StreamOnTheFly

Technical Documentation

This documentation covers the needs of developers and designers to work with the StreamOnTheFly (SOTF) Application Framework.

The Document does not cover basic help for users. It's addressed to the technicans working with StreamOnTheFly and written by persons with programm- and systemadministration skills.

The main aims of this document:

- ✔ Support for installation of NODE and PORTAL
- ✔ Configuration of the Application Framework
- ✔ Customization of Layout and Features using the smarty template engine
- ✔ Illustration of Code / Classes and Database structure
- ✔ The Need of Unix Helpers for encoding and transport of data
- ✔ The Database Interface PEAR
- ✔ PostgreSQL as backend

Special Thanks to all persons helped in creating this paper: Dr. Roland Anton-Scheidl, Wolfgang Reutz, Andras Micsik, Alexander Kulinkov, The StreamOnTheFly Community.

```
/*****
* @package: StreamOnTheFly / Documentation
* @author: juergen schmidt (www.strg.at / juergen@strg.at)
* @author: michael dosser (www.strg.at / mic@strg.at)
* @version: 0.5
* @license: GPL (www.gpl.org)
*****/
```

www.strg.at
Stollgasse 8/5
1070 Wien
email: office@strg.at
Phone: ++43 1 526 56 29
Fax: ++43 1 526 56 49

Content

1) What is StreamOnTheFly (SOTF) 4
2) System Requirements 5
2.1) Unix Daemons 5
2.2) Unix Helpers 5
2.3) PHP Libraries 6
3) Installation and Configuration 6
3.1) Get the Sources 6
3.2) Installation 6
3.2.1) File permissions 7
3.2.2) Install script and first tests 7
3.2.3) Configuration / config.inc.php 8
4) The Node 9
4.1) SOTF id's / Identifier Scheme 10
4.2) Stations / Series / Programms 10
4.3) Basics and Features 11
4.3.1) Search Engine 11
4.3.1.1) Create Query 12
4.3.1.2) Run Query 12
4.3.1.3) Save Query 13
4.3.2) Topic Tree / Genres / Roles 13
4.3.3) RSS-Feeds 14
4.3.4) Statistik 15
4.3.5) Feedback 15
4.3.6) My playlist 16
4.3.7) Rigths and permissions 17
4.3.8) Editors' console 17
4.4) Dataimport // XBMF 17
4.4.1) Specifications // Metadata Example 18
4.4.2) XBMF Structure (new) 19
4.5) Communication 20
4.5.1) Syncronisation and Network 20
4.5.1.1) Adding new nodes 20
4.5.1.2) Detect nodes that are down 20
4.5.1.3) Network load 20
4.5.1.4) Speed of replication 21
4.5.1.5) Redundancy in updates 21
4.5.1.6) Node IDs 21
4.5.1.7) Getting item details 21
4.5.2) XMLRPC 21
4.6) Cron Jobs 21
4.6.1) Cron jobs for SOTF 22
4.7) Database 22
4.7.1) Relational Schema 22
4.7.2) Tableslist and usage 22
4.8) Codestructure 24
4.8.1) Files 24
4.8.2) Classes / Objects 25
4.9) NODE Interface 26
4.9.1) Interface HTML Files 27
4.9.2) Interface language files 28
5) The Portal 29
5.1) Principles 29
5.2) Installation 29
5.2.1) Requisits: 29
5.2.2) Preinstall: 29
5.3) Configuration 30

5.4) Code Structure 30
5.4.1) Classes 30
5.4.2) Executable Files 30
5.5) Database Strucutre 31
5.5.1) List of tables 31
5.5.2) Table description 32
6) UNIX Daemons 35
6.1) Apache 1.3.x/PHP4.x 35
6.2) PostgreSQL 35
6.3) ProFTPd (optional) 36
7) UNIX Helpers 36
7.1) OGG – Tools 36
7.2) transcode 36
7.3) Lame 37
7.4) Sox 37
7.5) rsync 37
7.6) ImageMagick 38
8) PHP Libraries 38
8.1) PEAR // The Database Interface 38
8.1.1) Installation 39
8.1.2) Mode of operation 39
8.1.3) PEAR DB-Object for PostgreSQL 39
8.1.3.1) Provided Methods 39
8.1.3.2) Error Object 40
8.2) Smarty // the Template Engine 41
8.2.1) Code and structure 42
8.2.2) general Methods and way of operation 42
8.2.3) config files and variables 43
8.2.4) plugins (functions and modifiers) 44
8.2.4.1) Modifiers 44
8.2.4.2) Functions 44
8.3) GetID3 45
9) Debugging and Tools 46
9.1) error messages 46
9.2) psql / commandline interface 46
9.3) phpPgAdmin 47
9.4) DbVisualizer / The Universal Database Tool 48
9.5) phpdocumentor 49

1) What is StreamOnTheFly (SOTF)

