Preside CMS Documentation

Release 0.1.0

Pixl8 Interactive

August 11, 2015

Contents

Preside CMS is a free and open source content management system written for the Railo platform and built with the Coldbox framework.

The documentation here aims to be both reference and guide for all aspects of the system and has been organised into the followlwing sections:

CHAPTER 1

Quickstart (TODO)

TODO

CHAPTER 2

Developer guides & recipes

2.1 Preside Data Objects

2.1.1 Overview

Preside Data Objects are the data layer implementation for PresideCMS. Just about everything in the system that persists data to the database uses Preside Data Objects to do so.

The Preside Data Objects system is deeply integrated into the CMS:

- Input forms and other administrative GUIs can be automatically generated for your preside objects (see `formlayouts`)
- Preside Data Object Views provide a way to present your data to end users without the need for handler or service layers

- The see `datamanager` provides a GUI for managing your client specific data and is based on entirely on Preside Data Objects
- Your preside objects can have their data tied to individual Working with multiple sites, without the need for any extra programming of site filters, see *Organising data by sites*

The following guide is intended as a thorough overview of Preside Data Objects. For API reference documentation, see Preside Object Service (service layer) and System Preside Objects (system provided data objects).

2.1.2 Object CFC Files

Data objects are represented by ColdFusion Components (CFCs). A typical object will look something like this:

```
component output=false {
    property name="name"          type="string" dbtype="varchar" maxlength="200" required=true;
    property name="email_address" type="string" dbtype="varchar" maxlength="255" required=true unique

    property name="tags" relationship="many-to-many" relatedto="tag";
}
```

A singe CFC file represents a table in your database. Properties defined using the `property` tag represent fields and/or relationships on the table (see *Properties*, below).

Database table names

By default, the name of the database table will be the name of the CFC file prefixed with **pobj_**. For example, if the file was `person.cfc`, the table name would be **pobj_person**.

You can override these defaults with the `tablename` and `tableprefix` attributes:

```
component tablename="mytable" tableprefix="mysite_" output=false {
    // .. etc.
}
```

Note: All of the preside objects that are provided by the core PresideCMS system have their table names prefixed with **psys_**.

Registering objects

The system will automatically register any CFC files that live under the `/application/preside-objects` folder of your site (and any of its sub-folders). Each .cfc file will be registered with an ID that is the name of the file without the ".cfc" extension.

For example, given the directory structure below, *four* objects will be registered with the IDs *blog*, *blogAuthor*, *event*, *eventCategory*:

```
/application
    /preside-objects
        /blogs
            blog.cfc
            blogAuthor.cfc
        /events
            event.cfc
            eventCategory.cfc
```

Note: Notice how folder names are ignored. While it is useful to use folders to organise your Preside Objects, they carry no logical meaning in the system.

Extensions and core objects

For extensions, the system will search for CFC files in a `/preside-objects` folder at the root of your extension.

Core system Preside Objects can be found at `/preside/system/preside-objects`. See System Preside Objects for reference documentation.

2.1.3 Properties

Properties represent fields on your database table or mark relationships between objects (or both).

Attributes of the properties describe details such as data type, data length and validation requirements. At a minimum, your properties should define a *name*, *type* and *dbtype* attribute. For *varchar* fields, a *maxLength* attribute is also required. You will also typically need to add a *required* attribute for any properties that are a required field for the object:

```
component output=false {
    property name="name"          type="string"  dbtype="varchar" maxLength="200" required=true;
    property name="max_delegates" type="numeric" dbtype="int"; // not required
}
```

Standard attributes

While you can add any arbitrary attributes to properties (and use them for your own business logic needs), the system will interpret and use the following standard attributes:

Name	Required	Default	Description
name	Yes	*N/A*	Name of the field
type	No	"string"	CFML type of the field. Valid values: *string*, *numeric*, *boolean*, *date*
dbtype	No	"varchar"	Database type of the field to be define on the database table field
maxLength	No	0	For dbtypes that require a length specification. If zero, the max size will be used.
required	No	**false**	Whether or not the field is required.
default	No	""	A default value for the property. Can be dynamically created, see *Default values for properties*
indexes	No	""	List of indexes for the field, see *Defining indexes and unique constraints*
uniqueindexes	No	""	List of unique indexes for the field, see *Defining indexes and unique constraints*
control	No	"default"	The default form control to use when rendering this field in a Preside Form. If set to 'default', the value for this attribute will be calculated based on the value of other attributes. See `/devguides/formcontrols` and `/devguides/formlayouts`.
renderer	No	"default"	The default content renderer to use when rendering this field in a view. If set to 'default', the value for this attribute will be calculated based on the value of other attributes. (reference needed here).
minLength	No	*none*	Minimum length of the data that can be saved to this field. Used in form validation, etc.
minValue	No	*none*	The minumum numeric value of data that can be saved to this field. *For numeric types only*.
maxValue	No	*N/A*	The maximum numeric value of data that can be saved to this field. *For numeric types only*.
format	No	*N/A*	Either a regular expression or named validation filter (reference needed) to validate the incoming data for this field
pk	No	**false**	Whether or not this field is the primary key for the object, *one field per object*. By default, your object will have an *id* field that is defined as the primary key. See *Default properties* below.
generator	No	"none"	Named generator for generating a value for this field when inserting a new record with the value of this field ommitted. Valid values are *increment* and *UUID*. Useful for primary key generation.
relationship	No	"none"	Either *none*, *many-to-one* or *many-to-many*. See *Defining relationships with properties*, below.
relatedTo	No	"none"	Name of the Preside Object that the property is defining a relationship with. See *Defining relationships with properties*, below.
relatedVia	No	""	Name of the object through which a many-to-many relationship will pass. If it does not exist, the system will created it for you. See *Defining relationships with properties*, below.
relationshipIsSource	No	**true**	In a many-to-many relationship, whether or not this object is regarded as the "source" of the relationship. If not, then it is regarded as the "target". See *Defining relationships with properties*, below.
relatedViaSourceFk	No	""	The name of the source object's foreign key field in a many-to-many relationship's pivot table. See *Defining relationships with properties*, below.
relatedViaTargetFk	No	""	The name of the target object's foreign key field in a many-to-many relationship's pivot table. See *Defining relationships with properties*, below.

Default properties

The bare minimum code requirement for a working Preside Data Object is:

```
component output=false {}
```

Yes, you read that right, an "empty" CFC is an effective Preside Data Object. This is because, by default, Preside Data Objects will be automatically given `id`, `label`, `datecreated` and `datemodified` properties. The above example is equivalent to:

```
component output=false {
    property name="id"           type="string" dbtype="varchar"   required=true maxLength="35" genera
    property name="label"        type="string" dbtype="varchar"   required=true maxLength="250";
    property name="datecreated"  type="date"   dbtype="timestamp" required=true;
    property name="datemodified" type="date"   dbtype="timestamp" required=true;
}
```

The ID Field

The ID field will be the primary key for your object. We have chosen to use a UUID for this field so that data migrations between databases are achievable. If, however, you wish to use an auto incrementing numeric type for this field, you could do so by overriding the `type`, `dbtype` and `generator` attributes:

```
component output=false {
    property name="id" type="numeric" dbtype="int" generator="increment";
}
```

The same technique can be used to have a primary key that does not use any sort of generator (you would need to pass your own IDs when inserting data):

```
component output=false {
    property name="id" generator="none";
}
```

Tip: Notice here that we are just changing the attributes that we want to modify (we do not specify `required` or `pk` attributes). All the default attributes will be applied unless you specify a different value for them.

The Label field

The **label** field is used by the system for building automatic GUI selectors that allow users to choose your object records.

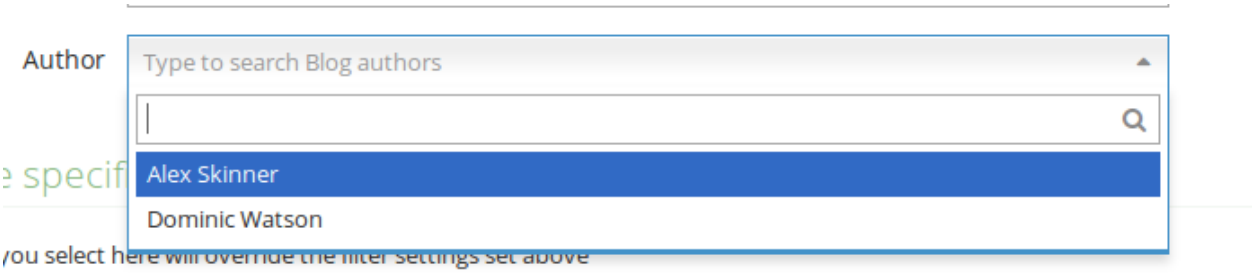

Fig. 2.1: Screenshot showing a record picker for a "Blog author" object

If you wish to use a different property to represent a record, you can use the `labelfield` attribute on your CFC, e.g.:

```
component output=false labelfield="title" {
    property name="title" type="string" dbtype="varchar" maxlength="100" required=true;
    // etc.
}
```

If you do not want your object to have a label field at all (i.e. you know it is not something that will ever be selectable, and there is no logical field that might be used as a string representation of a record), you can add a `nolabel=true` attribute to your CFC:

```
component output=false nolabel=true {
    // ... etc.
}
```

The DateCreated and DateModified fields

These do exactly what they say on the tin. If you use the APIs to insert and update your records, the values of these fields will be set automatically for you.

Default values for properties

You can use the `default` attribute on a property tag to define a default value for a property. This value will be used during an *InsertData()* operation when no value is supplied for the property. E.g.

```
component output=false {
    // ...
    property name="max_attendees" type="numeric" dbtype="int" required=false default=100;
}
```

Dynamic defaults

Default values can also be generated dynamically at runtime. Currently, this comes in two flavours:

1. Supplying raw CFML to be evaluated at runtime
2. Supplying the name of a method defined in your object that will be called at runtime, this method will be passed a 'data' argument that is a structure containing the data to be inserted

For raw CFML, prefix your value with `cfml:`, e.g. `cfml:CreateUUId()`. For methods that are defined on your object, use `method:methodName`. e.g.

```
component output=false  {
    // ...
    property name="event_start_date" type="date"   dbtype="timestamp"                required=false
    property name="slug"             type="string" dbtype="varchar"   maxlength="200" required=false

    public string function calculateSlug( required struct data ) output=false {
        return LCase( ReReplace( data.label ?: "", "\W", "_", "all" ) );
    }
}
```

Defining relationships with properties

Relationships are defined on **property** tags using the `relationship` and `relatedTo` attributes. For example:

```
// eventCategory.cfc
component output=false {}

// event.cfc
component output=false {
    property name="category" relationship="many-to-one" relatedto="eventCategory" required=true;
}
```

If you do not specify a `relatedTo` attribute, the system will assume that the foreign object has the same name as the property field. For example, the two objects below would be related through the `eventCategory` property of the `event` object:

```
// eventCategory.cfc
component output=false {}

// event.cfc
component output=false {
    property name="eventCategory" relationship="many-to-one" required=true;
}
```

One to Many relationships

In the examples, above, we define a **one to many** style relationship between `event` and `eventCategory` by adding a foreign key property to the `event` object.

The `category` property will be created as a field in the `event` object's database table. Its datatype will be automatically derived from the primary key field in the `eventCategory` object and a Foreign Key constraint will be created for you.

Note: The `event` object lives on the **many** side of this relationship (there are *many events* to *one category*), hence why we use the relationship type, *many-to-one*.

You can also declare the relationship on the other side (i.e. the 'one' side). This will allow you to traverse the relationship from either angle (see *Making use of relationships*). e.g. we could add a 'one-to-many' property on the *eventCategory.cfc* object; this will not create a field in the database table, but will allow you to query the relationship from the category viewpoint:

```
// eventCategory.cfc
component output=false {
    // note that the 'relationshipKey' property is the FK in the event object
    // this will default to the name of this object
    property name="events" relationship="one-to-many" relatedTo="event" relationshipKey="eventCategor
}

// event.cfc
component output=false {
    property name="eventCategory" relationship="many-to-one" required=true;
}
```

Many to Many relationships

If we wanted an event to be associated with multiple event categories, we would want to use a **Many to Many** relationship:

```
// eventCategory.cfc
component output=false {}

// event.cfc
component output=false {
    property name="eventCategory" relationship="many-to-many";
}
```

In this scenario, there will be no `eventCategory` field created in the database table for the `event` object. Instead, a "pivot" database table will be automatically created that looks a bit like this (in MySQL):

```
-- table name derived from the two related objects, delimited by __join__
create table `pobj_event__join__eventcategory` (
    -- table simply has a field for each related object
      `event`         varchar(35) not null
    , `eventcategory` varchar(35) not null

    -- plus we always add a sort_order column, should you care about
    -- the order in which records are related
    , `sort_order`    int(11)     default null

    -- unique index on the event and eventCategory fields
    , unique key `ux_event__join__eventcategory` (`event`,`eventcategory`)

    -- foreign key constraints on the event and eventCategory fields
    , constraint `fk_1` foreign key (`event`        ) references `pobj_event`         (`id`) on delet
    , constraint `fk_2` foreign key (`eventcategory`) references `pobj_eventcategory` (`id`) on delet
) ENGINE=InnoDB;
```

Note: Unlike **many to one** relationships, the **many to many** relationship can be defined on either or both objects in the relationship. That said, you will want to define it on the object(s) that make use of the relationship. In the event / eventCategory example, this will most likely be the event object. i.e. `event.insertData( label=eventName, eventCategory=listOfCategoryIds ).`

"Advanced" Many to Many relationships

You can excert a little more control over your many-to-many relationships by making use of some extra, non-required, attributes:

```
// event.cfc
component output=false {
    property name                = "eventCategory"
             relationship        = "many-to-many"
             relatedTo           = "eventCategory"
             relationshipIsSource = false              // the event object is regarded as the 'target
             relatedVia          = "event_categories" // create a new auto pivot object called "even
             relatedViaSourceFk  = "cat"              // name the foreign key field to the source ob
             relatedViaTargetFk  = "ev";              // name the foreign key field to the target ob
}
```

TODO: explain these in more detail. In short though, these attributes control the names of the pivot table and foreign

keys that get automatically created for you (see *Standard attributes* for more details on each of the attributes). If you leave them out, PresideCMS will figure out sensible defaults for you.

As well as controlling the automatically created pivot table name with "relatedVia", you can also use this attribute to define a relationship that exists through a pre-existing pivot object.

Tip: If you have multiple many-to-many relationships between the same two objects, you will **need** to use the `relatedVia` attribute to ensure that a different pivot table is created for each context.

Defining indexes and unique constraints

The Preside Object system allows you to define database indexes on your fields using the `indexes` and `uniqueindexes` attributes. The attributes expect a comma separated list of index definitions. An index definition can be either an index name or combination of index name and field position, separated by a pipe character. For example:

```
// event.cfc
component output=false {
    property name="category" indexes="category,categoryName|1" required=true relationship="many-to-or
    property name="name"     indexes="categoryName|2"           required=true type="string" dbtype="va
    // ...
}
```

The example above would result in the following index definitions:

```
create index ix_category     on pobj_event( category );
create index ix_categoryName on pobj_event( category, name );
```

The exact same syntax applies to unique indexes, the only difference being the generated index names are prefixed with `ux_` rather than `ix_`.

2.1.4 Keeping in sync with the database

When you reload your application (see `reloading`), the system will attempt to synchronize your object definitions with the database. While it does a reasonably good job at doing this, there are some considerations:

- If you add a new, required, field to an object that has existing data in the database, an exception will be raised. This is because you cannot add a `NOT NULL` field to a table that already has data. *You will need to provide upgrade scripts to make this type of change to an existing system.*
- When you delete properties from your objects, the system will rename the field in the database to `_deprecated_yourfield`. This prevents accidental loss of data but can lead to a whole load of extra fields in your DB during development.
- The system never deletes whole tables from your database, even when you delete the object file

2.1.5 Working with the API

The Preside Object Service service object provides methods for performing CRUD operations on the data along with other useful methods for querying the metadata of each of your data objects. There are two ways in which to interact with the API:

1. Obtain an instance the Preside Object Service and call its methods directly, see *Getting an instance of the Service API*

2. Obtain an "auto service object" for the specific object you wish to work with and call its decorated CRUD methods as well as any of its own custom methods, see *Using Auto Service Objects*

You may find that all you wish to do is to render a view with some data that is stored through the Preside Object service. In this case, you can bypass the service layer APIs and use the Preside Data Object Views system instead.

Getting an instance of the Service API

We use Wirebox to auto wire our service layer. To inject an instance of the service API into your service objects and/or handlers, you can use wirebox's "inject" syntax as shown below:

```
// a handler example
component output=false {
    property name="presideObjectService" inject="presideObjectService";

    function index( event, rc, prc ) output=false {
        prc.eventRecord = presideObjectService.selectData( objectName="event", id=rc.id ?: "" );

        // ...
    }
}

// a service layer example
// (here at Pixl8, we prefer to inject constructor args over setting properties)
component output=false {

    /**
     * @presideObjectService.inject presideObjectService
     */
     public any function init( required any presideObjectService ) output=false {
        _setPresideObjectService( arguments.presideObjectService );

        return this;
     }

     public query function getEvent( required string id ) output=false {
        return _getPresideObjectService().selectData(
              objectName = "event"
            , id         = arguments.id
        );
     }

     // we prefer private getters and setters for accessing private properties, this is our house sty
     private any function _getPresideObjectService() output=false {
         return variables._presideObjectService;
     }
     private void function _setPresideObjectService( required any presideObjectService ) output=false
         variables._presideObjectService = arguments.presideObjectService;
     }

}
```

Using Auto Service Objects

An auto service object represents an individual data object. They are an instance of the given object that has been decorated with the service API CRUD methods.

Calling the CRUD methods works in the same way as with the main API with the exception that the objectName argument is no longer required. So:

```
record = presideObjectService.selectData( objectName="event", id=id );

// is equivalent to:
eventObject = presideObjectService.getObject( "event" );
record      = eventObject.selectData( id=id );
```

Getting an auto service object

This can be done using either the *GetObject()* method of the Preside Object Service or by using a special Wirebox DSL injection syntax, i.e.

```
// a handler example
component output=false {
    property name="eventObject" inject="presidecms:object:event";

    function index( event, rc, prc ) output=false {
        prc.eventRecord = eventObject.selectData( id=rc.id ?: "" );

        // ...
    }
}

// a service layer example
component output=false {

    /**
     * @eventObject.inject presidecms:object:event
     */
     public any function init( required any eventObject ) output=false {
        _setPresideObjectService( arguments.eventObject );

        return this;
     }

     public query function getEvent( required string id ) output=false {
        return _getEventObject().selectData( id = arguments.id );
     }

     // we prefer private getters and setters for accessing private properties, this is our house sty
     private any function _getEventObject() output=false {
         return variables._eventObject;
     }
     private void function _setEventObject( required any eventObject ) output=false {
         variables._eventObject = arguments.eventObject;
     }

}
```

CRUD Operations

The service layer provides core methods for creating, reading, updating and deleting records (see individual method documentation for reference and examples):

- *SelectData()*
- *InsertData()*
- *UpdateData()*
- *DeleteData()*

In addition to the four core methods above, there are also further utility methods for specific scanarios:

- *DataExists()*
- *SelectManyToManyData()*
- *SyncManyToManyData()*
- *GetDeNormalizedManyToManyData()*
- *GetRecordVersions()*

Specifying fields for selection

The *SelectData()* method accepts a `selectFields` argument that can be used to specify which fields you wish to select. This can be used to select properties on your object as well as properties on related objects and any plain SQL aggregates or other SQL operations. For example:

```
records = newsObject.selectData(
    selectFields = [ "news.id", "news.title", "Concat( category.label, category$tag.label ) as catand
);
```

The example above would result in SQL that looked something like:

```
select      news.id
          , news.title
          , Concat( category.label, tag.label ) as catandtag

from        pobj_news     as news
inner join  pobj_category as category on category.id = news.category
inner join  pobj_tag      as tag      on tag.id      = category.tag
```

Note: The funky looking `category$tag.label` is expressing a field selection across related objects - in this case **news -> category -> tag**. See *Making use of relationships* for full details.

Filtering data

All but the **insertData()** methods accept a data filter to either refine the returned recordset or the records to be updated / deleted. The API provides two arguments for filtering, `filter` and `filterParams`. Depending on the type of filtering you need, the `filterParams` argument will be optional.

Simple filtering A simple filter consists of one or more strict equality checks, all of which must be true. This can be expressed as a simple CFML structure; the structure keys represent the object fields; their values represent the expected record values:

```
records = newsObject.selectData( filter={
      category             = chosenCategory
    , "category$tag.label" = "red"
} );
```

Note: The funky looking `category$tag.label` is expressing a filter across related objects - in this case **news -> category -> tag**. We are filtering news items whos category is tagged with a tag who's label field = "red". See *Making use of relationships*.

Complex filters More complex filters can be achieved with a plain SQL filter combined with filter params to make use of parametized SQL statements:

```
records = newsObject.selectData(
      filter       = "category != :category and DateDiff( publishdate, :publishdate ) > :daysold and
    , filterParams = {
          category             = chosenCategory
        , publishdate          = publishDateFilter
        , "category$tag.label" = "red"
        , daysOld              = { type="integer", value=3 }
      }
);
```

Note: Notice that all but the *daysOld* filter param do not specify a datatype. This is because the parameters can be mapped to fields on the object/s and their data types derived from there. The *daysOld* filter has no field mapping and so its data type must also be defined here.

Making use of relationships

As seen in the examples above, you can use a special field syntax to reference properties in objects that are related to the object that you are selecting data from / updating data on. When you do this, the service layer will automatically create the necessary SQL joins for you.

The syntax takes the form: `(relatedObjectReference).(propertyName)`. The related object reference can either be the name of the related object, or a `$` delimited path of property names that navigate through the relationships (see examples below).

This syntax can be used in:

- Select fields, see *Specifying fields for selection*
- Filters. see *Filtering data*
- Order by statements
- Group by statements

To help with the examples, we'll illustrate a simple relationship between three objects:

```
// tag.cfc
component output=false {}

// category.cfc
component output=false {
    property name="category_tag" relationship="many-to-one" relatedto="tag"  required=true;
    property name="news_items"   relationship="one-to-many" relatedTo="news" relationshipKey="news_ca
    // ..
}

// news.cfc
component output=false {
    property name="news_category" relationship="many-to-one" relatedto="category" required=true;
```

```
    // ..
}
```

Auto join example

```
// update news items who's category tag = "red"
presideObjectService.updateData(
      objectName = "news"
    , data       = { archived = true }
    , filter     = { "tag.label" = "red" } // the system will automatically figure out the relationsh
);
```

Property name examples

```
// delete news items who's category label = "red"
presideObjectService.deleteData(
      objectName = "news"
    , data       = { archived = true }
    , filter     = { "news_category.label" = "red" }
);

// select title and category tag from all news objects, order by the category tag
presideObjectService.selectData(
      objectName   = "news"
    , selectFields = [ "news.title", "news_category$category_tag.label as tag" ]
    , orderby      = "news_category$category_tag.label"
);

// selecting categories with a count of news articles for each category
presideObjectService.selectData(
      objectName   = "category"
    , selectFields = [ "category.label", "Count( news_items.id ) as news_item_count" ]
    , orderBy      = "news_item_count desc"
);
```

Warning: While the auto join syntax can be really useful, it is limited to cases where there is only a single relationship path between the two objects. If there are multiple ways in which you could join the two objects, the system can have no way of knowing which path it should take and will throw an error.

Caching

By default, all *SelectData()* calls have their recordset results cached. These caches are automatically cleared when the data changes.

You can specify *not* to cache results with the `useCache` argument.

See `caching` for a full guide to configuring and creating caches, including how to configure the default query cache used here.

2.1.6 Extending Objects

Tip: You can easily extend core data objects and objects that have been provided by extensions simply by creating `.cfc` file with the same name.

Objects with the same name, but from different sources, are merged at runtime so that you can have multiple extensions all contributing to the final object definition.

Take the Sitetree Page object, for example. You might write an extension that adds an **allow_comments** property to the object. That CFC would look like this:

```
// /extensions/myextension/preside-objects/page.cfc
component output=false {
    property name="allow_comments" type="boolean" dbtype="boolean" required=false default=true;
}
```

After adding that code and reloading your application, you would find that the **psys_page** table now had an **allow_comments** field added.

Then, in your site, you may have some client specific requirements that you need to implement for all pages. Simply by creating a `page.cfc` file under your site, you can mix in properties along with the **allow_comments** mixin above:

```
// /application/preside-objects/page.cfc
component output=false {
    // remove a property that has been defined elsewhere
    property name="embargo_date" deleted=true;

    // alter attributes of an existing property
    property name="title" maxLength="50"; // strict client requirement?!

    // add a new property
    property name="search_engine_boost" type="numeric" dbtype="integer" minValue=0 maxValue=100 defau
}
```

Note: To have your object changes reflected in GUI forms (i.e. the add and edit page forms in the example above), you will likely need to modify the form definitions for the object you have changed. See `formlayouts` for a full guide and reference (hint: the same system of mixed in extensions is used for form layouts).

2.1.7 Versioning

By default, Preside Data Objects will maintain a version history of each database record. It does this by creating a separate database table that is prefixed with `_version_`. For example, for an object named 'news', a version table named **_version_pobj_news** would be created.

The version history table contains the same fields as its twin as well as a few specific fields for dealing with version numbers, etc. All foreign key constraints and unique indexes are removed.

Opting out

To opt out of versioning for an object, you can set the `versioned` attribute to **false** on your CFC file:

```
component versioned=false output=false {
    // ...
}
```

Interacting with versions

Various admin GUIs such as the `datamanager` implement user interfaces to deal with versioning records. However, if you find the need to create your own, or need to deal with version history records in any other way, you can use methods provided by the service api:

- *GetRecordVersions()*
- *GetVersionObjectName()*
- *ObjectIsVersioned()*
- *GetNextVersionNumber()*

In addition, you can specify whether or not you wish to use the versioning system, and also what version number to use if you are, when calling the *InsertData()*, *UpdateData()* and *DeleteData()* methods by using the `useVersioning` and `versionNumber` arguments.

Finally, you can select data from the version history tables with the *SelectData()* method by using the `fromVersionTable`, `maxVersion` and `specificVersion` arguments.

Many-to-many related data

By default, auto generated `many-to-many` data tables will be versioned along with your record changes. You can opt out of this by adding a `versioned=false` attribute to the `many-to-many` property:

```
property name="categories" relationship="many-to-many" relatedTo="category" versioned=false;
```

Inversely, you may have a `many-to-many` relationship for which you have an explicit join table that you'd like versioned along with the parent record. In this scenario, you can explicitly set `versioned=true`:

```
property name="categories" relationship="many-to-many" relatedTo="category" relatedVia="explicit_cate
```

Ignoring changes

By default, when the data actually changes in your object, a new version will be created. If you wish certain fields to be ignored when it comes to determining whether or not a new version should be created, you can add a `ignoreChangesForVersioning` attribute to the property in the preside object.

An example scenario for this might be an object who's data is synced with an external source on a schedule. You may add a helper property to record the last sync check date, if no other fields have changed, you probably don't want a new version record being created just for that sync check date. In this case, you could do:

```
property name="_last_sync_check" type="date" dbtype="datetime" ignoreChangesForVersioning=true;
```

2.1.8 Organising data by sites

You can instruct the Preside Data Objects system to organise your objects' data into your system's individual sites (see Working with multiple sites). Doing so will mean that any data reads and writes will be specific to the currently active site.

To enable this feature for an object, simply add the `siteFiltered` attribute to the `component` tag:

```
component output=false siteFiltered=true {
    // ...
}
```

2.2 Preside Data Object Views

2.2.1 Overview

PresideCMS provides a feature that allows you to autowire your data model to your views, completely bypassing hand written handlers and service layer objects. Rendering one of these views looks like this:

```
#renderView(
      view          = "events/preview"
    , presideObject = "event"
    , filter        = { event_category = rc.category }
)#
```

In the expample above, the `/views/events/preview.cfm` view will get rendered for each *event* record that matches the supplied filter, `{ event_category = rc.category }`. Each rendered view will be passed the database fields that it needs as individual arguments.

In order for the `renderView()` function to know what fields to select for your view, the view itself must declare what fields it requires. It does this using the `<cfparam>` tag. Using our "event preview" example from above, our view file might look something like this:

```
<cfparam name="args.label"                                    /><!--- I need the 'label' field --->
<cfparam name="args.teaser"                                   /><!--- I need the 'teaser' field --->
<cfparam name="args.image"                                    /><!--- I need the 'image' field --->
<cfparam name="args.event_type_id" field="event_type"         /><!--- I need the 'event_type' field, bu
<cfparam name="args.event_type"    field="event_type.label" /><!--- I need the 'label' field from the

<cfparam name="_counter" type="numeric" /><!--- current row in the recordset being rendered --->
<cfparam name="_records" type="numeric" /><!--- total records in the recordset being rendered --->

<cfoutput>
    <div class="preview-pane">
        <h3>#args.label#</h3>
        <p class="event-type">
            <a href="#event.buildLink( pageId=args.event_type_id )#">
                #args.event_type#
            </a>
        </p>

        #renderAsset( assetId=args.image, context="previewPane" )#

        <p>#args.teaser#</p>
    </div>
</cfoutput>
```

Given the examples above, the SQL you would expect to be automatically generated and executed for you would look something like this:

```
select      event.label
          , event.teaser
          , event.image
          , event.event_type as event_type_id
          , event_type.label as event_type

from        pobj_event      event
inner join pobj_event_type event_type on event_type.id = event.event_type

where       event.event_category = :event_category
```

2.2.2 Filtering the records to display

Any arguments that you pass to the `renderView()` method will be passed on to the Preside Object `selectData()` method when retrieving the records to be rendered.

