

spunQ 1.3

TECHNICAL DOCUMENTATION

Documentation is the essential of a software project

/*****

```
* @package: spunq 1.3 / technical documentation
* @author: juergen schmidt (www.strg.at / juergen@strg.at)
* @author: martin schmidt (www.strg.at / martin@strg.at)
* @version: 1.3
* @license: copyright strg.at
```

*****/

www.strg.at

Liniengasse 20/12

1060 Wien

email: office@strg.at

Phone: ++43 1 526 56 29

Fax: ++43 1 526 56 49

Content

1) General Information	4
2) Installation and Configuration	4
2.1) Requirements	4
2.2) Installation Basics	4
2.3) Database Installation	5
2.3.1) PostgreSQL	5
2.3.2) MySQL	6
2.4) Configuration	6
2.5) Filesystem Permissions	7
2.6) Run the sample site	8
3) How spunQ works	9
3.1) Software-Architecture	9
3.1.1) Layer Model	9
3.1.2) Data Structure	10
3.1.3) Permissions and Security	11
3.1.3.1) System Groups	11
3.1.3.2) Content Groups	12
3.1.4) Relations between Objects	12
3.1.4.1) Technical details	13
3.2) Backend and Frontend	13
3.3) Multilingual Projects	14
4) The spunQ Backend	15
4.1) General Description	15
4.2) Displaying Classes	15
4.3) Editing Class Objects and Folders	17
4.3.1) Creating Relations	17
4.3.2) Creating Translations	18
4.3.3) Creating Bookmarks	19
4.3.4) Object Folders	19
4.3.5) Class Tree View	20
4.4) Searching Objects	21
4.4.1) AJAX Live Quick Search	21
4.4.2) AJAX Live Class Full Text Search	21
4.4.3) Advanced Search	22
4.5) "Tools" Menu	22
4.5.1) ProjectsAdmin	22
4.5.2) Object Statistics	22
4.5.3) RSS Feeds	23
4.5.3.1) How to configure RSS Feeds	23
4.5.4) DB Backup	25
4.5.5) Preferences	25
4.6) "System Settings" Menu	25
4.6.1) DB Config	25
4.6.1.1) Create Classes	25
4.6.1.2) Edit Classes	26
4.6.2) User	27
4.6.3) System Groups	27
4.6.4) Content Groups	29
4.6.5) Controls	29
4.6.5.1) Howto create new control elements	29
4.6.5.2) How to create new AJAX checks	30
4.6.6) System Info	31
4.7) Personalizing the menu	31
5) The spunQ Frontend	32
5.1) General Description	32
5.2) Frontend Generation	32

5.3) Template Structure	33
5.4) Useful Methods of the Frontend Class	33
5.4.1) clearGet()	33
5.4.2) debug()	34
5.4.3) doSearch()	34
5.4.4) getClassStructure()	35
5.4.5) getCookieValues()	35
5.4.6) getLang()	35
5.4.7) getParentFolder()	36
5.4.8) getPureObject()	36
5.4.9) getRelatedObjects()	37
5.4.10) getRelatedObjectsReverse()	37
5.4.11) getStructureObjects()	38
5.4.12) getStructureObjectsPure()	38
5.4.13) getTableName()	38
5.4.14) getTextById()	39
5.4.15) insertObjectData()	39
5.4.16) insertPureObject()	40
5.4.17) saveObject()	40
5.4.18) switchLanguage()	41
5.4.19) switchStyle()	41
5.4.20) updateObjectData()	41
5.4.21) updatePureObject()	42
5.4.22) writeSessionValues()	42
6) PEAR and Smarty	43
6.1) PEAR Basics	43
6.1.1) getAll()	43
6.1.2) getOne()	44
6.1.3) getRow()	44
6.1.4) getCol()	44
6.1.5) Fetch Modes	45
6.1.5.1) DB_FETCHMODE_ORDERED	45
6.1.5.2) DB_FETCHMODE_ASSOC	45
6.1.5.3) DB_FETCHMODE_OBJECT	46
6.1.6) Handling database results	46
6.2) Smarty Basics	47
6.2.1) Assigning Variables	47
6.2.2) Displaying Templates	47
6.2.3) Using language variables	47
6.2.4) Arrays and Objects	48
6.2.5) Using Conditions	48
6.2.6) Using Loops	49
6.2.7) Using Smarty Modifiers	49
7) System Architecture and Design	51
7.1) Filestructure	51
7.1.1) The spunQ Backend	51
7.1.2) The spunQ Frontend	52
7.1.3) Additional Files	52
7.2) DB Structure	53

1) General Information

spunQ is a database modelling software which has been developed since 1999 and continuously extended with various functionality. It can be used for numerous different types of websites and applications. In 2005 the software experienced a complete rewrite by the development team of the Viennese software and system company strg.at Dosser Iacopino Schmidt OEG.

Version 1.3 provides you a lot of features for classical portal-software and typical options for Web 2.0 applications. Native AJAX implementations, an advanced user-system and permissions on different levels in the system are able to fit the needs of a modern styled web.

SpunQ is written in PHP4/5 and works on databases like postgresQL, mySQL and msSQL. More or less it is the typical LAMP environment with Apache and Linux. Some features and tools are written in Perl and shell scripts. The project is well tested under Linux and FreeBSD. It should be possible to run this system in a Microsoft Environment. We just made short tests about this.

This paper refers to version 1.3 of the software. It covers the need of web-developers, programmers and system administrators. It is not related to the needs of editors and content-creators.

For your first steps we offer you source and database-dump for usage as a CMS. The example is the running spunq.com site. The needed template and code files in the frontend are also included.

2) Installation and Configuration

2.1) Requirements

- Apache 1.3 or 2.0 with mod_rewrite enabled
- PHP4 or PHP5
- PostgreSQL or MySQL database server
- PEAR DB and XML Classes
- ImageMagick

2.2) Installation Basics

In order to install spunQ you will first have to download the current tarball from

spunq.com/download and untar it on your server. The folder structure of the extracted files will look like this:

```
spunq13/  
  backend/  
  frontend/
```

The installation and configuration is divided into two main parts. First, you will have to configure the administration interface in `backend/common/config.inc.php`, then set up the frontend by editing `frontend/common/config.inc.php`. For both files there are samples named `config.inc.php.tpl` available which you can use as a base for your system configuration.

We recommend you to use two different vHosts for frontend and backend to run spunQ. For example use `yourdomain.com` pointing to the `frontend/` and `spunq.yourdomain.com` pointing to the `backend/` directory.

2.3) Database Installation

The next step is the installation of the database. As spunQ uses the PEAR DB package as an abstraction layer, both PostgreSQL and MySQL databases are supported. However we recommend you to use PostgreSQL for performance reasons.

2.3.1) PostgreSQL

Create a database as PostgreSQL system user – commonly named “pgsql” or “postgres”:

```
# createdb spunq_db; // or an other database-name according to your setup
```

For security reasons create two users for backend and frontend. Grant all privileges for the admin user and just SELECT for the web user.

```
# createuser spunq_adm;  
# createuser spunq_web;
```

Answer the questions for creating of new databases and users with NO!

You need to grant permissions to this users now. For the sample-database dump you find two SQL scripts under `backend/install`

- `pg_perm_spunq_adm.sql`
- `pg_perm_spunq_web.sql`

Run this scripts against your new database:

```
# psql spunq_db < /YOUR/PATH/backend/db/pg_perm_spunq_adm.sql
# psql spunq_db < /YOUR/PATH/backend/db/pg_perm_spunq_web.sql
```

Finally, you need to set a password for both users:

```
#psql spunq_db
spunq_db=# ALTER USER spunq_adm WITH PASSWORD 'your_password';
spunq_db=# ALTER USER spunq_web with password 'your_password';
```

Note: If you create new tables (classes) in spunQ, permissions are not granted for the web-user. you have do that manually!!

2.3.2) MySQL

Login to MySQL as root:

```
# mysql -u root -p
```

Create a database for this project:

```
mysql> CREATE DATABASE spunq_db; // or an other database-name according to your setup
```

Create a user with all permissions to create and modify tables:

```
mysql> GRANT ALL PRIVILEGES ON spunq_db.* TO spunq_adm@localhost identified by 'pwd'
mysql> GRANT SELECT ON spunq_db.* TO spunq_web@localhost identified by 'pwd'
```

Leave the MySQL and fill the database with the provided dump:

```
cd backend/install/
mysql -u root -p spunq_db < spunq_db_mysql.dump
```

2.4) Configuration

Now it's time to edit the configuration settings. Therefore, you will find templates named config.inc.php.tpl in the common directory of both the frontend and the backend.

Please walk through the file and see the inline comments for settings. Additionally, here are some explanations:

Set `DEBUG_IP` to your private IP. This offers the possibility to have the smarty-debug interface popup without bothering others working on the project.

Set `APP_PATH_ROOT` to the path on your system.

Set APP_BASE_URL to the main vHost pointing to the admin-interface.

As a second part you have to define the database connect. Please enter database_name, user and password and define the database type (pgsql or mysql). Use the second user you setup just with select-permissions.

After editing, rename them to config.inc.php and you are done.

As the configuration of the FCKEditor is done in a javascript file, you will find these setting somewhere else, precisely in backend/FCKeditor/fckconfig.js

Set FCKConfig.CustomConfigurationsPath = '/relative/path/to/spunq13/backend'.

See the file fckconfig.js for further configuration of the editor.

There is a special FCKeditor plugin to use a spunq mediaclass as filebrowser. You will find this plugin under FCKeditor/editor/plugin/spunqbrowser. You need to set the path you want into fck_spunqbrowser.php.

