For example, an application administrator might need to list, view, add, and edit all records within that application. A functional role named *Example Application Administrator* would be created with authorisations for all of those activities, and then the user roles for people who are required to administer that application would be granted membership of that functional role (and so inherit those authorisations).

The *ID* of a functional role can be anything unique to an instance of the framework. It is recommended that functional roles be given descriptive *ID*s to make roles administration easier. The recommended functional *ID* style is *APPLICATION_ROLEDESCRIPTION*, for example *COSI_ADMIN* for the *COSI Administrator* functional role, or *ORCA_ADMIN* for the *ORCA Administrator* functional role.

A functional role can be a member of other functional roles (and so inherit those authorisations). This enables a hierarchy of authorisation to be defined. This has the potential to simplify management of authorisations, but also to build structures that are very complicated and not readily transparent (so have a think about what you're doing). The COSI Administration application doesn't allow the creation of circular references within the roles structure, but application developers need to be careful to avoid this (when inserting roles directly into the database on installation).

The framework uses two special functional roles that aren't visible in the administration application, *PUBLIC*, and *COSI_BUILT_IN_USERS*. The *PUBLIC* role is a default anonymous role that is assumed for every request. Every activity must at the very least have an authorisation for the *PUBLIC* role defined in the database in order for it to be accessible. The *COSI_BUILT_IN_USERS* role is automatically given to users who have authenticated using the built-in authentication service, and exists to provide access to the *Change Built-in Passphrase* activity for those users.

Organisational Roles

An organisational role represents an organisational entity or group of people (company, division, department, school, faculty, etc). Applications within the framework can optionally store organisational role information and use it to restrict access to data. For example, to only let members of an organisation see records flagged as being owned by that organisation.

The *Name* of an organisational role should describe the entity that it represents, and it is recommended that a fully qualified name be used to prevent ambiguity and confusion with other roles. For example use *Company X Accounting* instead of just *Accounting* to prevent confusion with the accounting departments of other companies. The *ID* of an organisational role can be anything unique to an instance of the framework. The recommended style for organisational *ID*s is similar to that for functional roles, but using colons as separators between abbreviations for the areas in the qualified name. For example *COMPX:ACCTNG*.

An organisational role can be a member of other organisational roles (and so inherit their authorisations).

The Administration Application

The framework is provided with an application for managing roles and authorisations. This administration application provides for the listing, viewing, adding, editing, and deleting of user, functional, and organisational roles. Any changes made to roles (including changes to authorisations) are effective immediately (i.e. they will be applied at a user's next page request, and are not postponed until a user's next session).

List Roles

The *List Roles* activity displays a list of all of the roles in the framework database. The list can be sorted by column in either direction by clicking on the column headers. The list will be paginated if there are more records than can be displayed on one page. Clicking on a record will take the user to the *View Role* activity for that record. The role list can be filtered on *ID*, *Name*, *Type*, or *Auth. Service* using the form at the top of the page.

Figure 3 (below) shows an example of the *List Roles* activity.

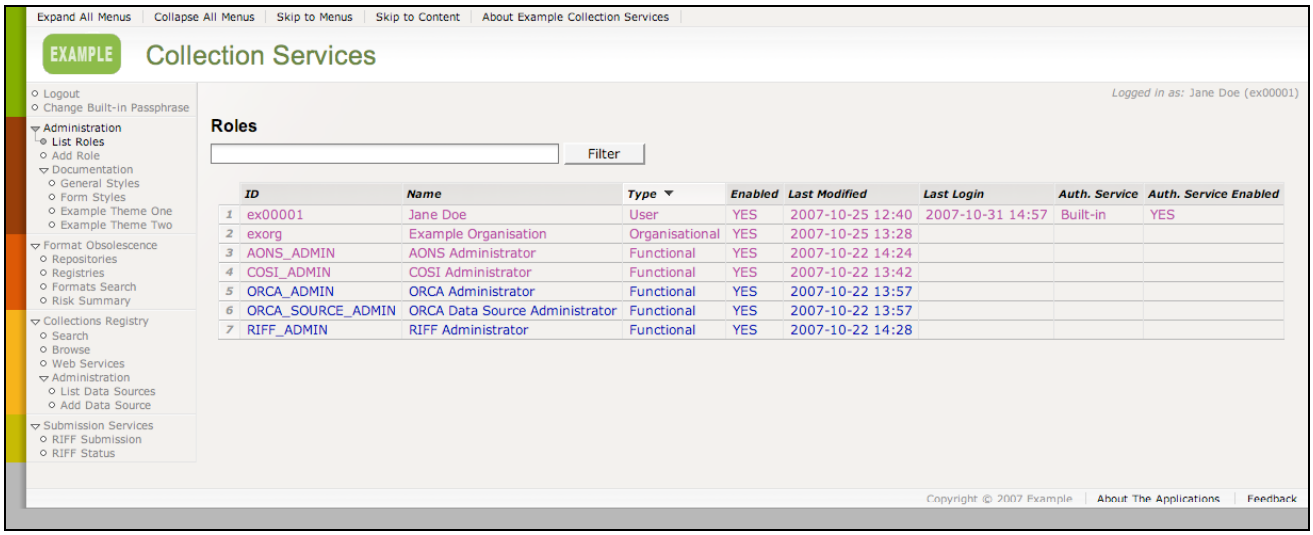

Figure 3: COSI Administration, *List Roles* example.