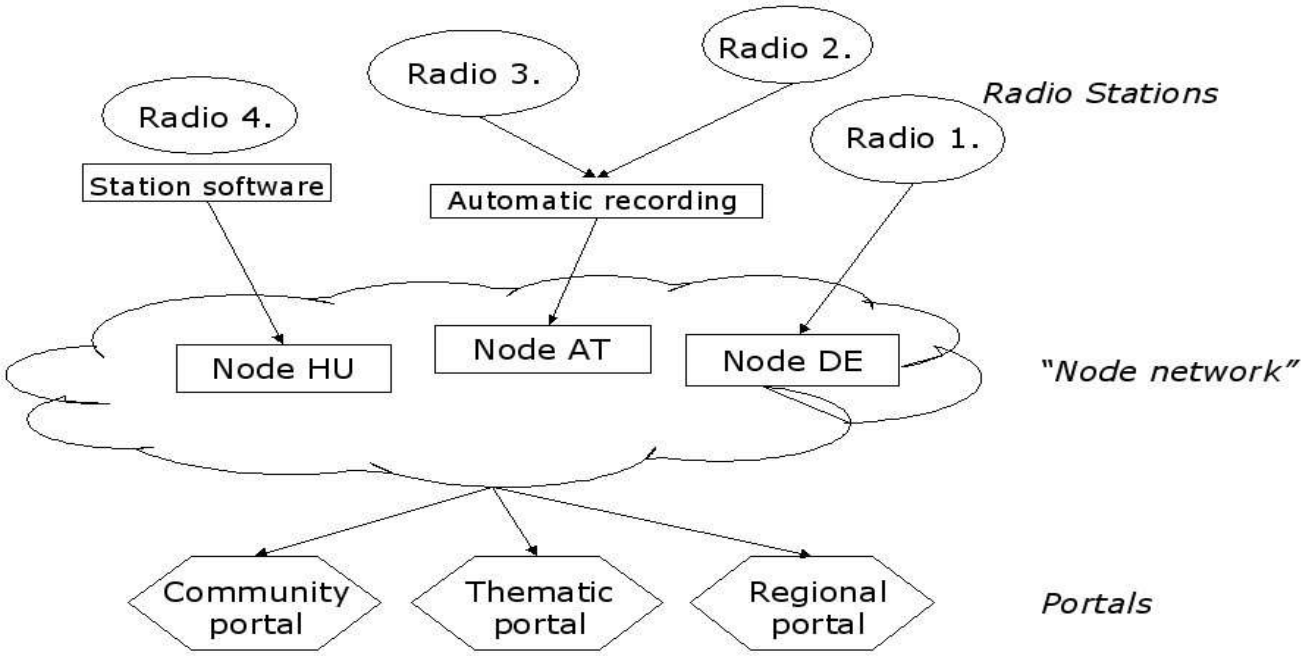
Distributed Archive and Web Services for Audio Content.

StreamOnTheFly is a multisite audio archive, radio station management and audio publication system. It's main objective is to find and experiment with new methods for the exchange and reuse of radio shows. It targeted the crowing number of free- and community radios.

One of the causes for this is the lack of technical support for easy exchange of radio programs. A significant part of the programs produced at radio stations are too valuable to be broadcasted only once. The StreamOnTheFly project was set out to find and demonstrate new ways for archival, management and personalization of audio content.

Technically it was mainly developed by Sztaki (Hungary/Budapest) in a Cluster with Public Voice LAB (Vienna/Austria http://www.pvl.at), Team Teichenberg and Radio Orange. SOTF was part of the EUTIST-AMI cluster activity of the European IST Program (IST-2001-32226)
As a collaborative platform it works on the level of web-based applications. SOTF is written in PHP and UNIX-Shell Scripts. It uses postgreSQL as a database backend. The interface is translated in English, germane, trench and Hungarian language and works on the basis of the smarty template engine (http://smarty.php.net). As a general database interface pear is used as a flexible extension to the SOTF-classes. (http://pear.php.net)

In a short description it is a decentralized and self-organizing network of nodes which forms the basis of the StreamOnTheFly environment. Each node hosts a set of radio stations. For each station the node archives radio programmes with a rich set of metadata and other associated content (Audio in several formats, photos, scripts and videos).
All metadata are automatically replicated on each node of the network. As there is no central server in the network, each node has a set of neighbours, and nodes periodically exchange new and modified metadata with their neighbours.



1. Figure: Basic Principles of Node-Network

The aims of this paper is a documentation for developers and designers. Please visit streamonthefly.org for a basic overview what you can do with this framework, and to see some casestudies about successful commitment.

2) System Requirements

SOTF needs a webserver, PHP, PostgreSQL as a database backend and a cupple of helpers to encode files, run synchronisation between nodes, GetID3, Imagemagick and Transcode if you want to extend the framework with features for video as binary content.
SOTF framework is running on all Unix and Unixlike operating systems on which the necessary tools are already ported. This documentation covers mainly our experiences on Debian GNU/Linux and FreeBSD.

2.1) Unix Daemons

- Apache1.3.x/PHP4.x (webserver & scripting engine)
- PostgreSQL (database backend)
- ProFTPd (to upload big binary files to the node)

2.2) Unix Helpers

SOTF needs a cupple of UNIX Helpers for file-encoding and transport. You can install them on all Linux/Unix* (OSX included) distributions.