2.5) Filesystem Permissions

Set the right permissions on all files (for security reasons) for admin-interface and frontend.

First, create a directory called templates_c in the /frontend and /backend directories.

```
--> *NEVER* do this as ROOT <--  
su - username  
cd public_html/spunq/  
find public_html/spunq -type f -exec chmod 644 {} \  
find public_html/spunq -type d -exec chmod 755 {} \  
;
```

Take care: In this step you change permissions recursively! If you do it as root in the wrong place (worst case in /) you would change permissions on the hole system!!!

Some directories need to be writable by the user your webserver is running under.

If you have root-access to the webserver change the ownership of the following directories:

```
cd public_html/spunq13/backend/  
chown www-data:www-data templates_c // compiled smarty templates
```

```
chown www-data:www-data upload // directory for uploaded files
chown www-data:www-data backup // directory for database backups
chown -R www-data:www-data logs // directory for log-files

cd public_html/spunq13/frontend/
chown www-data:www-data templates_c // compiled smarty templates
chown www-data:www-data documents // directory for document upload
chown www-data:www-data image // directory for image upload
chown www-data:www-data rss // directory for rss feeds
chown -R www-data:www-data logs // directory for log-files
```

If you cannot get root-access provide write permissions to the folders for the web-user:

```
cd public_html/spunq13/backend
chmod 777 templates_c
chmod 777 upload
chmod 777 backup
chmod -R 777 logs
```

```
cd public_html/spunq/frontend/
chmod 777 templates_c
chmod 777 documents
chmod 777 image
chmod 777 rss
chown -R 777 logs/
```

2.6) Run the sample site

Once these settings are carried out, you are able to run the sample site. First log in into the backend using root as 'user' and 'root!' as password. Then select the "Projects Admin" from the main menu (in the section "Tools") and add your domain into the list of the spunq project.

You might want to immediately start with a new project. Then, you will have to rename one file and one folder in the frontend/ directory.

```
mv frontend/templates/spunq frontend/templates/[your_project_name]
mv frontend/spunq.php frontend/[your_project_name].php
```

You are then ready to use the system!

3) How spunQ works

SpunQ is an object-relational management system. According to legacy data this means in general that each row of data is represented in an object-table. You are able to identify each row of data system-wide by a single object ID. The object relation spunQ is using is created on an application level. You can even use a normal-legacy database like MySQL which does not use server-side object-relation functions or features. The main goal of this architecture is the possibility to reference and associate objects.

spunQ is written in a straight forward object-oriented style. Since its PHP object-orientation is limited to the representation of data and the creation of domain-classes and the layer model.

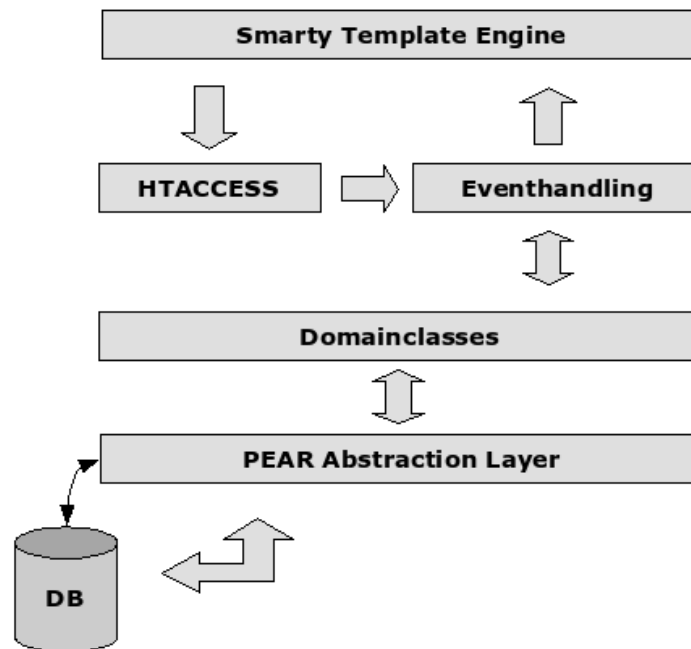
3.1) Software-Architecture

3.1.1) Layer Model

The software is designed in a straight forward layer model. All calls are processed by an event-handler. This handler uses htaccess to create a \$_GET array out of the used link. So you can work with human-readable URLs for all calls. From this handler the domain classes are called to run through a logical data flow. The domain classes then are using the PEAR abstraction-layer to access the used database. Once the data flow is finished the event handler relays needed data-objects to the presentation layer to display the data. Since spunQ is an open source software we use PEAR (pear.php.net) for database abstraction and Smarty (smarty.php.net) as presentation layer and template engine.

As you can easily see in the model, logic code and presentation are strictly separated. A workflow between programmers and designers is therefore easy to manage. All available variables together with their current values are always shown in the Smarty debugging interface. In the case of an array or object, the exact structure is also represented. By the used API for accessing the data in a possible frontend, these structures are mostly the same for all spunQ objects.

This given as a fact, frontend designers will be able to quickly create templates with the available data without being overwhelmed by the underlying code.



3.1.2) Data Structure

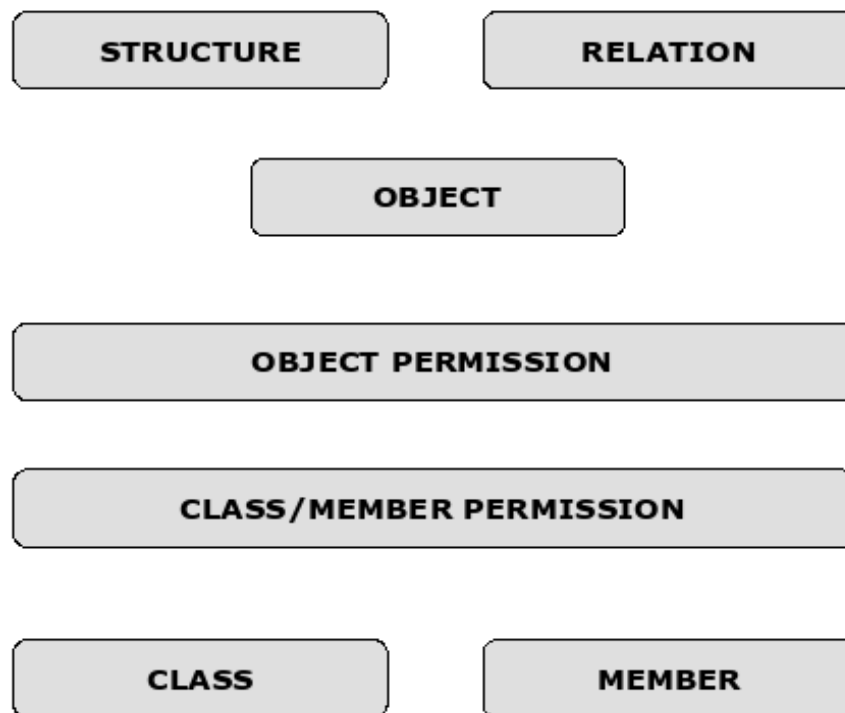
The data structure in spunQ represents the ideas of object-relational databases.

The structures of an object-relational database management offers us the possibility to represent data in a different way then in legacy databases. A spunQ object refers to a class (a table in legacy view) to store the metadata of this object. The reference is the `object_class_ref` in the object-table. To handle this views spunQ knows the attributes of each table.

Let's turn around and take a look at the object-oriented view on this idea. The object data itself is split into two tables: `object` and `object_adds`. The tables are combined by a SQL view. If you have any needs to extend the object core data for your project, use the `object_adds` table to create new columns.

A table in the view of relational databases is a class to store metadata of an object. An attribute (a column) of a table is a member of this class. spunQ knows every time all classes and all members of the system you created by the database modelling steps.

We use these skills to create the interfaces for the data and to have a low-level security and permission system.



spunQ stores the objects in a hierarchical structure. Behind this structure lays the same idea of relations and references between objects. A folder in spunQ is an object of the struct class. The connection between the struct object and the content object is handled by the relation table (see below in section 3.1.4) Those of you being familiar with the basic idea of file systems will easily understand. Technically it is done in a recursive integration.

3.1.3) Permissions and Security

Together with the idea of a strict layer model to prevent direct access from the presentation layer or an URL encoding to the domain classes or the database abstraction layer, we use a complex permission system on two levels.