Add Role

The *Add Role* activity displays a form for adding a new role to the framework database. The form indicates that *ID*, *Name* and *Type* are mandatory, and also that the *Authentication Service* is only required for roles of type *User* (in which case it is mandatory). *ID* must be unique for an instance of the framework, and for a user role that is using an external source of authentication, must be known by (and identical to the value in) that source.

Upon successfully saving the role, the user will be taken to the *View Role* activity for the new record.

Figure 4 (below) shows an example of the *Add Role* activity.

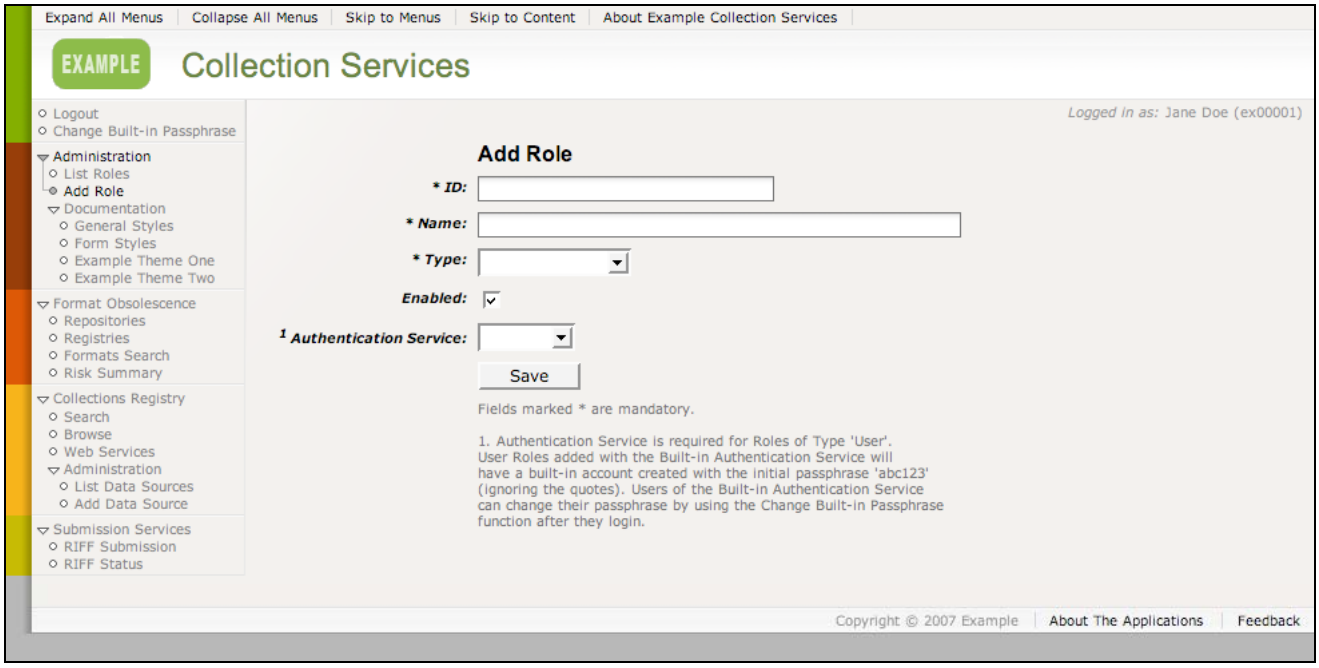

Figure 4: COSI Administration, *Add Role*.

View Role

The *View Role* activity displays detailed information for a role. Buttons are provided to take the user to the *Edit Role* and *Delete Role* activities for this record.

The bottom part of the page (below the *Edit* and *Delete* buttons) details the roles of which this role is a member and provides means for modifying role membership (via *add* and *remove* buttons and associated drop lists). Depending on the type of role being displayed there may also be provision for modifying (via *add* and *remove* buttons and associated drop lists) the activities for which membership of this role authorises access (in the case of functional roles); and presentation of any and all of the roles that are descendants of this role, and therefore inherit its authorisations (in the case of both functional and organisational roles).

Related roles are displayed as links which when followed will take the user to the *View Role* activity for that record.

Figures 5, 6, and 7 (below) show examples of the *View Role* activity for each type of role.

