

Globule Broker System Overview

Wilfred Dittmer

Version 1.0

<http://www.globule.org/>

Contents

1	Introduction	3
2	Globule Broker Concepts	3
2.1	Users	3
2.2	Groups	3
2.3	Sites	3
2.4	Servers	3
3	Assigning Access Rights	3
4	Limiting Resources	4
5	Document Locations on Disk	4
5.1	Origin Server Documents	5
5.1.1	Origin with DNS redirection	5
5.1.2	Origin on '/'	5
5.1.3	Origin on '/export/'	5
5.2	Replica & Backup Server Documents	5
5.3	Design decisions	5
6	Globule Broker Website	6
6.1	Structure of the Pages	6
6.1.1	User Tracking	6
6.1.2	Database Connection	6
6.1.3	Handle Action	6
6.1.4	Generate Page	7
6.2	Action	7
6.2.1	Login	7
6.2.2	New User	8
6.2.3	Express	8
6.2.4	Personal	9
6.2.5	Access Rights	10
6.2.6	Overview	11
6.2.7	Servers	11
6.2.8	Sites	12
6.2.9	Config	13
6.3	Cascading Style Sheet	13
6.4	Tool Tips	13

7	Database	14
7.1	Entity Types	14
7.1.1	Users	14
7.1.2	Groups	15
7.1.3	Servers	15
7.1.4	Sites	16
7.2	Relationships	16
7.2.1	Not Available	16
7.2.2	Member Of	17
7.2.3	Assigned Rights to Users	17
7.2.4	Assigned Rights To Groups	18
7.2.5	Has Replica	18
7.2.6	Has Backup	18
7.2.7	Has Redirector	19
7.2.8	Has Replica Qouta	19
7.2.9	Has Backup Quota	19
8	GBS, Database Interaction	20
8.1	Assigning Replicas & Backups by the GBS	20
8.2	Removing Replicas & Backups	21
8.3	Dirty Bit	21
8.4	Downloading Config Without User Interaction	22

1 Introduction

The goal of the Globule Broker System (GBS) is to be a meeting place for Globule users such that they can share resources in an easy way. The GBS will also provide information to the users about where their site is being replicated and who is using their servers. The GBS is able to generate Apache configurations files for users of this system.

To show the features of the GBS, we will provide hosting services in the form of (sub)domain names, DNS redirection and providing replica resources on our servers. *Comment: In the future we will remove this capability, hopefully by then others will provide domain names and DNS redirection.*

2 Globule Broker Concepts

There are four main concepts in the GBS: Users, Groups, Sites and Servers. We will go into details of each of them in the following sections.

2.1 Users

The user of the GBS will have a username, first name, last name, password and an e-mail address for identification. The user will be in charge of at least one server running Globule.

2.2 Groups

Groups are defined as a list of users under a certain name. There are two kinds of groups, private and public. Private groups are created by a user and are only visible to this user. Public groups are also created by a user, but the group is visible to all users of the GBS. The name will have the owner associated with it so it will be unique (e.g. <username>:<group name>).

With private groups, the user can add and remove anyone he wants, without needing their permission to do so. The user may change his private group to a public group.

With public groups, only the owner of the group can add and remove anyone he wants, without needing their permission to do so. Other users can request the owner to be added to or to be removed from the public group. The owner of the public group may decide to make the public group private.

2.3 Sites

A site consists of an origin server containing the documents of the site, zero or more replica servers, zero or more backup servers and zero or one DNS redirector.

2.4 Servers

A server is a machine, owned by a user, running Globule. The server can host origins, replicas, backups and/or provide DNS redirection. The server has finite resources like storage and bandwidth.

3 Assigning Access Rights

In order to easily assign access rights to users and groups, we maintain a list of users and groups for each user. For each entry in this list, the user can specify if he

will:

Resource	Options
Host replicas for this entry	Yes, No
Host backups for this entry	Yes, No
Host redirectors for this entry	Yes, No

Where *Yes* means to always host a replica, backup or redirector for that user or group without asking. Resources permitting of course. *No* means never host a replica, backup or redirector for that user or group.

By default this list will contain:

User name or Group	Host Replica for	Host Backup for	Host Redirector for
The user himself	Yes	Yes	Yes
<i>Globule Team</i>	Yes	No	No
<i>Default Action</i>	No	No	No

The user himself will be shown by his username. The *Globule Team* is a public group defined by us. The *Default Action* is there to define a default action to be taken for all users and groups not in this list.

The user will always be able to host his own replicas, backups or use its own redirector on his own servers. The *Globule Team* is allowed to host a replica on any of the servers of this user. By default, the user is asked if he wants to host a replica or backup or provide DNS redirection for a user not in this list.

The user can add or remove users (except himself), private groups or public groups (except *Default Action*) to or from this list. The user may change the permission options for all entries in this list.

These access rights will be applied to all the user's servers.

4 Limiting Resources

A user can define per server, the total amount of disk space available for hosting replicas. The user then specifies the number of replicas.

Backups are more difficult to limit in their resources as backups take as much space as they would need. At the moment we will let the user only specify the number of backups available on this server. By default this will be zero.

5 Document Locations on Disk

We will now explain where the documents of the origins, replicas and backups will be located on the user's server. We assume that all the servers have a `<ServerRoot>` of `/etc/httpd/` and a `<DocumentRoot>` of `/var/www/html/`.