This means that you can specify any number of valid `selectData()` arguments to filter and sort the records for display. e.g.

```
rendered = renderView(
      view          = "event/detail"
    , presideObject = "event"
    , id            = eventId
);

rendered = renderView(
      view          = "event/preview"
    , presideObject = "event"
    , filter        = "event_type != :event_type or comment_count < :comment_count"
    , filterParams  = { event_type=rc.type, comment_count=10 }
    , startRow      = 11
    , maxRows       = 10
    , orderBy       = "datepublished desc"
);
```

2.2.3 Declaring fields for your view

As seen in the examples above, the `<cfparam>` tag is used by your view to specify what fields it needs to render. Any variable that is declared that starts with "args." will be considered a field on your preside object by default.

If we are rendering a view for a **news** object, the following param will lead to `news.headline` being retrieved from the database:

```
<cfparam name="args.headline" />
```

Aliases

You may find that you need to have a different variable name to the field that you need to select from the data object. To achieve this, you can use the `field` attribute to specify the name of the field:

```
<cfparam name="args.headline" field="news.label" />
```

You can use the same technique to do aggregate fields and any other SQL select goodness that you want:

```
<cfparam name="args.headline"      field="news.label" />
<cfparam name="args.comment_count" field="Count( comments.id )" />
```

Getting fields from other objects

For one to many style relationships, where your object is the many side, you can easily select fields from the related object using the `field` attribute shown above. Simply prefix the column name with the name of the foreign key field on your object. For example, if our **news** object has a single **news_category** field that is a foreign key to a category lookup, we could get the title of the category with:

```
<cfparam name="args.headline" field="news.label" />
<cfparam name="args.category" field="news_category.label" />
```

Front end editing

If you would like a field to be editable in the front end website, you can set the `editable` attribute to **true**:

```
<cfparam name="args.label" editable="true" />
```

Accepting arguments that do not come from the database

Your view may need some variables that do not come from the database. For example, in the code below, the view is being passed the `showComments` argument that does not exist in the database.

```
#renderView( view="myview", presideObject="news", args={ showComments=false } )#
```

To allow this to work, you can specify `field="false"`, so:

```
<cfparam name="args.headline"     field="news.label" />
<cfparam name="args.category"     field="news_category.label" />
<cfparam name="args.showComments" field="false" type="boolean" />
```

Defining renderers

Each of the fields fetch from the database for your view will be pre-rendered using the default renderer for that field. So fields that use a richeditor will have their Widgets and embedded assets all ready rendered for you. To specify a different renderer, or to specify renderers on calculated fields, do:

```
<cfparam name="args.comment_count" field="Count( comments.id )" renderer="myNumberFormatter" />
```

2.3 Preside viewlets

2.3.1 Overview

Coldbox has a concept of viewlets (see what they have to say about it here: http://wiki.coldbox.org/wiki/Layouts-Views.cfm#Viewlets_(Portable_Events)).

Preside builds on this concept and provides a concrete implementation with the `renderViewlet()` method. This implementation is used throughout Preside and is an important concept to grok when building custom Preside functionality (widgets, form controls, etc.).

2.3.2 The Coldbox Viewlet Concept

Conceptually, a Coldbox viewlet is a self contained module of code that will render some view code after performing handler logic to fetch data. The implementation of a Coldbox viewlet is simply a private handler action that returns the rendered view (the handler must render the view itself). This action will be directly called using the `runEvent()` method. For example, the handler action might look like this:

```
private any function myViewlet( event, rc, prc, id=0 ) output=false {
    prc.someData = getModel( "someService" ).getSomeData( id=arguments.id );
    return getPlugin( "renderer" ).renderView( "/my/viewlets/view" );
}
```

And you could render that viewlet like so:

```
#runEvent( event="SomeHandler.myViewlet", prePostExempt=true, private=true, eventArguments={ id=2454
```

2.3.3 The Preside renderViewlet() method

Preside provides a concrete implementation of viewlets with the `renderViewlet()` method. For the most part, this is simply a wrapper to `runEvent()` with a clearer name, but it also has some other differences to be aware of:

1. If the passed event does not exist as a handler action, `renderViewlet()` will try to find and render the corresponding view
2. It defaults the `prePostExempt` and `private` arguments to `true` (this is the usual recommended behaviour for viewlets)
3. It formalizes how viewlet arguments are passed to the handler / view. When passing arguments to a handler action or view, those arguments will be available directly in the `args` structure

Example viewlet handler

Below is an example of a Preside viewlet handler action. It is much the same as the standard Coldbox viewlet handler action but receives an additional `args` structure that it can make use of and also passes any data that it gathers directly to the view rather than relying on the `prc` / `rc` (this is recommendation for Preside viewlets).

```
private any function myViewlet( event, rc, prc, args={} ) output=false {
    args.someData = getModel( "someService" ).getSomeData( id=( args.id ?: 0 ) );

    return getPlugin( "renderer" ).renderView( view="/my/viewlets/view", args=args );
}
```

You could then render the viewlet with:

```
#renderViewlet( event="SomeHandler.myViewlet", args={ id=5245 } )#
```

Example viewlet without a handler (just a view)

Sometimes you will implement viewlets in Preside without a handler. You might find yourself doing this for custom form controls or widgets (which are implemented as viewlets). For example:

```
<cfparam name="args.title" type="string" />
<cfparam name="args.description" type="string" />

<cfoutput>
    <h1>#args.title</h1>
    <p>#args.description#</p>
</cfoutput>
```

Rendering the viewlet:

```
#renderViewlet( event="viewlets.myViewlet", args={ title="hello", description="world" } )#
```

2.3.4 Reference

The `renderViewlet()` method is available to your handlers and views directly. In any other code, you will need to use `getController().renderViewlet()` where `getController()` would return the Coldbox controller instance. It takes the following arguments:

Argument name	Type	Required	Description
event	string	Yes	Coldbox event string, e.g. "mymodule:myHandler.myAction"
args	struct	No	A structure of arguments to be passed to the viewlet
prePostExempt	boolean	No	Whether or not pre and post events should be fired when running the handler action for the viewlet
private	boolean	No	Whether or not the handler action for the viewlet is a private method

2.4 Working with page types

2.4.1 Overview

Page types allow developers to wire *structured content* to website pages that are stored in the *site tree*. They are implemented in a way that is intuitive to the end-users and painless for developers.

Architecture

Pages

Pages in a site's tree are stored in the 'page' preside object (see Sitetree Page). This object stores information that is common to all pages such as *title* and *slug*.

Page types

All pages in the tree must be associated with a page *type*; this page type will define further fields that are specific to its purpose. Each page type will have its own Preside Object in which the specific data is stored. For example, you might have an "event" page type that had *Start date*, *End date* and *Location* fields.

A one-to-one relationship exists between each page type object and the page obejct. This means that every **page type** record must and will have a corresponding **page** record.

2.4.2 Creating a page type

There are four essential parts to building a page type. The data model, view layer, i18n properties file and form layout(s).

Hint: You can scaffold all the parts of a page template very quickly using the Developer console (see `developerconsole`). Once in the console, type `new pagetype` and follow the prompts.

The data model

A page type is defined by creating a **Preside Data Object** (see Preside Data Objects) that lives in a subdirectory called "page-types". For example: `/preside-objects/page-types/event.cfc`:

```
// /preside-objects/page-types/event.cfc
component output=false {
    property name="start_date" type="date"   dbtype="date"                  required=true;
    property name="end_date"   type="date"   dbtype="date"                  required=true;
    property name="location"   type="string" dbtype="varchar" maxLength=100 required=false;
}
```

Under the hood, the system will add some fields for you to cement the relationship with the 'page' object. The result would look like this:

```
// /preside-objects/page-types/event.cfc
component output=false labelfield="page.title" {
    property name="start_date" type="date"   dbtype="date"                  required=true;
    property name="end_date"   type="date"   dbtype="date"                  required=true;
    property name="location"   type="string" dbtype="varchar" maxLength=100 required=false;

    // auto generated property (you don't need to create this yourself)
    property mame="page" relationship="many-to-one" relatedto="page" required=true uniqueindexes="pag
}
```

Note: Notice the "page.title" **labelfield** attribute on the component tag. This has the effect of the 'title' field of the related 'page' object being used as the labelfield (see *The Label field*).

You do not need to specify this yourself, written here as an illustration of what gets added under the hood.

View layer

The page types system takes advantage of auto wired views (see Preside Data Object Views). What this means is that we do not need to create a service layer or a coldbox handler for our page type, PresideCMS will take care of wiring your view to your page type data object.

Using our "event" page type example, we would create a view file at `/views/page-types/event/index.cfm`. A simplified example might then look something like this:

```
<!--- /views/page-types/event/index.cfm --->
<cfparam name="args.title"      field="page.title"       editable="true" />
<cfparam name="args.start_date" field="event.start_date" editable="true" />
<cfparam name="args.end_date"   field="event.end_date"   editable="true" />
<cfparam name="args.location"   field="event.location"   editable="true" />

<cfoutput>
    <h1>#page.title#</h1>
    <div class="dates-and-location">
        <p>From #args.start_date# to #args.end_date# @ #args.location#</p>
    </div>
</cfoutput>
```

Using a handler

If you need to do some handler logic before rendering your page type, you take full control of fetching the data and rendering the view for your page type.

You will need to create a handler under a 'page-types' folder who's filename matches your page type object, e.g. `/handlers/page-types/event.cfc`. The "index" action will be called by default and will be called as a Preside Viewlet (see Preside viewlets). For example:

```
component output=false {

    private string function index( event, rc, prc, args ) output=false {
        args.someValue = getModel( "someServiceOrSomesuch" ).getSomeValue();

        return renderView(
              view          = "/page-types/event/index"
            , presideObject = "event"
            , id            = event.getCurrentPageId()
            , args          = args
        );
    }
}
```

Multiple layouts

You can create layout variations for your page type that the users of the CMS will be able to select when creating and editing the page. To do this, simply create multiple views in your page type's view directory. For example:

```
/views
    /page-types
        /event
            _ignoredView.cfm
            index.cfm
            special.cfm
```

Note: Any views that begin with an underscore are ignored. Use these for reusable view snippets that are not templates in themselves.

If your page type has more than one layout, a drop down will appear in the page form, allowing the user to select which template to use.

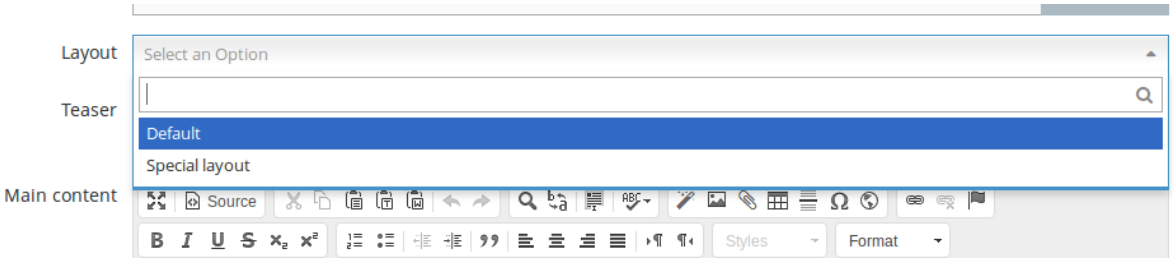

Fig. 2.2: Screenshot of a layout picker.

You can control the labels of your layouts that appear in the dropdown menu by adding keys to your page type's i18n properties file (see UI and i18n below).

UI and i18n

In order for the page type to appear in a satisfactory way for your users when creating new pages (see screenshot below), you will also need to create a `.properties` file for the page type.

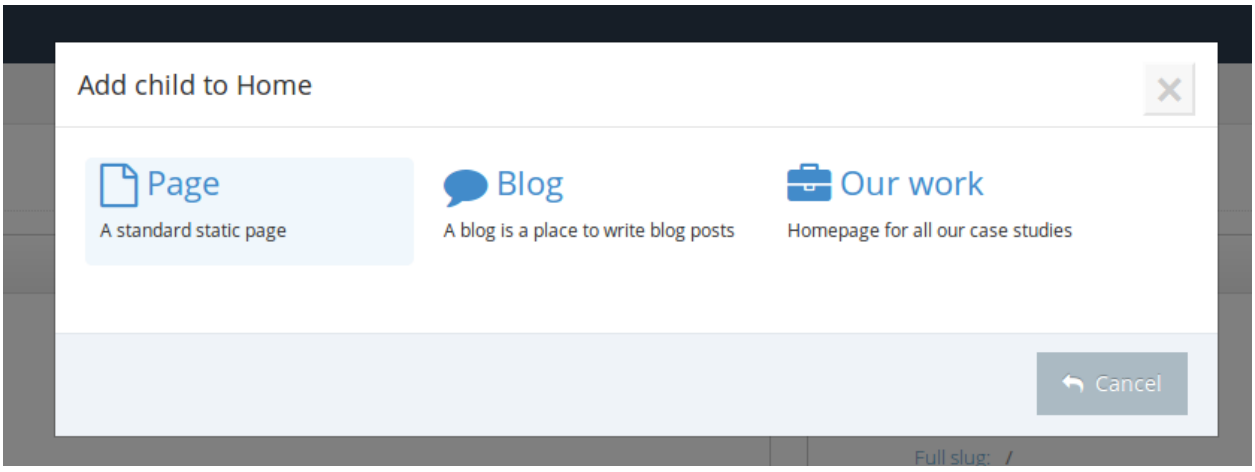

Fig. 2.3: Screenshot of a typical page type picker that appears when adding a new page to the tree.

For example, if your page type **Preside data object** was, `/preside-objects/page-types/event.cfc`, you would need to create a `.properties` file at, `/i18n/page-types/event.properties`. In it, you will need to add *name*, *description* and *iconclass* keys, e.g.

```
# mandatory keys
name=Event
description=An event page
iconclass=fa-calendar

# keys for the add / edit page forms (completely up to you, see below)
```

```
tab.title=Event fields
field.title.label=Event name
field.start_date.label=Start date
field.end_date.label=End date
field.location.label=Location

# keys for the layout picker
layout.index=Default
layout.special=Special layout
```

Add and edit page forms

The core PresideCMS system ships with default form layouts for adding and editing pages in the site tree. The page types system allows you to modify those forms for specific page types. See `formlayouts` for detailed documentation on creating and merging form layouts.

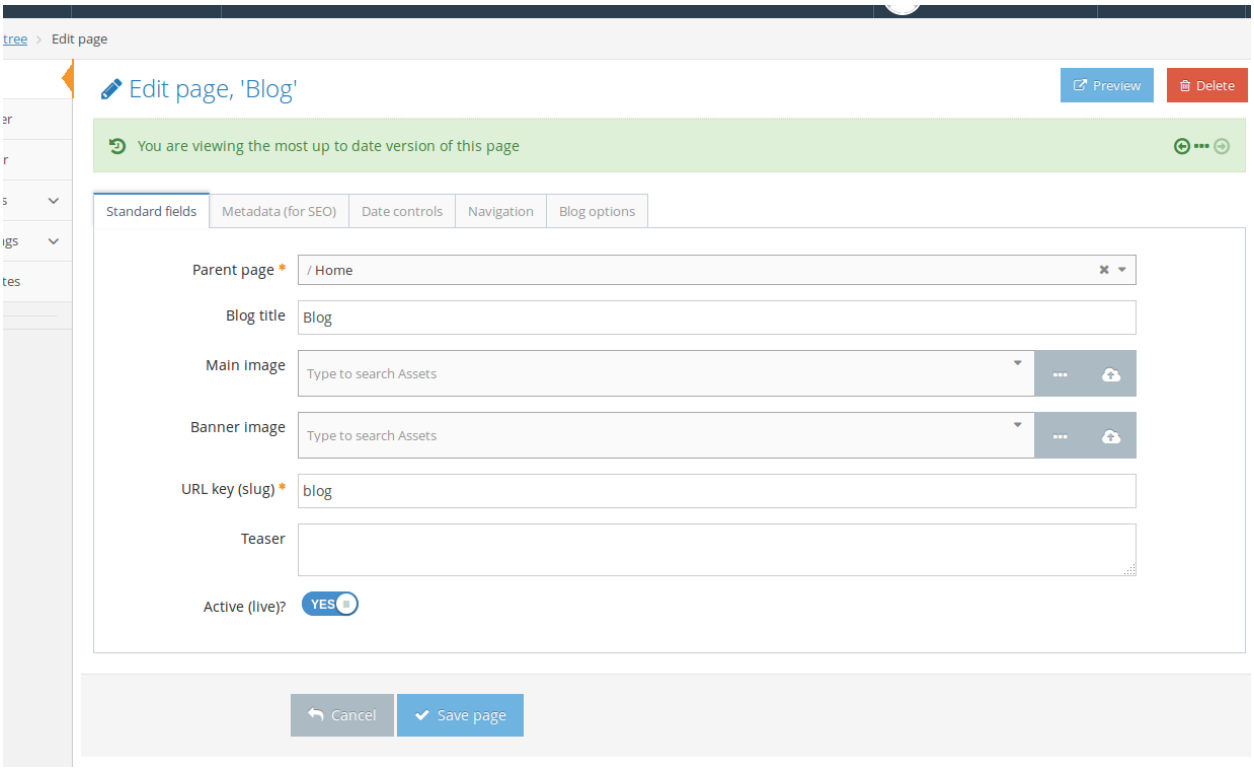

Fig. 2.4: Screenshot of a typical edit page form.

To achieve this, you can either create a single form layout that will be used to modify both the **add** and **edit** forms, or a layout for each form. For example, the following form layout will modify the layout forms for our "event" page type example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    To use this layout for both edit and add modes, the file would be:

        /application/forms/page-types/event.xml

    For individual add / edit forms:
```

```
        /application/forms/page-types/event.add.xml
        /application/forms/page-types/event.edit.xml
-->
<form>
    <tab id="main">
        <fieldset id="main">
            <!-- modify the label for the 'title' field to be event specific (uses a key from our i1
            <field name="title" label="page-types.event:field.title.label" />

            <!-- delete some fields that we don't want to see for event pages -->
            <field name="parent_page" deleted="true" />
            <field name="active"      deleted="true" />
            <field name="slug"        deleted="true" />
            <field name="layout"      deleted="true" />
        </fieldset>
    </tab>

    <!-- add some new fields in a new tab -->
    <tab id="event-fields" title="page-types.event:tab.title">
        <fieldset id="event-fields">
            <field binding="event.start_date" label="page-types.event:field.start_date.label" />
            <field binding="event.end_date"   label="page-types.event:field.end_date.label" />
            <field binding="event.location"   label="page-types.event:field.location.label" />
        </fieldset>
    </tab>
</form>
```

2.5 Working with multiple sites

2.5.1 Overview

PresideCMS allows users to create and manage multiple sites. This is perfect for things like microsites, different language sites and any other organisation of workflows and users.

From a development standpoint, the CMS allows developers to create and maintain multiple *Site templates*. A site template is very similar to a Preside Extension, the difference being that the site template is only active when the currently active site is using the template.

Finally, the CMS allows you to easily segment the data in your Preside Data Objects by site (see *Organising data by sites*). By doing so, each site will only have access to the data that is unique to it. The developers are in control of which data objects have their data shared across all sites and which objects have their data segmented per site.

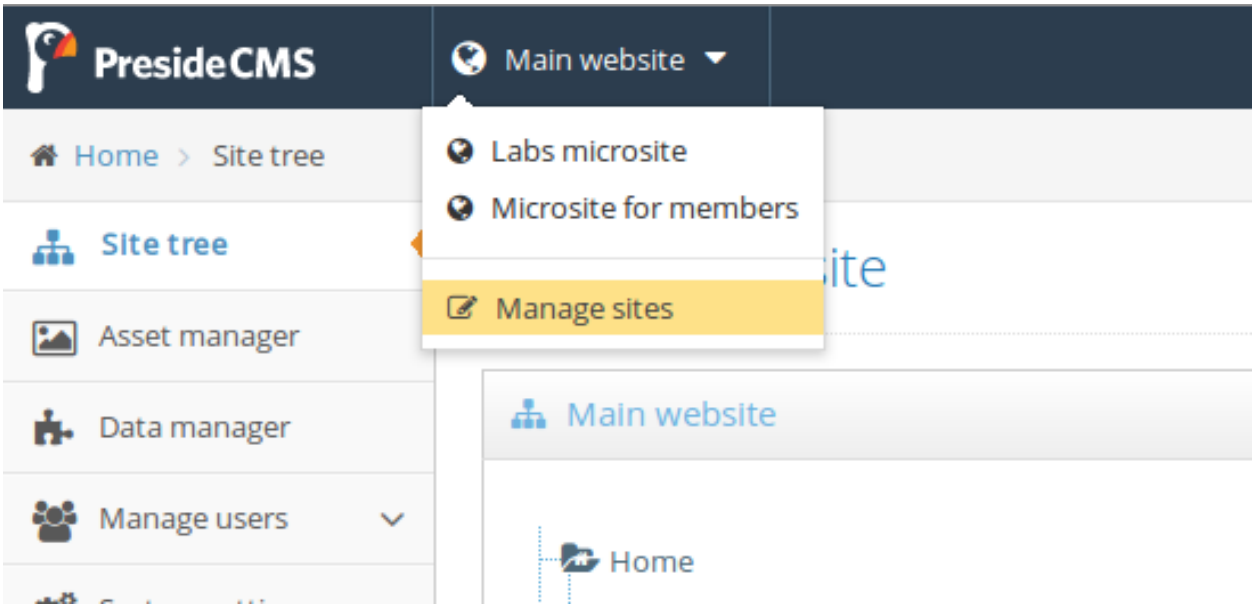

Fig. 2.5: Screenshot showing the site picker that appears in the administrator for users with access to multiple sites and / or users with access to the site manager.

2.5.2 Site templates

Site templates are like a PresideCMS application within another PresideCMS application. They can contain all the same folders and concepts as your main application but are only active when the currently active site is using the template. This means that any widgets, page types, views, etc. that are defined within your site template, will only kick in when the site that uses the template is active. CMS administrators can apply a single template to a site.

Creating a barebones site template

To create a new site template, you will need to create a folder under your application's `application/site-templates/` folder (create one if it doesn't exist already). The name of your folder will become the name of the template, e.g. the following folder structure will define a site template with an id of 'microsite':

```
/application
    /site-templates
        /microsite
```

In order for the site template to appear in a friendly manner in the UI, you should also add an i18n properties file that corresponds to the site id. In the example above, you would create `/application/i18n/site-templates/microsite.properties`:

```
title=Microsite template
description=The microsite template provides layouts, widgets and page types that are unique to the si
```

Overriding layouts, views, forms, etc.

To override any PresideCMS features that are defined in your main application, you simply need to create the same files in the same directory structure within your site template.

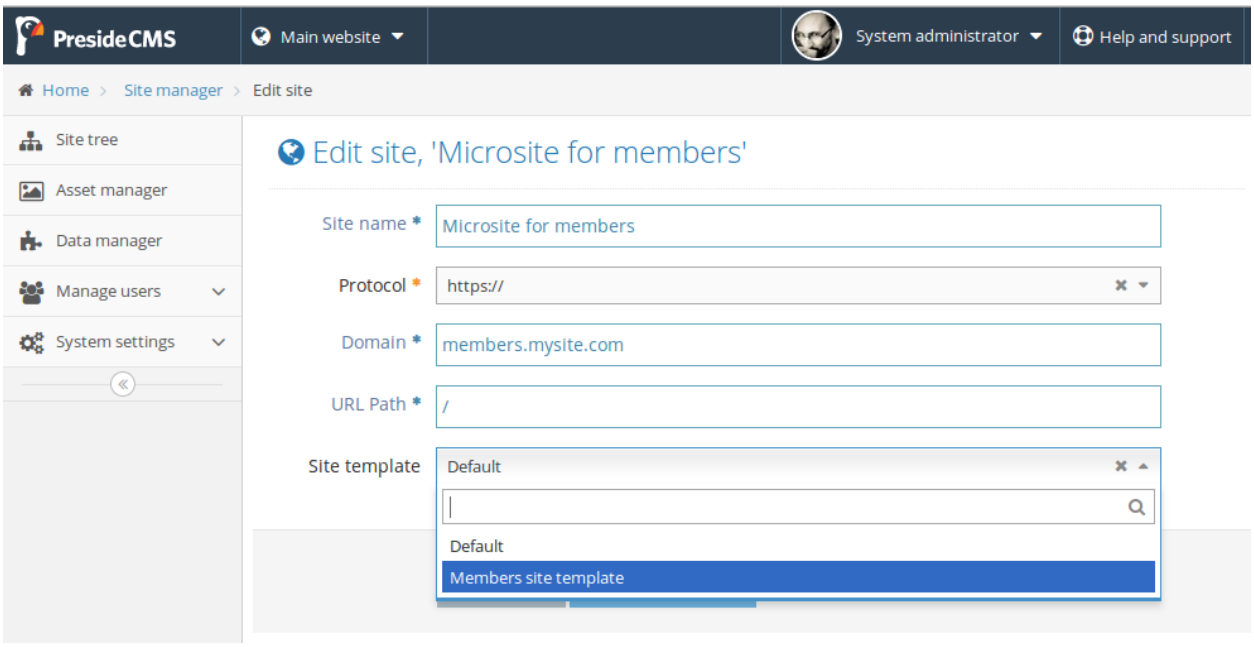

Fig. 2.6: Screenshot of an edit site form where the user can choose which template to apply to the site.

For example, if you wanted to create a different page layout for a site template, you might want to override the main application's `/application/layouts/Main.cfm` file. To do so, simply create `/application/site-templates/mytemplate/layouts/Main.cfm`:

```
/application
    /layouts
        Main.cfm <-- this will be used when the active site is *not* using the 'microsite' site templ
    /site-templates
        /microsite
            /layouts
                Main.cfm <-- this will be used when the active site is using the 'microsite' site ter
```

This technique can be used for Form layouts (see `formlayouts`), Widgets (see `widgets`), Page types (see Working with page types) and i18n. It can also be used for Coldbox views, layouts and handlers.

Warning: You cannot make modifications to Preside Data Objects with the intention that they will only take affect for sites using the current site template. Any changes to Preside Data Objects affect the database schema and will always take affect for every single site and site template.
If you wish to have different fields on the same objects but for different site templates, we recommend defining all the fields in your core application's object and providing different form layouts that show / hide the relevent fields for each site template.

Creating features unique to the site template

To create features that are unique to the site template, simply ensure that they are namespaced suitably so as not to conflict with other extensions and site templates. For example, to create an "RSS Feed" widget that was unique to your site template, you might create the following file structure:

```
/application
    /site-templates
```

```
        /microsite
            /forms
                /widgets
                    microsite-rss-widget.xml
            /i18n
                /widgets
                    microsite-rss-widget.properties
            /views
                /widgets
                    microsite-rss-widget.cfm
```

2.6 Working with the richeditor

2.6.1 Overview

PresideCMS uses CKEditor for its richeditor.

Beyond the standard install, PresideCMS provides custom plugins to interact with the CMS such as inserting images and documents from the Asset Manager, linking to pages in the site tree, etc. It also allows you to customize and configure the editor from your CFML code.

2.6.2 Configuration

Default settings and toolbar sets can be configured in your site's `Config.cfc`. For example,

```
public void function configure() output=false {
    super.configure();

    // ...

    settings.ckeditor = {};

    // default settings
    settings.ckeditor.defaults = {
          width       = "auto"                                  // default width of the editor, in pixe
        , minHeight   = "auto"                                  // default height of the editor, in pi
        , maxHeight   = 600                                     // maximum autogrow height of the edito
        , toolbar     = "full"                                  // default toolbar set, see below
        , stylesheets = [ "/specific/richeditor/", "/core/" ] // array of stylesheets to be included
        , configFile  = "/ckeditorExtensions/config.js"       // path is relative to the compiled ass
    };
```

```
    // toolbar sets, see further documentation below
    settings.ckeditor.toolbars = {};
    settings.ckeditor.toolbars.full = 'Maximize,-,Source,-,Preview'
                                    & '|Cut,Copy,Paste,PasteText,PasteFromWord,-,Undo,Redo'
                                    & '|Find,Replace,-,SelectAll,-,Scayt'
                                    & '|Widgets,ImagePicker,AttachmentPicker,Table,HorizontalRule,Spec
                                    & '|Link,Unlink,Anchor'
                                    & '|Bold,Italic,Underline,Strike,Subscript,Superscript,-,RemoveFor
                                    & '|NumberedList,BulletedList,-,Outdent,Indent,-,Blockquote,Create
                                    & '|Styles,Format,Font,FontSize'
                                    & '|TextColor,BGColor';

    settings.ckeditor.toolbars.boldItalicOnly = 'Bold,Italic';
}
```

Configuring toolbars

PresideCMS uses a light-weight syntax for defining sets of toolbars that translates to the full CKEditor toolbar definition. The following two definitions are equivalent:

CKEditor config.js

```
CKEDITOR.editorConfig = function( config ) {
    config.toolbar = "mytoolbar";

    config.toolbar_mytoolbar = [
        [
            [ 'Source', '-', 'NewPage', 'Preview', '-', 'Templates' ],                    // Defines
            [ 'Cut', 'Copy', 'Paste', 'PasteText', 'PasteFromWord', '-', 'Undo', 'Redo' ], // Defines
            '/',                                                                          // Line br
            [ 'Bold', 'Italic' ]                                                          // Defines
        ]
    ];
};
```

Config.cfc equivalent

```
public void function configure() output=false {
    super.configure();

    // ...

    settings.ckeditor.defaults = {
        , toolbar = "mytoolbar"
    };

    // in the PresideCMS version of the toolbar configuration, toolbar groups
    // are simply comma separated lists of buttons and dividers. Toolbar groups
    // are then delimited by the pipe ('|') symbol.
    settings.ckeditor.toolbars.mytoolbar = 'Source,-,NewPage,Preview,-,Templates'
                                         & '|Cut,Copy,Paste,PasteText,PasteFromWord,-,Undo,Redo'
                                         & '|/'
                                         & '|Bold,Italic';

    // the above toolbar string all on one line: 'Source,-,NewPage,Preview,-,Templates|Cut,Copy,Paste
}
```

Specifying non-default toolbars for form fields

You can define multiple toolbars in your configuration and then specify which toolbar to use for individual form fields (if you do not define a toolbar, the default will be used). An example, using a PresideCMS form definition:

```
<?xml version="1.0" encoding="UTF-8"?>
<form>
    <tab>
        <fieldset>
            <field name="description" control="richeditor" toolbar="boldItalicOnly" label="widgets.my
        </fieldset>
    </tab>
</form>
```

You can also define toolbars inline:

```
<?xml version="1.0" encoding="UTF-8"?>
<form>
    <tab>
        <fieldset>
            <field name="description" control="richeditor" toolbar="Bold,Italic,Underline|Cut,Copy,Pa
        </fieldset>
    </tab>
</form>
```

Configuring stylesheets

The stylesheets configuration effects how content within the editor is displayed during editing. You will likely want to include your site's core styles so that the WYSIWYG experience is as close to the final product as possible.

Default stylesheets are configured as an array of stylesheet includes (see Config.cfc example above). Each item in the array will be expanded as a CfStatic include resource (see `cssandjs`). The example below gives a sample folder structure along with the configuration required to include the site's core styles + richeditor specific styles:

```
/app
    /assets
        /css
            /core
                00_reset.less
                01_bootstrap.less
                02.typography.less
                03.forms.less
                ... etc.