- Oggtools (for soundfile encoding into the OGG file format)
- Lame (for soundfile encoding into the mpg file format)
- transcode (if you play with video encoding)
- Sox (to encode between different sound formats)
- rsync (to synchronice binary content between nodes)
- ImageMagick (for re-rendering of images)

2.3) PHP Libraries

SOTF needs three PHP components which are provided from the community to include in webapplications. This Libraries are composed as a cupple of PHP-classes to include in projects. They are widley generalized and written for reuse in several applications.

- PEAR
- SMARTY
- GetID3

3) Installation and Configuration

We describe the installation and configuration of the SOTF Framework. Installation of Helpers and Daemons are covered later in this document.

3.1) Get the Sources

Since SOTF is a sourceforge hosted project one will find all necessary sources on http://sourceforge.net/projects/sotf/. SOTF uses the file-publishing functions on sourceforge to release tar-balls for node and portal. You will find just stable releases in this tar-balls. For the latest changes please use the CVS system hosted on sourceforge.

3.2) Installation

There are two ways to get the sources from SOTF. Since SOTF is hosted by sourceforge (http://sourceforge.net) you can use the CVS provided by sourceforge or download a tar-ball. Go to a directory where the stream on the fly files should reside and enter the following commands. If prompted for a password just hit enter.

```
cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/sotf login
cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/sotf co node
```

You will now have a directory named "node" in the current directory / the directory you issued the above commands. Make the whole directory readable by the webserver. Create a vhost in your httpd.conf and set documentroot to the node containing directory. Make sure the vhost-domainname is included in a dns!

Copy node/www/config.inc.php.template into node/www/config.inc.php, and edit it according to your local settings.

Note: The nodeId is a number between 1 and 999. You schould ask the approval for your node id from the node network. In case of nodeID clash you have to reinstall the database.

Open PostgreSQL client from terminal:

```
=>createdb nodedb
=>psql -U <user> <databasename>
=> CREATE USER <username> WITH PASSWORD '<passwd>';
=> CREATE DATABASE <node db name> WITH OWNER '<username>';
=> \q
```

In a terminal:

```
# cd <path to node directory>/code/share
# psql -U <username> <node db name>
=> \i db.sql
=> \q
```

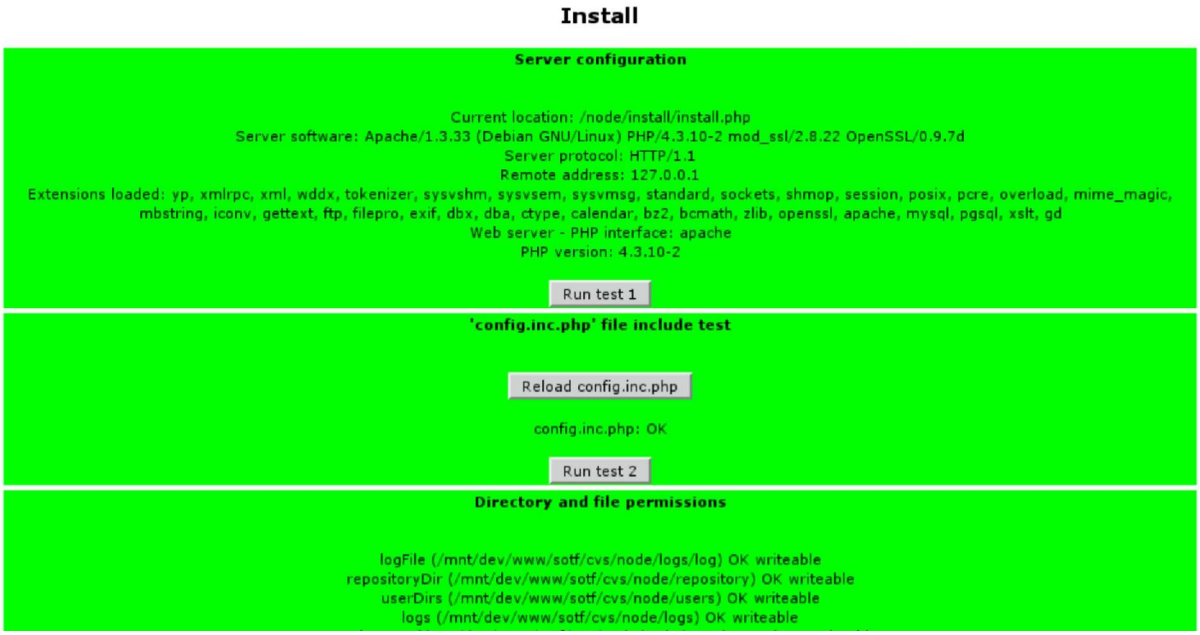
3.2.1) File permissions

You need to set for a cupple of directorys read and write permissions for your webserver. Under Debian this is default *'www-data'* for user and group (on BSD-Systems this is *www*). To set permissions do:

```
chown www-data:www-data node/logs
chown www-data:www-data node/repository
chown www-data:www-data node/users
chown www-data:www-data node/incoming
chown www-data:www-data node/www/tmp
```

3.2.2) Install script and first tests

Run install/install.php from your browser. This will test the mainconfiguration of the node. Such as database connection, filepath and permissions.