3.1.3.1) System Groups

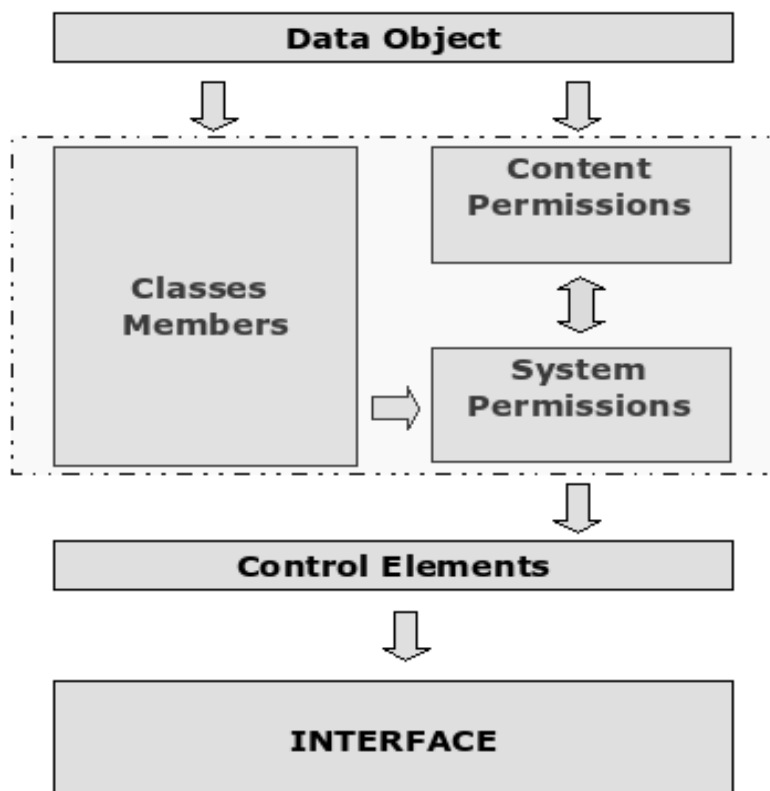
First is the general class and member permission (we call it system permission):

Users are part of a system group. An administrator can grant permissions to this group on class and on members level. Permissions can be defined as read, write and delete coming directly from SQL logic, like SELECT, UPDATE/INSERT, DELETE statements. According to the group settings one can define the interface elements for the presentation of data in the backend of spunQ (the edit interface). This is the way spunQ creates the interface for users.

Apart from the data in classes and members this is stored via db_groups in the tables class_perm and member_perm.

3.1.3.2) Content Groups

This is an implementation of the well known inheritance of object properties. An object is stored in a hierarchical data flow like we annotated above. Content permissions can be set for each content group as read/write/delete possibilities. To get started with content permissions in a special class you need to set the permissions for the root object. This object is a virtual presentation of the class and is not based on a real object. It is the start point for inheritance in this class.



3.1.4) Relations between Objects

Relations between objects (e.g. a news article with an associated image) can be easily handled with spunQ as it uses an abstracted relation table in the database being able to combine any objects among each other.

3.1.4.1) Technical details

The table “_relation”, which is the core of this functionality contains four columns:

...	_relation_class_ref1	_relation_object_ref1	_relation_class_ref2	_relation_object_ref2	...
	81	4527	82	4863	

This exemplary entry means that the object with the id 4863 in the class holding the id 82 is related to the object with the 4527 in the class holding the id 81.

This fact given, a SQL query to get all associated objects to a given class object – which is often needed in spunQ – would look like that:

```
SELECT object.* FROM _relation, object
WHERE _relation_class_ref1 = [given class_id] AND
      _relation_object_ref1 = [given object_id] AND
      _relation_class_ref2 = [id of the class you are looking for relation in] AND
      _relation_object_ref2 = object_id
```

Note: Rarely, the object_ref is entered in the table _relation instead of the object_id. This depends on the context of use, so when you are not getting what you wanted try comparing the _relation_object_1/2 with the field object_ref instead.

A special case is a search for the folder an object resides in. This situation is also mapped in the table _relation. As the struct class (which is responsible for all folders) has the ID 10, just use _relation_class_ref2 = 10 within your query. In this case, the struct_id has to be taken as _relation_object_ref2.

3.2) Backend and Frontend

Basically, spunQ is divided into two parts: The backend and the frontend.

The backend is the place where you can easily create and manage all content, set user permissions and administrate the whole system. Also, various search methods are included to quickly find the content objects you are looking for.

The frontend is what the visitors of your website or application will see. Various types of websites and applications are possible: simple websites, online shops, portals, intranet solutions and many more. The frontend code will therefore have to be adapted to fit your system.

However, we provide a comprehensive collection of useful methods to access and manipulate the data structures created in the backend. This way, development of applications with spunQ is extremely fast and flexible.

Once the frontend has been created, editorial users of your website will be able to change data in the backend and immediately see the results of their modifications in the frontend.

It is however possible to use original backend functions within the frontend. You might for example want to build a community platform where content created by users in the frontend will immediately be written in the database. In this case, remember to not just save the pure metadata but only to create spunQ objects out of the given data for all frontend methods to be still applicable (such frontend methods are built in ready to use).

On the other hand, there might be software projects (e.g. user management tools) where it will be better to use the existing backend as the application's main interface. In this case, the code and templates of the backend will just have to be extended – then, a typical frontend part is not needed any more. If you are interested in skinning the backend: This can be done by simply edit the backend/common/spunq.css style sheet file. As we annotated above the presentation layer in the backend is also created with the flexible Smarty template engine.

3.3) Multilingual Projects

With spunQ, it is no problem to create multilingual applications or websites. Once the languages have been defined in the system configuration, spunQ will display one tab for every language when editing an object in the backend. spunQ creates a child object for each translation and stores the metadata as a normal data row in the class-table.

The language definition has to be done in common/config.inc.php in the frontend as well as in the backend and both settings has to be equal for the multilingual functionality to work.

In the frontend, the built-in functions respect this fact and are able to perform language switches by writing a cookie on the client side. Thus, the whole application will immediately be translated using the smarty language variables for static content and the translated objects edited in the backend for dynamic content. If an object should not have been translated yet, the language version defined as fallback will automatically be displayed.

Technically, the different object versions are part of the object table in the database. When an english object is translated into german, the new object will get a new object_id, but has the

reference to its parent in the column 'object_parent' which then will contain the object_id of the original English version. Additionally, the field 'object_version' is e.g. set to '1' to identify the language according to the ID set in the system configuration.

4) The spunQ Backend

4.1) General Description

The spunQ menu consists of three main parts: First, there is the bar on the very top where you can search objects using an AJAX quick search or go to an advanced search form as well as leave the system. Second, you will find the so called "top menu" underneath where defined object classes may appear as quick links.

The main menu on the left however is the most important navigation tool in spunQ. It lists all classes separated into categories, followed by three special sections: "Tools", "Links" and "System Settings". Your personal bookmarks will also appear in the main menu.

In the "Tools" section, you will find the enabled spunQ modules such as object statistics or RSS feeds. Additionally, you can access the database backup tool, the "Projects Admin" and your personal user settings here.

For system administrators, the most relevant section of the main menu is the third one called "System Settings". This is the place where to manage all classes, users and groups as well as adding new control elements.

4.2) Displaying Classes



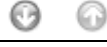







Once a class is properly set up, you can click on the accordant icon or class name in the main menu to manage its objects. You will then get to a screen where all relevant folders and objects are displayed.

For getting objects and folders for displaying in this section, only the object table containing object_name, object_version, object_parent, ... is read out. The metadata are not available for performance reasons. Only if a live class search is performed the according metadata will be considered on this screen.

Notice: In the case of an image class, thumbnails for each object will also be shown. To define

image classes, use the `$imageClasses` variable in `backend/common/config.inc.php`.

Here is a list of icons appearing on this screen:

Icon	Description
	creates a new class object, the icon underlying the yellow asterisk depends on the current class.
	displays / hides the "Advanced Editing Functions" of a class
	changes the order of class objects
	shows the current permission status of the corresponding object
	displays an object's context menu. The actions represented by the following icons will be available to choose from.
	is the edit button.
	adds an object to a user's bookmarks.
	deletes the object.
	leads to the content permission management of an object.
	is the read button. Clicking it will show an object's metadata without the possibility to edit it.

To create a new object, press the "New Object" button in the class objects header line. To edit an object simply click on its name.

There are two grey arrow buttons in each object line allowing you to change the order of objects within a class. This is relevant as it is mostly the order as they will appear in the frontend application unless the data is retrieved by a user-defined function.

Then, you will see a red, yellow or green icon representing the current permission level of the object. Green (or perm level 2) means that the object is visible in the frontend, while a red (perm level 0) status hides the object to visitors. Yellow (perm level 1) is a value in between which can be useful for editorial purposes.

When you move your mouse over the little wrench at the end of each object line, some more actions will be offered to you in a context menu: an alternative edit button, the possibility to create a bookmark for this object and the deletion of an object. Furthermore, you can set the permissions for this object. If someone has only the rights to read the contents of this object, he can do this by clicking the "Read" button. Finally, the creation and change dates of the object are displayed as well as the user who last changed it.

For deleting and moving multiple objects in a convenient way, use the "Advanced Editing Functions". By choosing this option, a checkbox will appear in front of each object. Select as many objects as you want before clicking on "Delete" or "Move" to perform the chosen action (in the latter case after you have selected the target folder as well).

4.3) Editing Class Objects and Folders

When having chosen to create a new or edit an existing object, you will get to the "Edit Object" screen.

The interface of this page depends on which members and controls have been defined for the class. There might be simple HTML input elements as well as WYSIWYG editor areas.

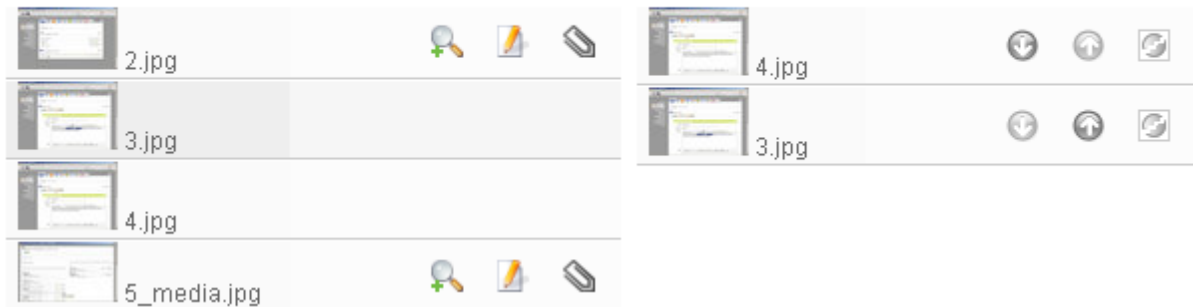
However, there are four buttons which all edit screens have in common: the first one allows you to save the entered data and return to the class view, the second to save and stay in the edit screen, the third to reset your specifications to the last saved version and the fourth to return to the class view without saving.