We define the following:

<code><ServerRoot></code>	<code>/etc/httpd/</code>
<code><DocumentRoot></code>	<code>/var/www/html/</code>
<code><server name></code>	the real name of the server. E.g. <code>wilfred.cs.vu.nl</code>
<code><path></code>	web directory of the site. E.g. <code>'/'</code> or <code>'/export/'</code>
<code><DNS name></code>	name of server using a DNS redirector. E.g. <code>wilfred.globeworld.net</code>
<code><site name></code>	concatenation of <code><DNS name></code> and <code><path></code> or <code><server name></code> and <code><path></code> if no DNS redirector is used. E.g. <code>wilfred.globeworld.net/</code> or <code>wilfred.cs.vu.nl/export/</code>
<code><SiteRoot></code>	<code><DocumentRoot></code> but then one level up, appended with <code><site name></code> . E.g. <code>/var/www/wilfred.globeworld.net/</code>
<code><SiteDocumentRoot></code>	<code><SiteRoot></code> appended with <code>/htdocs/</code> E.g. <code>/var/www/wilfred.globeworld.net/htdocs/</code>

5.1 Origin Server Documents

We will split this up in 3 scenarios. The first scenario is when the origin site uses DNS redirection. The last two scenarios describe when the origin site does not use DNS redirection.

5.1.1 Origin with DNS redirection

If DNS redirection is used, the site has its own `<SiteRoot>` and `<SiteDocumentRoot>` as defined above. Using the example above, a document called `a.html` in the `<SiteDocumentRoot>` can be accessed via the web as: `http://wilfred.globeworld.net/a.html` and is stored as `/var/www/wilfred.globeworld.net/htdocs/a.html`.

5.1.2 Origin on '/'

This time we do not use a DNS redirector. We will use the `<DocumentRoot>` of the server. A document `a.html` in the `<DocumentRoot>` can be accessed via the web as: `http://wilfred.cs.vu.nl/a.html` and is stored as `/var/www/html/a.html`.

5.1.3 Origin on '/export/'

Again no DNS redirector is used and the `<path>` is now `/export/`. This means that a document `b.html` can be accessed via the web as: `http://wilfred.cs.vu.nl/export/b.html` and is stored as `/var/www/html/export/b.html`.

5.2 Replica & Backup Server Documents

If the origin server uses a DNS redirector, then the documents will be stored in their `<SiteDocumentRoot>` and accessed in the same way as you would access the origin.

If the origin server does not use a DNS redirector, then the documents are stored in the `<DocumentRoot>` appended with the `<site name>`. E.g. for the origin `'/export/'`, its documents are stored in `/var/www/html/wilfred.cs.vu.nl/export/` and accessed via:

`http://replica.cs.vu.nl/wilfred.cs.vu.nl/export/`.

5.3 Design decisions

We chose this way of directory naming to minimize the overlap in directories belonging to sites. If two sites use the same `<path>`, you want to make sure these are kept separate in any circumstance.

This model does mean there are some situations that are not handled correctly. If the user creates a site, not using a DNS redirector and export on `'/'`, then we can not allow this server to be a replica for other sites that also do not use a DNS redirector. The reason is that the replica's documents are stored in a subdirectory of the origin's export path. This means that the origins starts to export the replica's files as if they were his own.

Another drawback is that we have a `<SiteRoot>` for every site that uses DNS redirection. Apache does not want to start if these directories do not exist. But, we have written scripts to automatically create these directories when the server is restarted.

6 Globule Broker Website

To manage and access the users, groups, servers and sites, we build a website which provides access to a MySQL database. The pages are generated using PHP and the layout is handled by a Cascading Style Sheet.

6.1 Structure of the Pages

To keep the URL clean and the same for all pages, we always need to go through *index.php* and use POST requests.

When a request comes in, it goes through *index.php*. Before anything is done or generated there is a user tracking method called (**handleTracking()**). After this a connection is made to the database (**handleDatabase()**). When the database connection is established it is time to look at what the user wants (**handleAction()**). And finally, we generate a new webpage for the user (**generatePage()**). These methods can be found in *main.php*.

We will now take a closer look at these methods.

6.1.1 User Tracking

To keep track of a user during a session, we set a cookie once the user has logged in with a username and password. The cookie stores a session identifier, which allows us to retrieve information about this user, like userID or email address. When a user logs out, the cookie and tracking information is removed.

Session support is enabled in PHP by default. Settings for the cookie and tracking things can be found in *constants.php*.

6.1.2 Database Connection

To access the MySQL database, **handleDatabase()** is called. It connects to the MySQL server as denoted by the constants DBH, DBU, DBP (see *constants.php*) and then selects the database for our usage, DBN. After a successful connection, we lock *ALL* the tables in the database to give us exclusive access. The lock is automatically released and the connection automatically closed when we are done.

6.1.3 Handle Action

The POST request will always contain the *action* variable, except when accessing the webpage for the first time (it will then default to go to the login page). The *action* variable tells us what we need to do and delegates this to a number of methods. Below is a table with the available options, their corresponding methods and in which file the method can be found.

action value	method	file
login	loginAction()	login.php
newuser	newUserAction()	newuser.php
express	expressAction()	express.php
overview	overviewAction()	overview.php
site	siteAction()	site.php
server	serverAction()	server.php
personal	personalAction()	personal.php
config	configAction()	config.php
rights	rightsAction()	rights.php

The described methods will then further refine the action to call the specific methods in the file. These methods can modify the action before it is used with **generatePage()**.