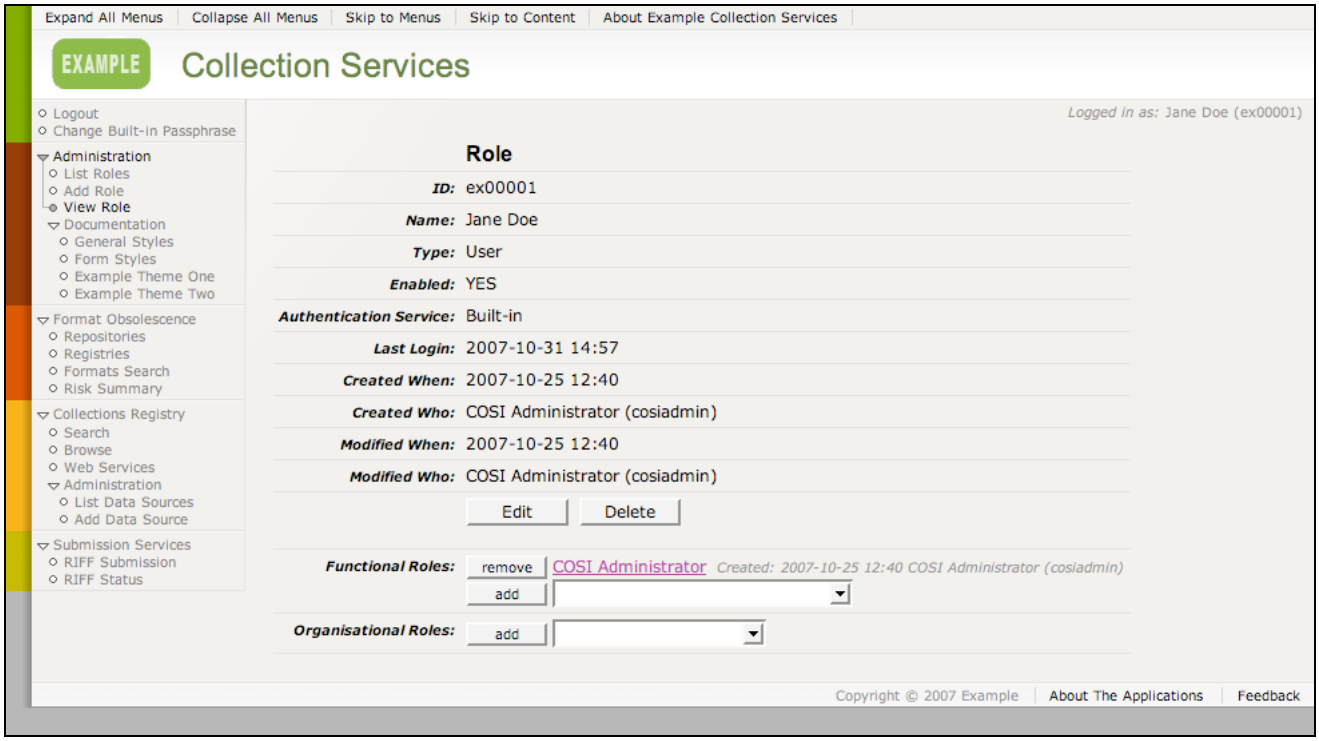

Figure 5: COSI Administration, *View Role*, user role example.

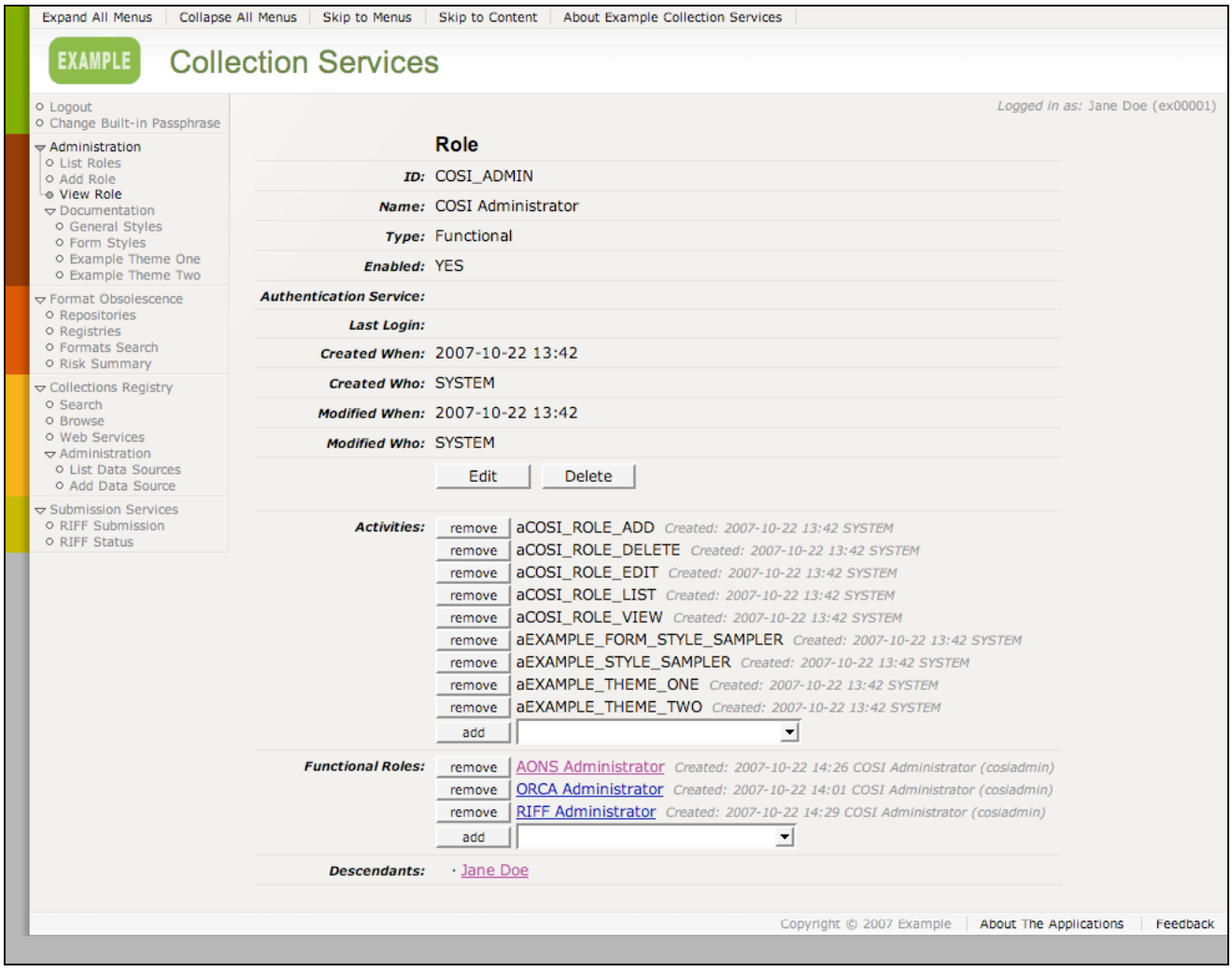

Figure 6: COSI Administration, *View Role*, functional role example.