If the object you are editing has already been saved before (which technically means it has got an object ID in the database), some more functions are available for selection. First, there will be a star to add the current object to the user's bookmarks. Two other areas depend on the class or system configuration, respectively: Relations and translations.

4.3.1) Creating Relations

If the class has got m:n relation members assigned to (and the "m:n view" as correctly been selected as corresponding interface element), the defined relation classes will appear as icons with green border. Clicking it will open the m:n view.



Here, all available objects are shown – possibly grouped in folders if they belong to a hierarchical class – on the left. To create relations, simply click on the paper click icon. You might also want to edit those objects which is possible by clicking the edit button. A little magnifying class is additionally shown in the case of an image class to view the accordant image in its original size.

Once you have chosen an object to create a relation to, it will appear on the right side. As it has already been the case in the class view, you can change the order of the objects by using the grey arrows. In the case you have accidentally chosen a wrong object, you can dissolve the created relation by using the little grey circle-shaped icon on the right.

Technically, the creation of relations is done by inserting a new row in the table `_relation` where all relations of the systems are mapped.

The insert statement will consist of five fields being `'_relation_class_ref1'`, `'_relation_object_ref1'`, `'_relation_class_ref2'`, `'_relation_object_ref2'` and `'_relation_data_order'` whose values will be as follows:

Column	Value
<code>_relation_class_ref1</code>	The ID of the class the object being edited belongs to.
<code>_relation_object_ref1</code>	The ID of the object currently being edited.
<code>_relation_class_ref2</code>	The ID of the class the object being associated belongs to.
<code>_relation_object_ref2</code>	The ID of the object currently being associated.
<code>_relation_data_order</code>	A value which controls the relation ordering described above. Initially, this value will be equal with the ID of the current row of the <code>_relation</code> table (<code>'_relation_id'</code>).

4.3.2) Creating Translations

If your spunQ has been configured to be multilingual, there will be tabs for all defined languages once your object has been saved to the database. Choosing one of them will lead you to the same

form where you can enter your translated content.

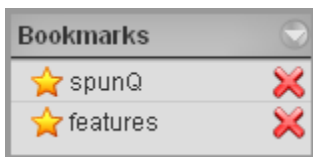
For translated versions, the same rules apply concerning the creation of relations. You have to save the new object before you can do it.

To mark an translated object in the database, it will be inserted with the object ID of the original object set as `object_parent` and the `object_version` being the ID of the new language in the system's configuration file.

4.3.3) Creating Bookmarks

To create a bookmark, chose the star icon in either one object's context menu in the class view or the same icon in the edit view.

Bookmarks are separately saved for each user and will appear in the main menu as first section – of course only if some exist.




In the database, bookmarks have their own table. By creating a bookmark, a new row consisting of five fields is inserted:

Column	Value
bookmarks_oid	the ID of the object being bookmarked
bookmarks_class_id	the ID of the class the object being bookmarked belongs to
bookmarks_name	the name of the bookmark which is equal to the object or folder name, respectively
bookmarks_path	the complete path to the bookmark object in the backend for creating the link in the bookmark menu
bookmarks_id	the ID of the user having created the bookmark

4.3.4) Object Folders

If a class has been defined to be a hierarchical one on its creation, there is the possibility to create

a folder by clicking on the "Create New Folder" icon . The following screen only consists of one field, asking you for the name of the folder.

Folders in spunQ are not only useful for keeping your classes tidy, but also – and mostly – for grouping objects according to certain criteria for different usage in the frontend. In your image class, you might for example store all pictures belonging to a gallery module in one folder while images related to current events in another one. This will allow you to select the images you need by calling the frontend's `getStructureObjects()` method.

When you are inside a folder, you may change its name and enter additional information by pressing the "Edit Folder" button. You will find it in the header of the folder objects, next to the "New Class Object" icon. The additional fields you will get are the unixname and a short description. The unixname field might be helpful in the case the folder name contains special chars or spaces. If you want to create nice URLs in the frontend containing folder names, you should provide a clean unixname which can be used for that purpose.

All folder information is stored in the 'struct' table of the database. Subfolders are stored with their parent folder's ID as `struct_parent` while first-level folders of each class have 0 as `struct_parent` value set. So, you can get the complete folder structure of a class by only regarding the `struct_id` and `struct_parent` fields in this table. The ID of the class a specific folder resides in is saved as `struct_class_ref`.

The spunQ struct class itself has the ID 10 and is treated as a normal content class in terms of relation creations. This means that the information to which folder an object belongs is equally saved in the `_relation` table using 10 as '`_relation_class_ref2`' and the `struct_id` as '`_relation_object_ref2`'.

4.3.5) Class Tree View

To comfortably edit the folder structure of a class, you can use the tree view of a class.



Click on the folder which you want to move within the class structure. In the second step choose

the new parent folder by clicking the paste icon to move the folder and all containing objects to the new place.

4.4) Searching Objects

4.4.1) AJAX Live Quick Search



The Live Quick Search is the fastest way to find objects and folders in spunQ. It searches through all classes but only regards the object names. Notice that you will have to enter at least three letters for the search process to begin.

If your query returns results, you will get a list of found objects which you can navigate easily by pressing the up/down arrows on your keyboard. Press enter or click on the object to perform the main action. In the case of a folder you will jump right to the folder view, in the case of a class object to the edit object screen.

Additionally, there are edit buttons and – in the case of a class object a link to the folder where the object resides.

4.4.2) AJAX Live Class Full Text Search

When you are looking for more, you can perform a class full text search. The respective query input field is located at the top of the folder and object list in each class.

The class full text search looks up all the metadata of all objects of a class. Therefore it may be the case that an object is listed which on the first view does not contain the search term. However, be sure that somewhere in its metadata, the query string you were looking for appears...

4.4.3) Advanced Search

If you want to control in which fields your search term is contained, use spunQ's advanced search. The link to the search is located at the very top of the interface, right above the quick search field.

Click it and you will first receive a list of all classes of which you will have to choose the one you want to search.

After that, you will reach the search form where every member is listed together with an appropriate input field to build complex queries.

4.5) "Tools" Menu

4.5.1) ProjectsAdmin

On the ProjectsAdmin screen, you can define and manage your websites and applications running with spunQ. Note that it is possible to manage several spunQ projects in one backend instance.













The project name is important as the event handler in the frontend (named [project name].php) and the according Smarty templates directory depend on it.

To add a new project, just enter its name under "New Project". It will immediately appear in the project list underneath. There, you will have to enter all domains your application frontend can be reached under. This way, the preparing code of the frontend will be able to chose the right event handler by analysing the request URL.

To delete a whole project, press the corresponding delete button. All associated objects and classes will also be deleted. However, a database backup dump will be created before performing the deletion.

4.5.2) Object Statistics

In the module Object Statistics (available since spunQ 1.3), you will see several information about all registered content classes:

	sum of objects	translated	not translated	 not locked	 semi locked	 locked
 Documents	2	0	2	2	0	0
 Download	9	1	8	9	0	0
 Features	14	2	12	12	2	0
 Help Content	1	0	1	1	0	0
 Images	110	0	110	110	0	0
 Institutions	1	0	1	1	0	0
 Media	1	0	1	1	0	0
 spunq	48	33	15	33	1	14
 spunQ Versions	5	0	5	5	0	0

First the sum of contained objects, second the number of objects which has already been translated to another language (as well as those being untranslated) and a summary of the class objects' object perms (0, 1 or 2).

By clicking on a field you will get to a list of all concerned objects which you can use as initial point to edit them and perform the usual object actions (permissions, deletion, ...). You can also delete multiple objects by using the "Advanced Editing Functions " as in the class view.

4.5.3) RSS Feeds

spunQ is perfectly suited for creating RSS feeds out of existing data.

In the "RSS Feeds" screen, you will see a list of all feeds with their currently contained objects. To manually update all feeds at once, press "Generate RSS Feeds".

However, it is recommended to update the feeds every time they are called in the frontend by using the frontend's RSS class. You will do this by placing these two lines in the frontend:

```
$rss = new rss($RSS, $frontend);
$rss->createRSS();
```

4.5.3.1) How to configure RSS Feeds

For configuring the feeds, you will have to edit the \$RSS array which is defined in the backend/common/rss.setting.inc.php in the following way:

```

$RSS['feeds'][0]['title'] = "The title of your feed";
$RSS['feeds'][0]['link'] = "http://www.link.to.your.frontend";
$RSS['feeds'][0]['language'] = "en"; // or de, fr, it, ...
$RSS['feeds'][0]['description'] = "A description of your feed";
$RSS['feeds'][0]['rights'] = "If you want to set rights/licenses for your feed put it here";
$RSS['feeds'][0]['publisher'] = "The publisher of your feed – may be your frontend URL";
$RSS['feeds'][0]['creator'] = "spunQ RSS-generator";
$RSS['feeds'][0]['subject'] = "The subject(s) of your feed";
$RSS['feeds'][0]['extension'] = "xml";
$RSS['feeds'][0]['linkcode'] = "\$link = sprintf(\"http://URL/view/%s\", \$item->id);";
$RSS['feeds'][0]['img'] = "URL to feed icon";
$RSS['feeds'][0]['file'] = "rc_works";
$RSS['feeds'][0]['query'] = "
    SELECT
    object_id as id,
    [classtable]_title as title,
    [classtable]_description as abstract
    FROM [classtable], object
    WHERE object_ref =[classtable]_id AND
           object_class_ref = [classid] AND
           .....[your personal conditions].....";

```

Some comments:

- In the field 'linkcode', the URL for each feed item is created using PHP's `sprintf()` function. As a first parameter set the link including the wild card '%s' which refers to the second variable which will probably be a property of the icon object (id, title). As the `sprintf()` command will be passed to an `eval()` function, all dollars and double quotes will have to be escaped as shown in the example.
- The SQL query creates the properties of the icon object so be sure that you use the three aliases 'id', 'title' and 'abstract' for your result columns. Apart from that rule, you may select any data you want. One frequently used query structure is shown above being the selection of an object title and description with the according object id.
- Rights for your feed may be e.g. CreativeCommons licenses.
- You can define as many feeds as you want by repeating the procedure and incrementing the original [0] key.