6.1.4 Generate Page

The GBS site is laid out in five main parts:

1. Page Head, name of the site.
2. Title, name of the site and location in the pages.
3. Login Status, user name and log out button.
4. Menu/Left Options
5. Main Page

Depending on the *action*, the Title, Menu and Main Page are generated accordingly, just like in Handle Action. The methods are *actionGetTitle()*, *actionGenerateLeftOptions()*, *actionGenerateMainPage()*. We will go into more detail of the actions below.

6.2 Action

The action determines which methods are called. We will show per action which methods are available to it.

6.2.1 Login

The *action* is further subdivided, as shown in the paragraphs below. Login handles everything to do with logging in or out of the website.

main This generates the main login page. It has two text fields for user name and password and a submit button.

register This generates the register page. This page has text fields for user name, first name, insertion, last name and e-mail address.

forgot This generates the forgot password page. It has a textfield in which the user can fill in his or hers user name.

do-login This processes the username and password that the user filled in. The user name could be used as a DNS name, so it must comply with RFC 1035. The user is informed if the user name is incorrect.

Next, we check that a password is filled in and try to retrieve the user's data with the given user name and the SHA1 of the given password. It should result in exactly one database line. If there are no results from the database, the user is informed that the user name or password is incorrect.

If the user is known and the password correct, we start a new session for this user by setting a cookie. The user's personal data is stored with this session, such that we do not have to retrieve this information everytime.

Action changed to: *newuser change-password* if this is a new user, *overview main* if this is a returning user, and *login main* on error.

do-register Here we check the registration information that the user filled in. The user name should conform to RFC 1035. The email address is checked to contain a '@' and something in front and after the '@' sign. The complete definition for an email address can be found in RFC 822, but it is too complicated for a simple site as ours (for the moment). The user name should be unique in the GBS, so we check that the name is still available. If the name is available we generate a 13 character random password for this user. The user is then added to the database and an email is sent to the email address given, containing the password.

Action changed to: login main if success, login register on error.

do-forgot We check that the user name is valid and retrieve the user's email address from the database. Then we generate a new random 13 character password and update our database with the new password. And finally, we send the new password to the user's email address.

Action changed to: login main on success, login forgot on err.

do-logout This removes the session and cookie for this user.

Action changed to: login main.

6.2.2 New User

New User handles users who log in for the first time. The main purpose is to add the user to the database and introduce the user with the Express Setup option.

change-password This generates a page where the user must change his password.

choice-page This generates a page where a user can choose between using the express setup or to go directly to the overview page.

do-change-password We check that the password fields are the same and not empty. We then update the database to put in the password and mark this user as not new anymore. If this was successful we do the following:

- Assign access rights to the user from himself.
- Assign access rights to the *Globule Team*, so it can place its replica on this user's server(s).
- Assign access rights to *Default Action*. By default, *Default Action* is not allowed to do anything.

Action changed to: newuser choice-page on success, newuser change-password on error.

6.2.3 Express

Express is used to generate the Express Setup pages. It allows the user to easily install unique site and server. The user also uses our DNS redirector and chooses an unique name.

main Generates the main page where the user is asked for some details about his server and site. We fill in the hostname or IP address using the IP address of the computer that is requesting the webpages. The port number is set to 80 because we are going to use DNS redirection and our replica servers are also running on port 80. We have filled in a default <ServerRoot> and <DocumentRoot> and via the tooltip we also show these defaults for Windows. The sitename is filled in with the user name.

info Generates an info page with the server and site that the user just added.

do-add Here we verify what the user has filled in on the main page. We first check that the hostname or IP address is valid. If the user filled in a hostname, we try to obtain the IP address. Next, we check that the <ServerRoot> and <DocumentRoot> are filled in. And finally the site name is checked to be a proper DNS name and that the name is unique.

When everything checks out, the server is added to our database. Next, we add replica and backup quota for this server into the database. By default a server allows one replica and 10MB of storage and zero backups. These defaults are defined in the *constants.php*.

Then the site is added. Using the same defaults as above it will request only 1 replica and no backups. The replica will be requested from the *Globule Team* group.

Action changes to: *express info* on success, *express main* on error.

6.2.4 Personal

Allows the user to check and change his personal details and password.

main Generates the main page on which the user can change his first name, last name and email address.

change-password Generates a page on which the user can change his password.

change-personal-ok Tells the user that the update of the user's first name, last name and email address was successful.

change-password-ok Tells the user that the update of the user's password was successful.

do-change-personal We check that the user's first name, last name and email address are filled in and that the email address is valid. Then we update our database.

Action changes to: *personal change-personal-ok* on success, *personal main* on error.

do-change-password We check that the password are non-empty and that they are the same in both fields. The database is then updated with the new password. Action changes to: *personal change-password-ok* on success, *personal change-password* on error.

6.2.5 Access Rights

Management of access rights by allowing the user to view, update access rights and add/remove users and groups to/from his list. The user can also create new groups and manage members of the groups.

main First we retrieve from the database the access rights that the user assigned to himself. Then we retrieve the access rights that the user has assigned to other users (with `add-user`).

Next we show the rights the user has assigned to his private and public groups (with `create-group`) and the rights the user has assigned to the public groups from others (with `add-group`).