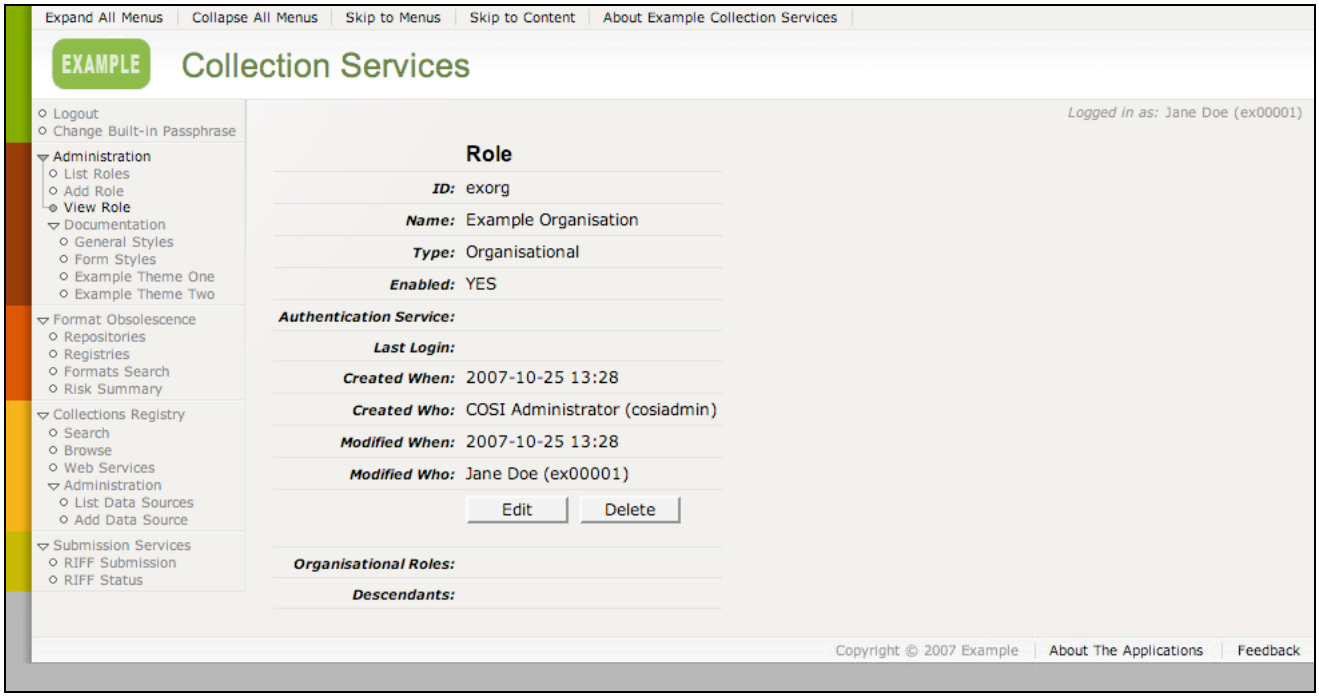

Figure 7: COSI Administration, *View Role*, organisational role example.

Edit Role

The *Edit Role* activity provides a form for modifying an existing role. The role *ID* cannot be changed—if there is an error with the role *ID* then the record must be deleted and a new one with the correct *ID* created using the *Add Role* activity. After successfully saving the changes, the user will be taken to the *View Role* activity for that record. Clicking the *Cancel* button will take the user to the *View Role* activity for that record (without saving any changes).

Figure 8 (below) shows an example of the *Edit Role* activity.