            /specific
                /richeditor
                    00_richeditorReset.less
                /newspage
                /eventspage
                ... etc
        /js
```

```
ckeditor.defaults.stylesheets = [ "/specific/richeditor/", "/core/" ]; // note how the paths are rela
```

Specifying non-default stylesheets for form fields

You can define specific stylesheets for individual form controls by supplying a comma separated list:

```
<?xml version="1.0" encoding="UTF-8"?>
<form>
    <tab>
        <fieldset>
            <field name="description" control="richeditor" stylesheets="/specific/myCustomEditorStyle
        </fieldset>
    </tab>
</form>
```

Configuring a custom CKEditor config file

For the most flexible configuration tweaking, you can define your own CKEditor `config.js` file:

```
ckeditor.defaults.configFile = "/path/to/my/custom/config/file.js"; // relative to your root assets
```

You can also define this inline:

```
<?xml version="1.0" encoding="UTF-8"?>
<form>
    <tab>
        <fieldset>
            <field name="description" control="richeditor" customConfig="/path/to/my/custom/config/fi
        </fieldset>
    </tab>
</form>
```

Note: The default configuration file can be found at `/preside/system/assets/ckeditorExtensions/config.js`

2.6.3 Where the code lives (for maintainers and contributers)

We manage a custom build of the editor, including all the core plugins that we require, through our own repository on GitHub. In addition, any Preside specific extensions to the editor are developed and maintained in the core repository, they can be found at: `/system/assets/ckeditorExtensions`.

Finally, we have our own custom javascript object for building instances of the editor. It can be found at `/system/assets/js/admin/core/preside.richeditor.js`.

2.7 Routing

2.7.1 Overview

Routing is the term used to describe how a URL gets mapped to actions and input variables in your application. In PresideCMS, the action will be a Coldbox event handler and the input variables will appear in your request context.

We use Coldbox's own routing system along with a PresideCMS addition for handling dynamic routes. When creating your own custom routes, you are free to use either system.

URLs can be built with `event.buildLink()`. Different routing URLs will be generated depending on the arguments passed to the `buildLink()` function.

2.7.2 Creating custom routes

To create custom routes for your site, you must create a `Routes.cfm` file in your `/application/config/` directory. In this file, you can create regular ColdBox routes as well as PresideCMS routes. The following routes.cfm file registers a couple of PresideCMS route handlers:

```
addRouteHandler( getModel( "myCustomRouteHandler" ) );
addRouteHandler( CreateObject( "app.routeHandlers.anotherCustomRouteHandler" ).init() );
```

PresideCMS Route Handlers

A PresideCMS Route Handler is any CFC that implements a simple interface to handle routing. The interface looks like this:

```
interface {
    // match(): return true if the incoming URL path should be handled by this route handler
    public boolean function match( required string path, required any event ) output=false {}

    // translate(): take an incoming URL and translate it - use the ColdBox event object to set varia
    public void    function translate( required string path, required any event ) output=false {}

    // reverseMatch(): return true if the incomeing set of arguments passed to buildLink() should be
    public boolean function reverseMatch( required struct buildArgs ) output=false {}

    // build(): take incoming buildLink() arguments and return a URL string
    public string  function build( required struct buildArgs ) output=false {}
}
```

An example route handler, that deals with custom URLs for a "My Profile" area of a website, might look like this:

```
component implements="preside.system.routeHandlers.iRouteHandler" output=false {

    public boolean function match( required string path, required any event ) output=false {
        return ReFindNoCase( "^/my-profile/", arguments.path );
    }

    public void function translate( required string path, required any event ) output=false {
        var coldboxEventName = ReReplace( arguments.path, "^/my-profile/", "myprofilemodule:myprofile

        coldboxEventName = ListChangeDelims( coldboxEventName, ".", "/" );

        if ( ListLen( coldboxEventName, "." ) lt 2 ) {
            coldboxEventName = coldboxEventName & "." & "index";
        }

        event.setValue( "event", coldboxEventName );
```

```
    }

    public boolean function reverseMatch( required struct buildArgs ) output=false {
        return Len( Trim( buildArgs.linkTo ?: "" ) ) and ListFirst( buildArgs.linkTo, "." ) eq "mypro
    }

    public string function build( required struct buildArgs ) output=false {
        var link = "/my-profile/#ListChangeDelims( ListRest( buildArgs.linkTo, "." ), "/", "." )#/";

        if ( Len( Trim( buildArgs.queryString ?: "" ) ) ) {
            link &= "?" & buildArgs.queryString;
        }

        return link;
    }
}
```

2.7.3 URL Rewriting

In order for the core routes to work, URL rewrites need to be in place. PresideCMS server distributions ship with the Tuckey URL rewrite filter installed and expect to find a `urlrewrite.xml` file in your webroot. The PresideCMS site skeleton builder creates one of these for you with the following rules which you are then free to modify and/or augment:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE urlrewrite PUBLIC "-//tuckey.org//DTD UrlRewrite 4.0//EN" "http://www.tuckey.org/res/dtds/u
<urlrewrite>
    <rule>
        <note>
            All request to system static assets that live under /preside/system/assets
            should go through Railo and will be rewritten to /index.cfm
        </note>
        <from>^/preside/system/assets/.*$</from>
        <to>%{context-path}/index.cfm</to>
    </rule>

    <rule>
        <note>
            All request to *.html or ending in / will be rewritten to /index.cfm
        </note>
        <from>^(/((.*?)(\.html|/))?)$</from>
        <to>%{context-path}/index.cfm</to>
    </rule>

    <rule>
        <note>
            Disable Railo Context except for local requests
        </note>
        <condition type="remote-addr" operator="notequal">^(127\.0\.0\.1|0:0:0:0:0:0:0:1)$</condition
        <from>^/railo-context/.*$</from>
        <set type="status">404</set>
        <to>null</to>
    </rule>

     <rule>
        <note>
            All the following requests should not be allowed and should return with a 404
```

```
            We block any request to:

            * the application folder (where all the logic and views for your site lives)
            * the uploads folder (should be configured to be somewhere else anyways)
            * this url rewrite file!
        </note>
        <from>^/(application/|uploads/|urlrewrite\.xml\b)</from>
        <set type="status">404</set>
        <to>null</to>
    </rule>
</urlrewrite>
```

2.7.4 Out-of-the-box routes

Site tree pages

Any URL that ends with `.html` followed by an optional query string, will be routed as a site tree page URL. The "directories" and "filename" will correspond to the slugs of the pages in your tree. For example:

```
/about-us/meet-the-team/alex-skinner.html?showComments=true
```

will be routed to:

```
Coldbox event : core.SiteTreePageRequestHandler
Coldbox RC    : { showComments : true }
Coldbox PRC   : { slug : "about-us.meet-the-team.alex-skinner" }
```

and map to the site tree page:

```
/about-us
    /meet-the-team
        alex-skinner
```

Tip: You can build a link to a site tree page with `event.buildLink( page=idOfThePage )`

PresideCMS Admin pages and actions

Any URL that begins with `/(adminPath)` and ends in a forward slash followed by an optional query string, will be routed as a PresideCMS admin request. Directory nodes in the URL will be translated to the ColdBox event.

Note: Your admin path can be configured in your site's `Config.cfc` file with the `settings.preside_admin_path` setting. The setting defaults to "preside_admin".

For example, assuming that `settings.preside_admin_path` has been set to "acme_cmsarea", the URL `/acme_cmsarea/sitetree/editPage/?id=F4554E4C-9347-4F7E-B5F862595BFC9EBF` will be routed to:

```
Coldbox event : admin.sitetree.editPage
Coldbox RC    : { id : "F4554E4C-9347-4F7E-B5F862595BFC9EBF" }
```

Tip: You can build a link to an admin event with `event.buildAdminLink( linkTo="sitetree.editPage", queryString="id=#pageId#" )` or `event.buildLink( linkTo="admin.sitetree.editPage", queryString="id=#pageId#" )`

Asset manager assets

Assets stored in the asset manager are served through the application. Any URL that starts with `/asset` and ends with a trailing slash will be routed to the asset manager download action. URLs take the form: `/asset/(asset ID)/` or `/asset/(asset ID)/(ID or name of derivative)/`. So the URL, `/asset/F4554E4C-9347-4F7E-B5F862595BFC9EBF/`, is routed to:

```
Coldbox event : core.assetDownload
Coldbox RC    : { assetId : "F4554E4C-9347-4F7E-B5F862595BFC9EBF" }
```

and `/asset/F4554E4C-9347-4F7E-B5F862595BFC9EBF/headerImage/` becomes:

```
Coldbox event : core.assetDownload
Coldbox RC    : { assetId : "F4554E4C-9347-4F7E-B5F862595BFC9EBF", derivativeId : "headerImage" }
```

Tip: You can build a link to an asset with `event.buildAdminLink( assetId=myAssetId )` or `event.buildLink( assetId=myAssetId, derivative=derivativeId )`

2.8 CMS Permissioning

2.8.1 Overview

CMS Permissioning is split into three distinct concepts in PresideCMS:

Permissions and roles These are defined in configuration and are not editable through the CMS GUI.

- **Permissions** allow you to grant or deny access to a particular action
- **Roles** provide convenient grant access to one or more permissions

Users and groups Users and groups are defined through the administrative GUI and are stored in the database.

- An *active* **user** must belong to one or more groups
- A **group** must have one or more *roles*

Permissions are granted to a user through the roles that are associated with the groups that she belongs to.

Contextual permissions Contextual permissions are fine grained permissions implemented specifically for any given area of the CMS that requires them.

For example, you could deny the "*Freelancers*" user group the "*Add pages*" permission for a particular page and its children in the sitetree; in this case, the context is the ID of the page.

Contextual permissions are granted or denied to user **groups** and always take precedence over permissions granted through groups and roles.

Note: If a feature of the CMS requires context permissions, it must supply its own views and handlers for

managing them. PresideCMS helps you out here with a viewlet and action handler for some common UI and saving logic, see *ContextPermGUIHelpers*.

2.8.2 Configuring permissions and roles

Permissions and roles are configured in your site or extension's `Config.cfc` file. An example configuration might look like this:

```
public void function configure() output=false {
    super.configure();

// PERMISSIONS
    // here we define a feature, "analytics dashboard" with a number of permissions
    settings.adminPermissions.analyticsdashboard = [ "navigate", "share", "configure" ];

    // features can be organised into sub-features to any depth, here
    // we have a depth of two, i.e. "eventmanagement.events"
    settings.adminPermissions.eventmanagement = {
          events = [ "navigate", "view", "add", "edit", "delete" ]
        , prices = [ "navigate", "view", "add", "edit", "delete" ]
    };

    /* The settings above will translate to the following permission keys being
       available for use in your Railo code, i.e. if ( hasCmsPermission( userId, permissionKey ) ) {

       analyticsdashboard.navigate
       analyticsdashboard.share
       analyticsdashboard.configure

       eventmanagement.events.navigate
       eventmanagement.events.view
       eventmanagement.events.add
       eventmanagement.events.edit
       eventmanagement.events.delete

       eventmanagement.prices.navigate
       eventmanagement.prices.view
       eventmanagement.prices.add
       eventmanagement.prices.edit
       eventmanagement.prices.delete
    */

// ROLES
    // roles are simply a named array of permission keys
    // permission keys for roles can be defined with wildcards (*)
    // and can be excluded with the ! character:

    // define a new role, with all event management perms except for delete
    settings.adminRoles.eventsOrganiser = [ "eventmanagement.*", "!*.delete" ];

    // another new role specifically for analytics viewing
    settings.roles.analyticsViewer = [ "analyticsdashboard.navigate", "analyticsdashboard.share" ];

    // add some new permissions to some existing core roles
    settings.adminRoles.administrator = settings.roles.administrator ?: [];
    settings.adminRoles.administrator.append( "eventmanagement.*" );
```

```
    settings.adminRoles.administrator.append( "analyticsdashboard.*" );

    settings.adminRoles.someRole = settings.roles.someRole ?: [];
```

Defining names and descriptions (i18n)

Names and descriptions for your roles and permissions must be defined in i18n resource bundles.

For roles, you should add *name* and *description* keys for each role to the `/i18n/roles.properties` file, e.g.

```
eventsOrganiser.title=Events organiser
eventsOrganiser.description=The event organiser role grants aspects to all aspects of event managemer

analyticsViewer.title=Analytics viewer
analyticsViewer.description=The analytics viewer role grants permission to view statistics in the ana
```

For permissions, add your keys to the `/i18n/permissions.properties` file, e.g.

```
eventmanagement.events.navigate.title=Events management navigation
eventmanagement.events.navigate.description=View events management navigation links

eventmanagement.events.view=title=View events
eventmanagement.events.view=description=View details of events that have been entered into the syster
```

Note: For permissions, you may only want to create resource bundle entries when the permissions will be used in contextual permission GUIs. Otherwise, the translations will never be used.

2.8.3 Applying permissions in code with hasCmsPermission()

When you wish to permission control a given system feature, you should use the `hasCmsPermission()` method. For example:

```
// a general permission check
if ( !hasCmsPermission( permissionKey="eventmanagement.events.navigate" ) ) {
    event.adminAccessDenied(); // this is a preside request context helper
}

// a contextual permission check. In this case:
// "do we have permission to add folders to the asset folder with id [idOfCurrentFolder]"
if ( !hasCmsPermission( permissionKey="assetManager.folders.add", context="assetmanagerfolders", cont
    event.adminAccessDenied(); // this is a preside request context helper
}
```

Note: The `hasCmsPermission()` method has been implemented as a ColdBox helper method and is available to all your handlers and views. If you wish to access the method from your services, you can access it via the `permissionService` service object, the core implementation of which can be found at `/preside/system/api/security/PermissionService.cfc`.

2.8.4 Rolling out Context Permission GUIs

Should a feature you are developing for the admin require contextual permissions management, you can make use of a viewlet helper to give you a visual form and handler code to manage them.

For example, if we want to be able to manage permissions on event management *per* event, we might have a view at `/views/admin/events/managePermissions.cfm`, that contained the following code:

```
#renderViewlet( event="admin.permissions.contextPermsForm", args={
      permissionKeys = [ "eventmanagement.events.*", "!*.managePerms" ] <!--- permissions that you wa
    , context        = "eventmanager"
    , contextKey     = eventId
    , saveAction     = event.buildAdminLink( linkTo="events.saveEventPermissionsAction", querystring=
    , cancelAction   = event.buildAdminLink( linkTo="events.viewEvent", querystring="id=#eventId#" )
} )#
```

Our `admin.events.saveEventPermissionsAction` handler action might then look like this:

```
function saveEventPermissionsAction( event, rc, prc ) output=false {
  var eventId = rc.id ?: "";

  // check that we are allowed to manage the permissions of this event, or events in general ;)
  if ( !hasCmsPermission( permissionKey="eventmanager.events.manageContextPerms", context="eventmanag
      event.adminAccessDenied();
  }

  // run the core 'admin.Permissions.saveContextPermsAction' event
  // this will save the permissioning configured in the
  // 'admin.permissions.contextPermsForm' form
  var success = runEvent( event="admin.Permissions.saveContextPermsAction", private=true );

  // redirect the user and present them with appropriate message
  if ( success ) {
      messageBox.info( translateResource( uri="cms:eventmanager.permsSaved.confirmation" ) );
      setNextEvent( url=event.buildAdminLink( linkTo="eventmanager.viewEvent", queryString="id=#event
  }

  messageBox.error( translateResource( uri="cms:eventmanager.permsSaved.error" ) );
  setNextEvent( url=event.buildAdminLink( linkTo="events.managePermissions", queryString="id=#eventId
}
```

2.8.5 System users

Users that are defined as **system users** are excempt from all permission checking. In effect, they are granted access to **everything**. This concept exists to enable web agencies to manage every aspect of a site while setting up more secure access for their clients.

System users are only configurable through your site's `Config.cfc` file as a comma separated list of login ids. The default value of this setting is 'sysadmin'. For example, in your site's Config.cfc, you might have:

```
public void function configure() output=false {
   super.configure();

   // ...

   settings.system_users = "sysadmin,developer"; // both the 'developer' and 'sysadmin' users are nou
 }
```

2.9 Website users and permissioning

2.9.1 Overview

PresideCMS supplies a basic core system for setting up user logins and permissioning for your front end websites. This system includes:

- Membership management screens in the administrator
- Ability to create users and user "benefits" (synonymous with user groups)
- Ability to apply access restrictions to site pages and assets through user benefits and individual users
- Core system for dealing with access denied responses (see *401 Access denied pages*)
- Core handlers for processing login, logout and forgotten password

The expectation is that, for more involved sites, these core systems will be extended and interacted with to create a fuller membership experience.

2.9.2 Users and Benefits

We provide a simple model of **users** and **benefits** with two core preside objects, Website user and Website user benefit. A user can have multiple benefits. User benefit's are analogous to user groups.

Note: We have kept the fields for both objects to a bare minimum so as to not impose unwanted logic to your sites. You are encouraged to extend these objects to add your site specific data needs (see *Extending Objects*).

2.9.3 Login

The user object (see Website user) provides core fields for handling login and displaying the currently logged in user's name:

- `login_id`
- `email_address`
- `password`

- `display_name`

Passwords are hashed using BCrypt and the default login procedure checks the supplied login id for a match against either the *login_id* or *email_address* field before checking the validity of the password with BCrypt.

The core login logic can be found in the Website login service.

Core handler actions

In addition to the core service logic, PresideCMS also provides a thin handler layer for processing login and logout and for rendering a login page. The handler can be found at `/system/handlers/Login.cfc`. It provides the following direct actions and viewlets:

Default (index)

The default action will render the loginPage viewlet. It will also redirect the user if they are already logged in. You can access this action with the URL: mysite.com/login/ (generate the URL with `event.buildLink( linkTo="login" )`).

AttemptLogin

The `attemptLogin()` action will process a login attempt, redirecting to the default action on failure or redirecting to the last page accessed (or the default post login page if no last page can be calculated) on success. You can use `event.buildLink( linkTo='login.attemptLogin' )` to build the URL required to access this action.

The action expects the required POST parameters `loginId` and `password` and will also process the optional fields `rememberMe` and `postLoginUrl`.

Logout

The `logout()` action logs the user out of their session and redirects them either to the previous page or, if that cannot be calculated, to the default post logout page.

You can build a logout link with `event.buildLink( linkTo='login.logout' )`.

Viewlet: loginPage

The `loginPage` viewlet is intended to render the login page.

The core view for this viewlet is just a stub and requires a specific implementation per site (see *Example login page implementation* below).

The core handler ensures that the following arguments are passed to the view:

`args.allowRememberMe`	Whether or not remember me functionality is allowed
`args.postLoginUrl`	URL to redirect the user to after successful login
`args.loginId`	Login id that the user entered in their last login attempt (if any)
`args.rememberMe`	Remember me preference that the user chose in their last login attempt (if any)
`args.message`	Message ID that can be used to render a message to the user. Core message IDs are `LOGIN_REQUIRED` and `LOGIN_FAILED`

Note: The default implementation of the access denied error handler renders this viewlet when the cause of the access denial is "LOGIN_REQUIRED" so that your login form will automatically be shown when login is required to access some resource. See *401 Access denied pages* for more detail.

Example login page implementation

The bare minimum requirement to creating a working login system is to create a view that will render your login form. This view will be part of the `login.loginPage` viewlet, so will need to live at `/yoursite/application/views/login/loginPage.cfm`:

```
<cfparam name="args.loginId"         default="" />
<cfparam name="args.password"        default="" />
<cfparam name="args.rememberMe"      default="" />
<cfparam name="args.postLoginUrl"    default="" />
<cfparam name="args.message"         default="" />
<cfparam name="args.allowRememberMe" default=getSystemSetting( "website_users", "allow_remember_me",

<cfoutput>
    <!--- display an alert message based on the args.message parameter --->
    <cfswitch expression="#args.message#">
        <cfcase value="LOGIN_REQUIRED">
            <p class="alert-message">The resource you are attempting to access requires a secure logi
        </cfcase>
        <cfcase value="LOGIN_FAILED">
            <p class="alert-message">The email address and password combination you supplied did not
        </cfcase>
    </cfswitch>

    <h2>Member Login</h2>

    <!--- the form action needs to be the the login.attemptLogin handler action --->
    <form action="#event.buildLink( linkTo="login.attemptLogin" )#" method="post">
        <!--- include the postLoginUrl so that it can be maintained across login attempts --->
        <input type="hidden" name="postLoginUrl" value="#args.postLoginUrl#" />

        <!--- the core login.attemptLogin handler action expects a 'loginId' field --->
        <label for="loginId">Email address <span class="required">*</span></label>
        <input type="email" id="loginId" name="loginId" value="#args.loginId#" class="form-control">

        <!--- the core login.attemptLogin handler action expects a 'password' field --->
        <label for="password">Password <span class="required">*</span></label>
        <input type="password" id="password" name="password" class="form-control">

        <!--- only show remember me checkbox if the feature is enabled --->
        <cfif args.allowRememberMe>
            <input type="checkbox" name="rememberMe" id="rememberMe" value="1"<cfif IsBoolean( args.r
            <label for="rememberMe">Keep me logged in</label>
        </cfif>

        <input type="submit" value="Log in">
    </form>
</cfoutput>
```

Checking login and getting logged in user details

You can check the logged in status of the current user with the helper method, `isLoggedIn()`. Additionally, you can check whether the current user is only auto logged in from a cookie with, `isAutoLoggedIn()`. User details can be retrieved with the helper methods `getLoggedInUserId()` and `getLoggedInUserDetails()`.

For example:

```
// an example 'add comment' handler:
public void function addCommentAction( event, rc, prc ) output=false {
    if ( !isLoggedIn() || isAutoLoggedIn() ) {
        event.accessDenied( "LOGIN_REQUIRED" );
    }

    var userId       = getLoggedInUserId();
    var emailAddress = getLoggedInUserDetails().email_address ?: "";

    // ... etc.
}
```

Login impersonation

CMS administrative users, with sufficient privileges, are able to "impersonate" the login of website users through the admin GUI. Once they have done this, they are treated as a fully logged in user in the front end.

If you wish to restrict these impersonated logins in any way, you can use the `isImpersonated()` method of the Website login service object to check to see whether or not the current login is merely an impersonated one.

2.9.4 Permissions

A permission is something that a user can do within the website. PresideCMS comes with two permissions out of the box, the ability to access a restricted page and the ability to access a restricted asset. These are configured in `Config.cfc` with the `settings.websitePermissions` struct:

```
// /preside/system/config/Config.cfc
component output=false {

    public void function configure() output=false {
        // ... other settings ... //

        settings.websitePermissions = {
              pages  = [ "access" ]
            , assets = [ "access" ]
        };

        // ... other settings ... //

    }

}
```

The core settings above produces two permission keys, "pages.access" and "assets.access", these permission keys are used in creating and checking applied permissions (see below). The permissions can also be directly applied to a given user or benefit in the admin UI:

The title and description of a permission key are defined in `/i18n/permissions.properties`:

```
# ... other keys ...

pages.access.title=Access restricted pages
pages.access.description=Users can view all restricted pages in the site tree unless explicitly denie
```

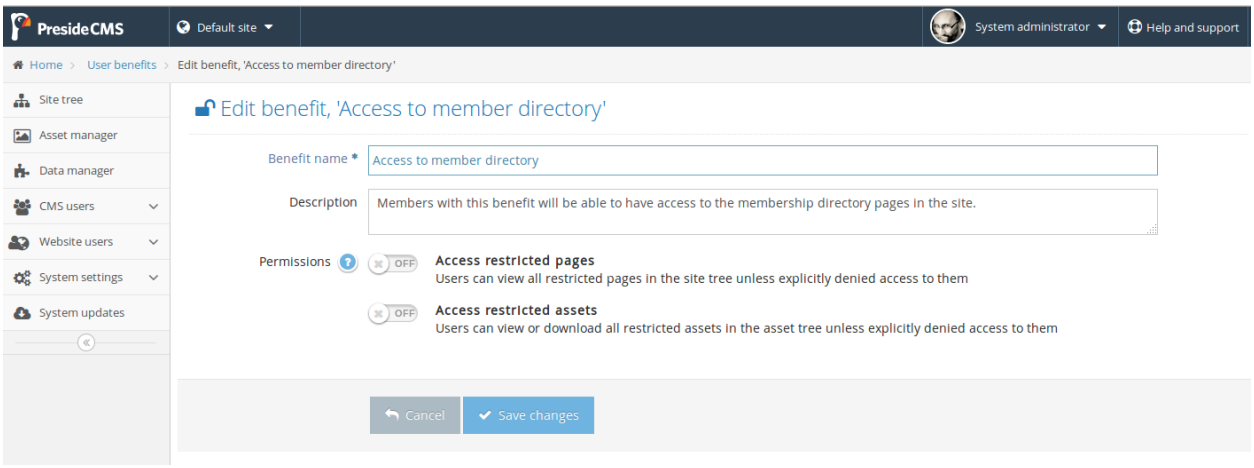

Fig. 2.7: Screenshot of the default edit benefit form. Benefits can have permissions directly applied to them.

```
assets.access.title=Access restricted assets
assets.access.description=Users can view or download all restricted assets in the asset tree unless e
```

Applied permissions and contexts

Applied permissions are instances of a permission that are granted or denied to a particular user or benefit. These instances are stored in the Website applied permission preside object.

Contexts

In addition to being able to set a grant or deny permission against a user or benefit, applied permissions can also be given a **context** and **context key** to create more refined permission schemes.

For instance, when you grant or deny access to a user for a particular **page** in the site tree, you are creating a grant or deny instance with a context of "page" and a context key that is the id of the page.

Defining your own custom permissions

It is likely that you will want to define your own permissions for your site. Examples might be the ability to add comments, or upload documents. Creating the permission keys requires modifying both your site's Config.cfc and permissions.properties files:

```
// /mysite/application/config/Config.cfc
component output=false extends="preside.system.config.Config" {

    public void function configure() output=false {
        super.configure();

        // ... other settings ... //

        settings.websitePermissions.comments = [ "add", "edit" ];
        settings.websitePermissions.documents = [ "upload" ];

        // ... other settings ... //
```

```
    }

}
```

The settings above would produce three keys, `comments.add`, `comments.edit` and `documents.upload`.

```
# /mysite/application/i18n/permissions.properties

comments.add.title=Add comments
comments.add.description=Ability to add comments in our comments system

comments.edit.title=Edit comments
comments.edit.description=Ability to edit their own comments after they have been submitted

documents.upload.title=Upload documents
documents.upload.description=Ability to upload documents to share with other privileged members
```

With the permissions configured as above, the benefit or user edit screen would appear with the new permissions added:

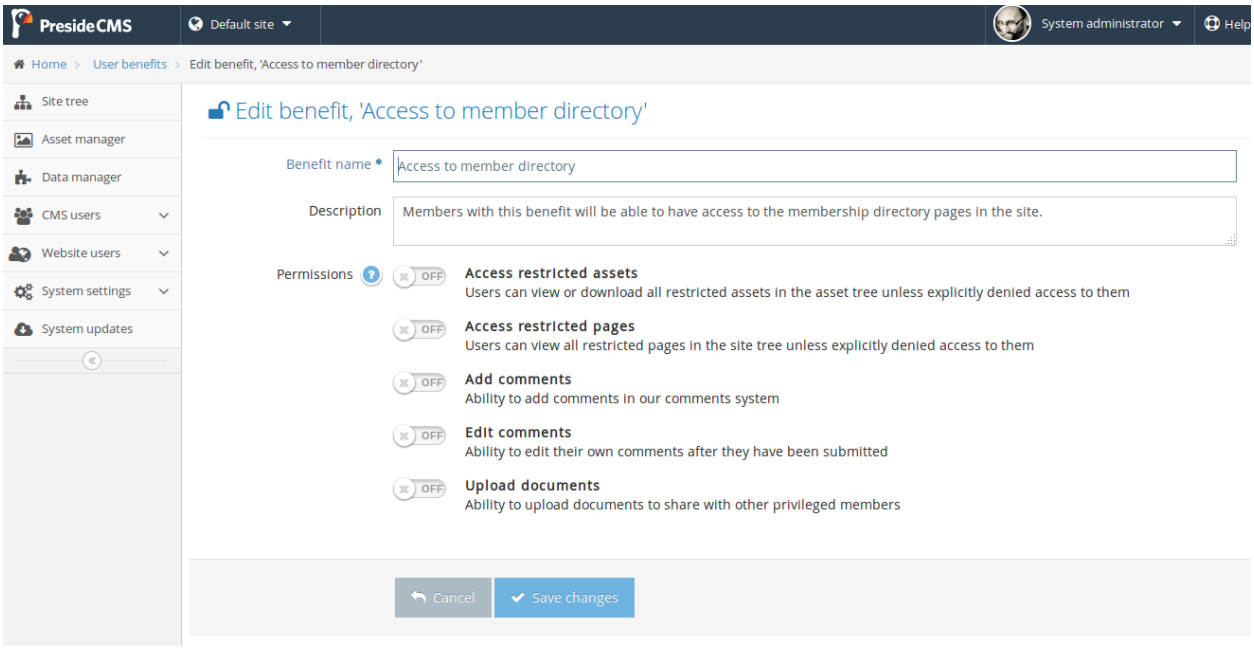

Fig. 2.8: Screenshot of the edit benefit form with custom permissions added.