2. Figure: /node/www/install/install.php

Run Test 1: checks the server engine itself and controls if needed helpers are present on the

system

Run Test 2: controls syntax and usage of config.inc.php

Run Test 3: checks file permissions on files and directories

Run Test 4: controls database connection to postgreSQL

Run Test 5: controls database connection to the node-database

Run Test 6: controls database connection to the users-database

Run Test 7: controls vocabulary and topic tree (create it)

Run Test 8: check the authentification model for the node-admin

Run all Tests: runs all above described test cases

3.2.3) Configuration / config.inc.php

For fine-tuning of the node setup it is recommended to read the config.inc.php file and its comments. There is a number of settings and features you can en- or disable. The file resides under: */www/config.inc.php*

Database Connection:

according to your setup you need to

```
$config['nodeDbUser'] = 'DBUSER';
$config['nodeDbHost'] = 'localhost';
$config['nodeDbPort'] = '5432';
$config['nodeDbPasswd'] = 'PASSWORD';
$config['nodeDbName'] = 'node_db';
```

Authentication:

You can set the authentication model against SADM or SOTF itself by changing the array element *$config['selfUserDb']*.

```
$config['selfUserDb'] = true;
if ($config['selfUserDb']) $config['userDbClass'] = 'userdb_node';
else $config['userDbClass'] = 'userdb_sadm';
```

Main Node Settings:

```
$config['nodeId'] = "200";
// the short name of this node in the network, example: HU5, AT3
$config['nodeName'] = "NODE :: StremOnTheFly";
// whether imported files are published by default
$config['publishXbmf'] = true;
```

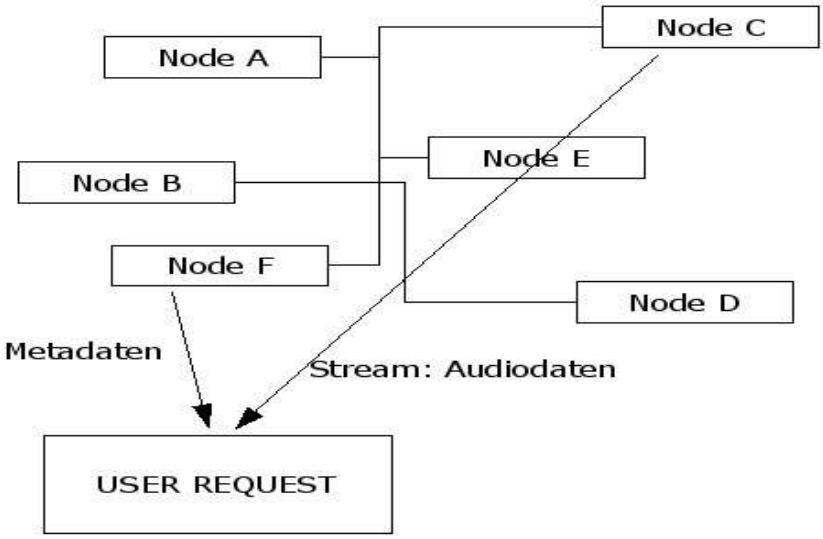
Required Formats:

```
$config['audioFormats'] = array(
          array(
            'format' => 'mp3',
            'bitrate' => '24',
            'channels' => '1',
            'samplerate' => '22050'),
          array(
```

```
            'format' => 'mp3',
            'bitrate' => '128',
            'channels' => '2',
            'samplerate' => '44100'),
          //array(
          //  'format' => 'ogg',
          //  'bitrate' => '64',
          //  'channels' => '2',
          //  'samplerate' => '22050'),
        );
```

The file config.inc.php is well documented in the file itself. Please read carefuly the comments in the file and setup the framework with your values such as pathes, language and helpers.

4) The Node



3. Figure: Basic PEAR TO PEAR Architecture

SOTF Nodes are working as a *pear to pear* network(see Figure 1). All Binary Files are stored on the node where Radiostations or other contentproducers are conntected to. All Metadata is synchroniced through the hole network of nodes using the XMLRPC standard.
On UserRequest an audiofile it is streamed from the node which is storing it. All Metadata comes from the next node in the neighborhood.

4. Figure: Node Main Interface

4.1) SOTF id's / Identifier Scheme

There is a global unique id for all SOTF objects in the network. Each global object receives a globally unique identifier. It starts with the node ID defined in config.inc.php. This ID must be a unique number with 3 digits in the nodenetwork. For the running nodes ex. this is

- 666: St. Pölten
- 667: Dornbirn
- 011: Budapest

The object ID's are using the following scheme: <node-id><obj-type><obj-id>

```
011st3 (a radio station)
```

Obj-type: 2 letters (pr = programme, se = series, etc.) Obj-id: up to 7 digits, locally unique for object-type.