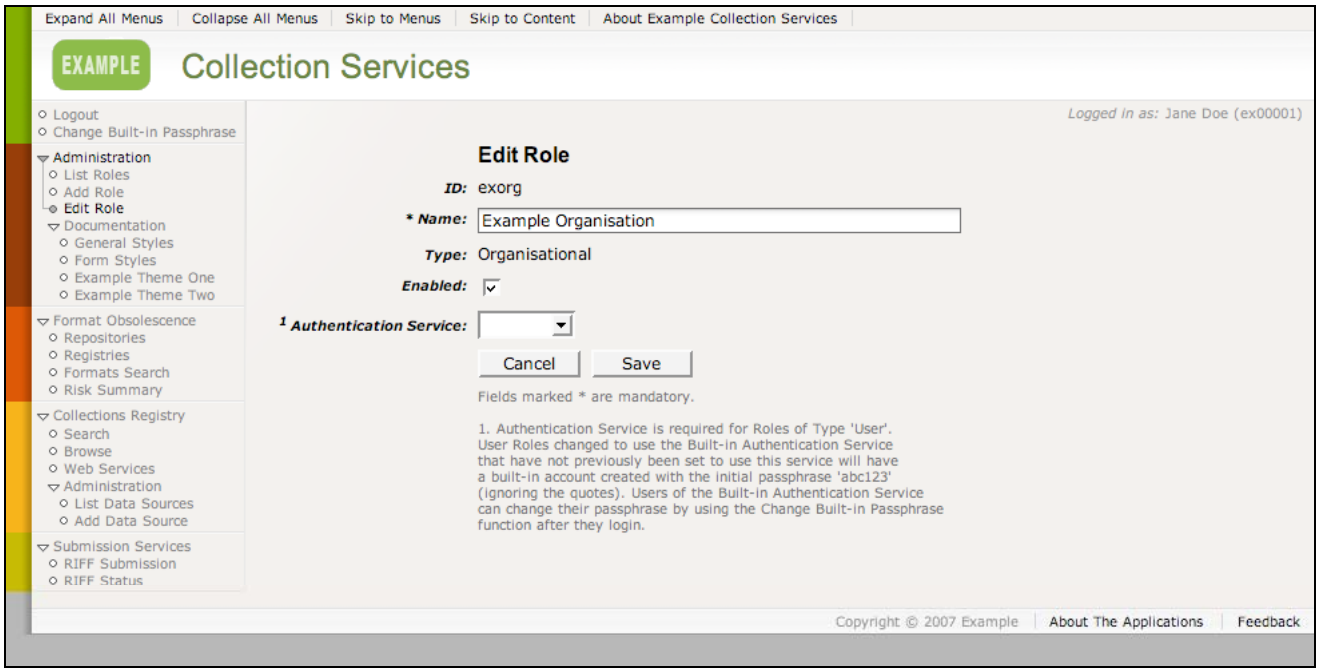

Figure 8: COSI Administration, *Edit Role* example.

Delete Role

The *Delete Role* activity prompts for confirmation that an activity is to be deleted. Clicking the *Delete* button will remove the identified activity, and records of its relation to other roles, from the database, and then take the user to the *List Roles* activity. Clicking the *Cancel* button will take the user to the *View Role* activity for that record (without deleting any records).

Figure 9 (below) shows an example of the *Delete Role* activity.

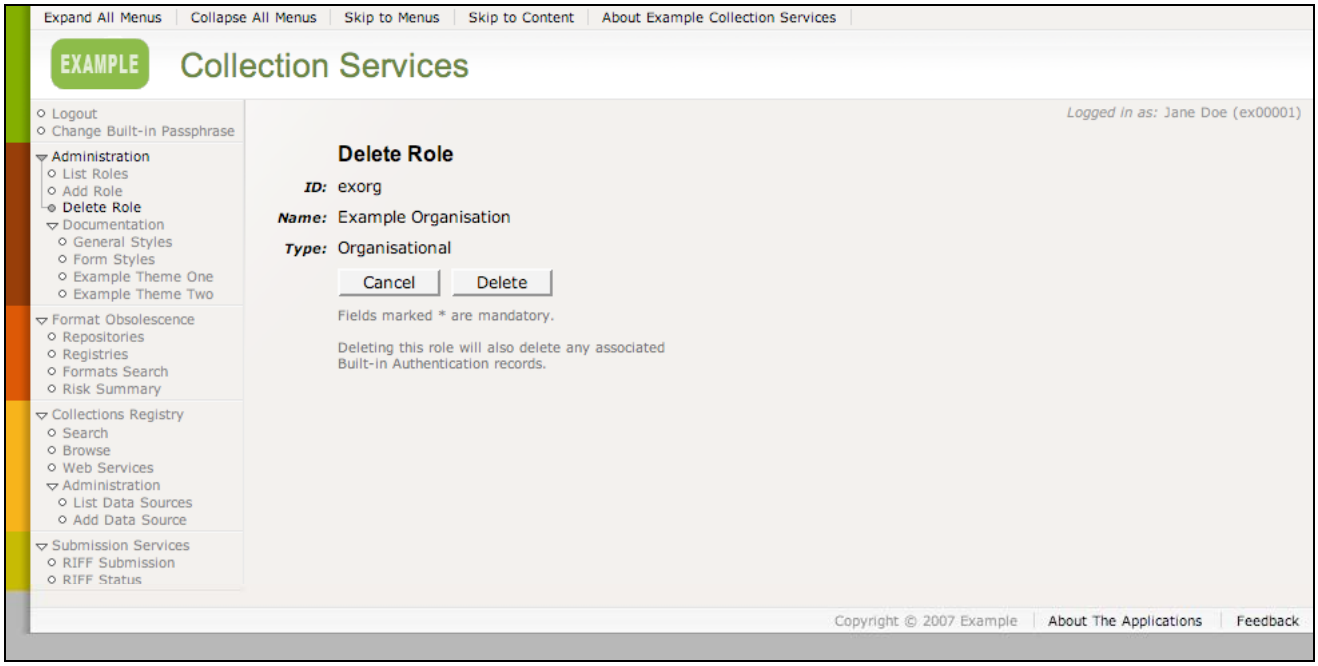

Figure 9: COSI Administration, *Delete Role* example.

Brief Technical Overview

System Requirements

The COSI-Framework is a PHP/PostgreSQL web application. It was developed and tested running under Apache HTTP Server version 2.2.4, with PostgreSQL version 8.2.3, and PHP version 5.2.1 (configured with pgsql, libxml, xsl, ldap, and openssl). It has been installed successfully on Red Hat Linux version 9, and Mac OS X versions 10.4 and 10.5. The framework has been built avoiding the use of platform specific libraries, and so it should work on operating systems other than those on which it was tested—potentially even Windows, and maybe even under the Internet Information Server (IIS) web server.