Checking permissions

Note: The core system already implements permission checking for restricted site tree page access and restricted asset access. You should only require to check permissions for your own custom permission schemes.

You can check to see whether or not the currently logged in user has a particular permission with the `hasWebsitePermission()` helper method. The minimum usage is to pass only the permission key:

```
<cfif hasWebsitePermission( "comments.add" )>
    <button>Add comment</button>
</cfif>
```

You can also check a specific context by passing in the `context` and `contextKeys` arguments:

```
public void function addCommentAction( event, rc, prc ) output=false {
    var hasPermission = hasWebsitePermission(
          permissionKey = "comments.add"
        , context       = "commentthread"
        , contextKeys   = [ rc.thread ?: "" ]
    );

    if ( !hasPermission ) {
        event.accessDenied( reason="INSUFFIENCT_PRIVILEGES" );
    }
}
```

Note: When checking a context permission, you pass an array of context keys to the `hasWebsitePermission()` method. The returned grant or deny permission will be the one associated with the first found context key in the array.

This allows us to implement cascading permission schemes. For site tree access permissions for example, we pass an array of page ids. The first page id is the current page, the next id is it's parent, and so on.

2.10 Editable System settings

2.10.1 Overview

Editable system settings are settings that effect the working of your entire system and that are editable through the CMS admin GUI.

They are stored against a single data object, `system_config`, and are organised into categories.

2.10.2 Categories

A category groups configuration options into a single form. To define a new category, you must:

1. Create a new form layout file at `/forms/system-config/my-category.xml`. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<form>
    <tab>
        <fieldset>
            <field name="hipchat_api_key"              control="textinput"   required="true" label="
            <field name="hipchat_room_name"            control="textinput"   required="true" label="
            <field name="hipchat_use_html_notification" control="yesNoSwitch" required="true" label="
        </fieldset>
    </tab>
</form>
```

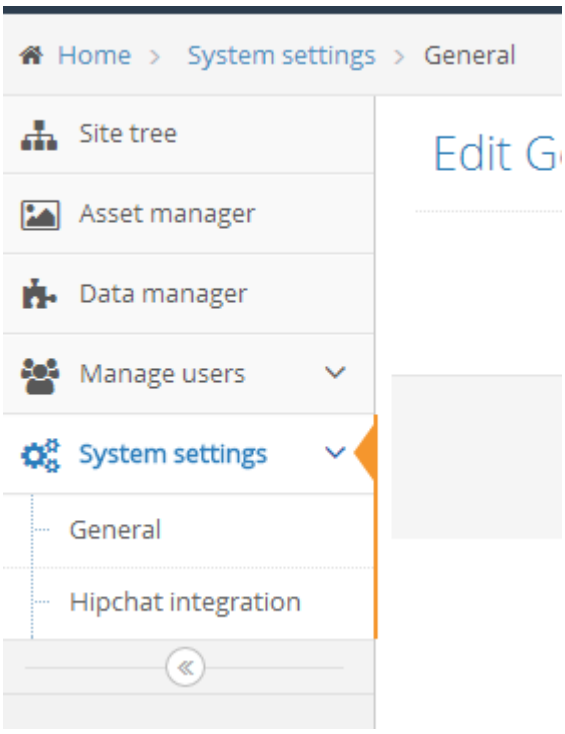

Fig. 2.9: Screenshot showing system settings with two categories, "General" and "Hipchat integration"

2. Create an i18n resource bundle file at `/i18n/system-config/my-category.properties`. This should at least contain `name`, `description` and `iconClass` properties to describe the category. For example:

```
name=Hipchat integration
description=Configure notifications from PresideCMS into your Hipchat rooms
iconClass=fa-comment

api_key.label=API Key
room_name.label=Room name
use_html_notification.label=Use HTML notifications
```

2.10.3 Retrieving settings

From handlers and views

Settings can be retrieved from within your handlers and views with the `getSystemSetting()` method. For example:

```
function myHandler( event, rc, prc ) output=false {
    prc.hipchatApiKey = getSystemSetting(
          category = "hipchat-integration"
        , setting  = "hipchat_api_key"
        , default  = "someDefaultApiKey"
    );
}
```

From withing your service layer

Settings can be injected into your service layer components using the PresideCMS custom WireBox DSL. For example:

```
component output=false {
    property name="hipchatApiKey" inject="presidecms:systemsetting:hipchat-integration.hipchat_api_ke

    ...
}
```

Warning: If you inject settings this way into a singleton, any changes to the settings through the admin will not be reflected in your service object until it is reinstantiated (i.e. a full application reload). In this case, you may wish to use the method described below.

You can also inject the `systemConfigurationService` object itself into your services and use its `getSetting()` method. This will mean that you can always get the latest setting stored in the database at runtime. For example:

```
component output=false {
    property name="systemConfigurationService" inject="systemConfigurationService";

    ...

    private string function _getApiKey() output=false {
        return systemConfigurationService.getSetting(
              category = "hipchat-integration"
            , setting  = "hipchat_api_key"
            , default  = "nokeyselected"
        );
    }
}
```

2.11 Email templating

2.11.1 Overview

PresideCMS comes with a very simple email templating system that allows you to define email templates by creating ColdBox handlers.

Emails are sent through the core email service (see Email service) which in turn invokes template handlers to render the emails and return any other necessary mail parameters.

2.11.2 Creating an email template handler

To create an email template handler, you must create a regular Coldbox handler under the `/handlers/emailTemplates` directory. The handler needs to implement a single *private* action,

`prepareMessage()` that returns a structure containing any message parameters that it needs to set. For example:

```
// /mysite/application/handlers/emailTemplates/adminNotification.cfc
component output=false {

        private struct function prepareMessage( event, rc, prc, args={} ) output=false {
                return {
                          to      = [ getSystemSetting( "email", "admin_notification_address", "" ) ]
                        , from    = getSystemSetting( "email", "default_from_address", "" )
                        , subject = "Admin notification: #( args.notificationTitle ?: '' )#"
                        , htmlBody = renderView( view="/emailTemplates/adminNotification/html", layou
                        , textBody = renderView( view="/emailTemplates/adminNotification/text", args=
                };
        }

}
```

An example send() call for this template might look like this:

```
emailService.send( template="adminNotification", args={
          notificationTitle   = "Something just happened"
        , notificationMessage = "Some message"
} );
```

2.11.3 Supplying message arguments to the send() method

Your email template handlers are not required to supply all the details of the message; these can be left to the calling code to supply. For example, we could refactor the above example so that the `to` and `subject` parameters need to be supplied by the calling code:

```
// /mysite/application/handlers/emailTemplates/adminNotification.cfc
component output=false {

        private struct function prepareMessage( event, rc, prc, args={} ) output=false {
                return {
                          htmlBody = renderView( view="/emailTemplates/adminNotification/html", layou
                        , textBody = renderView( view="/emailTemplates/adminNotification/text", args=
                };
        }

}
```

```
emailService.send(
          template = "adminNotification"
        , args     = { notificationMessage = "Some message" }
        , to       = user.email_address
        , subject  = "Alert: something just happend"
);
```

Note: Note the missing "from" parameter. The core send() implementation will attempt to use the system configuration setting `email.default_from_address` when encountering messages with a missing **from** address. This default address can be configured by users through the PresideCMS administrator (see Editable System settings).

2.11.4 Mail server and other configuration settings

The core system comes with a system configuration form for mail server settings. The form definition can be found here: System config form: Email. See Editable System settings for more details on how this is implemented.

The system uses these configuration values to set the server and port when sending emails. The "default from address" setting is used when sending mail without a specified from address.

This form may be useful to extend in your site should you want to configure other mail related settings. i.e. you might have default "to" addresses for particular admin notification emails, etc.

2.12 Notifications

2.12.1 Overview

PresideCMS comes with a system for raising notifications for the CMS admin users. These notifications may appear in a user's notification feed (see screenshot, below) and/or trigger notification emails. It is also possible to extend the notifications system so that you can have notifications raised in your team's IM tool of choice (Hipchat, Slack, etc.) or any other integration you can think of.

2.12.2 Topics

Notifications are organised into *topics*. A topic might be something like 'Event booking cancelled', or 'User complaint'. In the screenshot above, you can see four notification topics, 'Bookings checked out', 'Invalid CRM contact data', 'Invoice paid' and 'New contact created'.

Creating a topic

The first step is to register the topic in your application's config file. This can be done by appending its unique id to the `settings.notificationTopics` array. For example:

```
// /application/config/Config.cfc
component extends="preside.system.config.Config" output=false {

    public void function configure() output=false {
        super.configure();

        // other settings...

        settings.notificationTopics.append( "customerComplaintFiled" );
    }
}
```

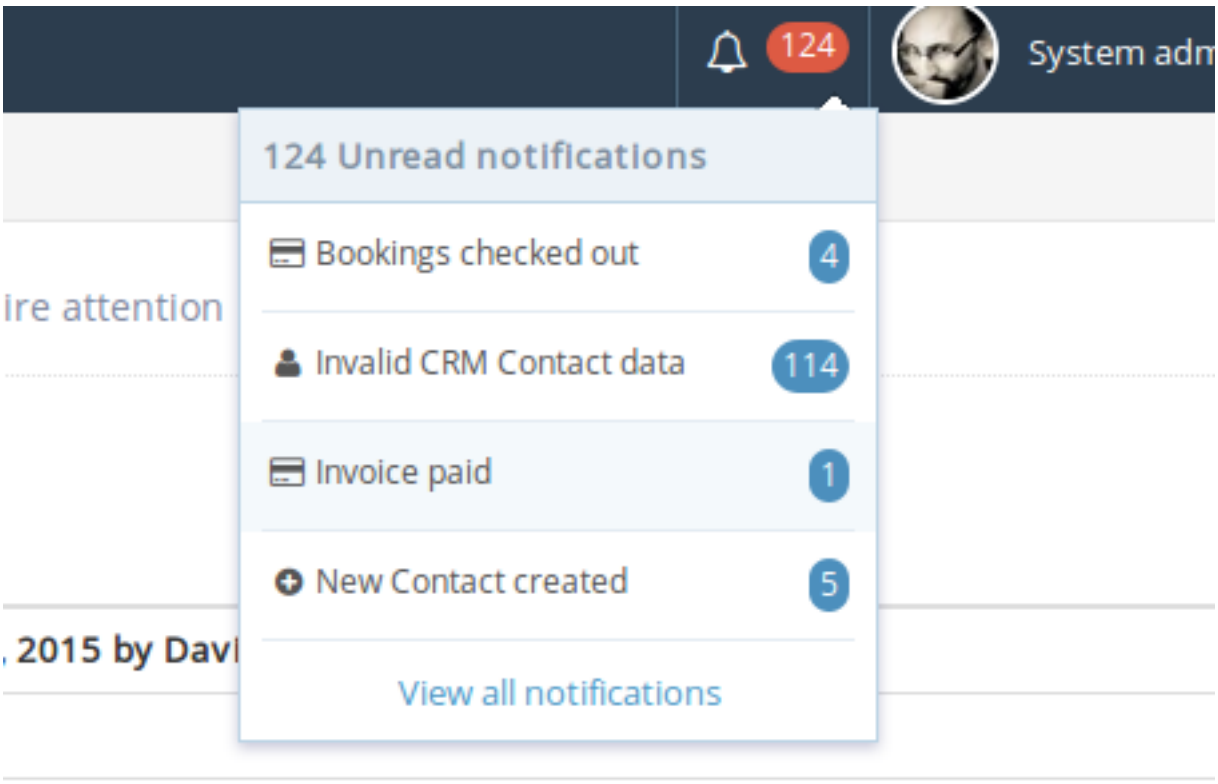

Fig. 2.10: Screenshot showing various programatically raised user notifications.

In order for the topic to render in the notifications panel, it then needs its own i18n .properties file at `/application/i18n/notifications/idOfTopic.properties`. This file needs to contain keys for `title`, `description` and `iconClass`. For example:

```
# /application/i18n/notifications/customerComplaintFiled.properties
title=Customer complaint filed
description=Notifications are raised when customers file complaints through the complaints procedure
iconClass=fa-user
```

2.12.3 Raising a notification

Notifications are raised using the Notification Service object's *CreateNotification()* method. For example, in a ColdBox handler, you might have:

```
component {

    property name="notificationService" inject="notificationService";

    public void function someAction( event, rc, prc ) {
        // some code
        // ...

        notificationService.createNotification(
              topic = "customerComplaintFiled"
            , type  = "ALERT"
            , data  = { complaintId=newlyCreatedComplaintId }
        );

        // some more code...
    }

}
```

2.12.4 Rendering notifications

Notifications can appear in various different *contexts* each of which requires its own renderer. These renderers are implemented as Preside viewlets that take the convention of: `renderers.notifications.{idOfNotification}.{context}`. The `args` struct passed to the viewlet, will contain any data that was passed to the *CreateNotification()* method.

At a bare minimum you must implement viewlets for the **full** and **datatable** contexts (see screenshots below). Additionally, if you want to use a non-default email notification, you can also supply viewlets for the **emailSubject**, **emailHtml** and **emailText** contexts.

Fig. 2.11: The 'datatable' context is shown in the notifications browser screen when showing many notifications in a table view.

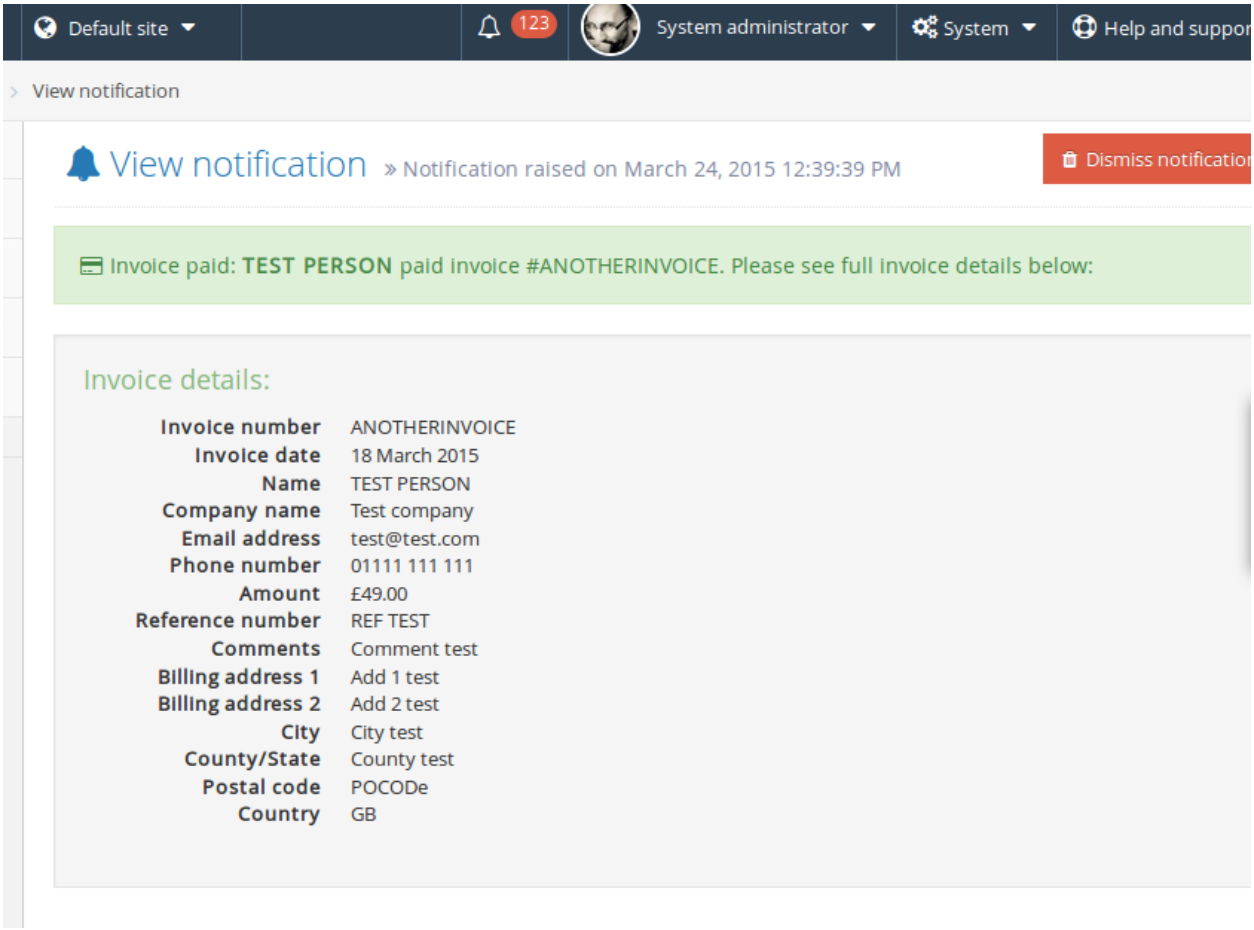

Fig. 2.12: The 'full' context allows you to show full details of the notification within the admin interface. The contents of this view is entirely up to you.

Example renderers

The following code provides an example for our 'customer complaint' notification using both a handler and view files for the various renderer viewlets:

```
// /application/handlers/renderers/notifications/CustomerComplaintFiled.cfc
component {

    property name="customerComplaintsService" inject="customerComplaintsService";

    private string function datatable( event, rc, prc, args={} ) {
        var complaint    = customerComplaintsService.getComplaint( args.complaintId ?: "" );
        var customerName = complaint.customerName ?: "Unknown customer";

        return "A complaint was filed by " & HtmlEditFormat( customerName );
    }

    private string function full( event, rc, prc, args={} ) {
        args.complaint = customerComplaintsService.getComplaint( args.complaintId ?: "" );

        return renderView(
              view = "/renderers/notifications/customerComplaintFiled/full"
            , args = args
        );
    }

    private string function emailSubject( event, rc, prc, args={} ) {
        return "A customer complaint was filed through the website";
    }

    private string function emailHtml( event, rc, prc, args={} ) {
        args.complaint = customerComplaintsService.getComplaint( args.complaintId ?: "" );

        return renderView(
              view = "/renderers/notifications/customerComplaintFiled/emailHtml"
            , args = args
        );
    }

    private string function emailText( event, rc, prc, args={} ) {
        args.complaint = customerComplaintsService.getComplaint( args.complaintId ?: "" );

        return renderView(
              view = "/renderers/notifications/customerComplaintFiled/emailText"
            , args = args
        );
    }

}
```

```
<!--- /views/renderers/notifications/customerComplaintFiled/full.cfm --->
<cfparam name="args.complaint.customerName" type="string" />
<cfparam name="args.complaint.complaint"    type="string" />
<cfparam name="args.complaint.dateMade"     type="string" />

<cfoutput>
    <div class="alert alert-danger">
        <h3><i class="fa fa-fw fa-user"></i> Customer complaint made by #args.complaint.customerName#
```

```
        <p>#HtmlEditFormat( args.complaint.complaint )#</p>
    </div>
</cfoutput>
```

```
<!--- /views/renderers/notifications/customerComplaintFiled/emailHtml.cfm --->
<cfparam name="args.complaint.customerName" type="string" />
<cfparam name="args.complaint.complaint"    type="string" />
<cfparam name="args.complaint.dateMade"     type="string" />

<cfoutput>
    <p><bold>Customer complaint made by #args.complaint.customerName# on #args.complaint.dateMade#</b

    <blockquote>#HtmlEditFormat( args.complaint.complaint )#</blockquote>
</cfoutput>
```

```
<!--- /views/renderers/notifications/customerComplaintFiled/emailText.cfm --->
<cfparam name="args.complaint.customerName" type="string" />
<cfparam name="args.complaint.complaint"    type="string" />
<cfparam name="args.complaint.dateMade"     type="string" />

<cfoutput>
Customer complaint made by #args.complaint.customerName# on #args.complaint.dateMade#:

-----

#args.complaint.complaint#
</cfoutput>
```

2.12.5 Creating notification extensions

TODO. If you have a requirement to do this, please get in touch.

2.13 Custom error pages & Maintenance mode

2.13.1 Overview

PresideCMS provides a simple mechanism for creating custom 401, 404 and 500 error pages while providing the flexibility to allow you to implement more complex systems should you need it.

2.13.2 404 Not found pages

Creating a 404 template

The 404 template is implemented as a Preside Viewlet (see Preside viewlets) and a core implementation already exists. The name of the viewlet is configured in your application's Config.cfc with the `notFoundViewlet` setting. The default is "errors.notFound":

```
// /application/config/Config.cfc
component extends="preside.system.config.Config" output=false {

    public void function configure() output=false {
        super.configure();

        // other settings...
        settings.notFoundViewlet = "errors.notFound";
    }
}
```

For simple cases, you will only need to override the `/errors/notFound` view by creating one in your application's view folder, e.g.

```
<!--- /application/views/errors/notFound.cfm --->
<h1>These are not the droids you are looking for</h1>
<p> Some pithy remark.</p>
```

Implementing handler logic

If you wish to perform some handler logic for your 404 template, you can simply create the Errors.cfc handler file and implement the "notFound" action. For example:

```
// /application/handlers/Errors.cfc
component output=false {

    private string function notFound( event, rc, prc, args={} ) output=false {
        event.setHTTPHeader( statusCode="404" );
        event.setHTTPHeader( name="X-Robots-Tag", value="noindex" );

        return renderView( view="/errors/notFound", args=args );
    }
}
```

Defining a layout template

The default layout template for the 404 is your site's default layout, i.e. "Main" (/application/layouts/Main.cfm). If you wish to configure a different default layout template for your 404 template, you can do so with the `notFoundLayout` configuration option, i.e.

```
// /application/config/Config.cfc
component extends="preside.system.config.Config" output=false {

    public void function configure() output=false {
        super.configure();

        // other settings...

        settings.notFoundLayout  = "404Layout";
        settings.notFoundViewlet = "errors.my404Viewlet";
    }
}
```

You can also programatically set the layout for your 404 template in your handler (you may wish to dynamically pick the layout depending on a number of variables):

```
// /application/handlers/Errors.cfc
component output=false {

    private string function notFound( event, rc, prc, args={} ) output=false {
        event.setHTTPHeader( statusCode="404" );
        event.setHTTPHeader( name="X-Robots-Tag", value="noindex" );
        event.setLayout( "404Layout" );

        return renderView( view="/errors/notFound", args=args );
    }
}
```

Programatically responding with a 404

If you ever need to programatically respond with a 404 status, you can use the `event.notFound()` method to do so. This method will ensure that the 404 statuscode header is set and will render your configured 404 template for you. For example:

```
// someHandler.cfc
component output=false {

    public void function index( event, rc, prc ) output=false {
        prc.record = getModel( "someService" ).getRecord( rc.id ?: "" );

        if ( !prc.record.recordCount ) {
            event.notFound();
        }

        // .. carry on processing the page
    }
}
```

Direct access to the 404 template

The 404 template can be directly accessed by visiting /404.html. This is achieved through a custom route dedicated to error pages (see Routing).

This is particular useful for rendering the 404 template in cases where PresideCMS is not producing the 404. For example, you may be serving static assets directly through Tomcat and want to see the custom 404 template when one of these assets is missing. To do this, you would edit your `${catalina_home}/config/web.xml` file to define a rewrite URL for 404s:

```
<!-- ... -->

        <welcome-file-list>
        <welcome-file>index.cfm</welcome-file>
    </welcome-file-list>

    <error-page>
        <error-code>404</error-code>
        <location>/404.html</location>
    </error-page>

</web-app>
```

Another example is producing 404 responses for secured areas of the application. In PresideCMS's default urlrewrite.xml file (that works with Tuckey URL Rewrite), we block access to files such as Application.cfc by responding with a 404:

```
<rule>
    <name>Block access to certain URLs</name>
    <note>
        All the following requests should not be allowed and should return with a 404:

        * the application folder (where all the logic and views for your site lives)
        * the uploads folder (should be configured to be somewhere else anyways)
        * this url rewrite file!
        * Application.cfc
    </note>
    <from>^/(application/|uploads/|urlrewrite\.xml\b|Application\.cfc\b)</from>
    <set type="status">404</set>
    <to last="true">/404.html</to>
</rule>
```

2.13.3 401 Access denied pages

Access denied pages can be created and used in exactly the same way as 404 pages, with a few minor differences. The page can be invoked with `event.accessDenied( reason=deniedReason )` and will be automatically invoked by the core access control system when a user attempts to access pages and assets to which they do not have permission.

Hint: For a more in depth look at front end user permissioning and login, see Website users and permissioning.

Creating a 401 template

The 401 template is implemented as a Preside Viewlet (see Preside viewlets) and a core implementation already exists. The name of the viewlet is configured in your application's Config.cfc with the `accessDeniedViewlet` setting. The default is "errors.accessDenied":

```
// /application/config/Config.cfc
component extends="preside.system.config.Config" output=false {

    public void function configure() output=false {
        super.configure();

        // other settings...
        settings.accessDeniedViewlet = "errors.accessDenied";
    }
}
```

The viewlet will be passed an `args.reason` argument that will be either `LOGIN_REQUIRED`, `INSUFFICIENT_PRIVILEGES` or any other codes that you might make use of.

The core implementation sets the 401 header and then renders a different view, depending on the access denied reason:

```
// /preside/system/handlers/Errors.cfc
component output=false {

    private string function accessDenied( event, rc, prc, args={} ) output=false {
        event.setHTTPHeader( statusCode="401" );
        event.setHTTPHeader( name="X-Robots-Tag"    , value="noindex" );
        event.setHTTPHeader( name="WWW-Authenticate", value='Website realm="website"' );

        switch( args.reason ?: "" ){
            case "INSUFFICIENT_PRIVILEGES":
                return renderView( view="/errors/insufficientPrivileges", args=args );
            default:
                return renderView( view="/errors/loginRequired", args=args );
        }
    }
}
```

For simple cases, you will only need to override the `/errors/insufficientPrivileges` and/or `/errors/loginRequired` view by creating them in your application's view folder, e.g.

```
<!--- /application/views/errors/insufficientPrivileges.cfm --->
<h1>Name's not on the door, you ain't coming in</h1>
<p> Some pithy remark.</p>
```

```
<!--- /application/views/errors/loginRequired.cfm --->
#renderViewlet( event="login.loginPage", message="LOGIN_REQUIRED" )#
```

Implementing handler logic

If you wish to perform some handler logic for your 401 template, you can simply create the Errors.cfc handler file and implement the "accessDenied" action. For example:

```
// /application/handlers/Errors.cfc
component output=false {
    private string function accessDenied( event, rc, prc, args={} ) output=false {
        event.setHTTPHeader( statusCode="401" );
        event.setHTTPHeader( name="X-Robots-Tag"    , value="noindex" );
        event.setHTTPHeader( name="WWW-Authenticate", value='Website realm="website"' );

        switch( args.reason ?: "" ){
            case "INSUFFICIENT_PRIVILEGES":
                return renderView( view="/errors/my401View", args=args );
            case "MY_OWN_REASON":
                return renderView( view="/errors/custom401", args=args );
            default:
                return renderView( view="/errors/myLoginFormView", args=args );
        }
    }
}
```

Defining a layout template

The default layout template for the 401 is your site's default layout, i.e. "Main" (/application/layouts/Main.cfm). If you wish to configure a different default layout template for your 401 template, you can do so with the `accessDeniedLayout` configuration option, i.e.