And, finally, we show the rights assigned to *Default Action*.

add-user Shows a list of users, that do not have access rights assigned to them by this user.

add-group Shows a list of public groups, that do not have access rights assigned to them by this user.

create-group This generates a page on which the user can add a private or public group.

view-group:<group type>:<groupID> If the user is the owner of the group, it will show a page with the current members of the group and gives the user the ability to add or remove users. Otherwise it only shows the members of the group.

add-user-to-group:<group type>:<groupID> Shows a list of users that are not member of the selected group. The users can then select new members and add them.

do-add-user The users selected are added to the user's access rights list. By default they have all rights disabled.
Action changes to: *rights main*.

do-update-user:<userID> The rights assigned to the user are updated.
Action changes to: *rights main*.

do-remove-user:<userID> The user is removed from the access rights list.
Action changes to: *rights main*.

do-add-group The groups selected are added to the user's access rights list. By default they have not rights yet.
Action changes to: *rights main*.

do-create-group A new group is created (if the name and type are unique). By default this group has no rights.
Action changes to: *rights view-group:<group type>:<groupID>* on success, or *rights create-group* on error.

do-update-group:<group type>:<groupID> Rights of the group are updated. The `<group type>` remains the same.
Action changes to: *rights main*.

do-remove-group:<group type>:<groupID> The selected group is removed from the access rights list. If the user is the owner of the group then all the members of the group are removed from the group first.

All the sites that were using this group as their resources group will now use the <No-one> group. The next time these users fetch a configuration for the server that hosts these sites, their replicas and backups will be removed. **FIXME: this is a very silent process and users might not notice nor understand why they lost their replicas...**

Action changes to: *rights main*.

do-change-group:<group type>:<groupID> The group name is changed and the <group type> is changed as well, but only if the group name, group type remain unique.

Action changes to: *rights view-group:<group type>:<groupID>*.

do-add-user-to-group:<group type>:<groupID> The selected users are made a member of the group.

Action changes to: *rights view-group:<group type>:<groupID>*.

do-remove-user-from-group The selected users are removed from the group.

Action changed to: *rights view-group:<group type>:<groupID>*.

6.2.6 Overview

The overview only has *main* as action. It generates a page that lists the user's sites and servers and some status information, like the number of assigned replicas and backups per site and the number of origins, replicas and backups on your servers. An exclamation mark in front of a server name, notifies the user that that server will need a new configuration file.

6.2.7 Servers

Everything about the servers the user owns can be viewed and updated here. Some information is provided about who is using the servers and for what purpose.

add Generates a page on which the user can add a server by providing information about the server. This is a bit more for the advanced users. They can enable PHP support and enable/disable the auto fetching and restart of the configuration file. This is just a flag in the config file: **# GlobuleAutoRestart y/n** and programs should check this flag. The ApacheMonitor for Windows does check and adhere to this flag.

view: Displays a list of all the servers owned by this user. It also shows the number of origins per server and the number of replicas or backups assigned.

view:<serverID> Displays all information about the <serverID> given. The user may change most of the information that was given when adding the server, except for the port number.

Information is displayed on the origins, replicas, backups and sites that use the redirector on this server. It also displays the locations on disk where data for these sites are stored.

remove-ask:<serverID> Asks the user if he is certain that he wants to remove the server indicated by <serverID>.

do-add First we check that the hostname or IP address is OK and that the server root and document root are filled in. If PHP is enabled, we check that those paths are set as well. Then we check that the disk space available for replicas is a number.

Then we add the server to our database and also insert the replica quota and backup quota values.

Action changes to: server view:<serverID> on success, server add on error.

do-update:<serverID> Here we update the server's information in our database. We check that the server root is filled in and that the disk space available to replicas is a number.

If the server stops supporting PHP, when it did before, then all the origin sites that require PHP should have been removed before. All replicas and backups that required PHP will be removed.

If the number of replicas/backups allowed on this server is decreased, we remove the superfluous replicas/backups.

Action changes to: server view:<serverID>.

do-remove:<serverID> Checks that there are no origin sites are placed on this server. If there are, these need to be removed first (but not here). We remove all trace of this server in the various tables.

Action changes to: server view.

6.2.8 Sites

Information about the sites the user has created can be viewed and updated here. Information is provided to show where replicas and backups are located.

add Generates a page where a user can add a site. The user selects a server which he owns. The user can choose to use a redirector and fill in a prefix for that redirector or leave these fields blank and fill in a path to use with the server name. If the site requires PHP, a checkbox can be checked to indicate this. The user can then fill in how many replicas and backups he wants and from which group he wants to have the replicas and backups. The user can only choose groups that he owns or public groups from other users or the '<Every One>' group.

view: Generates a page with a list of all the sites this user has and the number of replicas or backups assigned and requested.

view:<siteID> Generates a page with all the information about the site indicated by <siteID>. The user may change the prefix and redirector used. The user may also change the number of replicas/backups requested and from which group to request them from. Regarding PHP, a checkbox can be checked or unchecked.

remove-ask:<siteID> Asks if the user is certain that he wants to remove the site <siteID>.

do-add Adds the site to the database. If DNS redirection is not used, we check that the path is OK. If DNS redirection is used, we check that this server does not both host the origin for this site and be a redirector for this site. Then we check that the prefix is valid and available. If this all works, then we check that the port number the redirector uses is the same as the port number that the origin server uses. If the site wants to use PHP, then the origin server also needs to support PHP. Then finally the site is added to our database.