Web browser requirements basically consist of support for recent World Wide Web Consortium (W3C) standards, and support for cookies. Javascript support will enable expanding/collapsing menus, and the use of the DHTML controls, but is not essential.

The framework has been tested successfully on the following web browsers:

- Firefox 2 on Mac OS X 10.4
- Safari 2 on Mac OS X 10.4
- Safari 3 on Mac OS X 10.5
- Firefox 2 on Windows XP SP2
- Internet Explorer 7 on Windows XP SP2

The framework has also been tested on (and functions correctly with) Internet Explorer 6 on Windows XP SP2 (IE6). However, IE6 has limited support for Cascading Style Sheets (CSS) and Portable Network Graphics (PNG) images, so some page elements are not displayed properly.

File Structure

Files at the root level of the install, along with those in directories prefixed with an underscore (‘_’), form the framework proper. Applications housed within the framework (including the COSI Administration application in the *admin* directory) are located in directories at the root level of the install that are not prefixed with an underscore.

Figure 10 (below) shows an example file structure for a COSI Framework install.

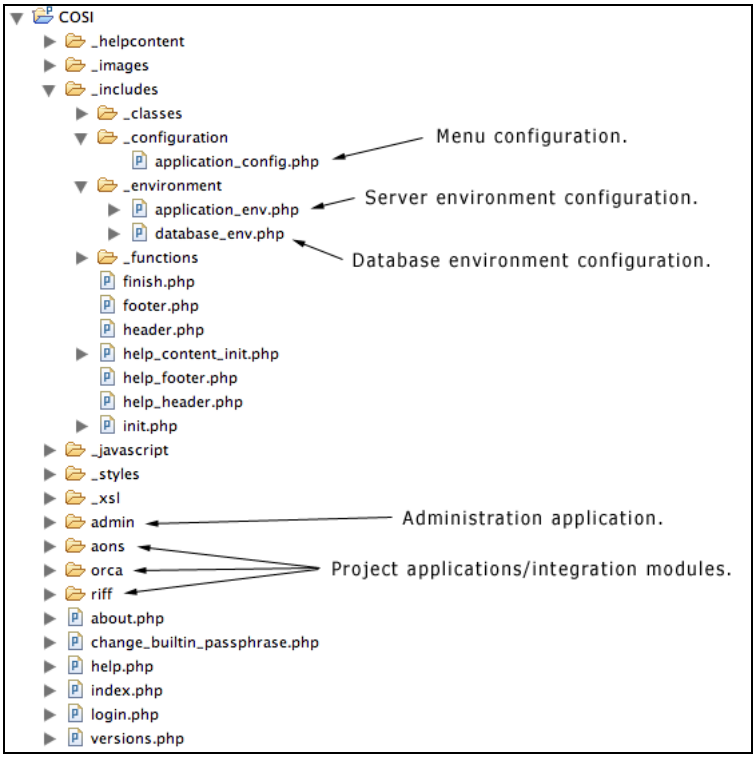

Figure 10: COSI-Framework file structure example.

Within the *_includes* directory are the directories *_configuration* and *_environment*. These house files that control the application configuration (menus, activities, and navigation), and define environment settings for an instance of the framework (refer to the installation instructions contained in the software download for more information).

The file structure of an application housed in the framework mimics that of the framework. Activities are defined in files at the application's root, and within folders that are not prefixed with an underscore. Folders that are prefixed with an underscore are used to hold function libraries, help content etc.

Figure 11 (below) shows an example file structure for an application housed within the framework.

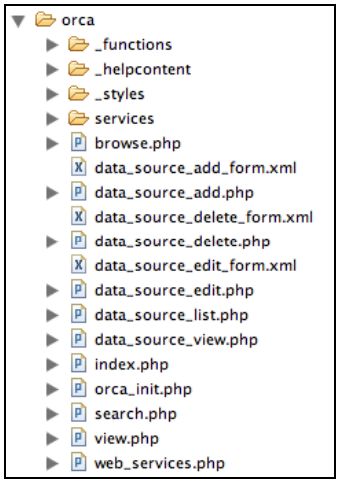

Figure 11: COSI project application file structure example.

Database Structure

The framework database consists of a small number of tables and views that are used to store information about roles and activities. All framework access to the database is through parameterised calls to user-defined functions.

Figure 12 (below) shows the table structure of the database (including constraints).