4.2) Stations / Series / Programms

The Objects within in the SOTF network are devided into three main elements.

5. Figure: Station/Series/Programm

Stations: Station is for example a radio station, connected to a node-network. XBMF files are produced by stations and uploaded to the next node in the neighbourhood. Stations are the main content producers in the architecture of a node network.

Series: Continous broadcasts from a station.

Programms: The literal content of the hole network. Audiofiles and metadata as described in XBMF. Programms can be streamed and listend from the node network.

4.3) Basics and Features

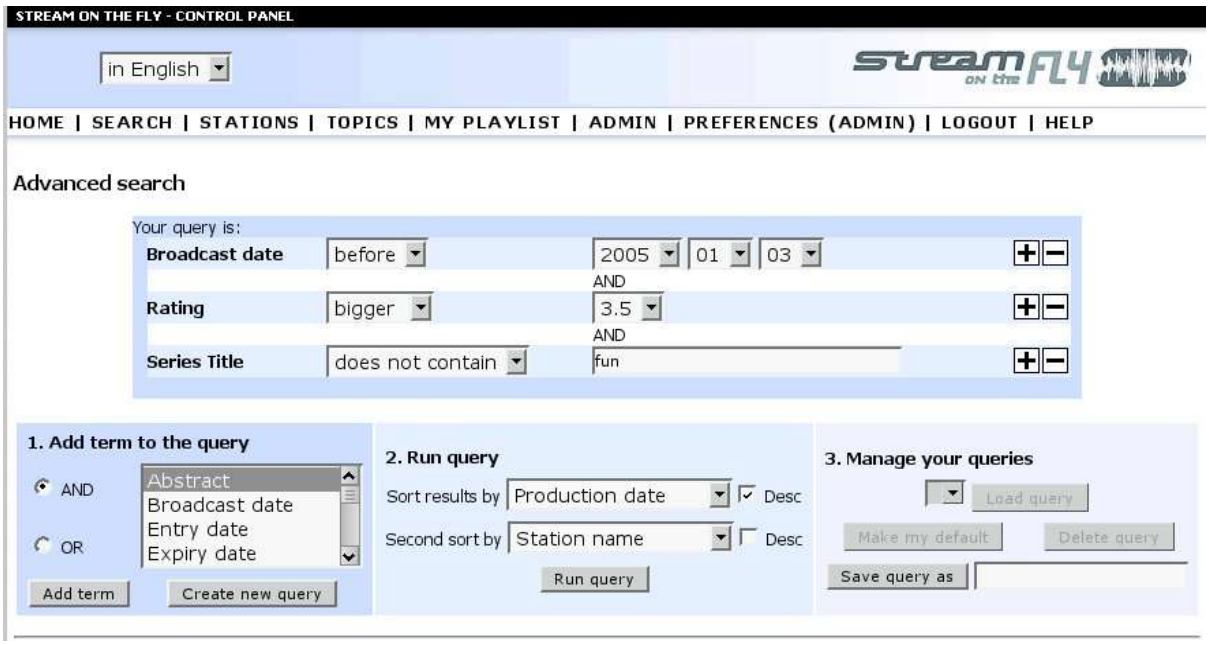
4.3.1) Search Engine

There are two main search-engines.

Files used for search engines:

```
node/www/search.php // simple search engine
node/www/advsearch.php // advanced search engine
node/www/advsearchresults.php // returns results
node/code/classes/sotf_AdvSearch.class.php // search object class
```

Please read inline-comments in the related files.



6. Figure: Advanced search engine

4.3.1.1) Create Query

On Adding a search path it is written to the object for parameter caching:

```
$paramcache = & new sotf_ParamCache();
```

....

```
$advsearch = new sotf_AdvSearch($SQLquery);
```

New queries are written to the session for forther usage:

```
$_SESSION["SQLquerySerial"] = $advsearch->Serialize();
```

Note: it's not possible to change queries. You just can save it as new and delete the old one.

4.3.1.2) Run Query

On Running a query, the node reads from the sotf_paramCache object and generates a sql-query using the provided values. This query returns a search result and calls *advsearchresults.php* to print the search results into the webinterface:

Example Search Query:

```
SELECT count(*)
  FROM (SELECT distinct programmes.*
    FROM ( SELECT sotf_programmes.*, sotf_stations.name as station,
         sotf_series.name as seriestitle,
         sotf_series.description as seriesdescription,
         sotf_prog_rating.rating_value as rating FROM sotf_programmes
    LEFT JOIN sotf_stations
      ON sotf_programmes.station_id = sotf_stations.id
    LEFT JOIN sotf_series
      ON sotf_programmes.series_id = sotf_series.id
    LEFT JOIN sotf_prog_rating
```

```
     ON sotf_programmes.id = sotf_prog_rating.id) as programmes
  WHERE published = 't'
    AND broadcast_date >= '2005-1-3'
    AND abstract  ~* '.*funstuff.*'
  ORDER BY production_date DESC, station) as count
```

4.3.1.3) Save Query

The Interface provides possibilities to generate and store database queries to run against the node-network. This queries can be stored for each single user. They are saved in the table sotf_user_prefs. On returning a user can be call back this queries and run them again against the node database.