```
// /application/config/Config.cfc
component extends="preside.system.config.Config" output=false {

    public void function configure() output=false {
        super.configure();

        // other settings...

        settings.accessDeniedLayout  = "401Layout";
        settings.accessDeniedViewlet = "errors.my401Viewlet";
    }
}
```

You can also programatically set the layout for your 401 template in your handler (you may wish to dynamically pick the layout depending on a number of variables):

```
// /application/handlers/Errors.cfc
component output=false {
    private string function accessDenied( event, rc, prc, args={} ) output=false {
        event.setHTTPHeader( statusCode="401" );
        event.setHTTPHeader( name="X-Robots-Tag"    , value="noindex" );
        event.setHTTPHeader( name="WWW-Authenticate", value='Website realm="website"' ); // this head

        event.setLayout( "myCustom401Layout" );

        // ... etc.
    }
}
```

Programatically responding with a 401

If you ever need to programatically respond with a 401 access denied status, you can use the `event.accessDenied( reason="MY_REASON" )` method to do so. This method will ensure that the 401 statuscode header is set and will render your configured 401 template for you. For example:

```
// someHandler.cfc
component output=false {

    public void function reservePlace( event, rc, prc ) output=false {
        if ( !isLoggedIn() ) {
            event.accessDenied( reason="LOGIN_REQUIRED" );
        }
        if ( !hasWebsitePermission( "events.reserveplace" ) ) {
            event.accessDenied( reason="INSUFFICIENT_PRIVILEGES" );
        }

        // .. carry on processing the page
    }
}
```

2.13.4 500 Error Pages

The implementation of 500 error pages is more straight forward than the 40x templates and involves only creating a flat `500.htm` file in your webroot. The reason behind this is that a server error may be caused by your site's layout code, or may even occur before PresideCMS code is called at all; in which case the code to render your error template will not be available.

If you do not create a `500.htm` in your webroot, PresideCMS will use it's own default template for errors. This can be found at `/preside/system/html/500.htm`.

Bypassing the error template

In your local development environment, you will want to be able see the details of errors, rather than view a simple error message. This can be achieved with the config setting, `showErrors`:

```
// /application/config/Config.cfc
component extends="preside.system.config.Config" output=false {

    public void function configure() output=false {
        super.configure();

        // other settings...

        settings.showErrors = true;
    }
}
```

In most cases however, you will not need to configure this for your local environment. PresideCMS uses ColdBox's environment configuration (see `coldboxenvironments`) to configure a "local" environment that already has `showErrors` set to **true** for you. If you wish to override that setting, you can do so by creating your own "local" environment function:

```
// /application/config/Config.cfc
component extends="preside.system.config.Config" output=false {
```

```
    public void function configure() output=false {
        super.configure();

        // other settings...
    }

    public void function local() output=false {
        super.local();

        settings.showErrors = false;
    }
}
```

Note: PresideCMS's built-in local environment configuration will map URLs like "mysite.local", "local.mysite", "localhost" and "127.0.0.1" to the "local" environment.

2.13.5 503 Maintenance mode page

The administrator interface provides a simple GUI for putting the site into maintenance mode (see figure below). This interface allows administrators to enter a custom title and message, turn maintenance mode on/off and also to supply custom settings to allow users to bypass maintenance mode.

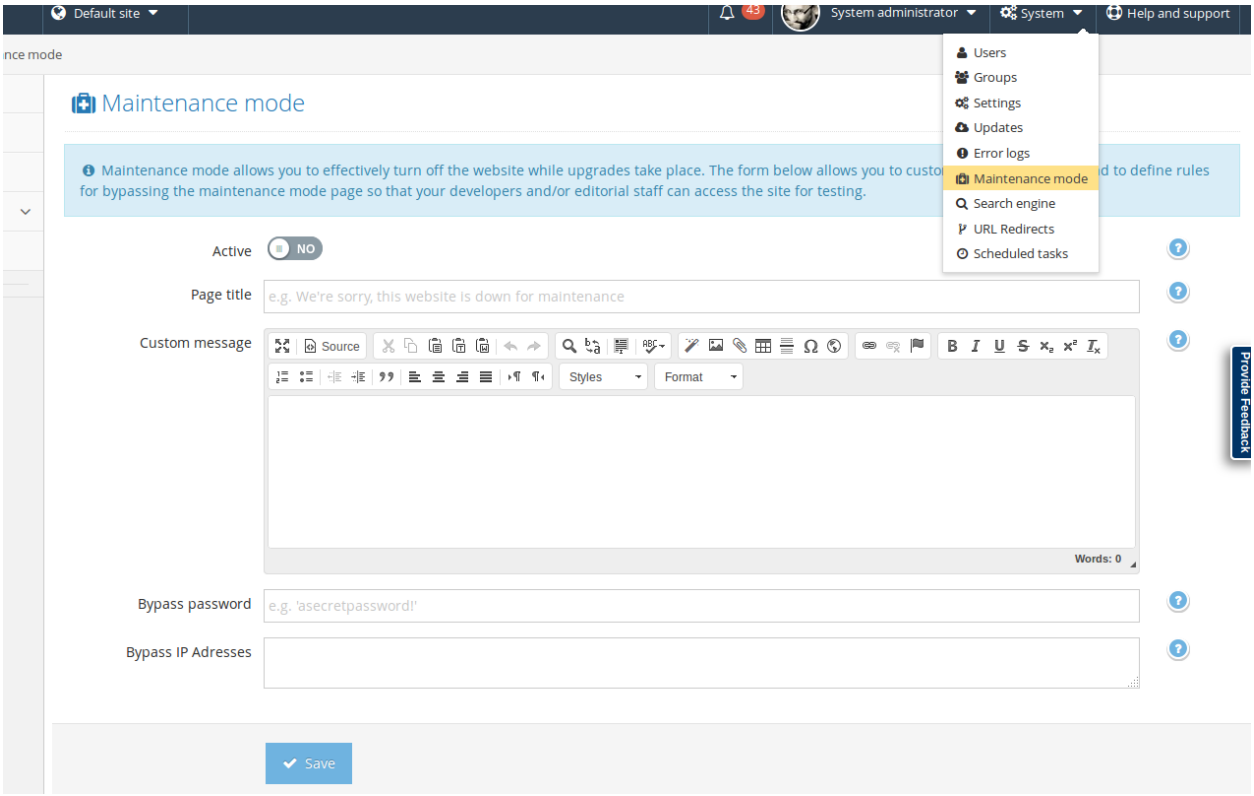

Fig. 2.13: Screenshot of maintenance mode management GUI

Creating a custom 503 page

The 503 template is implemented as a Preside Viewlet (see Preside viewlets) and a core implementation already exists. The name of the viewlet is configured in your application's Config.cfc with the `maintenanceModeViewlet` setting. The default is "errors.maintenanceMode":

```
// /application/config/Config.cfc
component extends="preside.system.config.Config" output=false {

    public void function configure() output=false {
        super.configure();

        // other settings...
        settings.maintenanceModeViewlet = "errors.maintenanceMode";
    }
}
```

To create a custom template, you can choose either to provide your own viewlet by changing the config setting, or by overriding the view and/or handler of the `errors.maintenanceMode` viewlet.

For example, in your site's `/application/views/errors/` folder, you could create a `maintenanceMode.cfm` file with the following:

```
<cfparam name="args.title"   />
<cfparam name="args.message" />

<cfoutput><!DOCTYPE html>
<html>
    <head>
        <title>#args.title#</title>
        <meta charset="utf-8">
        <meta name="robots" content="noindex,nofollow" />
    </head>
    <body>
        <h1>#args.title#</h1>
        #args.message#
    </body>
</html></cfoutput>
```

Note: The maintenance mode viewlet needs to render the entire HTML of the page.

Manually clearing maintenance mode

You may find yourself in a situation where you application is in maintenance mode and you have no means by which to access the admin because the password has been lost. In this case, you have two options:

Method 1: Set bypass password directly in the database

To find the current bypass password, you can query the database with:

```
select value
from   psys_system_config
where  category = 'maintenanceMode'
and    setting  = 'bypass_password';
```

If the value does not exist, create it with:

```
insert into psys_system_config (id, category, setting, `value`, datecreated, datemodified)
values( '{a unique id}', 'maintenancemode', 'bypass_password', '{new password}', now(), now() );
```

The bypass password can then be used by supplying it as a URL parameter to your site, e.g. `http://www.mysite.com/?thepassword`. From there, you should be able to login to the administrator and turn off maintenance mode.

Method 2: Delete the maintenance mode file

When maintenance mode is activated, a file is created at `/yoursite/application/config/.maintenance`. To clear maintenance mode, delete that file and restart the application.

2.14 Sitetree Navigation Menus

2.14.1 Overview

A common task for CMS driven websites is to build navigation menus based on the site tree. PresideCMS provides two extendable viewlets (see Preside viewlets) to aid in rendering such menus with the minimum of fuss; `core.navigation.mainNavigation` and `core.navigation.subNavigation`.

2.14.2 Main navigation

The purpose of the main navigation viewlet is to render the menu that normally appears at the top of a website and that is usually either one, two or three levels deep. For example:

```
<nav role="navigation">
    <ul class="nav navbar-nav">
        <li class="hiddex-sm home-nav"><a href="/"><span class="fa fa-home"></span></a></li>

        #renderViewlet( event="core.navigation.mainNavigation", args={ depth=2 } )#
    </ul>
</nav>
```

This would result in output that looked something like this:

```
<nav class="site-navigation" role="navigation">
    <ul class="nav navbar-nav">
        <li class="hiddex-sm home-nav"><a href="/"><span class="fa fa-home"></span></a></li>
```

```
        <!-- start of "core.navigation.mainNavigation" -->
        <li class="active">
            <a href="/news.html">News</a>
        </li>
        <li class="dropdown">
            <a href="/about.html">About us</a>
            <ul class="dropdown-menu" role="menu">
                <li><a href="/about/team.html">Our team</a></li>
                <li><a href="/about/offices.html">Our offices</a></li>
                <li><a href="/about/ethos.html">Our ethos</a></li>
            </ul>
        </li>
        <li>
            <a href="/contact.html">Contact</a>
        </li>
        <!-- end of "core.navigation.mainNavigation" -->
    </ul>
</nav>
```

Note: Notice how the core implementation does not render the outer `<ul>` element for you. This allows you to build navigation items either side of the automatically generated navigation such as login links and other application driven navigation.

Viewlet options

You can pass the following arguments to the viewlet through the `args` structure:

Name	Description
`rootPage`	ID of the page who's children make up the top level of the menu. This defaults to the site's homepage.
`depth`	Number of nested dropdown levels to drill into. Default is 1, i.e. just render the immediate children of the root page and have no drop downs

Overriding the view

You might find yourself in a position where the HTML markup provided by the core implementation does not suit your needs. You can override this markup by providing a view at `/views/core/navigaton/mainNavigation.cfm`. The view will be passed a single argument, `args.menuItems`, which is an array of structs who's structure looks like this:

```
[
    {
        "id"       : "F9923DE1-9B2D-4544-A4E7F8E198888211",
        "title"    : "News",
        "active"   : true,
        "children" : []
    },
    {
        "id"       : "F9923DE1-9B2D-4544-A4E7F8E198888A6F",
        "title"    : "About us",
        "active"   : false,
        "children" : [
            {
                "id"       : "F9923DE1-9B2D-4544-A4E7F8E198888000",
                "title"    : "Our team",
                "active"   : false,
```

```
                "children" : []
            },
            {
                "id"       : "F9923DE1-9B2D-4544-A4E7F8E198888FF8",
                "title"    : "Our offices",
                "active"   : false,
                "children" : []
            },
            {
                "id"       : "F9923DE1-9B2D-4544-A4E7F8E1988887FE",
                "title"    : "Our ethos",
                "active"   : false,
                "children" : []
            }
        ]
    },
    {
        "id"       : "F9923DE1-9B2D-4544-A4E7F8E19888834A",
        "title"    : "COntact us",
        "active"   : false,
        "children" : []
    }
]
```

This is what the core view implementation looks like:

```
<cfoutput>
    <cfloop array="#( args.menuItems ?: [] )#" index="i" item="item">
        <li class="<cfif item.active>active </cfif><cfif item.children.len()>dropdown</cfif>">
            <a href="#event.buildLink( page=item.id )#">#item.title#</a>
            <cfif item.children.len()>
                <ul class="dropdown-menu" role="menu">
                    <!--- NOTE the recursion here --->
                    #renderView( view='/core/navigation/mainNavigation', args={ menuItems=item.childr
                </ul>
            </cfif>
        </li>
    </cfloop>
</cfoutput>
```

2.14.3 Sub navigation

The sub navigation viewlet renders a navigation menu that is often placed in a sidebar and that shows siblings, parents and siblings of parents of the current page. For example:

```
News
*Events and training*
    Annual Conference
    *Online*
        Free webinars
        *Bespoke online training* <-- current page
About us
Contact us
```

This viewlet works in exactly the same way to the main navigation viewlet, however, the HTML output and the input arguments are very slightly different:

Viewlet options

Name	Description
`startLevel`	At what depth in the tree to start at. Default is 2. This will produce a different root page for the menu depending on where in the tree the current page lives
`depth`	Number of nested menu levels to drill into. Default is 3.

Overriding the view

Override the markup for the sub navigation viewlet by providing a view file at `/views/core/navigaton/subNavigation.cfm`. The view will be passed two arguments, `args.menuItems` and `args.rootTitle`. The `args.menuItems` argument is the nested array of menu items. The `args.rootTitle` argument is the title of the root page of the menu (who's children makeup the top level of the menu).

The core view looks like this:

```
<cfoutput>
    <cfloop array="#( args.menuItems ?: [] )#" item="item">
        <li class="<cfif item.active>active </cfif><cfif item.children.len()>has-submenu</cfif>">
            <a href="#event.buildLink( page=item.id )#">#item.title#</a>
            <cfif item.children.len()>
                <ul class="submenu">
                    #renderView( view="/core/navigation/subNavigation", args={ menuItems=item.childre
                </ul>
            </cfif>
        </li>
    </cfloop>
</cfoutput>
```

2.14.4 Crumbtrail

The crumbtrail is the simplest of all the viewlets and is implemented as two methods in the request context and as a viewlet with just a view (feel free to add your own handler if you need one).

The view looks like this:

```
<!--- /preside/system/views/core/navigation/breadCrumbs.cfm --->
<cfset crumbs = event.getBreadCrumbs() />
<cfoutput>
    <cfloop array="#crumbs#" index="i" item="crumb">
        <cfset last = i eq crumbs.len() />

        <li class="<cfif last>active</cfif>">
            <cfif last>
                #crumb.title#
            <cfelse>
                <a href="#crumb.link#">#crumb.title#</a>
            </cfif>
        </li>
    </cfloop>
</cfoutput>
```

Note: Note that again we are only outputting the `<li>` tags in the core view, leaving you free to implement your own list wrapper HTML.

Request context helper methods

There are two helper methods available to you in the request context, `event.getBreadCrumbs()` and *event.addBreadCrumb(title, link)*.

The `getBreadCrumbs()` method returns an array of the breadcrumbs that have been registered for the request. Each breadcrumb is a structure containing `title` and `link` keys.

The `addBreadCrumb()` method allows you to append a breadcrumb item to the current stack. It requires you to pass both a title and a link for the breadcrumb item.

Note: The core site tree page handler will automatically register the breadcrumbs for the current page.

2.15 Modifying the administrator left hand menu

2.15.1 Overview

PresideCMS provides a simple mechanism for configuring the left hand menu of the administrator, either to add new main navigational sections, take existing ones away or to modify the order of menu items.

2.15.2 Configuration

Each top level item of the menu is stored in an array that is set in `settings.adminSideBarItems` in `Config.cfc`. The core implementation looks like this:

```
component output=false {

    public void function configure() output=false {

        // ... other settings ...

        settings.adminSideBarItems = [
              "sitetree"
            , "assetmanager"
            , "datamanager"
            , "usermanager"
            , "websiteUserManager"
```

```
            , "systemConfiguration"
            , "updateManager"
        ];

        // ... other settings ...

    }
}
```

Each of these side bar items is then implemented as a view that lives under a `/views/admin/layout/sidebar/` folder. For example, for the 'sitetree' item, there exists a view at `/views/admin/layout/sidebar/sitetree.cfm` that looks like this:

```
// /views/admin/layout/sidebar/sitetree.cfm

if ( hasCmsPermission( "sitetree.navigate" ) ) {
    WriteOutput( renderView(
          view = "/admin/layout/sidebar/_menuItem"
        , args = {
              active  = ListLast( event.getCurrentHandler(), ".") eq "sitetree"
            , link    = event.buildAdminLink( linkTo="sitetree" )
            , gotoKey = "s"
            , icon    = "fa-sitemap"
            , title   = translateResource( 'cms:sitetree' )
          }
    ) );
}
```

2.15.3 Core view helpers

There are two core views that can be used when rendering your menu items, `/admin/layout/sidebar/_menuItem` and `/admin/layout/sidebar/_subMenuItem`.

/admin/layout/sidebar/_menuItem

Renders a top level menu item.

Arguments

Argument	Description
active	Boolean. Whether or not the current page lives within this part of the CMS.
link	Where this menu item points to. Not needed when the menu item has a submenu.
title	Title of the menu item
icon	Icon class for the menu item. We use font awesome, so "fa-users" for example.
subMenu	Rendered submenu items.
subMenuItems	Array of sub menu items to render (alternative to supplying a rendered sub menu). Each item should be a struct with `link`, `title` and optional `gotoKey` keys
gotoKey	Optional key that when used in combination with the `g` key, will send the user to the item's link. e.g. `g+s` takes you to the site tree.

Example

```
<cfset subMenuItems = [] />
<cfif hasCmsPermission( "mynewsubfeature.access" )>
    <cfset subMenuItems.append( {
          link  = event.buildAdminLink( linkTo="mynewsubfeature" )
        , title = event.translateResource( uri="mynewsubfeature.menu.title" )
    } ) />
</cfif>
<cfif hasCmsPermission( "myothernewsubfeature.access" )>
    <cfset subMenuItems.append( {
          link  = event.buildAdminLink( linkTo="myothernewsubfeature" )
        , title = event.translateResource( uri="myothernewsubfeature.menu.title" )
    } ) />
</cfif>


#renderView( view="/admin/layout/sidebar/_menuItem", args={
      active       = ReFindNoCase( "my(other)?newsubfeature$", event.getCurrentHandler() )
    , title        = event.translateResource( uri="mynewfeature.menu.title" )
    , icon         = "fa-world-domination"
    , subMenuItems = subMenuItems
} )#
```

/admin/layout/sidebar/_subMenuItem

Renders a sub menu item.

Arguments

Argument	Description
link	Where this menu item points to.
title	Title of the menu item
gotoKey	Optional key that when used in combination with the `g` key, will send the user to the item's link. e.g. `g+s` takes you to the site tree.

Example

```
<cfif hasCmsPermission( "mynewsubfeature.access" )>
    #renderView( view="/admin/layout/sidebar/_subMenuItem", args={
          link    = event.buildAdminLink( linkTo="mynewsubfeature" )
        , title   = event.translateResource( uri="mynewsubfeature.menu.title" )
        , gotoKey = "f"
    } )#
</cfif>
```

2.15.4 Examples

Adding a new item

Firstly, add the item to our array of sidebar items in your site or extension's Config.cfc:

```
// ...

settings.adminSideBarItems.append( "mynewfeature" );

// ...
```

Finally, create the view for the side bar item:

```
<!--- /views/admin/layout/sidebar/mynewfeature.cfm --->
<cfif hasCmsPermission( "mynewfeature.access" )>
    <cfoutput>
        #renderView( view="/admin/layout/sidebar/_menuItem", args={
              active       = ReFindNoCase( "mynewfeature$", event.getCurrentHandler() )
            , title        = event.translateResource( uri="mynewfeature.menu.title" )
            , link         = event.buildAdminLink( linkTo="mynewfeature" )
            , icon         = "fa-world-domination"
            , subMenuItems = subMenuItems
        } )#
    </cfoutput>
</cfif>
```

Note: In order for the calls to `hasCmsPermission()` and `translateResource()` to do anything useful, you will need to have setup the necessary permission keys (see CMS Permissioning) and resource bundle keys (see `i18n`).

Remove an existing item

In your site or extension's `Config.cfc` file:

```
// ...

// delete the site tree menu item, for example:
settings.adminSideBarItems.delete( "sitetree" );

// ...
```

2.16 Working with uploaded files

PresideCMS comes with its own Digital Asset Manager (see `assetmanager`) and in many cases this will meet your document / image uploading needs. However, there are scenarios in which the users of your website will upload files that will not warrant a presence in your asset manager and the following APIs and practices can be used to deal with these cases.

2.16.1 The storage provider interface

PresideCMS has a concept of a "Storage Provider" and provides an interface at `/system/services/fileStorage/StorageProvider.cfc`. A storage provider is a an API interface to any implementation of a system that can store and serve files. The system provides a concrete implementation using a regular file system which can be found at `/system/services/fileStorage/FileSystemStorageProvider.cfc`.

Note: The core asset manager system uses storage providers for its file storage.

Distinct storage provider instances can be created through Wirebox by mapping the storage provider class to an id and passing your custom configuration, i.e. the physical directories in which you will store files, or credentials for a CDN API, etc. Below is an example of creating a storage provider instance with your own file path in your application's Wirebox.cfc file (`/application/config/Wirebox.cfc`):

```
component extends="preside.system.config.WireBox" {

    public void function configure() {
        super.configure();

        var settings = getColdbox().getSettingStructure();

        map( "userProfileImageStorageProvider" ).to( "preside.system.services.fileStorage.FileSystemS
            .initArg( name="rootDirectory" , value=settings.uploads_directory & "/profilePictures" )
            .initArg( name="trashDirectory", value=settings.uploads_directory & "/.trash" )
            .initArg( name="rootUrl"       , value="" );
    }

}
```

Hint: Having individual storage provider instances with their own distinct paths is a good way to organise your uploaded files and can provide you with granularity when dealing with permissions, etc.

Example upload / download code

The following *example* code will upload a file into the storage provider we created in our example above:

```
property name="storageProvider" inject="userProfileImageStorageProvider";

public string function uploadProfilePicture(
      required string userId
    , required string fileExtension
    , required binary uploadedImageBinary
) {
    var filePath = "/#arguments.userId#.#arguments.fileExtension#";

    storageProvider.putObject( object=fileBinary, path=filePath );

    return filePath;
}
```

Downloading a file can be done through a specific core route (see Routing), i.e. you can build a link to the direct download / serving of the file. The syntax is as follows:

```
var downloadLink = event.buildLink(
      fileStorageProvider = nameOfStorageProvider
    , fileStoragePath     = storagePathAsStoredInStorageProvider
    , filename            = optionalFileNameUserWillSeeWhenDownloading
);
```

So, for the example above, we might have:

```
var imageUrl = event.buildLink(
      fileStorageProvider = "userProfileImageStorageProvider"
    , fileStoragePath     = user.profileImagePath
);
```

2.16.2 Applying access control

There is no built in access control for storage providers. However, the download logic served by the core route handler announces three interception points that you can use to inject your own access control logic. The interception points are:

- preDownloadFile
- onDownloadFile
- onReturnFile304

For access control, your most likely choice will be the `preDownloadFile` interception point. An example implementation might look like this:

```
component extends="coldbox.system.Interceptor" {

    // note: important to use Wirebox's 'provider' DSL here to delay
    // injection in our interceptors
    property name="websiteLoginService"    inject="provider:websiteLoginService";
    property name="myAccessControlService" inject="provider:myAccessControlService";

    public void function configure() {}

    public void function preDownloadFile( event, interceptData ) {
        var rc              = event.getCollection();
        var storageProvider = rc.storageProvider ?: "";
        var storagePath     = rc.storagePath     ?: "";
        var filename        = rc.filename        ?: ListLast( storagePath, "/" );

        if ( storageProvider == "myStorageProviderWithAccessControl" ) {
            if ( !websiteLoginService.isLoggedIn() ) {
                event.accessDenied( reason="LOGIN_REQUIRED" );
            }

            var hasAccess   = myAccessControlService.hasAccess(
                  documentPath = storagePath
                , userId       = websiteLoginService.getLoggedInUserId()
            );
            if ( !hasAccess ) {
                event.accessDenied( reason="INSUFFICIENT_PRIVILEGES" );
            }
        }
    }
}
```

2.17 Multilingual content

2.17.1 Overview

PresideCMS comes packaged with a powerful multilingual content feature that allows you to make your client's pages and other data objects translatable to multiple languages.

Enabling multilingual translations is a case of:

1. Enabling the feature in your `Config.cfc` file
2. Marking the preside objects that you wish to be multilingual with a `multilingual` flag
3. Marking the specific properties of preside objects that you wish to be multilingual with a `multilingual` flag
4. Optionally providing specific form layouts for translations
5. Providing a mechanism in the front-end application for users to choose from configured languages

Once the multilingual content feature is enabled, PresideCMS will provide a basic UI for allowing CMS administrators to translate content and to configure what languages are available. When selecting data for display in your application, PresideCMS will automatically select translations of your multilingual properties for you when available for the currently selected language. If no translation is available, the system will fall back to the default content.

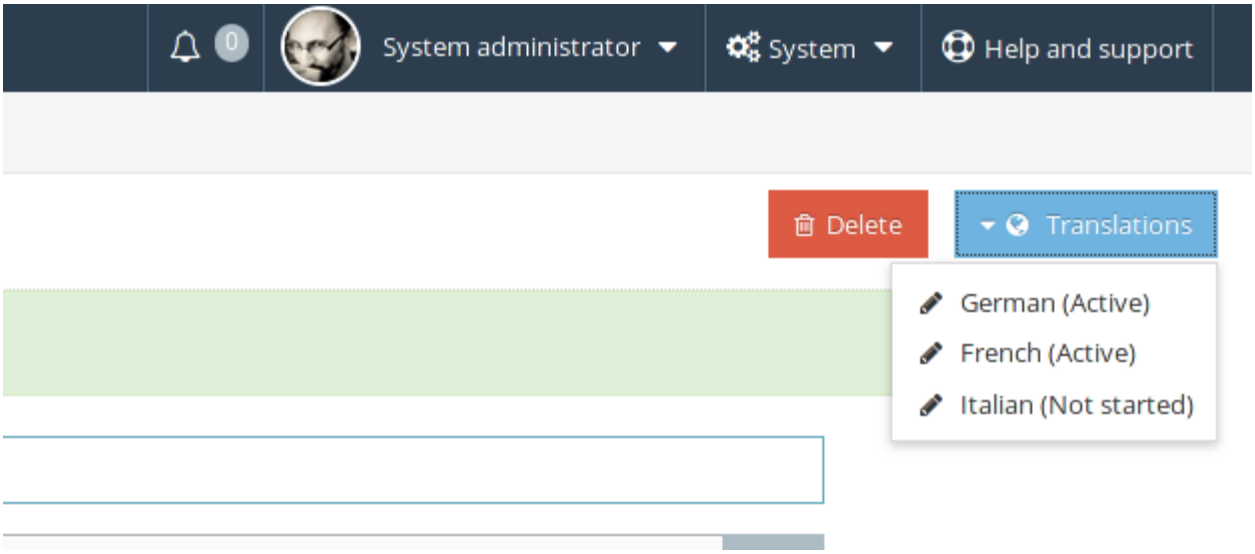

Fig. 2.14: Screenshot showing selection of configured languages

2.17.2 Enabling multilingual content

Global config

Enabling the feature in your applications's `Config.cfc` file is achieved as follows:

```
public void function configure() output=false {
    super.configure();

    // ...

    settings.features.multilingual.enabled = true;
```

Configuring specific data objects

Configuring individual Preside Objects is done using a `multilingual=true` flag on both the component itself and any properties you wish to be translatable:

```
/**
 * @multilingual true
 *
 */
component {
    property name="title" multilingual=true // ... (multilingual)
    property name="active" // ... (not multilingual)
}
```

2.17.3 Configuring languages

Configuring languages is done entirely through the admin user interface and can be performed by your clients if necessary. To navigate to the settings page, go to *System -> Settings -> Content translations*:

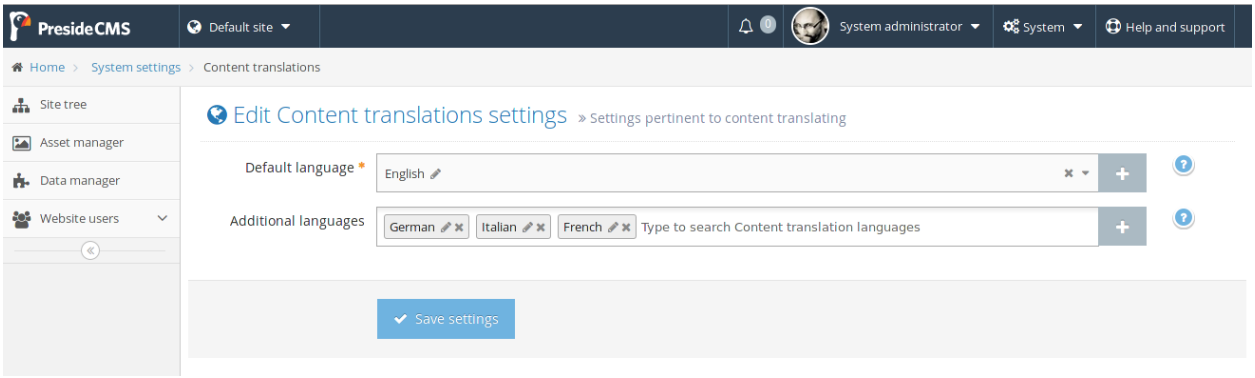

Fig. 2.15: Screenshot showing configuration of content translation languages in the admin user interface

2.17.4 Customizing translation forms

By default, the forms for translating records will be automatically generated. They will contain no tabs or fieldsets and the order of fields may be unpredictable.

To provide a better experience when dealing with records with many fields, you can define an alternative translation form at:

```
/forms/preside-objects/_translation_objectname/admin.edit.xml // where 'objectname' is the name of yo
```

When dealing with page types and pages, this will be:

```
/forms/preside-objects/_translation_page/admin.edit.xml // for the core page object
/forms/preside-objects/_translation_pagetypename/admin.edit.xml // where 'pagetypename' is the name o
```

2.17.5 Setting the current language

It is up to your application to choose the way in which it will set the language for the current request. One common way in which to do this would be to allow the user to pick from the available languages and to persist their preference.

The list of available languages can be obtained with the *ListLanguages()* method of the Multilingual Preside Object Service, e.g.:

```
component {
    property name="multilingualPresideObjectService" inject="multilingualPresideObjectService";

    function someHandlerAction( event, rc, prc ) {
        prc.availableLanguages = multilingualPresideObjectService.listLanguages()
    }
}
```

Setting the current language can be done with `event.setLanguage( idOfLanguage )`. An ideal place to do this would be at the beggining of the request. This can be achieved in the `/handlers/General.cfc` handler. For example:

```
component extends="preside.system.handlers.General" {

    // here, userPreferenceService would be some custom service
    // object that was written to get and set user preferences
    // it is for illustration purposes only and not a core service
    property name="userPreferencesService" inject="userPreferencesService";

    function requestStart( event, rc, prc ) {
        super.requestStart( argumentCollection=arguments );

        event.setLanguage( userPreferencesService.getLanguage() );
    }
}
```

Note: Notice how the `General.cfc` handler extends `preside.system.handlers.General` and calls `super.requestStart( argumentCollection=arguments )`. Without this logic, the core request start logic would not take place, and the system would likely break completely.

CHAPTER 3

Guide for maintainers and contributers

This guide is for those who wish to maintain or contribute to Preside CMS. Here you will find guides for how to work with a local build of the project as well as guidelines for coding standards, etc.

3.1 Building Preside locally

In order to run Preside from a local copy of the codebase, the system requires that external dependencies be pulled in to the expected locations in the project. Before continuing, you will need to make sure you have ant installed. Build steps:

1. Clone the GitHub repository (you probably want to fork it first)
2. Run the ant buildfile found at *rootdir/support/build/build.xml* with the **install-preside-deps** task

i.e.