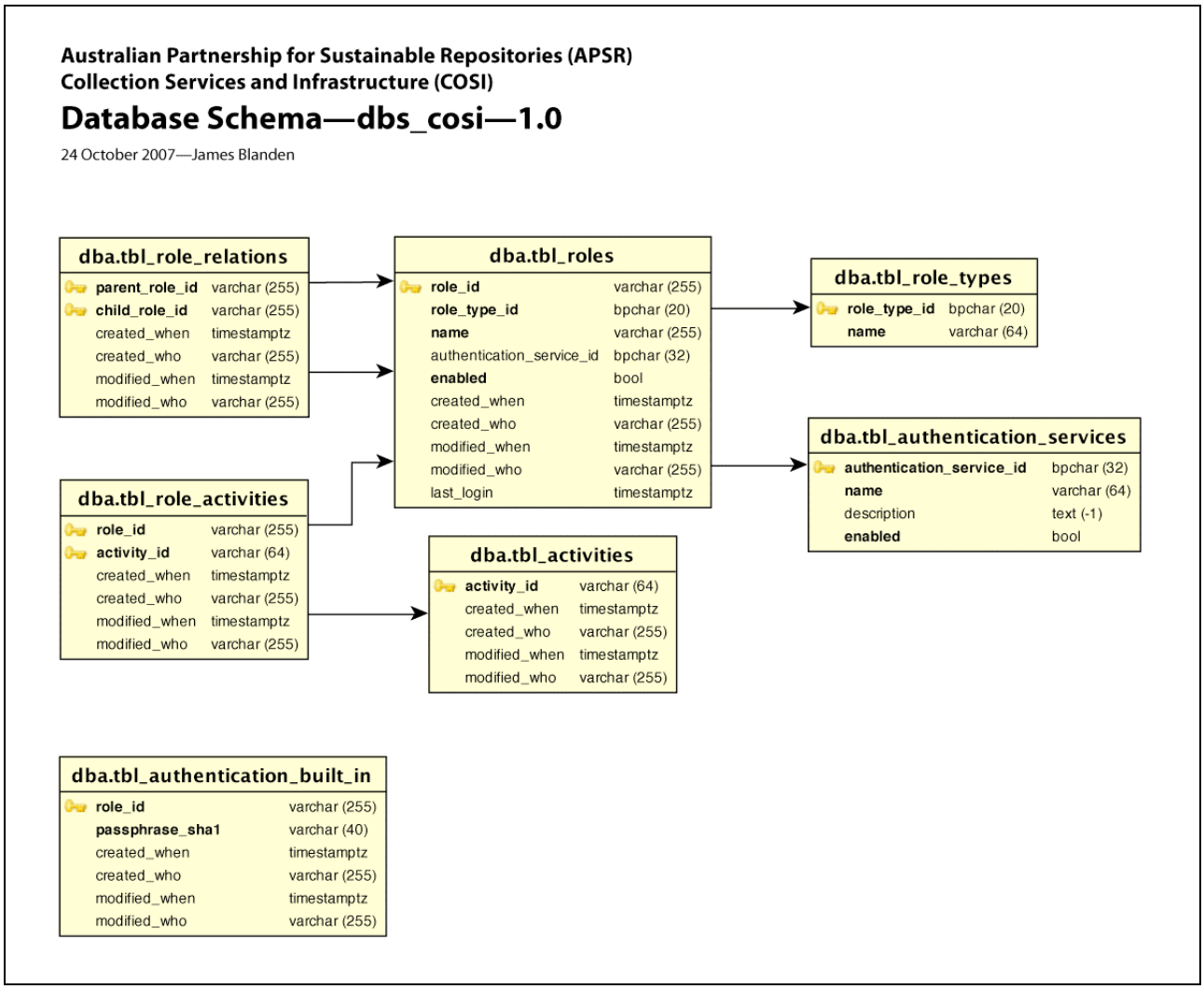

Figure 12: COSI-Framework database structure.

DHTML Controls

The framework provides two DHTML controls that can be utilised by applications housed within it to enhance the user experience.

The *Please Wait* Control

Page requests may take a while to process before a response is issued. Web browsers often display some indication that activity is occurring (a "spinner" animation or similar), but it can be nice to provide additional information to the user—such as how long they've been waiting and what process is being undertaken—the *Please Wait* control serves this purpose.

The *Please Wait* control is sitting invisible (and not taking any space: *display = none*) on every page. By utilising an *onsubmit* or *onclick* event at the instant of a page request (a form submission for example) that it is known may take some time to process (a search request for example), the developer can make the *Please Wait* control visible and begin an animation that indicates to the user that processing is being undertaken. Optionally, the *Please Wait* control can display the time that has elapsed since the request was made. If the request is terminated by the user (by pressing the browser's stop button for example), the *Please Wait* control can be hidden again by clicking the **X** in the top right hand corner.

Figure 13 (below) shows an example of the *Please Wait* control displaying the elapsed time and a message indicating the processing being undertaking is 'Importing…'.

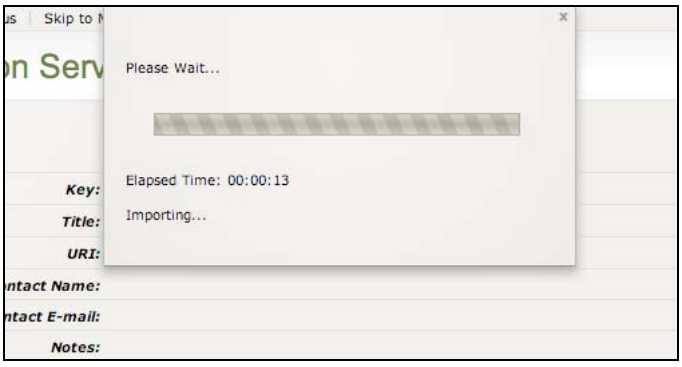

Figure 13: An example of the *Please Wait* dialog.

The *Calendar* Control

The framework provides developers with a *Calendar* control to assist users with the input of dates and times in form fields.

The *Calendar* control is sitting invisible (and not taking any space: *display = none*) on every page. Developers can create a form field that has an associated *Calendar* control for inputting time, date, or date and time values. These fields have an associated *Calendar* control icon, which when clicked will display the *Calendar* control; hide it again if it is already visible; or move it to the selected field if it was visible next to another field (and so bind input to the new field). Date and time values can still be entered manually into fields that provide the *Calendar* control (i.e. without using the control), or if Javascript is disabled—it is simply provided as a convenience.