Table: sotf_user_prefs

```
          Table "public.sotf_user_prefs"
  Column  |          Type          |  Modifiers
----------+------------------------+-------------
 id       | integer                | not null
 username | character varying(50)  | not null
 email    | character varying(100) |
 feedback | boolean                | default true
 prefs    | text                   |
```

4.3.2) Topic Tree / Genres / Roles

Topics and Genres in SOTF are used to classify the audio content in the node. Generally it is an m:n relation in the database structure. topics are stored in *sotf_topics* and *sotf_topic_trees* as parents.

Table: sotf_topics

```
   Column    |          Type          | Modifiers
-------------+------------------------+-----------
 id          | character varying(12)  | not null
 topic_id    | character varying(12)  | not null
 language    | character varying(10)  | not null
 topic_name  | character varying(255) | not null
 description | character varying(255) |
 url         | character varying(120) |
```

Table: sotf_topic_trees

```
   Column    |          Type          | Modifiers
-------------+------------------------+-----------
 id          | character varying(12)  | not null
 tree_id     | smallint               | not null
 subtopic_of | character varying(12)  |
 name        | character varying(255) |
 url         | character varying(100) |
 languages   | character varying(255) |
```

Related Files:

```
topicSearch.php
topics.php
```

```
topicTree.php
```

Topics and the hole vocabulary must be generated during the installation or afterwards.

Call: http://yournode/install/install.php and move to part 7. Click 'Create Vocublary' to fill the database tables with the right values.

4.3.3) RSS-Feeds

http://backend.userland.com/rss092

An RSS feed is a computer-readable index of your website. Instead of using HTML, which is designed for formatting on the screen, you use XML, which is designed to be easy for computer programs to read. It's a really simple way for a web site to *syndicate* its content, much like comic strip writers syndicate their strips so that they can be republished by lots of other publications. RSS is the classical way for syndication of webcontent.

```
xmlwriterclass.php       -> parse xml
rss_writer_class.php     -> write rss files
sotf_AdvSearch.class.php -> create searchresults for rss
rss.php                  -> handler file for rss calls
```

Using this code SOTF is able to generate RSS-Feeds on all Search-Querys you run against the node database. RSS Files are written to the filesystem and can be called from all other applications, designed to handle RSS.

The node does not store any RSS – data in the database.

It's very important to remember that RSS is just structured XML , that is, the elements, attributes and their order is defined by a specification. Their are three widely used RSS formats; RSS 0.91 RSS 0.91, RSS 1.0 and RSS 2.0. StreamOnTheFly uses basicaly the RSS 0.92 standard.

Example:

```
<rdf:RDF>
<channel rdf:about="http://radio.sztaki.hu/node/rss.php">
  <description>New programmes at StreamOnTheFly</description>
  <link>http://yourdomain/node/</link>
  <title>StreamOnTheFly</title>
  <dc:date>2005-01-04T14:53:17+01:00</dc:date>
  <image rdf:resource="http://yourdomain/sotflogosmall.gif"/>
  <items>
    <rdf:Seq>
      <rdf:li rdf:resource="http://radio.sztaki.hu/node"/>
    </rdf:Seq>
  </items>
  <textinput rdf:resource=
             "http://domain/search.php?language=any_language"/>
</channel>
<image rdf:about="http://yourdomain/static/sotflogosmall.gif">
  <url>
    http://radio.sztaki.hu/node/static/sotflogosmall.gif
  </url>
  <link>http://radio.sztaki.hu/node</link>
  <title>StreamOnTheFly logo</title>
  <description>World wide network of radio archives</description>
```

```
</image>
<item rdf:about="http://radio.sztaki.hu/node">
  <description/>
  <link>http://radio.sztaki.hu/node</link>
  <title>no new items</title>
</item>
<textinput rdf:about="http://yourdomain/search.php?language=any_language">
  <name>pattern</name>
  <link>
    http://yourdomain/node/search.php?language=any_language
  </link>
  <title>Search for:</title>
  <description>Search in StreamOnTheFly</description>
</textinput>
</rdf:RDF>
```

4.3.4) Statistik

related files

```
sotf_Statistics.class.php -> general statistic object description
```

This class works as an extension for sotf_Object. On each call related to statistics the table sotf_stats gets an update. On generating the index-page of the interface, the node reads the table and returns the values to the smarty object for further display in the interface.

Table sotf_stats:

```
 Column           | Type                    | Modifiers
-+-------------------------------------------------------------
 id               | integer                 | not null ...
 prog_id          | character varying(12)   | not null
 station_id       | character varying(12)   | not null
 year             | smallint                | not null
 month            | smallint                | not null
 week             | smallint                | not null
 day              | smallint                | not null
 listens          | integer                 | default 0
 downloads        | integer                 | default 0
 visits           | integer                 | default 0
 unique_listens   | integer                 | default 0
 unique_downloads | integer                 | default 0
 unique_visits    | integer                 | default 0
```

On each call related to statistics SOTF updates the table sotf_stats.