Action changes to: site view:<siteID> if success, site add on error.

do-update:<siteID> We update the information of the site here. Most changes causes the GBS to have to remove all assigned replicas and backups for this site. If the prefix is changed, we check that the new prefix is still available and release the old if it is not the username or the redirector server has changed.. We also make sure that the redirector server is not the same as the origin server.

Action changes to: site view:<siteID>.

do-remove:<siteID> The site is removed. All replicas and backups are removed as well. If the site used DNS redirection, the prefix is freed as well.

Action changes to: site view:.

6.2.9 Config

Generates the Apache configuration files for the servers and assigns/updates the replicas and backups assigned to the server.

create:<serverID> Generates a page with some information about the server and a button to download the configuration file.

do-create:<serverID> This will generate an Apache configuration file for this server. We will explain later how this is done precisely. The user downloads this file to the server in question.

Direct Access To enable the automatic fetching of the configuration files, by scripts/programs like the Apache Monitor, they can access the config.php file directly by giving serverID and userID parameters. We only check that the given userID has a server with the serverID indicated. We could also check that the IP address of the machine fetching the config, matches the IP address belonging to the serverID. Although if the user maintains a centralized status of all his servers, than this check would block doing so.

6.3 Cascading Style Sheet

The layout of the site is handled by a Cascading Style Sheet which can be found in *gbs.css*.

6.4 Tool Tips

Most things have tool tips to give the user extra information. For example when you see a user name, the tool tip will show the user's full name.

7 Database

The GBS database contains a number of tables to store all our data. We will define the entity tables and the relationship tables below that are used in our database. It is also worth noting that the database is case insensitive when matching fields. All the paths that are used in the tables are in UNIX-style notation. In *constants.php* you must define the hostname of the database (DBH), the username and password for the database (DBU, DBP) and the name of the database (DBN).

7.1 Entity Types

The entities of the GBS are defined below. Note that a particular ID will never be zero. We use zero to denote not used or inactive.

7.1.1 Users

This table holds basic information about the users and holds a unique ID per user.

```
CREATE TABLE Users (  
  userID          INT UNSIGNED  AUTO_INCREMENT,  
  userName       VARCHAR(16)   NOT NULL,  
  firstName      VARCHAR(50)   NOT NULL,  
  insertion      VARCHAR(50)  
  lastName       VARCHAR(50)   NOT NULL,  
  emailAddress   VARCHAR(255)  NOT NULL,  
  password       VARCHAR(50)   NOT NULL,  
  new            BOOLEAN       DEFAULT 1,  
  createdAt      DATETIME      NOT NULL,  
  PRIMARY KEY(userID),  
  UNIQUE(userName));
```

The password entry will contain the SHA1 of the user's password, which will be a 40 character hexadecimal string. If another password encryption scheme is used it may be up to 50 characters. The user name should be unique.

The database should contain two users before usage: *Default Action* and *Globule*. The *Default Action* user will be used to provide the system with a way to denote the default action to be taken. It must be inserted manually with:

```
INSERT INTO Users(userName, firstName, lastName, emailAddress, new, createdAt)  
  VALUES ('Default Action', 'Default', 'Action', 'wilfred@cs.vu.nl',  
          0, NOW());
```

Note that the password is not set, to make it impossible/hard to log in as this user. The userID of this user must be set in *constants.php* as **G_DEFAULT_ID**.

The *Globule* user will be responsible for our servers. Use the following to insert it manually:

```
INSERT INTO Users(userName, firstName, lastName, emailAddress, password,  
  new, createdAt)  
  VALUES ('Globule', 'Globule', 'Globule', 'wilfred@cs.vu.nl',  
          SHA1('XXXXXXXX'), 0, NOW());
```

The password should be set to something else than 'XXXXXXXX' of course. The userID of this user must be set in *constants.php* as **G_GLOBULE_ID**.

7.1.2 Groups

This table hold information pertaining groups.

```
CREATE TABLE Groups (  
  groupID      INT UNSIGNED      AUTO_INCREMENT,  
  name         VARCHAR(50)      NOT NULL,  
  type         ENUM('PRIV', 'PUB') NOT NULL,  
  ownerUserID  INT UNSIGNED,  
  PRIMARY KEY(groupID),  
  UNIQUE(name, type, ownerUserID));
```

The type of groups are 'PRIV' to indicate a private group and 'PUB' to indicate a public group. The combination of group name, type and owner should be unique.

The database should contain three groups before usage: *EveryOneDummy*, *NoOneDummy* and *Globule Team*. *EveryOneDummy* and *NoOneDummy* are a private groups owned by *Globule*. It will be used to provide the system with a 'Everyone' and a 'No-One' group. The *Globule Team* is also owned by *Globule*, but it will be a public group. Insert the three groups manually as follows:

```
INSERT INTO Groups(name, type, ownerUserID)  
VALUES ('EveryOneDummy', 'PRIV', G_GLOBULE_ID),  
      ('NoOneDummy', 'PRIV', G_GLOBULE_ID),  
      ('Globule Team', 'PUB', G_GLOBULE_ID);
```

The ownerUserID should be the userID of *Globule*. The groupIDs of the *EveryOneDummy*, *NoOneDummy* and *Globule Team* should be noted in the *constants.php* as **G_EVERYONE_GROUP_ID**, **G_NOONE_GROUP_ID** and **G_GROUP_ID** respectively. The names for the *EveryOneDummy* and *NoOneDummy* groups as shown on the web pages should be set in the *constants.php* with **G_EVERYONE_NAME** (currently we use '<Every One>') and **G_NOONE_NAME**.