```
/preside/support/build/>ant install-preside-deps
```

CHAPTER 4

Reference

4.1 System Service API

4.1.1 Update manager service

Overview

Full path: *preside.system.services.updateManager.UpdateManagerService*

The Update Manager Service provides the APIs for managing the installed version of core Preside for your application.

Public API Methods

4.1.2 Multilingual Preside Object Service

Overview

Full path: *preside.system.services.i18n.MultilingualPresideObjectService*

This service exists to provide APIs that make providing support for multilingual translations of standard preside objects possible in a transparent way. Note: You are unlikely to need to deal with this API directly.

Public API Methods

IsMultilingual()

```
public boolean function isMultilingual( required string objectName, string propertyName="" )
```

Returns whether or not the given object and optional property are multilingual enabled.

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object that we wish to check
propertyName	string	No (default="")	Optional name of the property that we wish to check

AddTranslationObjectsForMultilingualEnabledObjects()

```
public void function addTranslationObjectsForMultilingualEnabledObjects( required struct objects )
```

Performs the magic of creating extra database tables (preside objects) to store the translations of multilingual enabled objects.

Arguments

Name	Type	Required	Description
objects	struct	Yes	Objects as compiled and read by the preside object service.

CreateTranslationObject()

```
public struct function createTranslationObject( required string objectName, required struct sourceObj
```

Returns the meta data for our auto generated translation object based on a given source object

Arguments

Name	Type	Required	Description
objectName	string	Yes	The name of the source object
sourceObject	struct	Yes	The metadata of the source object

DecorateMultilingualObject()

```
public void function decorateMultilingualObject( required string objectName, required struct object )
```

Adds utility properties to the multilingual enabled source object so that its translations can be easily queried

Arguments

Name	Type	Required	Description
objectName	string	Yes	The name of the source object
object	struct	Yes	The metadata of the source object

MixinTranslationSpecificSelectLogicToSelectDataCall()

```
public void function mixinTranslationSpecificSelectLogicToSelectDataCall( required string objectName,
```

Works on intercepted select queries to discover and replace multilingual select fields with special IfNull(translation, original) syntax to automagically select translations without the developer having to do anything about it

Arguments

Name	Type	Required	Description
objectName	string	Yes	The name of the source object
selectFields	array	Yes	Array of select fields as passed into the presideObjectService.selectData() method
adapter	any	Yes	Database adapter to be used in generating the select query SQL

AddLanguageClauseToTranslationJoins()

```
public void function addLanguageClauseToTranslationJoins( required array tableJoins, required string
```

Works on intercepted select queries to discover and decorate joins on translation objects with an additional clause for the passed in language

Arguments

Name	Type	Required	Description
tableJoins	array	Yes	Array of table joins as calculated by the SelectData() logic
language	string	Yes	The language to filter on
preparedFilter	struct	Yes	The fully prepared and resolved filter that will be used in the select query

ListLanguages()

```
public array function listLanguages( boolean includeDefault=true )
```

Returns an array of actively supported languages. Each language is represented as a struct with id, name, native_name, iso_code and default keys

Arguments

Name	Type	Required	Description
includeDefault	boolean	No (default=true)	Whether or not to include the default language in the array

GetTranslationStatus()

```
public array function getTranslationStatus( required string objectName, required string recordId )
```

Returns an array of actively supported languages as per listLanguages() with an additional 'status' field indicating the status of the translation for the given object record

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object that has the record we wish to get the translation status of
recordId	string	Yes	ID of the record we wish to get the translation status of

GetLanguage()

```
public struct function getLanguage( required string languageId )
```

Returns a structure of language details for the given language. If the language is not an actively translatable language, an empty structure will be returned.

Arguments

Name	Type	Required	Description
languageId	string	Yes	ID of the language to get

GetTranslationObjectName()

```
public string function getTranslationObjectName( required string sourceObjectName )
```

Returns the name of the given object's corresponding translation object

Arguments

Name	Type	Required	Description
sourceObjectName	string	Yes	

4.1.3 TikaWrapper

Overview

Full path: *preside.system.services.assetManager.TikaWrapper*

The Tika Wrapper service object is a simple CFML API for interacting with Apache Tika (http://tika.apache.org). Its purpose in the context of PresideCMS is for metadata and content extraction from uploaded documents.

Public API Methods

4.1.4 Preside Object Service

Overview

Full path: *preside.system.services.presideObjects.PresideObjectService*

The Preside Object Service is the main entry point API for interacting with **Preside Data Objects**. It provides CRUD operations for individual objects as well as many other useful utilities.

For a full developer guide on using Preside Objects and this service, see Preside Data Objects.

Public API Methods

GetObject()

```
public any function getObject( required string objectName )
```

Returns an 'auto service' object instance of the given Preside Object.

See *Using Auto Service Objects* for a full guide.

Arguments

Name	Type	Required	Description
objectName	string	Yes	The name of the object to get

Example

```
eventObject = presideObjectService.getObject( "event" );


eventId     = eventObject.insertData( data={ title="Christmas", startDate="2014-12-25", endDate="201
event       = eventObject.selectData( id=eventId )
```

SelectData()

```
public query function selectData( required string objectName, string id="", array selectFields=[], ar
```

Selects database records for the given object based on a variety of input parameters

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object from which to select data
id	string	No (default="")	ID of a record to select
selectFields	array	No (default=[])	Array of field names to select. Can include relationships, e.g. ['tags.label as tag']
filter	any	No (default={})	Filter the records returned, see *Filtering data* in Preside Data Objects
filterParams	struct	No (default={})	Filter params for plain SQL filter, see *Filtering data* in Preside Data Objects
extraFilters	array	No (default=[])	An array of extra sets of filters. Each array should contain a structure with `filter` and optional *code:'filterParams* keys.
savedFilters	array	No	
orderBy	string	No (default="")	Plain SQL order by string
groupBy	string	No (default="")	Plain SQL group by string
maxRows	numeric	No (default=0)	Maximum number of rows to select
startRow	numeric	No (default=1)	Offset the recordset when using maxRows
useCache	boolean	No (default=true)	Whether or not to automatically cache the result internally
fromVersionTable	boolean	No (default=false)	Whether or not to select the data from the version history table for the object
maxVersion	string	No (default="HEAD")	Can be used to set a maximum version number when selecting from the version table
specificVersion	numeric	No (default=0)	Can be used to select a specific version when selecting from the version table
forceJoins	string	No (default="")	Can be set to "inner" / "left" to force *all* joins in the query to a particular join type

Examples

```
// select a record by ID
event = presideObjectService.selectData( objectName="event", id=rc.id );


// select records using a simple filter.
// notice the 'category.label as categoryName' field - this will
// be automatically selected from the related 'category' object
events = presideObjectService.selectData(
```

```
      objectName   = "event"
    , filter       = { category = rc.category }
    , selectFields = [ "event.name", "category.label as categoryName", "event.category" ]
    , orderby      = "event.name"
);


// select records with a plain SQL filter with added SQL params
events = presideObjectService.selectData(
      objectName   = "event"
    , filter       = "category.label like :category.label"
    , filterParams = { "category.label" = "%#rc.search#%" }
);
```

InsertData()

```
public any function insertData( required string objectName, required struct data, boolean insertMany
```

Inserts a record into the database, returning the ID of the newly created record

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object in which to to insert a record
data	struct	Yes	Structure of data who's keys map to the properties that are defined on the object
insertManyToManyRecords	boolean	No (default=false)	Whether or not to insert multiple relationship records for properties that have a many-to-many relationship
useVersioning	boolean	No (default=automatic)	Whether or not to use the versioning system with the insert. If the object is setup to use versioning (default), this will default to true.
versionNumber	numeric	No (default=0)	If using versioning, specify a version number to save against (if none specified, one will be created automatically)

Example:

```
newId = presideObjectService.insertData(
      objectName = "event"
    , data       = { name="Summer BBQ", startdate="2015-08-23", enddate="2015-08-23" }
);
```

UpdateData()

```
public numeric function updateData( required string objectName, required struct data, string id="", a
```

Updates records in the database with a new set of data. Returns the number of records affected by the operation.

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object who's records you want to update
data	struct	Yes	Structure of data containing new values. Keys should map to properties on the object.
id	string	No (default="")	ID of a single record to update
filter	any	No	Filter for which records are updated, see *Filtering data* in Preside Data Objects
filterParams	struct	No	Filter params for plain SQL filter, see *Filtering data* in Preside Data Objects
extraFilters	array	No	An array of extra sets of filters. Each array should contain a structure with `filter` and optional *code:'filterParams* keys.
savedFilters	array	No	
forceUpdateAll	boolean	No (default=false)	If no ID and no filters are supplied, this must be set to **true** in order for the update to process
updateManyToManyRecords	boolean	No (default=false)	Whether or not to update multiple relationship records for properties that have a many-to-many relationship
useVersioning	boolean	No (default=auto)	Whether or not to use the versioning system with the update. If the object is setup to use versioning (default), this will default to true.
versionNumber	numeric	No (default=0)	If using versioning, specify a version number to save against (if none specified, one will be created automatically)
forceVersionCreation	boolean	No (default=false)	

Examples

```
// update a single record
updated = presideObjectService.updateData(
      objectName = "event"
    , id         = eventId
    , data       = { enddate = "2015-01-31" }
);


// update multiple records
updated = presideObjectService.updateData(
      objectName     = "event"
    , data           = { cancelled = true }
    , filter         = { category = rc.category }
);


// update all records
updated = presideObjectService.updateData(
      objectName     = "event"
    , data           = { cancelled = true }
    , forceUpdateAll = true
);
```

DeleteData()

```
public numeric function deleteData( required string objectName, string id="", any filter, struct filt
```

Deletes records from the database. Returns the number of records deleted.

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object from who's database table records are to be deleted
id	string	No (default="")	ID of a record to delete
filter	any	No	Filter for records to delete, see *Filtering data* in Preside Data Objects
filterParams	struct	No	Filter params for plain SQL filter, see *Filtering data* in Preside Data Objects
extraFilters	array	No	An array of extra sets of filters. Each array should contain a structure with `filter` and optional *code:'filterParams* keys.
savedFilters	array	No	
forceDeleteAll	boolean	No (default=false)	If no id or filter supplied, this must be set to **true** in order for the delete to process

Examples

```
// delete a single record
deleted = presideObjectService.deleteData(
      objectName = "event"
    , id         = rc.id
);


// delete multiple records using a filter
// (note we are filtering on a column in a related object, "category")
deleted = presideObjectService.deleteData(
      objectName   = "event"
    , filter       = "category.label != :category.label"
    , filterParams = { "category.label" = "BBQs" }
);


// delete all records
// (note we are filtering on a column in a related object, "category")
deleted = presideObjectService.deleteData(
      objectName     = "event"
    , forceDeleteAll = true
);
```

DataExists()

```
public boolean function dataExists( required string objectName )
```

Returns true if records exist that match the supplied fillter, false otherwise.

Note: In addition to the named arguments here, you can also supply any valid arguments that can be supplied to the *SelectData()* method

---

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object in which the records may or may not exist

Example

```
eventsExist = presideObjectService.dataExists(
      objectName = "event"
    , filter     = { category = rc.category }
);
```

SelectManyToManyData()

```
public query function selectManyToManyData( required string objectName, required string propertyName,
```

Selects records from many-to-many relationships

Note: You can pass additional arguments to those specified below and they will all be passed to the *SelectData()* method

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object that has the many-to-many property defined
propertyName	string	Yes	Name of the many-to-many property
selectFields	array	No	Array of fields to select
orderBy	string	No (default="")	Plain SQL order by statement

Example

```
tags = presideObjectService.selectManyToManyData(
      objectName   = "event"
    , propertyName = "tags"
    , orderby      = "tags.label"
);
```

SyncManyToManyData()

```
public boolean function syncManyToManyData( required string sourceObject, required string sourcePrope
```

Synchronizes a record's related object data for a given property. Returns true on success, false otherwise.

Arguments

Name	Type	Required	Description
sourceObject	string	Yes	The object that contains the many-to-many property
sourceProperty	string	Yes	The name of the property that is defined as a many-to-many relationship
sourceId	string	Yes	ID of the record who's related data we are to synchronize
targetIdList	string	Yes	Comma separated list of IDs of records representing records in the related object

Example

```
presideObjectService.syncManyToManyData(
      sourceObject   = "event"
    , sourceProperty = "tags"
    , sourceId       = rc.eventId
    , targetIdList   = rc.tags // e.g. "635,1,52,24"
);
```

GetDeNormalizedManyToManyData()

```
public struct function getDeNormalizedManyToManyData( required string objectName, required string id,
```

Returns a structure of many to many data for a given record. Each structure key represents a many-to-many type property on the object. The value for each key will be a comma separated list of IDs of the related data.

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object who's related data we wish to retrieve
id	string	Yes	ID of the record who's related data we wish to retrieve
fromVersionTable	boolean	No (default=false)	Whether or not to retrieve the data from the version history table for the object
maxVersion	string	No (default="HEAD")	If retrieving from the version history, set a max version number
specificVersion	numeric	No (default=0)	If retrieving from the version history, set a specific version number to retrieve
selectFields	array	No	

Example

```
relatedData = presideObjectService.getDeNormalizedManyToManyData(
    objectName = "event"
  , id         = rc.id
);


// the relatedData struct above might look like { tags = "C3635F77-D569-4D31-A794CA9324BC3E70,3AA27F
```

GetRecordVersions()

```
public query function getRecordVersions( required string objectName, required string id, string field
```

Returns a summary query of all the versions of a given record (by ID), optionally filtered by field name

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object who's record we wish to retrieve the version history for
id	string	Yes	ID of the record who's history we wish to view
fieldName	string	No	Optional name of one of the object's property which which to filter the history. Doing so will show only versions in which this field changed.

DbSync()

```
public void function dbSync( )
```

Performs a full database synchronisation with your Preside Data Objects. Creating new tables, fields and relationships as well as modifying and retiring existing ones.

See *Keeping in sync with the database*.

Note: You are unlikely to need to call this method directly. See `/devguides/reloading`.

Arguments *This method does not accept any arguments.*

Reload()

```
public void function reload( )
```

Reloads all the object definitions by reading them all from file.

Note: You are unlikely to need to call this method directly. See `/devguides/reloading`.

Arguments *This method does not accept any arguments.*

ListObjects()

```
public array function listObjects( boolean includeGeneratedObjects=false )
```

Returns an array of names for all of the registered objects, sorted alphabetically (ignoring case)

Arguments

Name	Type	Required	Description
includeGeneratedObjects	boolean	No (default=false)	

ObjectExists()

```
public boolean function objectExists( required string objectName )
```

Returns whether or not the passed object name has been registered

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object that you wish to check the existance of

ObjectIsAutoGenerated()

```
public boolean function objectIsAutoGenerated( required string objectName )
```

Returns whether or not the object has been automatically created by the system

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object that you wish to check

FieldExists()

```
public boolean function fieldExists( required string objectName, required string fieldName )
```

Returns whether or not the passed field exists on the passed object

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object who's field you wish to check
fieldName	string	Yes	Name of the field you wish to check the existance of

GetObjectAttribute()

```
public any function getObjectAttribute( required string objectName, required string attributeName, st
```

Returns an arbritary attribute value that is defined on the object's `component` tag.

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object who's attribute we wish to get
attributeName	string	Yes	Name of the attribute who's value we wish to get
defaultValue	string	No (default="")	Default value for the attribute, should it not exist

Example

```
eventLabelField = presideObjectService.getObjectAttribute(
      objectName    = "event"
    , attributeName = "labelField"
    , defaultValue  = "label"
);
```

GetObjectPropertyAttribute()

```
public string function getObjectPropertyAttribute( required string objectName, required string proper
```

Returns an arbritary attribute value that is defined on a specified property for an object.

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the property who's attribute we wish to get
propertyName	string	Yes	
attributeName	string	Yes	Name of the attribute who's value we wish to get
defaultValue	string	No (default="")	Default value for the attribute, should it not exist

Example

```
maxLength = presideObjectService.getObjectPropertyAttribute(
      objectName    = "event"
    , propertyName  = "name"
    , attributeName = "maxLength"
    , defaultValue  = 200
);
```

GetVersionObjectName()

```
public string function getVersionObjectName( required string sourceObjectName )
```

This method, returns the object name that can be used to reference the version history object for a given object.

Arguments

Name	Type	Required	Description
sourceObjectName	string	Yes	Name of the object who's version object name we wish to retrieve

ObjectIsVersioned()

```
public boolean function objectIsVersioned( required string objectName )
```

Returns whether or not the given object is using the versioning system

Arguments

Name	Type	Required	Description
objectName	string	Yes	Name of the object you wish to check

GetNextVersionNumber()

```
public numeric function getNextVersionNumber( )
```

Returns the next available version number that can be used for saving a new version record.

This is an auto incrementing integer that is global to all versioning tables in the system.

Arguments *This method does not accept any arguments.*

4.1.5 FeatureService

Overview

Full path: *preside.system.services.features.FeatureService*

The Feature Service provides an API to preside's configured features. This allows other systems within PresideCMS to check the enabled statusof enabled status of features before proceeding to provide a page or perform some action

Public API Methods

IsFeatureEnabled()

```
public boolean function isFeatureEnabled( required string feature, string siteTemplate )
```

Returns whether or not the passed feature is currently enabled

Arguments

Name	Type	Required	Description
feature	string	Yes	name of the feature to check
siteTemplate	string	No	current active site template - can be used to check features that can be site template specific

4.1.6 Website login service

Overview

Full path: *preside.system.services.websiteUsers.WebsiteLoginService*

The website login manager object provides methods for member login, logout and session retrieval

See also: Website users and permissioning

Public API Methods

Login()

```
public boolean function login( required string loginId, string password="", boolean rememberLogin=fa
```

Logs the user in by matching the passed login id against either the login id or email address fields and running a bcrypt password check to verify the security credentials. Returns true on success, false otherwise.

Arguments

Name	Type	Required	Description
loginId	string	Yes	Either the login id or email address of the user to login
password	string	No (default="")	The password that the user has entered during login
rememberLogin	boolean	No (default=false)	Whether or not to set a "remember me" cookie
rememberExpiryInDays	any	No (default=90)	When setting a remember me cookie, how long (in days) before the cookie should expire
skipPasswordCheck	boolean	No (default=false)	

ValidatePassword()

```
public boolean function validatePassword( required string password, string userId )
```

Validates the supplied password against the a user (defaults to currently logged in user)

Arguments

Name	Type	Required	Description
password	string	Yes	The user supplied password
userId	string	No	The id of the user who's password we are to validate. Defaults to the currently logged in user.

Logout()

```
public void function logout( )
```

Logs the currently logged in user out of their session

Arguments *This method does not accept any arguments.*

IsLoggedIn()

```
public boolean function isLoggedIn( function securityAlertCallback )
```

Arguments

Name	Type	Required	Description
securityAlertCallback	function	No	

IsAutoLoggedIn()

```
public boolean function isAutoLoggedIn( )
```

Returns whether or not the user making the current request is only automatically logged in. This would happen when the user has been logged in via a "remember me" cookie. System's can make use of this method when protecting pages that require a full authenticated session, forcing a login prompt when this method returns true.

Arguments *This method does not accept any arguments.*

IsImpersonated()

```
public boolean function isImpersonated( )
```

Returns whether or not the user making the current request is only "impersonated" by an admin user. This method can then be used to hide sensitive information that even admin users impersonating a web user should not be able to see.

Arguments *This method does not accept any arguments.*

GetLoggedInUserDetails()

```
public struct function getLoggedInUserDetails( )
```

Returns the structure of user details belonging to the currently logged in user. If no user is logged in, an empty structure will be returned.

Arguments *This method does not accept any arguments.*

GetLoggedInUserId()

```
public string function getLoggedInUserId( )
```

Returns the id of the currently logged in user, or an empty string if no user is logged in

Arguments *This method does not accept any arguments.*

SendWelcomeEmail()

```
public boolean function sendWelcomeEmail( required string userId )
```

Sends welcome email to the supplied user. Returns true if successful, false otherwise.

Arguments

Name	Type	Required	Description
userId	string	Yes	

SendPasswordResetInstructions()

```
public boolean function sendPasswordResetInstructions( required string loginId )
```

Sends password reset instructions to the supplied user. Returns true if successful, false otherwise.

Arguments

Name	Type	Required	Description
loginId	string	Yes	Either the email address or login id of the user

ValidateResetPasswordToken()

```
public boolean function validateResetPasswordToken( required string token )
```

Validates a password reset token that has been passed through the URL after a user has followed 'reset password' link in instructional email.

Arguments

Name	Type	Required	Description
token	string	Yes	The token to validate

ResetPassword()

```
public boolean function resetPassword( required string token, required string password )
```

Resets a password by looking up the supplied password reset token and encrypting the supplied password

Arguments

Name	Type	Required	Description
token	string	Yes	The temporary reset password token to look the user up with
password	string	Yes	The new password

ChangePassword()

```
public boolean function changePassword( required string password, string userId )
```

Changes a password

Arguments

Name	Type	Required	Description
password	string	Yes	The new password
userId	string	No	ID of the user who's password we wish to change (defaults to currently logged in user id)

ListLoggedInUserBenefits()

```
public array function listLoggedInUserBenefits( )
```

Gets an array of benefit IDs associated with the logged in user

Arguments *This method does not accept any arguments.*

DoesLoggedInUserHaveBenefits()

```
public boolean function doesLoggedInUserHaveBenefits( required array benefits )
```

Returns true / false depending on whether or not a user has access to any of the supplied benefits

Arguments

Name	Type	Required	Description
benefits	array	Yes	Array of benefit IDs. If the logged in user has any of these benefits, the method will return true

RecordLogin()

```
public boolean function recordLogin( )
```

Sets the last logged in date for the logged in user

Arguments *This method does not accept any arguments.*

RecordLogout()

```
public boolean function recordLogout( )
```

Sets the last logged out date for the logged in user. Note, must be called before logging the user out

Arguments *This method does not accept any arguments.*

RecordVisit()

```
public boolean function recordVisit( )
```

Records the visit for the currently logged in user Currently, all this does is to set the last request made datetime value

Arguments *This method does not accept any arguments.*

4.1.7 Notification Service

Overview

Full path: *preside.system.services.notifications.NotificationService*

The notifications service provides an API to the PresideCMS administrator notifications system

Public API Methods

CreateNotification()

```
public string function createNotification( required string topic, required string type, required stru
```

Adds a notification to the system.

Arguments

Name	Type	Required	Description
topic	string	Yes	Topic that indicates the specific notification being raised. e.g. 'sync.jobFailed'
type	string	Yes	Type of the notification, i.e. 'INFO', 'WARNING' or 'ALERT'
data	struct	Yes	Supporting data for the notification. This is used, in combination with the topic, to render the alert for the end users.

GetUnreadNotificationCount()

```
public numeric function getUnreadNotificationCount( required string userId )
```

Returns a count of unread notifications for the given user id.

Arguments

Name	Type	Required	Description
userId	string	Yes	id of the admin user who's unread notification count we wish to retrieve

GetUnreadTopics()

```
public query function getUnreadTopics( required string userId )
```

Returns counts of unread notifications by topics for the given user

Arguments

Name	Type	Required	Description
userId	string	Yes	id of the admin user who's unread notifications we wish to retrieve

GetNotifications()

```
public query function getNotifications( required string userId, string topic="", numeric startRow=1,
```

Returns the latest unread notifications for the given user id. Returns an array of structs, each struct contains id and data keys.

Arguments

Name	Type	Required	Description
userId	string	Yes	id of the admin user who's unread notifications we wish to retrieve
topic	string	No (default="")	
startRow	numeric	No (default=1)	
maxRows	numeric	No (default=10)	maximum number of notifications to retrieve

GetNotification()

```
public struct function getNotification( required string id )
```

Returns a specific notification

Arguments

Name	Type	Required	Description
id	string	Yes	ID of the notification

GetNotificationsCount()

```
public numeric function getNotificationsCount( required string userId, string topic="" )
```

Returns the count of non-dismissed notifications for the given user id and optional topic

Arguments

Name	Type	Required	Description
userId	string	Yes	id of the admin user who's unread notifications we wish to retrieve
topic	string	No (default="")	topic by which to filter the notifications

RenderNotification()

```
public string function renderNotification( required string topic, required struct data, required str
```

Renders the given notification topic

Arguments

Name	Type	Required	Description
topic	string	Yes	Topic of the notification
data	struct	Yes	Data associated with the notification
context	string	Yes	Context of the notification

ListTopics()

```
public array function listTopics( )
```

Returns array of configured topics

Arguments *This method does not accept any arguments.*

MarkAsRead()

```
public numeric function markAsRead( required array notificationIds, required string userId )
```

Marks notifications as read for a given user

Arguments

Name	Type	Required	Description
notificationIds	array	Yes	Array of notification IDs to mark as read
userId	string	Yes	The id of the user to mark as read for

Dismiss()

```
public numeric function dismiss( required array notificationIds )
```

Completely discards the given notifications

Arguments

Name	Type	Required	Description
notificationIds	array	Yes	Array of notification IDs to dismissed

GetUserSubscriptions()

```
public array function getUserSubscriptions( required string userId )
```

Get subscribed topics for a user. Returns an array of the topic ids

Arguments

Name	Type	Required	Description
userId	string	Yes	ID of the user who's subscribed topics we want to fetch

GetGlobalTopicConfiguration()

```
public struct function getGlobalTopicConfiguration( required string topic )
```

Retrieves globally saved configuration settings for a given notification topic

Arguments

Name	Type	Required	Description
topic	string	Yes	ID of the topic

SaveGlobalTopicConfiguration()

```
public boolean function saveGlobalTopicConfiguration( required string topic, required struct configu
```

Saves configuration for a topic

Arguments

Name	Type	Required	Description
topic	string	Yes	ID of the topic
configuration	struct	Yes	Struct containing configuration data

SaveUserSubscriptions()

```
public void function saveUserSubscriptions( required string userId, required array topics )
```

Saves a users subscription preferences

Arguments

Name	Type	Required	Description
userId	string	Yes	ID of the user who's subscribed topics we want to save
topics	array	Yes	Array of topics to subscribe to

4.1.8 Email service

Overview

Full path: *preside.system.services.email.EmailService*

The email service takes care of sending emails through the PresideCMS's email templating system (see Email templating).

Public API Methods

Send()

```
public boolean function send( string template="", struct args, array to, string from="", string subje
```

Sends an email. If a template is supplied, first runs the template handler which can return a struct that will override any arguments passed directly to the function

Arguments

Name	Type	Required	Description
template	string	No (default="")	Name of the template who's handler will do the rendering, etc.
args	struct	No	Structure of arbitrary arguments to forward on to the template handler
to	array	No	Array of email addresses to send the email to
from	string	No (default="")	Optional from email address
subject	string	No (default="")	Optional email subject. If not supplied, the template handler should supply it
cc	array	No	Optional array of CC addresses
bcc	array	No	Optional array of BCC addresses
html-Body	string	No (default="")	Optional HTML body
textBody	string	No (default="")	Optional plain text body
params	struct	No	Optional struct of cfmail params (headers, attachments, etc.)

ListTemplates()

```
public array function listTemplates( )
```

Returns an array of email templates that have been dicovered from the /handlers/emailTemplates directory

Arguments *This method does not accept any arguments.*

4.1.9 Site service

Overview

Full path: *preside.system.services.siteTree.SiteService*

The site service provides methods for interacting with the core "Site" system

Public API Methods

ListSites()

```
public query function listSites( )
```

Returns a query of all the registered sites

Arguments *This method does not accept any arguments.*

GetSite()

```
public struct function getSite( required string id )
```

Returns a single site matched by id

Arguments

Name	Type	Required	Description
id	string	Yes	ID of the site to get

MatchSite()

```
public struct function matchSite( required string domain, required string path )
```

Returns the site record that matches the incoming domain and URL path.

Arguments

Name	Type	Required	Description
domain	string	Yes	The domain name used in the incoming request, e.g. testsite.com
path	string	Yes	The URL path of the incoming request, e.g. /path/to/somepage.html

GetActiveAdminSite()

```
public struct function getActiveAdminSite( )
```

Returns the id of the currently active site for the administrator. If no site selected, chooses the first site that the logged in user has rights to

Arguments *This method does not accept any arguments.*

SetActiveAdminSite()

```
public void function setActiveAdminSite( required string siteId )
```

Sets the current active admin site id

Arguments

Name	Type	Required	Description
siteId	string	Yes	

EnsureDefaultSiteExists()

```
public void function ensureDefaultSiteExists( )
```

Ensures that at least one site is registered with the system, called internally before checking valid routes

Arguments *This method does not accept any arguments.*

GetActiveSiteId()

```
public string function getActiveSiteId( )
```

Retrieves the current active site id. This is based either on the URL, for front-end requests, or the currently selected site when in the administrator

Arguments *This method does not accept any arguments.*

GetActiveSiteTemplate()

```
public string function getActiveSiteTemplate( )
```

Retrieves the current active site template. This is based either on the URL, for front-end requests, or the currently selected site when in the administrator

Arguments *This method does not accept any arguments.*

SyncSiteAliasDomains()

```
public boolean function syncSiteAliasDomains( required string siteId, required string domains )
```

Sync alias domains with the site record

Arguments

Name	Type	Required	Description
siteId	string	Yes	
domains	string	Yes	

SyncSiteRedirectDomains()