4.3.5) Feedback

This is a rating system on all programms in the network. Users are able to rate a programm. On synchronisation between the nodes this data is exchanged and updates all ratings in the network. Ratings are not immediately shown in the interface. Only after the next synchronisation run this is shown in the interface.

On code side there exists a class called *sotf_Rating.class.php*. This is an extension to the sotf_0bject class. No other files are called.

Ratings are stored in the table *sotf_rating* and *sotf_prog_rating*.
On a user rating request, the table sotf_prog_rating is updated. on synchronisation or on instant rating data from sotf_ratings is read. the method *getInstantRating()* is called and calculates the actual rating data.

Table sotf_ratings

```
    Column     |           Type           | Modifiers
---------------+--------------------------+-----------------------------
 id            | integer                  | not null default nextval ...
 prog_id       | character varying(12)    | not null
 user_node_id  | smallint                 |
 user_id       | integer                  |
 rate          | smallint                 | not null default 0::smallint
 host          | character varying(100)   | not null
 portal        | character varying(255)   |
 entered       | timestamp with time zone | not null default ...
 auth_key      | character varying(50)    |
 problem       | character varying(50)    |
```

Table sotf_prog_rating

```
      Column       |         Type          | Modifiers
-------------------+-----------------------+-----------
 id                | character varying(12) | not null
 prog_id           | character varying(12) | not null
 rating_value      | double precision      |
 nodes_only        | double precision      |
 alt_value         | double precision      |
 rating_count      | integer               | default 0
 rating_count_reg  | integer               | default 0
 rating_count_anon | integer               | default 0
 rating_sum_reg    | integer               | default 0
 rating_sum_anon   | integer               | default 0
 detail            | text                  |
```

4.3.6) My playlist

Related files:

```
playlist.php
playlistPopup.php
sotf_PlayList.class.php
```

Tabel sotf_playlists_u:

```
 Column  |         Type
---------+----------------------
 prog_id | character varying(12)
 user_id | integer
```

Tabel sotf_playlists:

```
  Column   |          Type           | Modifiers
-----------+-------------------------+-----------
 id        | integer                 | not null
 prog_id   | character varying(12)   | not null
 user_id   | integer                 |
 order_id  | integer                 |
 type      | character varying(10)   |
```

4.3.7) Rigths and permissions

Related files:

```
sotf_Permission.class.php
editPermissions.php
```

Table: sotf_permission

```
   Column   |          Type           | Modifiers
------------+-------------------------+-----------------------------------------------------------
 id         | integer                 | not null default nextval ...
 permission | character varying(20)   | not null
```

Table: sotf_user_permissions

```
    Column     |          Type           | Modifiers
---------------+-------------------------+-----------------------------------
 id            | integer                 | not null default nextval ...
 user_id       | integer                 |
 object_id     | character varying(12)   |
 permission_id | integer                 |
```

4.3.8) Editors' console

Admin and Userinterface are using the same interface. They are seperated only by permissions and rights.

```
editContact.php
editFiles.php
editLink.php
editMeta.php
editNeighbour.php
editor.php
editPermissions.php
editRight.php
editRole.php
editSeries.php
editStation.php
```

4.4) Dataimport // XBMF

The XBMF fileformat was designed to create an easily extensible file and metadata format for the exchange and transport of binary content including xml-metadata. While various formats for the exchange of audio data and metadata exist, there was not a specific format for broadcasts yet.

The metadata file uses XML to flexibly encode all necessary information. The data is modeled to provide all Dublin Core Elements.

4.4.1) Specifications // Metadata Example

```
XBMF/
   Metadata.xml
   Audio/
   Audio/SoundBinary1.mp3
   Audio/SoundBinary2.mp3
   Files/
   Files/Sound_Image.jpg
   Files/description.pdf
```

The tarred and gzipped directory is then considered to be an xbmf file. By using this standard all of which are realy available as libraries for inclusion into almost all programming environments. It is easy to write further applications in different languages to generate XBMF files and send them to the nodenetwork. In this way it is easy to create SOTF interfaces for different studio applications and content management systems.