7.1.3 Servers

This table contains all the information about the servers from users.

```
CREATE TABLE Servers (  
  serverID      INT UNSIGNED      AUTO_INCREMENT,  
  ip            VARCHAR(15)      NOT NULL,  
  hostname      VARCHAR(255)    NOT NULL,  
  port          SMALLINT,  
  serverRoot    VARCHAR(255)    NOT NULL,  
  serverDocumentRoot VARCHAR(255),  
  ownerUserID  INT UNSIGNED,  
  canDNSRedirect BOOLEAN        DEFAULT 0,  
  providesPHP  BOOLEAN        DEFAULT 0,  
  phpModule    VARCHAR(15),  
  phpPath      VARCHAR(255),  
  phpIniDir    VARCHAR(255),  
  autoRestart' BOOLEAN        DEFAULT 1,  
  lastConfig   DATETIME,  
  dirty        BOOLEAN        DEFAULT 1,  
  PRIMARY KEY(serverID),  
  UNIQUE(ip));
```

For the first iteration, we will allow each server to have only one IP address. If a server is added with two different IP addresses, they will be treated as if they were separate servers.

The database should contain a server to provide DNS redirection (baby.cs.vu.nl, 130.37.198.245). The DNS redirector uses a special domain name for our purposes: *globeworld.net*. It has to be added manually as follows:

```
INSERT INTO Servers(ip, hostname, port, serverRoot, serverDocumentRoot, ownerUserID,
                   canDNSRedirect, lastConfig, dirty)
VALUES ('130.37.198.245', 'globeworld.net', 80, '/etc/httpd/', '/var/www/html/',
        G_GLOBULE_ID, 1, NOW(), 0),
```

These servers is owned by *Globule*. The serverID of this server must be noted in the *constants.php* as **G_REDIRE_ID**. Also add to *constants.php* the values for **G_REDIRE_NAME** (baby.cs.vu.nl) and **G_BASE_NAME** (globeworld.net). Because of our trick, we can not store the server's real name in the database, but we do need it when generating the config file.

7.1.4 Sites

This tables contains all the basic information about a particular site.

```
CREATE TABLE Sites (
  siteID          INT UNSIGNED AUTO_INCREMENT,
  name           VARCHAR(255) NOT NULL,
  prefix         VARCHAR(50),
  redirectorServerID INT UNSIGNED,
  originServerID INT UNSIGNED,
  path           VARCHAR(255) NOT NULL,
  ownerUserID    INT UNSIGNED,
  resourcesGroupID INT UNSIGNED,
  replicasRequested SMALLINT DEFAULT 1,
  backupsRequested SMALLINT DEFAULT 0,
  phpRequired    BOOLEAN DEFAULT 0,
  PRIMARY KEY(siteID),
  UNIQUE(name));
```

Prefix is the name that the user has chosen to use with a DNS redirector. If prefix is set, then siteRoot and redirectorServerID are also set and path will be '/'. Otherwise siteRoot is empty and redirectorServerID will be zero, path will be set by the user. The resourcesGroupID indicates that this site only wants replicas and/or backups from members of this group.

Although *www.globeworld.net* is a site using the DNS redirector, we do not add it into our database because it can not use replicas and backups (due to the PHP and mysql database).

7.2 Relationships

The entities have the following relationships with each other.

7.2.1 Not Available

Some user names will not be available. These can be names that are already used by someone else (either as user name or as site name of our Globule redirector) or names we deem inappropriate.

```
CREATE TABLE NotAvailable (
  userName VARCHAR(50) NOT NULL,
  mayRemove BOOLEAN DEFAULT 1,
  PRIMARY KEY(userName));
```

The `mayRemove` bit is used to prevent accidental removal of names used by us. The list is automatically updated when users register and if they use our special redirector, the prefix names will also be inserted here. We do this to ensure that someone's username can always be used with our redirector.

The following names must be added manually to prevent others from using them. Manually add other names that might be offensive.

```
INSERT INTO NotAvailable(userName, mayRemove)
VALUES ('Globule', 0), ('Default Action', 0), ('Default-Action', 0),
       ('Globule Team', 0), ('Globule-Team', 0), ('www', 0),
       ('Every One', 0), ('Every-one', 0), ('Everyone', 0);
```

The username 'www' is included because the DNS redirector is still used for *www.globeworld.net* and we do not want other to claim this name.

7.2.2 Member Of

Users can be members of multiple groups. Groups can have multiple users as member. A user cannot be member of the same group more than once. M:N relationship.

```
CREATE TABLE MemberOf (
  userID INT UNSIGNED,
  groupID INT UNSIGNED,
  PRIMARY KEY(userID, groupID));
```

The *EveryOneDummy* group will never have members as it denotes everyone. The *NoOneDummy* group will also never have members. The *Globule Team* group will have *Globule* as member, so we manually add that here:

```
INSERT INTO MemberOf(userID, groupID)
VALUES (G_GLOBULE_ID, G_GROUP_ID);
```

7.2.3 Assigned Rights to Users

Users can assign rights to other users. This will also determine if a user will be visible in the user's access rights list. The user can assign access rights to multiple users. A user cannot assign access rights to the same user more than once. 1:N relationship.

```
CREATE TABLE RightsToUsers (
  userID INT UNSIGNED,
  toUserID INT UNSIGNED,
  hostReplicaFor ENUM('Y', 'A', 'N') DEFAULT 'N',
  hostBackupFor ENUM('Y', 'A', 'N') DEFAULT 'N',
  hostRedirectorFor ENUM('Y', 'A', 'N') DEFAULT 'N',
  PRIMARY KEY(userID, toUserID));
```

The letters 'Y', 'A', 'N' stand for Yes, Ask and No respectively. The Ask option will not yet be used. Note that the order of the ENUM is important for sorting. (We prefer having 'Y' before 'N').