```
public boolean function syncSiteRedirectDomains( required string siteId, required string domains )
```

Sync redirect domains with the site record

Arguments

Name	Type	Required	Description
siteId	string	Yes	
domains	string	Yes	

4.1.10 Site Templates service

Overview

Full path: *preside.system.services.siteTree.SiteTemplatesService*

The site templates service provides methods for discovering and listing out site templates which are self contained sets of widgets, page types, objects, etc. See Working with multiple sites.

Public API Methods

ListTemplates()

```
public array function listTemplates( )
```

Returns an array of SiteTemplate objects that have been discovered by the system

Arguments *This method does not accept any arguments.*

Reload()

```
public void function reload( )
```

Re-reads all the template directories to repopulate the internal list of templates

Arguments *This method does not accept any arguments.*

4.2 System Preside Objects

4.2.1 login

Overview

The login page type object is used to store any fields that are distinct to the system page type 'login'

Object name: login

Table name: psys_login

Path: /preside-objects/page-types/login.cfc

Properties

4.2.2 accessDenied

Overview

The accessDenied page type object is used to store any fields that are distinct to the system page type 'Access denied'

Object name: accessDenied

Table name: psys_accessDenied

Path: /preside-objects/page-types/accessDenied.cfc

Properties

4.2.3 forgotten_password

Overview

The forgotten password page type object is used to store any fields that are distinct to the system page type 'forgotten password'

Object name: forgotten_password

Table name: psys_forgotten_password

Path: /preside-objects/page-types/forgotten_password.cfc

Properties

4.2.4 notFound

Overview

The notFound page type object is used to store any fields that are distinct to the system page type '404 not found'

Object name: notFound

Table name: psys_notFound

Path: /preside-objects/page-types/notFound.cfc

Properties

4.2.5 homepage

Overview

The homepage page type object is used to store any fields that are distinct to pages of type 'homepage'

Object name: homepage

Table name: psys_homepage

Path: /preside-objects/page-types/homepage.cfc

Properties

4.2.6 reset_password

Overview

The reset password page type object is used to store any fields that are distinct to the system page type 'reset password'

Object name: reset_password

Table name: psys_reset_password

Path: /preside-objects/page-types/reset_password.cfc

Properties

4.2.7 Page type: Standard

Overview

The "Standard" page type object is used to store any fields that are distinct to pages of type 'homepage'

Object name: standard_page

Table name: psys_standard_page

Path: /preside-objects/page-types/standard_page.cfc

Properties

4.2.8 multilingual_language

Overview

The multilingual language object stores languages that can be available to the Presides core multilingual content system

Object name: multilingual_language

Table name: psys_multilingual_language

Path: /preside-objects/i18n/multilingual_language.cfc

Properties

```
property name="name"          type="string"  dbtype="varchar" maxlength=200 required=true uniqueindex
property name="iso_code"      type="string"  dbtype="varchar" maxlength=2   required=true uniqueindex
property name="native_name"   type="string"  dbtype="varchar" maxlength=200 required=true control="te
property name="right_to_left" type="boolean" dbtype="boolean"               required=false default=fa
```

4.2.9 Asset derivative

Overview

The asset derivative object represents a derived version of an Asset, storing the file path and named derivative used to transform the initial asset.

Object name: asset_derivative

Table name: psys_asset_derivative

Path: /preside-objects/assetManager/asset_derivative.cfc

Properties

```
property name="asset"         relationship="many-to-one" required=true  uniqueindexes="derivative|1";
property name="asset_version" relationship="many-to-one" required=false uniqueindexes="derivative|2";

property name="label" maxLength=200 required=true uniqueindexes="derivative|3";

property name="storage_path" type="string" dbtype="varchar" maxLength=255 required=true   uniqueindex
property name="trashed_path" type="string" dbtype="varchar" maxLength=255 required=false;
property name="asset_type"   type="string" dbtype="varchar" maxLength=10  required=true;
```

4.2.10 Asset folder

Overview

An asset folder is a hierarchy of named storage locations for assets (see Asset)

Object name: asset_folder

Table name: psys_asset_folder

Path: /preside-objects/assetManager/asset_folder.cfc

Properties

```
property name="label" uniqueindexes="folderName|2";
property name="original_label"       type="string"  dbtype="varchar" maxLength=200 required=false;
property name="allowed_filetypes"    type="string"  dbtype="text"                  required=false;
property name="max_filesize_in_mb"   type="numeric" dbtype="float"                 required=false max
property name="access_restriction"   type="string"  dbtype="varchar" maxLength="7" required=false def
property name="full_login_required"  type="boolean" dbtype="boolean"               required=false def
property name="is_system_folder"     type="boolean" dbtype="boolean"               required=false def
property name="system_folder_key"    type="string"  dbtype="varchar" maxLength=200 required=false inc
property name="hidden"               type="boolean" dbtype="boolean"               required=false def

property name="parent_folder" relationship="many-to-one" relatedTo="asset_folder" required="false" un

property name="created_by"  relationship="many-to-one" relatedTo="security_user" required="false" ger
property name="updated_by"  relationship="many-to-one" relatedTo="security_user" required="false" ger
```

4.2.11 Asset

Overview

The asset version object represents the file information for a specific version of a file for a given asset The active asset version's file details are duplicated in the asset object to reduce API and querying complexity

i.e. to get the file details of the active version of a given asset, one simply has to query the asset itself. This has also been done to make upgrades easier as this asset version feature has been added later.

Object name: asset_version

Table name: psys_asset_version

Path: /preside-objects/assetManager/asset_version.cfc

Properties

```
property name="asset"             relationship="many-to-one" relatedTo="asset"      required=true  un
property name="version_number"    type="numeric" dbtype="int"                       required=true  un

property name="storage_path"      type="string"  dbtype="varchar" maxLength=255     required=true  un
property name="size"              type="numeric" dbtype="int"                       required=true;
property name="asset_type"        type="string"  dbtype="varchar" maxLength=10      required=true;
property name="raw_text_content"  type="string"  dbtype="longtext";

property name="created_by"  relationship="many-to-one" relatedTo="security_user" required=false gener
property name="updated_by"  relationship="many-to-one" relatedTo="security_user" required=false gener
```

4.2.12 Asset

Overview

The asset object represents the core data associated with any file uploaded into the Asset manager

Object name: asset

Table name: psys_asset

Path: /preside-objects/assetManager/asset.cfc

Properties

```
property name="asset_folder" relationship="many-to-one"                          required=true   uniq

property name="title"             type="string"  dbtype="varchar" maxLength=150     required=true   u
property name="original_title"    type="string"  dbtype="varchar" maxLength=200     required=false;
property name="storage_path"      type="string"  dbtype="varchar" maxLength=255     required=true   u
property name="trashed_path"      type="string"  dbtype="varchar" maxLength=255     required=false;
property name="description"       type="string"  dbtype="text"    maxLength=0       required=false;
property name="author"            type="string"  dbtype="varchar" maxLength=100     required=false;
property name="size"              type="numeric" dbtype="int"                       required=true;
property name="asset_type"        type="string"  dbtype="varchar" maxLength=10      required=true;
property name="raw_text_content"  type="string"  dbtype="longtext";
property name="width"             type="numeric" dbtype="int"                       required=false;
property name="height"            type="numeric" dbtype="int"                       required=false;
property name="active_version"    relationship="many-to-one" relatedTo="asset_version" required=false

property name="access_restriction"  type="string"  dbtype="varchar" maxLength="7" required=false defa
property name="full_login_required" type="boolean" dbtype="boolean"               required=false defa

property name="created_by"  relationship="many-to-one" relatedTo="security_user" required=false gener
property name="updated_by"  relationship="many-to-one" relatedTo="security_user" required=false gener
```

4.2.13 Asset meta data

Overview

The asset meta object represents a single item of extracted meta data from an asset file

Object name: asset_meta

Table name: psys_asset_meta

Path: /preside-objects/assetManager/asset_meta.cfc

Properties

```
property name="asset"         relationship="many-to-one"                  required=true   uniqueinde
property name="asset_version" relationship="many-to-one"                  required=false  uniqueinde
property name="key"           type="string" dbtype="varchar" maxLength=150 required=true   uniqueinde
property name="value"         type="string" dbtype="text"                 required=false;
```

4.2.14 Website user login token

Overview

The website user login token object represents a "remember me" authentication token. This system is being implemented as advocated here: http://jaspan.com/improved_persistent_login_cookie_best_practice

Object name: website_user_login_token

Table name: psys_website_user_login_token

Path: /preside-objects/websiteUserManagement/website_user_login_token.cfc

Properties

```
property name="user"   relationship="many-to-one" relatedTo="website_user"          uniqueindexes="us
property name="series" type="string" dbtype="varchar" maxLength="35" required=true  uniqueindexes="us
property name="token"  type="string" dbtype="varchar" maxLength="60" required=true;
```

4.2.15 Website applied permission

Overview

A website applied permission records a grants or deny permission for a given user or benefit, permission key and optional context.

Object name: website_applied_permission

Table name: psys_website_applied_permission

Path: /preside-objects/websiteUserManagement/website_applied_permission.cfc

Properties

```
property name="permission_key" type="string"  dbtype="varchar" maxlength="100" required=true  unique
property name="granted"        type="boolean" dbtype="boolean"                  required=true;

property name="context"        type="string"  dbtype="varchar" maxlength="100" required=false unique
property name="context_key"    type="string"  dbtype="varchar" maxlength="100" required=false unique

property name="benefit" relationship="many-to-one" relatedto="website_benefit" required=false unique
property name="user"    relationship="many-to-one" relatedto="website_user"    required=false unique
```

4.2.16 Website user benefit

Overview

Website benefits can be tagged against website users (see Website user). Pages in the site tree, assets in the asset manager, and other custom access areas can then be tagged with member benefits to control users' access to multiple areas and actions in the site through their benefits. This is also a useful object to extend so that you could add other types of benefits other than page / asset access. For example, you could have a disk space field that can tell the system how much disk space a user has in an uploads folder or some such.

Object name: website_benefit

Table name: psys_website_benefit

Path: /preside-objects/websiteUserManagement/website_benefit.cfc

Properties

```
property name="label" uniqueindexes="benefit_name";
property name="priority"    type="numeric" dbtype="int"                      required=false default="
property name="description" type="string"  dbtype="varchar" maxLength="200"  required=false;

property name="combined_benefits" relationship="many-to-many" relatedTo="website_benefit" relatedVia=
```

Public API Methods

4.2.17 Website user

Overview

The website user object represents the login details of someone / something that can log into the front end website (as opposed to the admin)

Object name: website_user

Table name: psys_website_user

Path: /preside-objects/websiteUserManagement/website_user.cfc

Properties

```
property name="login_id"                    type="string"   dbtype="varchar" maxLength="255" required
property name="email_address"               type="string"   dbtype="varchar" maxLength="255" required
property name="password"                    type="string"   dbtype="varchar" maxLength="60"  required
property name="display_name"                type="string"   dbtype="varchar" maxLength="255" required
property name="active"                      type="boolean"  dbtype="boolean"                 required
property name="reset_password_token"        type="string"   dbtype="varchar" maxLength="35"  required
property name="reset_password_key"          type="string"   dbtype="varchar" maxLength="60"  required
property name="reset_password_token_expiry" type="datetime" dbtype="datetime"                required
property name="last_logged_in"              type="datetime" dbtype="datetime"                required
property name="last_logged_out"             type="datetime" dbtype="datetime"                required
property name="last_request_made"           type="datetime" dbtype="datetime"                required

property name="benefits" relationship="many-to-many" relatedTo="website_benefit";
```

4.2.18 System config

Overview

The system config object is used to store system settings (see Editable System settings). See CMS Permissioning for more information on permissioning.

Object name: system_config

Table name: psys_system_config

Path: /preside-objects/admin/configuration/system_config.cfc

Properties

```
property name="category" type="string" dbtype="varchar" maxlength="50" required="true"  uniqueindexes
property name="setting"  type="string" dbtype="varchar" maxlength="50" required="true"  uniqueindexes
property name="value"    type="string" dbtype="text"                   required="false";
```

4.2.19 Context permission

Overview

A context permission records a grant or deny permission for a given user user group, permission key and context. See CMS Permissioning for more information on permissioning.

Object name: security_context_permission

Table name: psys_security_context_permission

Path: /preside-objects/admin/security/security_context_permission.cfc

Properties

```
property name="permission_key" type="string" dbtype="varchar" maxlength="100" required=true uniqueinc
property name="context"        type="string" dbtype="varchar" maxlength="100" required=true uniqueinc
property name="context_key"    type="string" dbtype="varchar" maxlength="100" required=true uniqueinc
property name="security_group" relationship="many-to-one"                     required=true uniqueinc
property name="granted"        type="boolean" dbtype="boolean" required=true;
```

4.2.20 User

Overview

A user represents someone who can login to the website administrator. See CMS Permissioning for more information on users and permissioning.

Object name: security_user

Table name: psys_security_user

Path: /preside-objects/admin/security/security_user.cfc

Properties

```
property name="known_as"                        type="string"   dbtype="varchar" maxLength="50"  requ
property name="login_id"                        type="string"   dbtype="varchar" maxLength="50"  requ
property name="password"                        type="string"   dbtype="varchar" maxLength="60"  requ
property name="email_address"                   type="string"   dbtype="varchar" maxLength="255" requ
property name="active"                          type="boolean"  dbtype="boolean" required=false defau
property name="reset_password_token"            type="string"   dbtype="varchar" maxLength="35"  requ
property name="reset_password_key"              type="string"   dbtype="varchar" maxLength="60"  requ
property name="reset_password_token_expiry"     type="datetime" dbtype="datetime"                requ
property name="subscribed_to_all_notifications" type="boolean"  dbtype="boolean"                 requ
property name="last_logged_in"                  type="datetime" dbtype="datetime"                requ
property name="last_logged_out"                 type="datetime" dbtype="datetime"                requ
property name="last_request_made"               type="datetime" dbtype="datetime"                requ
```

```
property name="groups" relationship="many-to-many" relatedTo="security_group";
```

4.2.21 User group

Overview

User groups allow you to bulk assign a set of Roles to a number of users. See CMS Permissioning for more information on users and permissioning.

Object name: security_group

Table name: psys_security_group

Path: /preside-objects/admin/security/security_group.cfc

Properties

```
property name="label" uniqueindexes="role_name";
property name="description"  type="string"  dbtype="varchar" maxLength="200"  required="false";
property name="roles"        type="string"  dbtype="varchar" maxLength="1000" required="false" contro
```

4.2.22 Notification

Overview

The notification object is used to store notifications that can be consumed by admin users

Object name: admin_notification

Table name: psys_admin_notification

Path: /preside-objects/admin/notifications/admin_notification.cfc

Properties

```
property name="topic"     type="string"  dbtype="varchar" maxlength=200 required=true indexes="topic,
property name="type"      type="string"  dbtype="varchar" maxlength=10  required=true indexes="type,t
property name="data"      type="string"  dbtype="text"                  required=false;
property name="data_hash" type="string"  dbtype="varchar" maxlength=32  required=false indexes="topic
```

4.2.23 Notification consumer

Overview

The notification subscription object is used to store details of a single user's subscriptions to particular notification topics

Object name: admin_notification_subscription

Table name: psys_admin_notification_subscription

Path: /preside-objects/admin/notifications/admin_notification_subscription.cfc

Properties

```
property name="security_user"           required=true  uniqueindexes="notificationSubscriber|1" relat
property name="topic"                   required=true  uniqueindexes="notificationSubscriber|2" type=
property name="get_email_notifications" required=false type="boolean" dbtype="boolean" indexes="emai
```

4.2.24 Notification

Overview

The notification topic object is used to store global configuration for a given notification topic

Object name: admin_notification_topic

Table name: psys_admin_notification_topic

Path: /preside-objects/admin/notifications/admin_notification_topic.cfc

Properties

```
property name="topic"                type="string"  dbtype="varchar" maxlength=200 required=true uni
property name="send_to_email_address" type="string"  dbtype="text"                  required=false;
property name="save_in_cms"           type="boolean" dbtype="boolean"               required=false de
```

4.2.25 Notification consumer

Overview

The notification consumer object is used to store details of a single user's interactions with a notification

Object name: admin_notification_consumer

Table name: psys_admin_notification_consumer

Path: /preside-objects/admin/notifications/admin_notification_consumer.cfc

Properties

```
property name="admin_notification" relationship="many-to-one" required=true uniqueindexes="notificati
property name="security_user"      relationship="many-to-one" required=true uniqueindexes="notificati

property name="read" type="boolean" dbtype="boolean" required=false default=false indexes="read";
```

4.2.26 Audit log

Overview

The audit log object is used to store audit trail logs that are triggered by user actions in the administrator (or any other actions you wish to track).

Object name: audit_log

Table name: psys_audit_log

Path: /preside-objects/admin/audit/audit_log.cfc

Properties

```
property name="detail"     type="string"  dbtype="varchar" maxLength="200" required=true;
property name="source"     type="string"  dbtype="varchar" maxLength="100" required=true;
property name="action"     type="string"  dbtype="varchar" maxLength="100" required=true;
property name="type"       type="string"  dbtype="varchar" maxLength="100" required=true;
property name="instance"   type="string"  dbtype="varchar" maxLength="200" required=true;
property name="uri"        type="string"  dbtype="varchar" maxLength="255" required=true;
property name="user_ip"    type="string"  dbtype="varchar" maxLength="15"  required=true;
property name="user_agent" type="string"  dbtype="varchar" maxLength="255" required=true;

property name="user" relationship="many-to-one" relatedTo="security_user" required="true";

property name="datecreated" indexes="logged"; // add a DB index to the default 'datecreated' property
```

4.2.27 workflow_state

Overview

The workflow_state object saves the state and status of a single workflow item

Object name: workflow_state

Table name: psys_workflow_state

Path: /preside-objects/workflow/workflow_state.cfc

Properties

```
property name="workflow"  type="string" dbtype="varchar" maxlength=50 required=true uniqueindexes="wo
property name="reference" type="string" dbtype="varchar" maxlength=50 required=true uniqueindexes="wo
property name="owner"     type="string" dbtype="varchar" maxlength=50 required=true uniqueindexes="wo
property name="state"     type="string" dbtype="text";
property name="status"    type="string" dbtype="varchar" maxlength=50 required=true;
property name="expires"   type="date"   dbtype="datetime" required=false;
```

4.2.28 Site alias domain

Overview

The Site alias domain object represents a single domain that can also be used to serve the site. Good examples are when you have a separate domain for serving the mobile version of the site, i.e. www.mysite.com and m.mysite.com.

Object name: site_alias_domain

Table name: psys_site_alias_domain

Path: /preside-objects/core/site_alias_domain.cfc

Properties

```
property name="domain" type="string" dbtype="varchar" maxlength="255" required=true uniqueindexes="si
property name="site" relationship="many-to-one"                       required=true uniqueindexes="si
```

4.2.29 Draft

Overview

The draft object represents any draft data that is stored against a specific User.

Object name: draft

Table name: psys_draft

Path: /preside-objects/core/draft.cfc

Properties

```
property name="key" type="string"  dbtype="varchar" maxlength="200"        required="true" uniqueinde
property name="owner" relationship="many-to-one" relatedTo="security_user" required="true" uniqueinde
property name="content" type="string" dbtype="longtext" required="false";
```

4.2.30 Log entry

Overview

The log_entry object stores any log entries from logs using the the PresideDbAppender log appender through logbox.

Object name: log_entry

Table name: psys_log_entry

Path: /preside-objects/core/log_entry.cfc

Properties

```
property name="id"          type="numeric" dbtype="bigint"  generator="increment";
property name="severity"    type="string"  dbtype="varchar" maxLength="20" indexes="severity" require
property name="category"    type="string"  dbtype="varchar" maxLength="50" indexes="category" require
property name="message"     type="string"  dbtype="text";
property name="extra_info"  type="string"  dbtype="text";

property name="admin_user_id" relationship="many-to-one" relatedTo="security_user";
property name="web_user_id"   relationship="many-to-one" relatedTo="website_user";

property name="datemodified" deleted=true;
```

4.2.31 Site redirect domain

Overview

The Site redirect domain object represents a single domain that will permanently redirect to the default domain for a site.

Object name: site_redirect_domain

Table name: psys_site_redirect_domain

Path: /preside-objects/core/site_redirect_domain.cfc

Properties

```
property name="domain" type="string" dbtype="varchar" maxlength="255" required=true uniqueindexes="si
property name="site" relationship="many-to-one"                       required=true uniqueindexes="si
```

4.2.32 Link

Overview

The link object represents a link to just about anything, be it page in the site tree, an email address or plain link

Object name: link

Table name: psys_link

Path: /preside-objects/core/link.cfc

Properties

```
property name="internal_title" type="string" dbtype="varchar" maxlength="100" required=true  uniquein
property name="type"           type="string" dbtype="varchar" maxlength="20"  required=false default=
property name="title"          type="string" dbtype="varchar" maxlength="200" required=false;
property name="target"         type="string" dbtype="varchar" maxlength="20"  required=false format="
property name="text"           type="string" dbtype="varchar" maxlength="400" required=false;

property name="external_protocol" type="string" dbtype="varchar" maxlength="10"  required=false defau
property name="external_address"  type="string" dbtype="varchar" maxlength="255" required=false;
property name="email_address"     type="string" dbtype="varchar" maxlength="255" required=false;
property name="email_subject"     type="string" dbtype="varchar" maxlength="100" required=false;
property name="email_body"        type="string" dbtype="varchar" maxlength="255" required=false;

property name="page"  relationship="many-to-one" relatedto="page"  required=false;
property name="asset" relationship="many-to-one" relatedto="asset" required=false;
property name="image" relationship="many-to-one" relatedto="asset" required=false allowedTypes="image
```

4.2.33 Sitetree Page

Overview

The page object represents the core data that is stored for all pages in the site tree, regardless of page type.

Object name: page

Table name: psys_page

Path: /preside-objects/core/page.cfc

Properties

```
property name="title"         type="string"  dbtype="varchar"  maxLength="200" required=true control="
property name="main_content" type="string"  dbtype="text"                     required=false;
property name="teaser"        type="string"  dbtype="varchar"  maxLength="500" required=false;
property name="slug"          type="string"  dbtype="varchar"  maxLength="50"  required=false uniquein
property name="page_type"     type="string"  dbtype="varchar"  maxLength="100" required=true
property name="layout"        type="string"  dbtype="varchar"  maxLength="100" required=false
property name="sort_order"    type="numeric" dbtype="int"                      required=true
property name="active"        type="boolean" dbtype="boolean"                  required=false default=
property name="trashed"       type="boolean" dbtype="boolean"                  required=false default=
property name="old_slug"      type="string"  dbtype="varchar" maxLength="50"   required=false;

property name="main_image"  relationship="many-to-one" relatedTo="asset"                 required=f
property name="parent_page" relationship="many-to-one" relatedTo="page"                  required=f
property name="created_by"  relationship="many-to-one" relatedTo="security_user"         required=t
property name="updated_by"  relationship="many-to-one" relatedTo="security_user"         required=t

property name="internal_search_access"           type="string"  dbtype="varchar" maxLength="7"    rec
property name="search_engine_access"             type="string"  dbtype="varchar" maxLength="7"    rec
property name="author"                           type="string"  dbtype="varchar" maxLength="100"  rec
property name="browser_title"                    type="string"  dbtype="varchar" maxLength="100"  rec
property name="description"                      type="string"  dbtype="varchar" maxLength="255"  rec
property name="embargo_date"                     type="date"    dbtype="datetime"                 rec
property name="expiry_date"                      type="date"    dbtype="datetime"                 rec
property name="access_restriction"               type="string"  dbtype="varchar" maxLength="7"    rec
property name="full_login_required"              type="boolean" dbtype="boolean"                  rec
property name="exclude_from_navigation"          type="boolean" dbtype="boolean"                  rec
property name="exclude_from_sub_navigation"      type="boolean" dbtype="boolean"                  rec
property name="exclude_children_from_navigation" type="boolean" dbtype="boolean"                  rec
property name="exclude_from_sitemap"             type="boolean" dbtype="boolean"                  rec
property name="navigation_title"                 type="string"  dbtype="varchar" maxLength="200"  rec

property name="_hierarchy_id"                    type="numeric" dbtype="int"     maxLength="0"    rec
property name="_hierarchy_sort_order"            type="string"  dbtype="varchar" maxLength="200"  rec
property name="_hierarchy_lineage"               type="string"  dbtype="varchar" maxLength="200"  rec
property name="_hierarchy_child_selector"        type="string"  dbtype="varchar" maxLength="200"  rec
property name="_hierarchy_depth"                 type="numeric" dbtype="int"                      rec
property name="_hierarchy_slug"                  type="string"  dbtype="varchar" maxLength="2000" rec

property name="child_pages" relationship="one-to-many" relatedTo="page" relationshipKey="parent_page"
```

Public API Methods

UpdateChildHierarchyHelpers()

```
public void function updateChildHierarchyHelpers( required query oldData, required struct newData )
```

This method is used internally by the Sitetree Service to ensure that all child nodes of a page have the most up to date helper fields when the parent node changes. This is implemented using some funky SQL that was beyond the

capabilities of the standard Preside Object Service CRUD methods.

Arguments

Name	Type	Required	Description
oldData	query	Yes	Query record of the old parent node data
newData	struct	Yes	Struct containing the changed fields on the parent node

4.2.34 Site

Overview

The Site object represents a site / microsite that is managed by the CMS. Each site will have its own tree of Sitetree Page records.

Object name: site

Table name: psys_site

Path: /preside-objects/core/site.cfc

Properties

```
property name="name"     type="string" dbtype="varchar" maxlength="200" required=true  uniqueindexes=
property name="domain"   type="string" dbtype="varchar" maxlength="255" required=true  uniqueindexes=
property name="path"     type="string" dbtype="varchar" maxlength="255" required=true  uniqueindexes=
property name="protocol" type="string" dbtype="varchar" maxlength="5"   required=false
property name="template" type="string" dbtype="varchar" maxlength="50"  required=false;

property name="hide_from_search"     type="boolean" dbtype="boolean"                  required=false
property name="author"               type="string"  dbtype="varchar" maxLength="100"  required=false;
property name="browser_title_prefix" type="string"  dbtype="varchar" maxLength="100"  required=false;
property name="browser_title_suffix" type="string"  dbtype="varchar" maxLength="100"  required=false;
```

4.3 System form layouts

4.3.1 Asset: add form

/forms/preside-objects/asset/admin.add.xml

This form is used when adding assets in the asset manager section of the administrator. For multi file uploads, this form will be rendered once for each file.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="standard" sortorder="10" title="preside-objects.asset:standard.tab.title">
        <fieldset id="standard" sortorder="10">
            <field sortorder="10" binding="asset.title" />
            <field sortorder="20" binding="asset.author" control="textinput" />
            <field sortorder="30" binding="asset.description" control="textarea" />
        </fieldset>
    </tab>
</form>
```

4.3.2 Asset: add through picker form

/forms/preside-objects/asset/picker.add.xml

This form is used as the add asset form when a user is uploading assets through the asset picker. The form will be shown once for each file that has been uploaded.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="main" sortorder="10">
        <fieldset id="main" sortorder="10">
            <field sortorder="10" binding="asset.title" />
            <field sortorder="20" binding="asset.asset_folder" name="folder" control="assetFolderPick
            <field sortorder="30" binding="asset.author" control="textinput" />
            <field sortorder="40" binding="asset.description" control="textarea" />
        </fieldset>
    </tab>
</form>
```

4.3.3 Asset: edit form

/forms/preside-objects/asset/admin.edit.xml

This form is used when editing assets in the asset manager section of the administrator.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="standard" sortorder="10" title="preside-objects.asset:standard.tab.title">
        <fieldset id="standard" sortorder="10">
            <field sortorder="10" binding="asset.title" />
            <field sortorder="30" binding="asset.author" control="textinput" />
            <field sortorder="40" binding="asset.description" control="textarea" />
        </fieldset>
    </tab>

    <tab id="permissions" sortorder="20" title="preside-objects.asset:permissions.tab.title">
        <fieldset id="permissions" sortorder="10">
            <field sortorder="10" binding="asset.access_restriction" />
            <field sortorder="20" binding="asset.full_login_required" />
            <field sortorder="30" name="grant_access_to_benefits" control="objectPicker" object="webs
            <field sortorder="40" name="deny_access_to_benefits"  control="objectPicker" object="webs
            <field sortorder="50" name="grant_access_to_users"    control="objectPicker" object="webs
            <field sortorder="60" name="deny_access_to_users"     control="objectPicker" object="webs
        </fieldset>
    </tab>
</form>
```

4.3.4 Asset folder: add form

/forms/preside-objects/asset_folder/admin.add.xml

This form is used for adding folders in the asset manager section of the administrator.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<form>
    <tab id="basic" sortorder="10" title="preside-objects.asset_folder:basic.tab.title">
        <fieldset id="basic" sortorder="10">
            <field sortorder="10" binding="asset_folder.label" />
        </fieldset>
    </tab>
    <tab id="restrictions" sortorder="20" title="preside-objects.asset_folder:restrictions.tab.title"
        <fieldset id="restrictions" sortorder="10">
            <field sortorder="10" binding="asset_folder.allowed_filetypes" control="filetypepicker" r
            <field sortorder="20" binding="asset_folder.max_filesize_in_mb" />
        </fieldset>
    </tab>
    <tab id="permissions" sortorder="30" title="preside-objects.asset_folder:permissions.tab.title">
        <fieldset id="permissions" sortorder="10">
            <field sortorder="10" binding="asset_folder.access_restriction" />
            <field sortorder="20" binding="asset_folder.full_login_required" />
            <field sortorder="30" name="grant_access_to_benefits" control="objectPicker" object="webs
            <field sortorder="40" name="deny_access_to_benefits"  control="objectPicker" object="webs
            <field sortorder="50" name="grant_access_to_users"    control="objectPicker" object="webs
            <field sortorder="60" name="deny_access_to_users"     control="objectPicker" object="webs
        </fieldset>
    </tab>
</form>
```

4.3.5 Asset folder: edit form

/forms/preside-objects/asset_folder/admin.edit.xml

This form is used for editing folders in the asset manager section of the administrator.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" title="preside-objects.asset_folder:basic.tab.title">
        <fieldset id="basic" sortorder="10">
            <field sortorder="10" binding="asset_folder.parent_folder" control="AssetFolderPicker" />
            <field sortorder="20" binding="asset_folder.label" />
        </fieldset>
    </tab>
    <tab id="restrictions" sortorder="20" title="preside-objects.asset_folder:restrictions.tab.title"
        <fieldset id="restrictions" sortorder="10">
            <field sortorder="10" binding="asset_folder.allowed_filetypes" control="filetypepicker" r
            <field sortorder="20" binding="asset_folder.max_filesize_in_mb" />
        </fieldset>
    </tab>
    <tab id="permissions" sortorder="30" title="preside-objects.asset_folder:permissions.tab.title">
        <fieldset id="permissions" sortorder="10">
            <field sortorder="10" binding="asset_folder.access_restriction" />
            <field sortorder="20" binding="asset_folder.full_login_required" />
            <field sortorder="30" name="grant_access_to_benefits" control="objectPicker" object="webs
            <field sortorder="40" name="deny_access_to_benefits"  control="objectPicker" object="webs
            <field sortorder="50" name="grant_access_to_users"    control="objectPicker" object="webs
            <field sortorder="60" name="deny_access_to_users"     control="objectPicker" object="webs
        </fieldset>
    </tab>
</form>
```