4.5.4) DB Backup

This is the place where to manage your DB Backups. On this screen, all available backups are listed. Click on the "New Backup" icon (blue with yellow asterisk) to create a new database dump.

Notice that if you perform critical actions such as the deletion of a class or its members, the system will automatically create a backup before performing the action. These backups will also be listed on the "DB Backup" screen.

You may save storage place by selecting one or multiple of the original backups to tar them. Therefore, check the corresponding boxes, enter a file name and push the "Tar and Delete Dumps" button. Existing dump tar files will be listed at the bottom of the page.

Notice: It is possible to check/uncheck several dumps by using the green buttons (tick mark and minus) on top.

4.5.5) Preferences

The last option in the "Tools" menu is the place where to change your own personal data (including login name) and to set a new password. This menu item is available equally for all users who use the system.

4.6) "System Settings" Menu

4.6.1) DB Config

The "DB Config" is the place where all classes and members are defined. The original view will list all classes which have been defined. The possible actions in brief: To create a class, use the DB icon with the yellow asterisk in the top right corner of the screen, to edit a class just click its name and to remove a class from the system use the red cross icon.

4.6.1.1) Create Classes

On the "Create Class" screen, you will have to fill out some necessary fields:

Field	Description
table name	the name of the database table

Field	Description
class name	the name of the class as it will appear in the backend
lock	choose "public" for classes which should be visible in the interface or "private" for those only used for logical purposes
dim	"hierarchical" is used for classes which may contain folders while a "plane" class gives no possibility to group objects – they will appear as one long list (notice that a later change of this property is not possible, so you have to make an ultimate decision)
object name	the name of the first member (or column in terms of a database).
datatype	the data type of the first member
size	if the data type requires a size, use this field (for example, varchar expects a size between 1 and 255)
class type	here you can control where the class should appear in the menu
description	the description of the class will be displayed on top of the class view to help users

As a last step, you can select an appropriate icon for your new class which will represent it across the whole system. Finally, click on the "save and exit" symbol to confirm your class specifications.

4.6.1.2) Edit Classes

Once the class is created you will be redirected to the "Edit Class" screen, which you will also reach by choosing a class from the "DB-Config" main view. Here you may change the class name, its description and the position in the menu as well as the associated icon.

Furthermore, this is the place to add, edit and remove members. You will find them listed in the lower half of the page. As you will see it on many places across the system, you can here also change the order of the members to influence their order of appearance on the object edit screens.

To add a member, use the drop down box containing available data types and click on the "New member" icon next to it. This will lead you to a screen where you can set name, title and the NULL behaviour of the member. If you have chosen a relation (being of m:n or n:1 type), you will here get a selection of classes to which the relation should be build.

Important Notice: Before the class is displayed in the menu for users to create objects, you will first have to set control elements for each member in Menu->System Settings->System Groups.

4.6.2) User

In this section, you can manage the user accounts of the system. The main view shows a list of all users with corresponding edit and delete buttons. To create new users, there is again a button with a yellow asterisk.

In a first step, enter the main data (surname, first name, login, e-mail) of the user and set a password. By clicking on the save button, you will get back to the previous screen. Now the user object has been created and it is time to assign him to a system group.

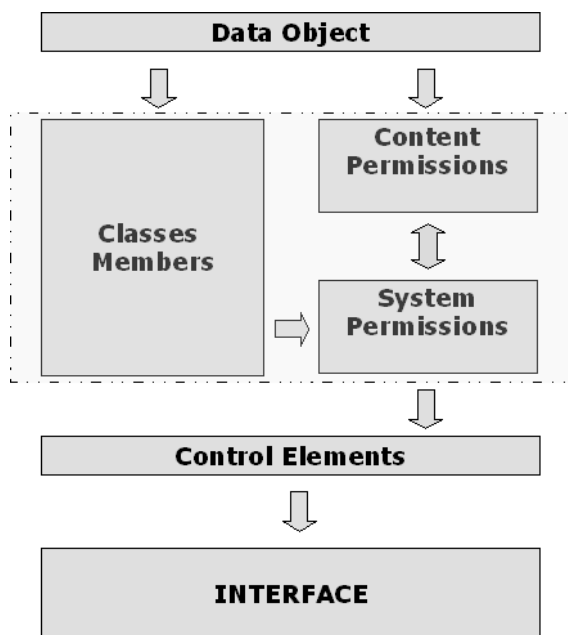
Therefore click the user's edit button to return to approximately the same screen as before with an additional select area letting you choose a system group for the user. Do this and click on the save icon to return.

Notice that you – being root – can always set a new password for any user of the system in the case he lost it by visiting one user's edit screen.

4.6.3) System Groups

This section lets you manage your system groups the users are assigned to in terms of configuring appropriate controls for each object class. Controls are these interface elements which serve to insert and modify data when editing an class object.

That's how the interface constructor works in general:



If you want to create a new system group just enter a name on top of the "System Groups" screen and press the "Add Group" button. To delete a group, use the red cross icon. The root group can not be removed.

To edit the control objects for a group use the permission icon consisting of two keys on the right. At least, you should carry out this step for the group "root" so you yourself are able to create and edit class objects in the interface.

Having chosen the group, you will receive the list of all currently defined spunQ object classes where you will have to chose one of to proceed.

Then, a screen appears to adapt the appearance of the class to the system users belonging to the initially chosen group: On top you can choose the class-wide rights for reading, writing and deleting objects. Underneath, a list of all members consisting of 6 columns appears:

Column	Description
element	the name of the current member (in square brackets the name of the corresponding table column)
read permission	check if you want to display this attribute to users
write permission	check if you want users to be able to edit this attribute
interface element	Select the element which best fits the member's data type. For relations, you will have to select the "m:n view" or the "n:1 select" element, respectively. For date fields the "date javascript" is recommended and if you want text data to be formatted by the users using a WYSIWYG editor chose "FCKeditor".
validation	Here you can select an AJAX validation which will be carried out already when leaving the associated form input field. To add personal checks, use the spunq1.3/backend/controls/check/ folder for your code.
context help	If you want type a short explanation of how data should be entered to help the users when they fill out the form.

It may be useful to select or deselect all checkboxes at once. To do this, use the two green buttons on top and on bottom of the page.

As soon as you have configured the class object interface according to your wishes, leave the screen by saving your modifications.

4.6.4) Content Groups

Additionally, you can set explicit content permissions (for reading, writing and deleting) for users using the system's "Content Groups".

Add a new group by entering its name and press the blue/green "Add Group" button. Then you can click on the group to add system users to it.

The content permissions can be set class-wide in Menu->System Settings->DB-Config by selecting a class and then pressing the accordant button at the right top of the screen symbolized with two keys.

To set content permissions for specific objects, use the object context menu in the class view and select "Rights Management". This will lead you to the same screen as above, only that the permissions will logically be set for the selected object only.

4.6.5) Controls

Control elements are the user interface parts allowing you to enter object data fitting different data types or the usability needs of less experienced user groups. Examples for controls are a simple one-row input fields, text area, document upload buttons, drop-down boxes, date selectors or m:n views.

4.6.5.1) Howto create new control elements

It might be interesting for you to add new, personal controls. So, that's the way this should be done:

First copy one control element of your choice (e.g. onerow.ctrl.php, to take the simplest one) inside the directory backend/controls/ to use it as a template. Then set the 6 containing variables to fit your needs:

Variable	Description
\$TITLE	the title of the control element
\$CALLNAME	the function being called in backend/classes/control.class.php to process the control file
\$SOURCE	the HTML code of the control element
\$ENDTAG	if an end tag is needed, write it down here (e.g. for the tag <textarea>, you need

Variable	Description
	a closing tag <code></textarea></code> after the field's content)
<code>\$VERGET</code>	PHP code ready to be passed to an <code>eval()</code> function in <code>backend/classes/control.class.php</code> . This code is used to prepare the data to be displayed in the user interface.
<code>\$VERPOST</code>	Same as above, but executed before the data is stored into the database.

If you need some more code to fulfil your wishes, use the `backend/controls/helpers` directory to store it there and include it in your control code.

When you're done, add the control element to the database by using the screen you find in Menu->System Settings->Controls. Just add a new object as you are used to it from other spunQ classes providing a name and the `*.ctrl.php` file name of your previously created control file.

4.6.5.2) How to create new AJAX checks

The AJAX checks you can associate with input fields are all stored in `backend/controls/checks`.