Every user will assign access rights to himself and to the *Default Action* (which we defined as a special user). So for user *Globule* we manually add:

```

INSERT INTO RightsToUsers(userID, toUserID, hostReplicaFor, hostBackupFor,
                        hostRedirectorFor)
VALUES (G_GLOBULE_ID, G_GLOBULE_ID, 'Y', 'Y', 'Y'),
      (G_GLOBULE_ID, G_DEFAULT_ID, 'Y', 'N', 'N');

```

The *Globule* user allows everyone to host a replica on his servers. This way we do not have to assign rights to every user in our system.

For other users, the GBS automatically inserts default values for themselves ('Y', 'Y', 'Y') and *Default Action* ('N', 'N', 'N') when they log in for the first time.

7.2.4 Assigned Rights To Groups

Users can assign rights to groups. This will also determine if a group will be visible in the user's group list. The user can assign access rights to multiple groups. The user cannot assign access rights to the same group more than once. 1:N relationship.

```

CREATE TABLE RightsToGroups (
  userID          INT UNSIGNED,
  groupID         INT UNSIGNED,
  hostReplicaFor ENUM('Y','A','N') DEFAULT 'N',
  hostBackupFor  ENUM('Y','A','N') DEFAULT 'N',
  hostRedirectorFor ENUM('Y','A','N') DEFAULT 'N',
  PRIMARY KEY(userID, groupID));

```

The letters 'Y', 'A', 'N' stand for Yes, Ask and No respectively. The Ask option will not yet be used. Note that the order of the ENUM is important for sorting. (We prefer having 'Y' before 'N').

For new users we always assign access rights to the *Globule Team* group to allow replicas. The owner of a group always has access rights assigned to that group, so we manually add for user *Globule*:

```

INSERT INTO RightsToGroups(userID, groupID, hostReplicaFor, hostBackupFor,
                        hostRedirectorFor)
VALUES (G_GLOBULE_ID, G_GROUP_ID, 'Y', 'N', 'N');

```

7.2.5 Has Replica

A site can have multiple replicas. A site can only use a server as replica once. 1:N relationship.

```

CREATE TABLE HasReplica (
  siteID  INT UNSIGNED,
  serverID INT UNSIGNED,
  path    VARCHAR(255) NOT NULL,
  password VARCHAR(32) NOT NULL,
  PRIMARY KEY(siteID, serverID));

```

The path will be set to the site's name with a leading '/' when no DNS redirector is used or '/' if it is. The password will automatically be assigned by the GBS and will be a 13 character string.

7.2.6 Has Backup

A site can have multiple backups. A site can only use a server as backup once. 1:N relationship.

```
CREATE TABLE HasBackup (
  siteID    INT UNSIGNED,
  serverID  INT UNSIGNED,
  path      VARCHAR(255) NOT NULL,
  password  VARCHAR(32)  NOT NULL,
  PRIMARY KEY(siteID, serverID));
```

The path will be set to the site's name with a leading '/' when no DNS redirector is used or '/' if it is. The password field will be assigned by the GBS and will be a 13 character string.

7.2.7 Has Redirector

A site can have multiple redirectors (but only one domain name). 1:N relationship.

```
CREATE TABLE HasRedirector (
  siteID    INT UNSIGNED,
  serverID  INT UNSIGNED,
  path      VARCHAR(255) NOT NULL,
  password  VARCHAR(32)  NOT NULL,
  PRIMARY KEY(siteID, serverID));
```

The path will be set to '/'. The password fields will be assigned by the GBS and will be a 13 character string.

7.2.8 Has Replica Quota

A server must have quota on number of replicas and disk space for replicas. A server can only be assigned replica quota once. 1:1 relationship.

```
CREATE TABLE HasReplicaQuota (
  serverID    INT UNSIGNED,
  nrOfReplicas TINYINT UNSIGNED,
  diskSpace   SMALLINT UNSIGNED,
  PRIMARY KEY(serverID));
```

The user may specify to host up to 255 replicas. The user may specify available disk space per megabyte, up to 65,535 megabytes (65GB). When a server is added, this table is updated. For the redirector server of *Globule* we manually add:

```
INSERT INTO HasReplicaQuota(serverID, nrOfReplicas, diskSpace)
VALUES (G_REDIR_ID, 0, 0);
```

The GBS currently allows a maximum of 10 replicas per server. This is defined in *constants.php* with **G_MAX_REPL**.