On its display (by clicking the control icon) the *Calendar* control will populate its associated field with current values for time, date or both (depending on the mode in which it is operating). The *Calendar* control can be moved by clicking and dragging on the bar at its top and can be hidden by clicking the **X** at its top right. In time only mode the control must be hidden again by either clicking the icon, or the **X**. In date only or date and time mode, the control will be hidden when a date is selected by clicking within the calendar part of the control. The current date is highlighted with a

green background in the calendar. Months and Years can be navigated to by using the drop lists, or by clicking the previous and next months on either side of the current month. Clicking the clockwise circular arrow icon (↻) will return the calendar to the current month. Clicking the day of the week column headers will re-order the days of the week to start from the day selected. Working examples of the *Calendar* control can be found in the documentation section of the administration application (*Administration* > *Documentation* > *Form Styles*), look at the third form down (titled 'Test Form').

Figures 14, 15 and 16 (below) show examples of the *Calendar Control* operating in *time*, *date*, and *datetime* modes.

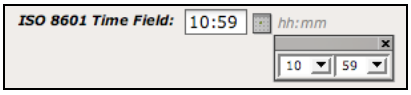

Figure 14: An example of the *Calendar Control* in *time* mode.

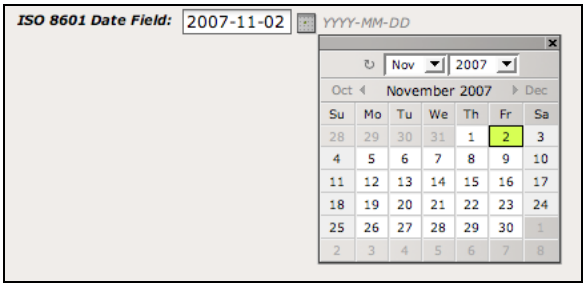

Figure 15: An example of the *Calendar Control* in *date* mode.

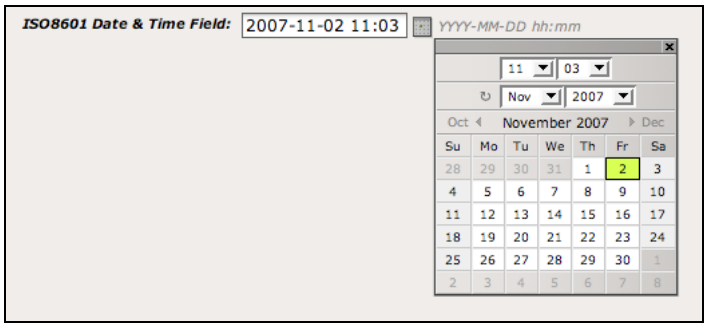

Figure 16: An example of the *Calendar Control* in *datetime* mode.

The format of the dates and times are controlled by settings in *application_env.php*, the relevant excerpt from which is shown below.

```
// Timezone Settings and Date/Time Format Constants
// ------------------------------------------------------------------------
// See http://www.php.net/manual/en/timezones.php
// for a list of supported timezones.
ini_set("date.timezone", "Australia/ACT");

// Format mask constants for use with the datetime DHTML control.
// As per /_javascript/datetime_control.js:
// These strings will be treated in a case-sensitive way.
// The code assumes that all masks requiring a date will include 'YYYY', 'MM', and 'DD'.
// The code also assumes that all masks requiring a time will include 'hh' and 'mm'.
// There is no support for time resolution greater than minutes
//(ie. no seconds or fractions of seconds).
// There is no support for years before year 0.
define("eDCT_FORMAT_ISO8601_DATE"          , 'YYYY-MM-DD');
define("eDCT_FORMAT_ISO8601_DATE_TIME"     , 'YYYY-MM-DD hh:mm');
define("eDCT_FORMAT_ISO8601_DATE_TIME_UTC" , 'YYYY-MM-DD hh:mmZ');
define("eDCT_FORMAT_ISO8601_TIME"          , 'hh:mm');
define("eDCT_FORMAT_ISO8601_TIME_UTC"      , 'hh:mmZ');
define("eDCT_FORMAT_ISO8601_DATETIME"      , 'YYYY-MM-DDThh:mm');
define("eDCT_FORMAT_ISO8601_DATETIME_UTC"  , 'YYYY-MM-DDThh:mmZ');
define("eDCT_FORMAT_AU_DATE"               , 'DD/MM/YYYY');
define("eDCT_FORMAT_AU_DATETIME"           , 'DD/MM/YYYY hh:mm AM');
define("eDCT_FORMAT_US_DATE"               , 'MM/DD/YYYY');
define("eDCT_FORMAT_US_DATETIME"           , 'MM/DD/YYYY hh:mm AM');
define("eDCT_FORMAT_TIME"                  , 'hh:mm AM');

$eDateFormat     = eDCT_FORMAT_ISO8601_DATE;
$eTimeFormat     = eDCT_FORMAT_ISO8601_TIME;
$eDateTimeFormat = eDCT_FORMAT_ISO8601_DATE_TIME;
```

UTC dates and times will be displayed as calculated based on the *date.timezone* setting.

Date and time settings for an instance of the framework are displayed on the *About The Applications* page (reached via the link at the right side of the footer).