Once again, the best way to create a new check procedure is copying an already existing file and editing it. (The name of the new file MUST have the extension `".check.php"`!) In this type of files, only three variables are important:

Variable	Description
<code>\$inputValue</code>	This is the string you have to evaluate.
<code>\$c_return</code>	The result of the check. If the check succeeds, it has to be set to 1 at the end of the script, else to 0.
<code>\$MESSAGE</code>	This error message is displayed in the interface if the check fails.

Finally, to add a check file for selection in the "Member Permission" screen in Menu->"System Settings"->"System Groups", execute the following SQL statement in the spunQ database:

```
spunq_db=> INSERT INTO control_check (control_check_title, control_check_source) VALUES
('['TITLE]', '['FILENAME_WITHOUT_.CHECK.PHP']')
```

You can simply do this in a database management tool with graphical user interface such as phpMyAdmin or phpPgAdmin – depending on the database type you are using with spunQ. In both cases, you will have to log in to the system by providing a user with write permissions for the table `control_check` (being 'spunq_adm' per default). Additionally, you will of course have to modify the

values in square brackets to fit the properties of your new check script.

In phpMyAdmin, choose the spunQ database from the drop down list on the left. Once you get the list of tables, choose "SQL" from the top menu and paste the command above there.

In phpPgAdmin, choose "SQL" from the menu at the left top of the screen. You will then get a popup window, where you just have to select your spunQ database in the corresponding drop down field and paste the command above into the main text area.

4.6.6) System Info

The last menu option will show the output of PHP's phpinfo function which lists all relevant system variables and extensive information on all installed modules.

4.7) Personalizing the menu

The spunQ menu is created automatically using the menu class.

However, there are two places where you can personalize its appearance – the files `backend/common/config.inc.php` and `backend/common/menu.inc.php`

Menu-related variables in `backend/common/config.inc.php` are `$application["CLASSTYPES"]` (giving you control over the wording of each menu section) and `$classtypes` (an associated array responsible for the order of classes in each menu). Finally, there is the `$topmenu` variable – being also an array – in which the class types appearing in the top menu are defined.

In the second file, `backend/common/menu.inc.php`, you can edit the class-independent menu sections – by default being the "Tools", "Links" and "System settings" areas. They all are there organized in the `$mainmenu` array.

5) The spunQ Frontend

5.1) General Description

Supposing you have created all classes and objects you want in the backend it is time to face the creation of the frontend application – in the case you are using spunQ as a classical content management system.

Therefore, use the provided code and templates as a base for everything you do. This will be the fastest way to create a fully functional frontend application based on spunQ.

5.2) Frontend Generation

As the frontend uses the same database as the backend, you have all objects readily available for use in your application. Let's take a look on the code flow:

When a user visits your frontend webpage, he will be redirected to the file frontend/frontend.php via .htaccess. In frontend.php, all needed objects (such as the Smarty engine) are initiated. Another part of this file is the creation of a special \$GET array by calling the clearGet() method of the frontend class. This function splits the current URL string into different parameters and pushes them into the array. Per default, a spunQ frontend URL fits the following schema, which simply can be changed by editing the clearGet() method:

```
http://www.yourdomain.com/action/class_id/parent/oid/version/page/errorcode
```

(Of course, not all parameters have to exist.)

Afterwards, the main event handler is included, where the whole frontend is controlled. This file is named [your_project_name].php. The one which belongs to the spunQ website and is placed in the tarball as an example therefore is the file frontend/spunq.php.

The event handler mainly consists of a switch/case construction which takes the \$GET['action'] as argument. This means that you will have to place your start page actions in the "default" branch of the switch/case structure as at this point no action will be defined.

Most methods you will need to get content out of the database can be found in the built-in frontend class (frontend/classes/frontend.class.php). The most used ones are getTextById() which returns the metadata of an object and getStructureObjects() which returns the objects existing within a folder. (For detailed descriptions of most frontend methods, have a look at chapter 5.4.)

After the switch/case construction, some general Smarty variables such as the POST array or the current URL are defined via `$smarty->assign("NAME", $value)`.

Finally, `$smarty->display("index.tpl")` is called to generate the output to the client.

5.3) Template Structure

The templates of the frontend resides in the `frontend/templates/[project name]` directory. There should be one main template (e.g. `index.tpl`) including the basic layout (menus, toolbars, css inclusion, etc.) which at some part include the content template which fits the current page. In smarty, this happens by setting a `{include file=$coretpl}` tag.

Therefore, you will have to define the `$coretpl` variable for each page in the according case inside the event handler.

For further information on Smarty language elements please refer to chapter 6.2 of this document.

5.4) Useful Methods of the Frontend Class

5.4.1) clearGet()

Parameters	none
Return Value	(array) <code>\$frontend->get</code> // the spunQ GET array

This method will create the spunQ GET Array by splitting the URL according to the following pattern:

`http://your.frontend.domain/action/class_id/parent/oid/version/page/errorcode`

What you will get is an associated array containing those seven parameters. Feel free to modify their naming to fit your needs.

5.4.2) debug()

Parameters	(string) \$data // the variable to debug (string) \$title // the title of the variable which will appear in the debug file (int) \$pre // whether or not to use the HTML <pre> tag in the output (0 or 1)
Return Value	(void)

The debug method of the spunQ frontend is very useful during development! It will append the variable to debug in the file defined as APP_DEBUGLOCATION in frontend/common/config.inc.php. This way, it won't touch your frontend output which might cause "headers already sent" errors or destroy the whole appearance of your application.

On Unix/FreeBSD systems, you can easily control the output by typing:

```
$ tail -f [APP_DEBUGLOCATION]
```

This will continuously show the new output produced by the debug() method.

5.4.3) doSearch()

Parameters	(string) \$query // the search term (array) \$searchclasses // IDs of all classes to search within
Return Value	(array) \$result // the search results

This method allows you to easily implement a full text search in your application. It will go through all the members and check whether the given search term appear somewhere within the member data.

If several terms separated by a space are given, all those terms will be conjuncted by an OR clause during each member's check in the SQL statement. So, if a user looks for "foo bar", the return value of the doSearch() method will be an array of objects which contain either "foo" or "bar" in one or many of their metadata fields.

The \$searchclasses array is defined in frontend/common/config.inc.php being an array of class IDs according to the content classes you want to search through, e.g. array(92, 125).

5.4.4) getClassStructure()

Parameters	(int) \$class_id // the class id (int) \$parent // the id of the parent folder (int) \$limit // to limit the number of results
Return Value	mixed: 1. (void) 2. (array) \$structure // the folder structure

If you want to know which folders a class main folder contains, use this method with 0 as value for \$parent. In another case, you may have a list of all subfolders of a given folder returned. Then set \$parent to the accordant folder ID.

The method will return an array of all found folders along with their object data.

5.4.5) getCookieValues()

Parameters	none
Return Value	(array) \$this->cookie // the current cookie values

The getCookieValues() method will provide an array with the current language and CSS style size set on the client side. Therefore, it will split the value set in the client cookie by the writeSessionValues() method.

For your project, you will probably want to alter the values which are saved in the cookie. Therefore, edit this method according to your needs as well as writeSessionValues().

5.4.6) getLang()

Parameters	(array) \$versions // the spunQ version array
Return Value	(string) \$language // language code ('en', 'de', ...)

If you ever need the abbreviation of the current interface language, use getLang(). It will take the \$version array as an argument as it has been defined in frontend/common/config.inc.php. This variable might look like this:

```
array (
  1 => 'en',
  0 => 'de'
)
```

For example, this method is used to choose the appropriate Smarty language config file, being 'en.wording' or 'de.wording' in frontend/templates/[projectname].

5.4.7) getParentFolder()

Parameters	(int) \$parent // the ID of the parent folder (int) \$class_id // the needed class ID
Return Value	(object) // the PEAR DB result object

getParentFolder() will give you all information about a folder as it is stored in the struct table (name, shortcut, parent, abstract). Typically used to get information about a object's parent folder, it can actually be used for all spunQ folders by providing the according struct_id as first parameter.

5.4.8) getPureObject()

Parameters	(int) \$oid // the object ID (int) \$ref // the object reference (int) \$classid // the class ID
Return Value	(object) \$object // the PEAR DB result object

This method will return all object information out of the database table 'object', e.g. object_name, object_filename, object_parent, object_version, object_created, ...

If the given \$oid is not 0, the object_id will be taken for the WHERE clause of the SQL statement, else the \$ref parameter will be regarded. In the latter case, the object_ref in the database will be compared. The object reference is the ID a object is represented by in its according class table.

The class ID is a necessary parameter in both cases.

5.4.9) getRelatedObjects()

Parameters	(int) \$class_ref1 // ID of class the target object resides in (int) \$class_ref2 // ID of class in which to search for relations (int) \$object_ref1 // the target object ID (int) \$limit // if given, the result will be limited to this number of objects
Return Value	mixed: 1. (array) \$object // array containing all found objects with their metadata 2. (boolean) FALSE

This is a very important spunQ core method to look for relations of an object out of the `_relation` table. You will often need it, for example to get all images related to a specific text content or similar cases.

You may also search for objects a folder contains. In this case, you will have to use the value 10 as `$class_ref1` and the according `struct_id` as `$object_ref1`.

The return array consists of the found objects and will include all their metadata – ready for use in your application.