7.2.9 Has Backup Quota

A server must have a quota on the number of backups on the server. A server can only be assigned backup quota once. 1:1 relationship.

```
CREATE TABLE HasBackupQuota (
  serverID    INT UNSIGNED,
  nrOfBackups TINYINT UNSIGNED,
  PRIMARY KEY(serverID));
```

The user may specify to host up to 255 backups. When a server is added, this table is updated. For the redirector servers for *Globule* we add manually:

```
INSERT INTO HasBackupQuota(serverID, nrOfBackups)
VALUES (G_REDIR_ID, 0);
```

The GBS currently allows a maximum of 4 backups on the server. This is defined in *constants.php* with **G_MAX_BACK**.

8 GBS, Database Interaction

At certain points the GBS does some maintenance or updates to the database. We will explain what happens below.

8.1 Assigning Replicas & Backups by the GBS

Now that we have all the information in our database, the GBS can start looking for sites that need replicas/backups and servers that will allow them.

This is done at the time that someone wants to generate a configuration file for his server. This way we are certain that that server will use the configuration we have chosen for it and that the setup adheres to the rules. This is the only time replicas and backups are assigned.

The rules to see if a site may have a replica on this server, are as follows (for backups this is about the same):

1. The server allows replicas and has space left for more replicas.
2. The site has requested replicas and still needs more replicas.
3. The owner of the server must have assigned the 'hostReplicaFor' access right to the owner of the site, either
 - directly to the site owner,
 - or to a group where this site owner is a member of,
 - or if the owner of the server set this access right for *Default Action*.
4. The owner of the server must be a member of the site's 'resourcesGroup' or the resourcesGroup is *<EveryOne>*.
5. The server must use the same port as the port of the site's origin server.
6. The server may not also be the origin server of the site.
7. The server may not also host a backup for the same site.
8. From all the sites that match these criteria, select the ones with the least number of replicas.

If the server does not provide PHP support, then only sites that do not require it are selected. If the server does provide PHP support, all sites are eligible, but we prefer sites that require PHP support.

In terms of database queries this goes as follows:

1. From the RightsToUsers table, select the users that have 'hostReplicaFor' = 'Y' assigned to them by the server owner.

2. From the RightToGroups table, select all the groups that have 'hostReplicaFor' = 'Y' assigned to them by the server owner. Take all the members from these groups.
3. If the server owner allows *Default Action* to host replicas, we just select all users.
4. From all the users selected in the first three queries, take all the sites they own and where the server owner is a member of the site's resourcesGroup.
5. The selected site has requested more than zero replicas.
6. The selected site's origin server runs on the same port as this server.
7. The selected site has no replica yet on this server.
8. The selected site has no backup yet on this server.
9. For all these sites count the number of replicas they already have.
10. From these sites, discard the ones where the origin server is the same as this server.
11. Select only the sites that have no replicas or less replicas than requested.
12. Sort the sites such that the site with the least number of replicas is first.
13. Select only the number of sites that this server can still host a replica for.

See the file *config.php* for the exact query.

8.2 Removing Replicas & Backups

There are three places in the GBS where replicas and backups are removed. These are:

1. Sites, when the number of replicas or backups is reduced or the site is removed or the site name is changed. Or when the site changes its PHP requirement.
2. Servers, when the number of replicas or backups is reduced or the server is removed. Or when the server stops supporting PHP if it did before.
3. Config, when checking if current replica/backup assignments still abide the rules (e.g. when access rights changed).

Note that these changes do not take effect in the real world until every server involved obtains a new configuration file. To make things a little easier, every server has a dirty bit, which we describe in the next section.

8.3 Dirty Bit

As mentioned above changes to our database are only propagated to the servers when they fetch a new configuration file. In the future, Globule servers will be able to fetch their own configuration file and restart and in the far future, Globule servers can dynamically update their configuration without the need to reboot.

For this to work, every server has a dirty bit. The dirty bit is set by the GBS everytime a change is made in the database that only takes effect when a new configuration file is downloaded. The triggers are as follows:

- When a replica is added/removed to/from a server, the dirty bit of the origin server of the replica is set.
- When a backup is added/removed to/from a server, the dirty bit of the origin server of the backup is set AND all the replica servers of this site get their dirty bit set.
- When a site adds/changes/removes the DNS redirector capability, the dirty bit of the redirector server is set.

At the moment, when a dirty bit is set an exclamation mark because visible on the web pages to indicate that the server needs a new configuration file.

The only way to clear the dirty bit is to download a new configuration file for the servers.

8.4 Downloading Config Without User Interaction

The Globule servers will automatically download new configuration files and restart (if necessary). To facilitate this, they can access the GBS via:

<http://www.globeworld.net/config.php?userID=x&serverID=y>

where x must be replaced with the owner userID of the server indicated by y.

This allows scripts to fetch a new configuration file. If the script checks the serial number in the configuration file, it can decide whether or not to restart the server. The serial number is only changed when the dirty bit is set.

For Windows we have modified the ApacheMonitor to check every ten minutes for a new configuration file. If Apache is running, then it will automatically be restarted when a new configuration file is ready. If Apache is not running, only the configuration file will be replaced. To disable the automatic fetching of new configuration files, the user may edit the httpd.conf file and set '# GlobuleAutoRestart' to 'n'. Make sure the GBS also knows that you do not want this, by updating the server. Then the next time you download a httpd.conf by hand, it will remember this setting.

On Unix/Linux platforms a globulectl utility program exists, which works largely like apachectl, but if the previous httpd.conf originates from the GBS, it is able to reload a newer configuration from the Globule Broker System. Refer to the Globule documentation for the usage of the globulectl utility.