4.3.6 Asset folder: edit root folder form

/forms/preside-objects/asset_folder/admin.edit.root.xml

This form is used to modify the default edit folder form for the root folder only. i.e. the root folder cannot have its name or parent changed

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" deleted="true" />
</form>
```

4.3.7 Asset: new version form

/forms/preside-objects/asset/newversion.xml

This form is used when uploading new versions of asset files through the asset manager interface.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="default" sortorder="10">
        <fieldset id="default" sortorder="10">
            <field name="file" sortorder="10" required="true" control="fileupload" label="cms:assetma
        </fieldset>
    </tab>
</form>
```

4.3.8 Link picker: add link form

/forms/preside-objects/link/admin.quickadd.xml

This form is used for the quick add link modal

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" title="preside-objects.link:basic.tab.title">
        <fieldset id="standard" sortorder="10">
            <field sortorder="10" binding="link.type" control="hidden" />
            <field sortorder="20" binding="link.internal_title" />
        </fieldset>

        <!-- we will show/hide these fieldsets depending on the selected link type -->
        <fieldset id="sitetree" sortorder="20">
            <field sortorder="10" binding="link.page" />
        </fieldset>

        <fieldset id="url" sortorder="30">
            <field sortorder="10" binding="link.external_protocol" control="select" values="http://,h
            <field sortorder="20" binding="link.external_address" control="textinput" />
        </fieldset>

        <fieldset id="email" sortorder="40">
            <field sortorder="10" binding="link.email_address" control="textinput" />
            <field sortorder="20" binding="link.email_subject" control="textinput" />
```

```
            <field sortorder="30" binding="link.email_body"     />
        </fieldset>

        <fieldset id="asset" sortorder="50">
            <field sortorder="10" binding="link.asset" />
        </fieldset>
    </tab>

    <tab id="content" sortorder="20" title="preside-objects.link:content.tab.title">
        <fieldset id="content" sortorder="10">
            <field sortorder="10" binding="link.title" control="textinput" />
            <field sortorder="20" binding="link.text" control="textinput" />
            <field sortorder="30" binding="link.image" />
            <field sortorder="40" binding="link.target" control="select" values="_self,_blank,_parent
        </fieldset>
    </tab>
</form>
```

4.3.9 Link picker: edit link form

/forms/preside-objects/link/admin.quickedit.xml

This form is used for editing links through the link picker form control

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" title="preside-objects.link:basic.tab.title">
        <fieldset id="standard" sortorder="10">
            <field sortorder="10" binding="link.type" control="hidden" />
            <field sortorder="20" binding="link.internal_title" />
        </fieldset>

        <!-- we will show/hide these fieldsets depending on the selected link type -->
        <fieldset id="sitetree" sortorder="20">
            <field sortorder="10" binding="link.page" />
        </fieldset>

        <fieldset id="url" sortorder="30">
            <field sortorder="10" binding="link.external_protocol" control="select" values="http://,h
            <field sortorder="20" binding="link.external_address" control="textinput" />
        </fieldset>

        <fieldset id="email" sortorder="40">
            <field sortorder="10" binding="link.email_address" control="textinput" />
            <field sortorder="20" binding="link.email_subject" control="textinput" />
            <field sortorder="30" binding="link.email_body"     />
        </fieldset>

        <fieldset id="asset" sortorder="50">
            <field sortorder="10" binding="link.asset" />
        </fieldset>
    </tab>

    <tab id="content" sortorder="20" title="preside-objects.link:content.tab.title">
        <fieldset id="content" sortorder="10">
            <field sortorder="10" binding="link.title" control="textinput" />
```

```
            <field sortorder="20" binding="link.text" control="textinput" />
            <field sortorder="30" binding="link.image" />
            <field sortorder="40" binding="link.target" control="select" values="_self,_blank,_parent
        </fieldset>
    </tab>
</form>
```

4.3.10 Maintenance mode: settings form

/forms/maintenance-mode/settings.xml

This form is used for managing maintenance mode settings such as IP whitelist, custom message and bypass password

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" >
        <fieldset id="basic" sortorder="10">
            <field name="active"          control="yesnoswitch" label="cms:maintenanceMode.form.activ
            <field name="title"           control="textinput"   label="cms:maintenanceMode.form.title
            <field name="message"         control="richeditor"  label="cms:maintenanceMode.form.messa
            <field name="bypass_password" control="textinput"   label="cms:maintenanceMode.form.bypas
            <field name="ip_whitelist"    control="textarea"    label="cms:maintenanceMode.form.ip_wh
        </fieldset>
    </tab>
</form>
```

4.3.11 Notifications: preferences form

/forms/notifications/preferences.xml

This form is used for managing a user's notification preferences

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" >
        <fieldset id="basic" sortorder="10">
            <field sortorder="10" name="subscriptions" control="notificationTopicPicker" label="cms:n
        </fieldset>
    </tab>
</form>
```

4.3.12 Notifications: topic global configuration form

/forms/notifications/topic-global-config.xml

This form is used for managing global notification preferences for a particular topic

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" >
        <fieldset id="basic" sortorder="10">
            <field sortorder="10" binding="admin_notification_topic.save_in_cms" />
```

```
            <field sortorder="20" binding="admin_notification_topic.send_to_email_address" control="t
        </fieldset>
    </tab>
</form>
```

4.3.13 Notifications: topic preferences form

/forms/notifications/topic-preferences.xml

This form is used for managing a user's notification preferences for specific topics

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" >
        <fieldset id="basic" sortorder="10">
            <field sortorder="10" binding="admin_notification_subscription.get_email_notifications" /
        </fieldset>
    </tab>
</form>
```

4.3.14 Richeditor: attachment form

/forms/richeditor/link.xml

This form is used for the add/edit link screen in the richeditor.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" title="cms:ckeditor.linkpicker.basic.tab">
        <!-- we will show/hide these fieldsets depending on the selected link type -->
        <fieldset id="sitetree" sortorder="10">
            <field sortorder="10" name="page" control="sitetreePagePicker" required="true" label="cms
        </fieldset>

        <fieldset id="url" sortorder="20">
            <field sortorder="10" name="protocol" control="select" required="true" label="cms:ckedito
            <field sortorder="20" name="address" control="textinput" required="true" label="cms:cked
        </fieldset>

        <fieldset id="email" sortorder="30">
            <field sortorder="10" name="emailaddress" control="textinput" required="true"  maxlength=
            <field sortorder="20" name="emailsubject" control="textinput" required="false" maxlength=
            <field sortorder="30" name="emailbody"    control="textarea"  required="false" maxlength=
        </fieldset>

        <fieldset id="anchor" sortorder="40">
            <field sortorder="10" name="anchor" control="select" required="true" label="cms:ckeditor
        </fieldset>
    </tab>

    <tab id="advanced" sortorder="20" title="cms:ckeditor.linkpicker.advanced.tab">
        <fieldset id="advanced" sortorder="10">
            <field sortorder="10" name="link_target" control="select" required="true" label="cms:cke
            <field sortorder="20" name="title" control="textinput" required="false" maxlength="200" 
```

```
        </fieldset>
    </tab>
</form>
```

4.3.15 Richeditor: attachment form

/forms/richeditor/link.xml

This form is used for the add/edit link screen in the richeditor.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" title="cms:ckeditor.linkpicker.basic.tab">
        <!-- we will show/hide these fieldsets depending on the selected link type -->
        <fieldset id="sitetree" sortorder="10">
            <field sortorder="10" name="page" control="sitetreePagePicker" required="true" label="cms
        </fieldset>

        <fieldset id="url" sortorder="20">
            <field sortorder="10" name="protocol" control="select" required="true" label="cms:ckedito
            <field sortorder="20" name="address" control="textinput" required="true" label="cms:cked
        </fieldset>

        <fieldset id="email" sortorder="30">
            <field sortorder="10" name="emailaddress" control="textinput" required="true"  maxlength=
            <field sortorder="20" name="emailsubject" control="textinput" required="false" maxlength=
            <field sortorder="30" name="emailbody"    control="textarea"  required="false" maxlength=
        </fieldset>

        <fieldset id="anchor" sortorder="40">
            <field sortorder="10" name="anchor" control="select" required="true" label="cms:ckeditor.
        </fieldset>
    </tab>

    <tab id="advanced" sortorder="20" title="cms:ckeditor.linkpicker.advanced.tab">
        <fieldset id="advanced" sortorder="10">
            <field sortorder="10" name="link_target" control="select" required="true" label="cms:cked
            <field sortorder="20" name="title" control="textinput" required="false" maxlength="200" l
        </fieldset>
    </tab>
</form>
```

4.3.16 Richeditor: image form

/forms/richeditor/image.xml

This form is used for the add/edit image screen in the richeditor.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="main" sortorder="10" title="cms:ckeditor.imagepicker.main.tab">
        <fieldset id="main" sortorder="10">
            <field sortorder="10" name="asset"       control="assetPicker"     required="true"  label
            <field sortorder="20" name="alt_text"    control="textinput"       required="false" label
```

```
            <field sortorder="30" name="dimensions"  control="imagedimensions" required="true"  label
            <field sortorder="40" name="quality"     control="select"  required="true"  label="cms:ck
            <field sortorder="50" name="alignment"   control="select"  required="false" label="cms:ck
            <field sortorder="60" name="spacing"     control="spinner" required="false" label="cms:ck
        </fieldset>
    </tab>
    <tab id="caption" sortorder="20" title="cms:ckeditor.imagepicker.caption.tab">
        <fieldset id="caption" sortorder="10">
            <field sortorder="10" name="copyright" control="textinput"   required="false"  label="cms
            <field sortorder="20" name="caption"   control="richeditor"  required="false"  label="cms
        </fieldset>
    </tab>
    <tab id="link" sortorder="30" title="cms:ckeditor.imagepicker.link.tab">
        <fieldset id="link" sortorder="10">
            <field sortorder="10" name="link" control="textinput" required="false" label="cms:ckeditc
            <field sortorder="20" name="link_target" control="select" required="false" label="cms:cke
        </fieldset>
    </tab>
</form>
```

4.3.17 Site: add form

/forms/preside-objects/site/admin.add.xml

This form is used for the "add site" form in the site manager

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" title="preside-objects.site:basic.tab.title">
        <fieldset id="basic" sortorder="10">
            <field binding="site.name"     sortorder="10" control="textinput" />
            <field binding="site.protocol" sortorder="20" control="select"    values="http,https" lab
            <field binding="site.domain"   sortorder="30" control="textinput" />
            <field binding="site.path"     sortorder="40" control="textinput" />
            <field binding="site.template" sortorder="50" control="sitetemplatepicker"  />
        </fieldset>
    </tab>
    <tab id="seo" sortorder="20" title="preside-objects.site:seo.tab.title">
        <fieldset id="seo" sortorder="10">
            <field binding="site.hide_from_search"     sortorder="10" />
            <field binding="site.author"               sortorder="20" />
            <field binding="site.browser_title_prefix" sortorder="30" />
            <field binding="site.browser_title_suffix" sortorder="40" />
        </fieldset>
    </tab>
    <tab id="advanced" sortorder="30" title="preside-objects.site:advanced.tab.title">
        <fieldset id="advanced" sortorder="10">
            <field name="alias_domains"    sortorder="10" control="textarea" label="preside-objects.s
            <field name="redirect_domains" sortorder="20" control="textarea" label="preside-objects.s
        </fieldset>
    </tab>
</form>
```

4.3.18 Site: edit form

/forms/preside-objects/site/admin.edit.xml

This form is used for the "edit site" form in the site manager

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" title="preside-objects.site:basic.tab.title">
        <fieldset id="basic" sortorder="10">
            <field binding="site.name"     sortorder="10" control="textinput" />
            <field binding="site.protocol" sortorder="20" control="select"    values="http,https" lab
            <field binding="site.domain"   sortorder="30" control="textinput" />
            <field binding="site.path"     sortorder="40" control="textinput" />
            <field binding="site.template" sortorder="50" control="sitetemplatepicker"  />
        </fieldset>
    </tab>
    <tab id="seo" sortorder="20" title="preside-objects.site:seo.tab.title">
        <fieldset id="seo" sortorder="10">
            <field binding="site.hide_from_search"     sortorder="10" />
            <field binding="site.author"               sortorder="20" />
            <field binding="site.browser_title_prefix" sortorder="30" />
            <field binding="site.browser_title_suffix" sortorder="40" />
        </fieldset>
    </tab>
    <tab id="advanced" sortorder="30" title="preside-objects.site:advanced.tab.title">
        <fieldset id="advanced" sortorder="10">
            <field name="alias_domains"    sortorder="10" control="textarea" label="preside-objects.s
            <field name="redirect_domains" sortorder="20" control="textarea" label="preside-objects.s
        </fieldset>
    </tab>
</form>
```

4.3.19 Sitetree Page: add form

/forms/preside-objects/page/add.xml

This form is used as the base "add page" form for Sitetree pages. See also Sitetree Page: edit form.

Note: When an add page form is rendered, it gets mixed in with any forms that are defined for the *page type* that is being added.

See `/devguides/formlayouts` for a guide on form layouts and mixing forms.

See Working with page types for a guide to page types.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="main" sortorder="10" title="preside-objects.page:editform.basictab.title" description="p
        <fieldset id="main" sortorder="10">
            <field sortorder="10" binding="page.title" />
            <field sortorder="20" binding="page.slug" control="autoslug" required="true" basedOn="tit
            <field sortorder="30" binding="page.active" />
            <field sortorder="40" binding="page.main_image" />
            <field sortorder="50" binding="page.layout" />
            <field sortorder="60" binding="page.teaser" />
```

```
            <field sortorder="70" binding="page.main_content" />
        </fieldset>
    </tab>

    <tab id="meta" sortorder="20" title="preside-objects.page:editform.metadatatab.title" description
        <fieldset id="meta" sortorder="10">
            <field sortorder="05" binding="page.internal_search_access" />
            <field sortorder="10" binding="page.search_engine_access" />
            <field sortorder="20" binding="page.browser_title" />
            <field sortorder="30" binding="page.author" />
            <field sortorder="40" binding="page.description" />
        </fieldset>
    </tab>

    <tab id="dates" sortorder="30" title="preside-objects.page:editform.dateControlTab.title" descrip
        <fieldset id="dates" sortorder="10">
            <field sortorder="10" binding="page.embargo_date" control="datepicker" />
            <field sortorder="20" binding="page.expiry_date"  control="datepicker" />
        </fieldset>
    </tab>

    <tab id="navigation" sortorder="40" title="preside-objects.page:editform.navigationtab.title" des
        <fieldset id="navigation" sortorder="10">
            <field sortorder="10" binding="page.navigation_title" control="textinput" />
            <field sortorder="20" binding="page.exclude_from_navigation" />
            <field sortorder="30" binding="page.exclude_from_sub_navigation" />
            <field sortorder="40" binding="page.exclude_children_from_navigation" />
            <field sortorder="50" binding="page.exclude_from_sitemap" />
        </fieldset>
    </tab>

    <tab id="access" sortorder="50" title="preside-objects.page:editform.accesstab.title" description
        <fieldset id="access" sortorder="10">
            <field sortorder="10" binding="page.access_restriction" />
            <field sortorder="20" binding="page.full_login_required" />
            <field sortorder="30" name="grant_access_to_benefits" control="objectPicker" object="webs
            <field sortorder="40" name="deny_access_to_benefits"  control="objectPicker" object="webs
            <field sortorder="50" name="grant_access_to_users"    control="objectPicker" object="webs
            <field sortorder="60" name="deny_access_to_users"     control="objectPicker" object="webs
        </fieldset>
    </tab>
</form>
```

4.3.20 Sitetree Page: edit form

/forms/preside-objects/page/edit.xml

This form is used as the base "edit page" form for Sitetree pages. See also Sitetree Page: add form.

Note: When an edit page form is rendered, it gets mixed in with any forms that are defined for the *page type* of the given page.

See `/devguides/formlayouts` for a guide on form layouts and mixing forms.

See Working with page types for a guide to page types.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="main" sortorder="10" title="preside-objects.page:editform.basictab.title" description="p
        <fieldset id="main" sortorder="10">
            <field sortorder="5" binding="page.parent_page" control="sitetreePagePicker" required="tr
            <field sortorder="10" binding="page.title" />
            <field sortorder="20" binding="page.slug" required="true" />
            <field sortorder="30" binding="page.active" />
            <field sortorder="40" binding="page.main_image" />
            <field sortorder="50" binding="page.layout" />
            <field sortorder="60" binding="page.teaser" />
            <field sortorder="70" binding="page.main_content" />
        </fieldset>
    </tab>

    <tab id="meta" sortorder="20" title="preside-objects.page:editform.metadatatab.title" description
        <fieldset id="meta" sortorder="10">
            <field sortorder="05" binding="page.internal_search_access" />
            <field sortorder="10" binding="page.search_engine_access" />
            <field sortorder="20" binding="page.browser_title" />
            <field sortorder="30" binding="page.author" />
            <field sortorder="40" binding="page.description" />
        </fieldset>
    </tab>

    <tab id="dates" sortorder="30" title="preside-objects.page:editform.dateControlTab.title" descrip
        <fieldset id="dates" sortorder="10">
            <field sortorder="10" binding="page.embargo_date" control="datepicker" />
            <field sortorder="20" binding="page.expiry_date"  control="datepicker" />
        </fieldset>
    </tab>

    <tab id="navigation" sortorder="40" title="preside-objects.page:editform.navigationtab.title" des
        <fieldset id="navigation" sortorder="10">
            <field sortorder="10" binding="page.navigation_title" control="textinput" placeholder="pr
            <field sortorder="20" binding="page.exclude_from_navigation" />
            <field sortorder="30" binding="page.exclude_from_sub_navigation" />
            <field sortorder="40" binding="page.exclude_children_from_navigation" />
            <field sortorder="50" binding="page.exclude_from_sitemap" />
        </fieldset>
    </tab>

    <tab id="access" sortorder="50" title="preside-objects.page:editform.accesstab.title" description
        <fieldset id="access" sortorder="10">
            <field sortorder="10" binding="page.access_restriction" />
            <field sortorder="20" binding="page.full_login_required" />
            <field sortorder="30" name="grant_access_to_benefits" control="objectPicker" object="webs
            <field sortorder="40" name="deny_access_to_benefits"  control="objectPicker" object="webs
            <field sortorder="50" name="grant_access_to_users"    control="objectPicker" object="webs
            <field sortorder="60" name="deny_access_to_users"     control="objectPicker" object="webs
        </fieldset>
    </tab>
</form>
```

4.3.21 Sitetree Page: restore form

/forms/preside-objects/page/restore.xml

This form is used in the 'restore page' screen in the sitetree section of the administrator. This occurs when a user wants to restore a page that has been previously deleted and stored in the recycle bin.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="main" sortorder="10">
        <fieldset id="main" sortorder="10">
            <field sortorder="10" binding="page.parent_page" control="sitetreePagePicker" required="t
            <field sortorder="20" binding="page.slug" />
            <field sortorder="30" binding="page.active" label="cms:sitetree.restore.active.label" />
        </fieldset>
    </tab>
</form>
```

4.3.22 Sitetree page type: Homepage: edit form

/forms/page-types/homepage/edit.xml

This form is used when editing pages in the site tree manager who's page type is "homepage". It gets mixed in with the Sitetree Page: edit form and its purpose is to remove a number of unwanted fields and tabs from the default form.

Note: There is no 'add' form for the homepage page type because there can only be and must always be a single homepage in a given site (subject to change).

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="main">
        <fieldset id="main">
            <field name="parent_page" deleted="true" />
            <field name="active"      deleted="true" />
            <field name="slug"        deleted="true" />
            <field name="layout"      deleted="true" />
        </fieldset>
    </tab>

    <tab id="dates" deleted="true" />
</form>
```

4.3.23 System config form: Asset manager

/forms/system-config/asset-manager.xml

This form is used for configuring aspects of the asset manager

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="default" sortorder="10">
        <fieldset id="default" sortorder="10">
            <field sortorder="10" name="retrieve_metadata" control="yesnoswitch" required="false" lab
```

```
        </fieldset>
    </tab>
</form>
```

4.3.24 System config form: Email

/forms/system-config/email.xml

This form is used for configuring the mail server and other mail related settings

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="default" sortorder="10">
        <fieldset id="default" sortorder="10">
            <field sortorder="10" name="server"               control="textinput" required="false"
            <field sortorder="20" name="port"                 control="spinner"   required="false" de
            <field sortorder="30" name="username"             control="textinput" required="false"
            <field sortorder="40" name="password"             control="password"  required="false"
            <field sortorder="50" name="default_from_address" control="textinput" required="false"
        </fieldset>
    </tab>
</form>
```

4.3.25 System config form: Multilingual settings

/forms/system-config/multilingual.xml

This form is used for configuring aspects of Preside's multlingual content capabilities

```
<?xml version="1.0" encoding="UTF-8"?>

<form feature="multilingual">
    <tab id="default" sortorder="10">
        <fieldset id="default" sortorder="10">
            <field sortorder="10" name="default_language"     control="objectpicker" object="multilin
            <field sortorder="20" name="additional_languages" control="objectpicker" object="multilin
        </fieldset>
    </tab>
</form>
```

4.3.26 System config form: Website users

/forms/system-config/website_users.xml

This form is used to configure the core website users system.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="default" sortorder="10">
        <fieldset id="default" sortorder="10">
            <field sortorder="10" name="allow_remember_me"           control="yesNoSwitch"        rec
            <field sortorder="20" name="remember_me_expiry"          control="spinner"            rec
            <field sortorder="30" name="reset_password_token_expiry" control="spinner"            rec
```

```
            <field sortorder="40" name="default_post_login_page"     control="siteTreePagePicker" req
            <field sortorder="50" name="default_post_logout_page"    control="siteTreePagePicker" req
        </fieldset>
    </tab>
</form>
```

4.3.27 Update manager settings form

/forms/update-manager/general.settings.xml

This form is used for updating general settings of the Update manager. i.e. Which release branch should updates be fetch from, etc.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="administrator" sortorder="10">
        <fieldset id="administrator" sortorder="10">
            <field  sortorder="10" name="branch"           control="select"   required="true"  label=
            <field  sortorder="20" name="railo_admin_pw"   control="password" required="false" label=
            <field  sortorder="30" name="download_timeout" control="spinner"  required="false" label=
        </fieldset>
    </tab>
</form>
```

4.3.28 User: add form

/forms/preside-objects/security_user/admin.add.xml

This form is used for the "add user" form in the user admin section of the administrator.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10">
        <fieldset id="basic" sortorder="10" title="preside-objects.security_user:fieldset.details" de
            <field binding="security_user.email_address" required="true" />
            <field binding="security_user.known_as" />
            <field binding="security_user.login_id" control="autoslug" basedOn="label" />
            <field binding="security_user.groups" />
        </fieldset>

        <fieldset id="welcome" sortorder="20" title="preside-objects.security_user:fieldset.welcome"
            <field name="send_welcome"    control="yesNoSwitch" default="true" label="preside-objects
            <field name="welcome_message" control="textarea"    label="preside-objects.security_user
        </fieldset>
    </tab>
</form>
```

4.3.29 User: edit form

/forms/preside-objects/security_user/admin.edit.xml

This form is used for the "edit user" form in the user admin section of the administrator.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10">
        <fieldset id="basic" sortorder="10" title="preside-objects.security_user:fieldset.details" de
            <field binding="security_user.email_address" required="true" />
            <field binding="security_user.known_as" />
            <field binding="security_user.active" />
            <field binding="security_user.groups" />
        </fieldset>

        <fieldset id="welcome" sortorder="20" title="preside-objects.security_user:fieldset.resend.we
            <field name="resend_welcome"  control="yesNoSwitch" default="false" label="preside-object
            <field name="welcome_message" control="textarea"    label="preside-objects.security_user
        </fieldset>
    </tab>
</form>
```

4.3.30 User: edit profile form

/forms/preside-objects/security_user/admin.edit.profile.xml

This form is used for the "edit my profile" form

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10">
        <fieldset id="basic" sortorder="10">
            <field binding="security_user.email_address" required="true" />
            <field binding="security_user.known_as" />
            <field name="password" control="password" required="false" label="preside-objects.securit
            <field name="confirm_password" control="password" required="false" label="preside-objects
                <rule validator="sameAs">
                    <param name="field" value="password" />
                </rule>
            </field>
        </fieldset>
    </tab>
</form>
```

4.3.31 User: edit self form

/forms/preside-objects/security_user/admin.edit.self.xml

This form is used for the "edit user" form in the user admin section of the administrator **when the user being edited is the same as the logged in user**.

Note: This form gets mixed in with User: edit form. Its purpose is to remove the "active" flag control, preventing the user from deactivating themselves (the service layer also prevents this).

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic">
```

```
        <fieldset id="basic">
            <field name="active" deleted="true" />
        </fieldset>
    </tab>
</form>
```

4.3.32 User group: add form

/forms/preside-objects/security_group/admin.edit.xml

This form is used for the "edit user group" form in the user admin section of the administrator.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10">
        <fieldset id="basic" sortorder="10">
            <field binding="security_group.label" />
            <field binding="security_group.description" />
            <field binding="security_group.roles"  />
        </fieldset>
    </tab>
</form>
```

4.3.33 User group: add form

/forms/preside-objects/security_group/admin.edit.xml

This form is used for the "edit user group" form in the user admin section of the administrator.

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10">
        <fieldset id="basic" sortorder="10">
            <field binding="security_group.label" />
            <field binding="security_group.description" />
            <field binding="security_group.roles"  />
        </fieldset>
    </tab>
</form>
```

4.3.34 Website benefit: add form

/forms/preside-objects/website_benefit/admin.add.xml

This form is used for the "add website benefit" form in the website user manager

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10">
        <fieldset id="basic" sortorder="10">
            <field binding="website_benefit.label"             sortorder="10"  />
            <field binding="website_benefit.description"       sortorder="20"  />
```

```
            <field binding="website_benefit.combined_benefits" sortorder="25"  />
            <field name="permissions"                   sortorder="30" control="websitePermissionsPi
        </fieldset>
    </tab>
</form>
```

4.3.35 Website benefit: edit form

/forms/preside-objects/website_benefit/admin.edit.xml

This form is used for the "edit website benefit" form in the website user manager

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10">
        <fieldset id="basic" sortorder="10">
            <field binding="website_benefit.label"             sortorder="10"  />
            <field binding="website_benefit.description"       sortorder="20"  />
            <field binding="website_benefit.combined_benefits" sortorder="25"  />
            <field name="permissions"                   sortorder="30" control="websitePermissionsPi
        </fieldset>
    </tab>
</form>
```

4.3.36 Website user: add form

/forms/preside-objects/website_user/admin.add.xml

This form is used for the "add website user" form in the website user manager

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" title="preside-objects.website_user:basic.tab.title">
        <fieldset id="basic" sortorder="10">
            <field binding="website_user.login_id"      sortorder="10" control="textinput"  />
            <field binding="website_user.email_address" sortorder="20" control="textinput"  />
            <field binding="website_user.display_name"  sortorder="30" control="textinput"  />
            <field binding="website_user.active"        sortorder="40"  />
        </fieldset>
    </tab>
    <tab id="security" sortorder="20" title="preside-objects.website_user:security.tab.title">
        <fieldset id="security" sortorder="10">
            <field binding="website_user.benefits"      sortorder="10"  />
            <field name="permissions"                   sortorder="20" control="websitePermissionsPic
        </fieldset>
    </tab>
</form>
```

4.3.37 Website user: edit form

/forms/preside-objects/website_user/admin.edit.xml

This form is used for the "edit website user" form in the website user manager

```
<?xml version="1.0" encoding="UTF-8"?>

<form>
    <tab id="basic" sortorder="10" title="preside-objects.website_user:basic.tab.title">
        <fieldset id="basic" sortorder="10">
            <field binding="website_user.login_id"      sortorder="10" control="textinput"  />
            <field binding="website_user.email_address" sortorder="20" control="textinput"  />
            <field binding="website_user.display_name"  sortorder="30" control="textinput"  />
            <field binding="website_user.active"        sortorder="40"  />
        </fieldset>
    </tab>
    <tab id="security" sortorder="20" title="preside-objects.website_user:security.tab.title">
        <fieldset id="security" sortorder="10">
            <field binding="website_user.benefits"      sortorder="10"  />
            <field name="permissions"                   sortorder="20" control="websitePermissionsPic
        </fieldset>
    </tab>
</form>
```

Contributing to the documentation where you find it lacking or problematic is as easy as:

- Creating a pull request on GitHub
- Commenting on the pages
- Pinging the authors on twitter (@presidecms)