5.4.10) getRelatedObjectsReverse()

Parameters	(int) \$class_ref1 // ID of class in which to search for relations (int) \$class_ref2 // ID of class the target object resides in (int) \$object_ref2 // the target object's ID (string) \$class_table1 // the table name of the class in which to search for relations (string) \$sort_field // the field to use for the SQL SORT clause
Return Value	(array) \$result // array containing all found objects with their metadata

`getRelatedObjectsReverse()` is similar to `getRelatedObjects()`, but works the other way round. So, reusing the example above, it would get all text objects an image is associated with.

Assuming 94 as image class id and 123 as text class id, to get both types of relations, use the following two lines in your application:

```
$related_img = $frontend->getRelatedObjects(94, 123, [id of text object]);
$related_text = $frontend->getRelatedObjectsReverse(94, 123, [id of image object]);
```

Additionally, if the name of the class table to find related objects in (as `$class_table1`) and a sort field is given, the result will be sorted according to these criteria.

5.4.11) `getStructureObjects()`

Parameters	(int) <code>\$class</code> // the class id (int) <code>\$folder</code> // the folder id (int) <code>\$limit</code> // if given, the result will be limited to this number of objects
Return Value	(array) <code>\$items</code> // found objects together with their metadata

This method will give you all objects a specified folder specified by its ID and the class it resides in. Once again, their metadata will also be available as `getTextById()` is called within the function.

5.4.12) `getStructureObjectsPure()`

Parameters	(int) <code>\$class</code> // the class id (int) <code>\$folder</code> // the folder id (int) <code>\$limit</code> // if given, the result will be limited to this number of objects
Return Value	(array) <code>\$result</code> // the PEAR DB result set

`getStructureObjectsPure()` is nearly the same as `getStructureObjects()` with the difference that it will get only the object information stored in the object table and omit the saved object metadata.

Use this method for example if you only need the object IDs and names for a tabular list of objects. In this case, it will be much faster than calling `getStructureObjects()`.

5.4.13) `getTableName()`

Parameters	(int) <code>\$classid</code> // the class id
Return Value	(string) // the table's name

This is a short but useful one. It will get you the name of a class table by a given class id for usage in SQL statements.

5.4.14) getTextById()

Parameters	(int) \$id // object id (int) \$classid // class id (int) \$rel // whether relations are returned or not
Return Value	(array) \$item // all metadata of the given object

This is – together with getStructureObjects() – usually the most used frontend method in any spunQ frontend application. It will fetch all metadata of a object identified by its ID and the ID of the class it belongs to.

The third parameter (\$rel) is per default set to 1. This way, the method will call getRelatedObjects() to return all related objects along with their metadata as well. If you need this, you may omit \$rel as it has its default value, but if you can do without it set it to 0 when calling the method. You may imagine that the method will speed up in the latter case.

5.4.15) insertObjectData()

Parameters	none
Return Value	mixed: 1. (boolean) TRUE 2. (string) 'ERROR'

The insertObjectData() method has originally been part of the spunQ backend only. It serves the purpose of inserting a new content class object into the database.

It takes no parameters as it will use the frontend class variables to get all needed data from. Mainly, it requires \$this->clasdata and the \$this->post array.

First, it will check the rights applying to the logged in user and will then insert all given fields – probably originating from a HTML form – into the class table. The name of the HTML input field will have to match the columns in the according class table for this to work.

Once the class objects has been inserted into the database, the script will determine the ID of the just inserted object in the class table. With this information – being used as the object_ref – the object is also inserted into the main 'object' table to complete the process.

5.4.16) insertPureObject()

Parameters	(int) \$class_ref // the class id (int) \$object_ref // the object's id in the class table (string) \$name // the object's name (int) \$user // the ID of the user having created the object (int) \$group // the ID of the group this user belongs to (int) \$perm // the perm level for this object (ranging from 0 to 2) (int) \$sort // the sort value for this object
Return Value	(object) // the PEAR DB result object

This method inserts an object in the object table only. This means that its metadata should already exist in the according class table. Most of the parameters are self-explaining. However, two interesting ones are \$perm and \$sort.

The first one controls the general permission level (per default being 0, 1 or 2) of this object influencing whether it is displayed or not in the frontend. So, if you want to have newly added content checked in the backend by a kind of moderator before showing it to the world, it is a good idea to initially set \$perm to 0. If you wish to have the new data being displayed in the application immediately after its creation, change its value to 2.

The second one (\$sort) usually equals the object_id. This guarantees that new objects are displayed on top of any lists as all spunQ sorting functions regarding this field will return the results in descending order.

5.4.17) saveObject()

Parameters	(array) \$rights // general rights for object save action
Return Value	(int) \$this->post['structid'] // folder id for displaying the interface after this action

This method will check whether a given object exists (by looking at the 'objectid' field of the \$this->post array) and call either insertObjectData() or updateObjectData() to add the given data to the database.

A class id will in both cases have to be provided (probably as a hidden form field) as \$this->post['classid'].

5.4.18) switchLanguage()

Parameters	(string) \$lang // language code ('en', 'de', ...) (array) \$version // the spunQ version array
Return Value	(string) \$reload // URL for redirect

switchLanguage() is called from within the frontend/frontend.php script in case the get parameter 'lang' is set in the current URL. It will create a client cookie based on the given value thus the value of the selected language will be available for the whole visit (but only if the user accept the cookie to be set in his browser).

5.4.19) switchStyle()

Parameters	(int) \$size // size ID
Return Value	(string) \$reload // URL for redirect

This method is similar to the previous one – switchLanguage() – only that it will set the selected CSS style size ID in the client cookie.

Bases on this ID the appropriate style sheet may be included in the Smarty main template by using this pattern:

```
{if $COOKIE.size == '2'}
  <link rel="stylesheet" href="css_2.css" type="text/css" />
{/if}
```

5.4.20) updateObjectData()

Parameters	none
Return Value	mixed: 1. (object) // the PEAR DB result object 2. (string) 'ERROR'

The updateObjectData() works similarly to the insertObjectData() method only that it updates an already existing object. It also takes no parameters as it will use the frontend class variables to get all needed data from.

5.4.21) updatePureObject()

Parameters	(int) \$object_id // the object id (int) \$class_ref // the class id (int) \$object_ref // the object's id in the class table (string) \$name // the object's name (int) \$user // the ID of the user having created the object (int) \$group // the ID of the group this user belongs to (int) \$perm // the perm level for this object (ranging from 0 to 2) (int) \$sort // the sort value for this object
Return Value	(object) // the PEAR DB result object

In the same way updateObjectData() relates to insertObjectData(), updateObject() relates to insertObject(). It will update an object in the object table only.

The parameters are exactly the same only that it will of course need the object ID to know which object has to be updated.

5.4.22) writeSessionValues()

Parameters	(int) \$langid // the ID of the currently selected language (int) \$size // the size ID used to select the corresponding CSS stylesheet
Return Value	(void)

This method is used to set the given values in the client cookie. This is done by creating a string of the parameters separated by colons. If you need to retrieve stored information, use getCookieValues() – the counterpart of this method.

Maybe you need some additional data saved on the client side for your spunQ frontend application. In this case, feel free to add parameters to writeSessionValues() but remember to modify the cookie array creation procedure in getCookieValues() as well.

6) PEAR and Smarty

6.1) PEAR Basics

spunQ uses the PHP Extension and Application repository PEAR to create an abstracted database layer. The needed object – in spunQ called `$db` – is created in `backend/common/prepare.inc.php` as well as in `frontend/common/globals.inc.php`.

It appears in almost every class to process database queries. In the frontend and other classes the database object is directly passed on to, you call the PEAR object by `$frontend->db`, in a class which takes the frontend itself as parameter you will have to use `$class->frontend->db` to reach the object.

To make a query, you mainly have to choose from four useful methods: `getAll()`, `getOne()`, `getRow()` and `getCol()`.

The full documentation of this PEAR package is available on: <http://pear.php.net/package/DB/docs>

6.1.1) getAll()

The most comprehensive function to use is `getAll()` which returns a result set consisting of all columns and rows.

Example:

```
$frontend->db->getAll("SELECT object_name, object_id FROM object") returns
```

```
Array (
  [0] => Array (
    [0] => "Object Name 1"
    [1] => 1
  )
  [1] => Array (
    [1] => "Object Name 2"
    [2] => 2
  )
)
```

6.1.2) `getOne()`

Another useful one is `getOne()` which returns the data from the first column of the first row. To avoid misunderstandings, it is recommended to build the query a priori in a way that it only will return one data field.

Example:

```
$frontend->db->getOne("SELECT object_name FROM object LIMIT 1") returns "Object Name 1"
```

This method may for example be the one to choose in case you want to know the last created ID after having performed an INSERT query. For this purpose use:

```
$frontend->db->getOne("SELECT object_id FROM object ORDER BY object_id DESC LIMIT 1")
```

6.1.3) `getRow()`

If you want to get the data from only the first row of the result set, you will have to use `getRow()`.

Example:

```
$frontend->db->getRow("SELECT object_name, object_id FROM object") returns
```

```
Array (  
  [0] => Array (  
    [0] => "Object Name 1"  
    [1] => 1  
  )  
)
```

6.1.4) `getCol()`

The other way round is also available: If you want to get the data only from the first column, use `getCol()`.

Example:

```
$frontend->db->getCol("SELECT object_name, object_id FROM object"); returns
```

```
Array (  
[0] => Array (  
  [0] => "Object Name 1"  
  [1] => "Object Name 2"  
)  
)
```

6.1.5) Fetch Modes

As an additional parameter for `getAll()` and `getRow()` you may indicate a so called fetch mode. By doing this, you are able to influence the way results are returned. For the following examples, always the method `getRow()` is used, however the behaviour will be the same when using `getAll()`.

6.1.5.1) DB_FETCHMODE_ORDERED

This is PEAR's default fetch mode, so it will not have to be set – and it is not very convenient to use anyway. It will return the result as an ordered array.

Example:

```
$frontend->db->getRow("SELECT object_name, object_id FROM object",  
DB_FETCHMODE_ORDERED) returns
```

```
Array (  
[0] => Array (  
  [0] => "Object Name 1"  
  [1] => 1  
)  
)
```

6.1.5.2) DB_FETCHMODE_ASSOC

This fetch mode is the one to use if you want an associated array to be returned.

Example:

```
$frontend->db->getRow("SELECT object_name, object_id FROM object, DB_FETCHMODE_ASSOC")  
returns
```

```
Array (  
[0] => Array (  
  ['object_name'] => "Object Name 1"  
  ['object_id'] => 1  
)  
)
```

6.1.5.3) DB_FETCHMODE_OBJECT

For the use within classes, it may also be interesting to have each result row available as an object. DB_FETCHMODE_OBJECT does the job.

Example:

```
$frontend->db->getRow("SELECT object_name, object_id FROM object",  
DB_FETCHMODE_OBJECT) returns
```

```
Array (  
[0] => stdClass Object (  
  [object_name] => "Object Name 1"  
  [object_id] => 1  
)  
)
```

6.1.6) Handling database results

To process database results in your PHP code, best use a foreach loop. For example, to create an array of all German objects (supposing that German is defined as language with id 1 in the spunQ system configuration) use the following code snippet:

```
$result = $frontend->db->getRow("SELECT * FROM object, DB_FETCHMODE_ASSOC");  
  
$german_objects = array();  
foreach ($result as $res) {  
  if ($res['object_version'] == 1) {  
    $german_objects[] = $res;  
  }  
}
```

6.2) Smarty Basics

Smarty is very comfortable to display PHP variables in pre-defined page templates. It is a powerful tool having many various functions implemented. Describing all of them would go far beyond the scope of this documentation, but some very common features will now be explained. For further documentation, please visit <http://smarty.php.net/manual>.

6.2.1) Assigning Variables

To use PHP variables, you will have to assign them first to Smarty.

You do this simply by writing `$smarty->assign('foo', $bar)` in your PHP code. Afterwards, you can display it in Smarty by putting `{foo}` in your template. The content of `$bar` will then be displayed at this place.

6.2.2) Displaying Templates

Once you have all variables assigned you need, the last thing to do will be calling the template in the PHP script. This can be done simply by writing `$smarty->display('index.tpl')` – in the case `index.tpl` is the template you want to be displayed.

6.2.3) Using language variables

A very convenient part of Smarty is its behaviour when it comes to multilingual applications. Therefore, so called config files will have to be prepared. In the spunQ frontend, these files can be found in `frontend/templates/[your_project_name]`. The files in question have the extension `.wording`, with the abbreviation of the language as prefix. The two default language configs in the spunQ therefore are called `en.wording` and `de.wording`.

The use of these files is simple – a part of them could be:

en.wording	de.wording
db = database	db = Datenbank
query = query	query = Abfrage

To use them in a template, use following syntax:

```
<h2>{#db#} {#query#}</h2>
```

This will create a header containing the string “database query” or “Datenbank Abfrage”, depending on which frontend language is the active one. (The current language config to load will automatically be chosen by frontend/frontend.php.)

6.2.4) Arrays and Objects

To display content of an PHP array or object, you will have to use the following syntax:

Assuming you create these PHP arrays:

```
$arr1 = array('text 1', 'text 2');  
$arr2 = array('key1' => 'text 3', 'key2' => 'text 4');  
$smarty->assign('arr1', $arr1);  
$smarty->assign('arr2', $arr2);
```

In Smarty, you can then access their contents by typing:

```
{$arr1.0} {$arr1.1} <br/> {$arr2.key1} {$arr2.key2}
```

The output would then be:

```
text 1 text 2  
text 3 text 4
```

The handling of objects is similar, just use an arrow (->) instead of the point. So, to display the PHP object attribute \$item->name in Smarty, just put the variable into curly brackets: {\$item->name}

6.2.5) Using Conditions

You can use conditional clauses in Smarty just like you would do it in PHP, only that the syntax is quite different, for example:

```
{if $sum > 0}  
  Do something.  
{/if}
```

It is also possible to write if-/else-constructs:

```
{if $sum > 0}
  Do the one thing.
{else}
  Do the other thing.
{/if}
```

6.2.6) Using Loops

If you have assigned a PHP variable being an array to Smarty, you may probably want to display all of its content.

Let's see how we could manage to display a list of spunQ objects. In the case the variable \$result being a PEAR DB query result set having used DB_FETCHMODE_OBJECT and assigned to Smarty as "objects", the loop will look like this:

```
{foreach from=$objects item=o}
  {o->object_name} has ID {o->object_id}<br/>
{/foreach}
```

There are mainly two parameters, being "from" (the array to use) and "item" (the variable representing one array element within an iteration).

6.2.7) Using Smarty Modifiers

When you are looking through the templates of spunQ's example frontend, you may find strange constructions like the following:

```
{${DOWNLOADS.0.object->object_created|date_format:"%d. %b. %Y"}}
```

This is how Smarty modifiers are called. Modifiers are pieces of PHP code which Smarty variables are passed on to. The according functions will modify the output (for example by producing a pre-defined date/time output as it is the case in the example above) and return it directly to Smarty.

There are many built-in modifiers (e.g. escape, date_format, nl2br, ...), however it is possible to define your own by editing the ssmarty.class.php in the classes directory of the frontend or the backend, respectively.

For example, we wrote a modifier to automatically transform any e-mail addresses in a text to appropriate links:

```
function smarty_modifier_createEmail($item) {
    return preg_replace('!(\S+)@([a-zA-Z0-9\.\-]+\.[a-zA-Z]{2,3}|[0-9]{1,3})!', "<a
    href=\"mailto:$1@$2\">$1 (at) $2</a>", $item);
} // function
```

The first parameter – being `$item` in this case – is always the smarty variable data the method will be applied on. Any further parameters you need in the PHP function will have to be separated by ':' in the call within the Smarty template.

After you have written a similar function for your needs, you will have to register it so Smarty knows the way you want to call it from the templates. You will do that in the `init_smarty` method by typing – in the case above:

```
$this->register_modifier("createEmail", "smarty_modifier_createEmail");
```

7) System Architecture and Design

7.1) Filestructure

This is a short introduction into the spunQ filesystem. Please refer to the CodeDocs for more information about each file and its usage. (<http://spunq.com/codedoc/>)

7.1.1) The spunQ Backend

Path	Description
spunq1.3/backend/	path into the backend. index.php as eventhandler
spunq1.3/backend/action/	files called as additional events, such as addrelation.php
spunq1.3/backend/backup_files	storage for spunQ backups created by class changes or the backup interface
spunq1.3/backend/classes	general domain classes for spunQ
spunq1.3/backend/common/	configuration for the spunQ backend, menuhandling and rss-feeds; creates all needed objects for spunQ runtime
spunq1.3/backend/controls	sources for control elements
spunq1.3/backend/controls/calendar	special control for javascript calendar item
spunq1.3/backend/controls/check/	files for ajax checks of the control elements
spunq1.3/backend/db/	databases for installation and permission settings for postgresSQL
spunq1.3/backend/doc/	documentation documents
spunq1.3/backend/logs/	log files for spunQ log and debug log
spunq1.3/backend/modules	additional modules for spunQ backend all modules includes the structure of spunQ: /event.php /classes/ /templates/ see documentation about modules for more information
spunq1.3/backend/templates/	all template files. smarty-templates, wording files and images
spunq1.3/backend/templates_c/	compiled smarty templates
spunq1.3/backend/tools/	different tools for spunQ administration. this files

Path	Description
	are not called by the interface. they can be called as command line interface. the use /tools/common/config.cli.php for configuration.
spunq1.3/backend/upload/	storage for uploaded files in the backend itself (note: most project use a storage folder in the frontend area)

7.1.2) The spunQ Frontend

Path	Description
spunq1.3/frontend/	path into the frontend. PROJECT.php as eventhandler
spunq1.3/frontend/classes/	all domain classes for frontend operations
spunq1.3/frontend/common/	configuration for the frontend application
spunq1.3/frontend/document/	upload storage for different documents
spunq1.3/frontend/image/	upload storage for images and all rendered files.
spunq1.3/frontend/logs/	log files for the frontend debugging
spunq1.3/frontend/rss/	rss files storage created by the backend
spunq1.3/frontend/templates/	all template files for the frontend application
spunq1.3/frontend/templates_c/	compiled frontend templates

7.1.3) Additional Files

Directory	Description
spunq1.3/backend/FCKeditor	Files for the fckeditor called by control elements
spunq1.3/smarty/	the smarty template engine. used for backend and frontend
spunq1.3/update/	update scripts. use the svn/cvs storage of spunQ and upgrades the hole environment. (handle with care!)

7.2) DB Structure

Table	Description
_message	Smarty config variables of messages and errors
_relation	m:n relations between two objects
bookmarks	bookmark entries of all users
c_group	content group information
class	all class metadata
class_perm	class-wide permissions
control	control elements
control_check	check functions for control elements (form validation)
datatype	available DB datatypes
db_group	information about system groups
db_user	user login data
member	class members
member_perm	member permissions
object	main object table
object_adds	additional data for objects (comments, etc.)
object_perm	permissions for every single object
projects	projects and vhosts
struct	folder elements
userdata	user metadata