Metadata.xml

```
<?xml version="1.0"?>
<sotfPublish>
  <title>Die Realitaet</title>
  <alternative>Ohne Kompromisse durchs Leben fliegen</alternative>
  <series>
    <id>200se2</id>
    <title>wort24</title>
    <description></description>
  </series>
  <stationid>200st3</stationid>
  <language>ger</language>
  <rights>admin</rights>
  <genre>1</genre>
  <topic>000td2</topic>
  <description></description>
  <contributor></contributor>
  <identifier>100</identifier>
  <creator>
    <entity type="organisation">
      <name type="organizationname">PublicVoiceLAB</name>
      <name type="organizationacronym">PVL</name>
      <e-mail>js@pvl.at</e-mail>
      <address>Vienna</address>
      <logo>http://www.pvl.at/logo.gif</logo>
      <uri>http://www.pvl.at/</uri>
    </entity>
  </creator>
  <publisher>
    <entity type="organisation">
      <name type="organizationname">PVL/wort24 - Testprojekt</name>
      <name type="organizationacronym">PublicVoiceLAB</name>
      <e-mail>js@pvl.at</e-mail>
      <address>Vienna</address>
      <logo>http://www.pvl.at/logo.gif</logo>
```

```
      <uri>http://www.pvl.at/</uri>
    </entity>
  </publisher>
  <date type="created">2004-02-16</date>
  <date type="issued">2003-02-16</date>
  <date type="available"></date>
  <date type="modified"></date>
  <owner>
    <auth_id>265</auth_id>
    <login>admin</login>
    <name>Admin</name>
    <role>1</role>
  </owner>
  <publishedby>
    <auth_id>265</auth_id>
    <login>admin</login>
    <name>Admin</name>
    <role>1</role>
  </publishedby>
</sotfPublish>
```

4.4.2) XBMF Structure (new)

Since 2003 there exists a new Scheme for Metadata.xml. According to the following example:

```
<metadata  xmlns="http://www.streamonthefly.org/"  xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:xbmf="http://www.streamonthefly.org/xbmf"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
<dc:title xml:lang="hun">Beszélgetés Vámos Tiborral</dc:title>
<dc:publisher  >Test_radio</dc:publisher>
<dcterms:isPartOf  >Intézetünk nagyjai</dcterms:isPartOf>
<dc:description  xml:lang="hun">Vámos Tibor a Kossuth rádió Aranyemberek című műsorának
vendége.</dc:description>
<dc:contributor  >Vámos Tibor</dc:contributor>
<dcterms:available  >2003-05-13</dcterms:available>
<dcterms:modified  >2004-06-09</dcterms:modified>
<dc:type xsi:type="dcterms:DCMIType" >Sound</dc:type>
<dc:format xsi:type="dcterms:IMT" >audio/mpeg</dc:format>
<dcterms:extent  >1844</dcterms:extent>
<dcterms:medium  >online</dcterms:medium>
<dc:language xsi:type="dcterms:ISO639-2" >hun</dc:language>
<dc:identifier  >streamonthefly:011pr7</dc:identifier>
<dc:identifier  >http://radio.sztaki.hu/node/get.php/011pr7</dc:identifier>
<xbmf:station  >Test_radio</xbmf:station>
<xbmf:series>
      <xbmf:seriestitle  >Intézetünk nagyjai</xbmf:seriestitle>
</xbmf:series>
<xbmf:genre  xml:lang="eng">Interview</xbmf:genre>
<xbmf:topic  xml:lang="eng">Sciences and Technologies / Computer
science</xbmf:topic>
<xbmf:contributor>
      <xbmf:role  xml:lang="eng">Interviewee</xbmf:role>
      <xbmf:name  >Vámos Tibor</xbmf:name>
      <xbmf:intro  >Vámos Tibor 1986 óta a Magyar Tudományos Akadémia
Számítástechnikai és Automatizálási Kutató Intézet Intézeti Tanácsának
elnöke. </xbmf:intro>
</xbmf:contributor>
</metadata>
```

4.5) Communication

The following text goes ahead with documentationwork done by the development team of SOTF during programming tasks.

- Keep table of available nodes & stations up-to-date
- Keep metadata synchronized
- Control of node network

Each node defines its id, and defines its so-called neighbours (with id and URL). The node will exchange data only with its neighbours. In this way neighbours provide authentication for nodes: you cannot connect to a node if your node is not in the neighbour list of that node.
Data exchange is done via so-called sync requests. Each node should periodically send sync requests to its neighbours.

4.5.1) Syncronisation and Network

A sync request contains the local changes on the node since the last sync. In return, the node will get all outside updates.
Each piece of data that is synced is provided with a timestamp of the last change date. So, if a node gets outdated data from a neighbour, it is simply dropped.
The fact that an item/station has been deleted are stored and propagated similarly to other replicated data. In this way deletions are done on remote nodes as well.
Extent of replication
It is unfeasible to replicate all data around the network. Current design replicates so-called discovery metadata only, i.e. What is needed for transparent searching in the network. It won't replicate the following:

- audio and other accompanying files (e.g. images)
- statistics and feedback on items (number of listens, comments to authors, etc.)
- logos and jingles of stations (though caching may be relevant)

4.5.1.1) Adding new nodes

A new node has to be entered as a neighbour on at least one other node. After this, the node will be automatically registered at the time of the first sync request from the new node.
As mentioned earlier, at least one other node has to register a new node as a neighbour. Otherwise the new node cannot participate in the network.

4.5.1.2) Detect nodes that are down

When a neighbour gets no answer for its sync request, it will mark the node as being 'down', and this information will spread across the network with other data. When a node is up again, this fact will spread across the network, similarly.

4.5.1.3) Network load

Nodes can control their network traffic:

- Will get inbound traffic mostly from approved neighbours

- Will get metadata updates only in return to their sync requests.

So fewer neighbours and less frequent syncing means less network traffic.

4.5.1.4) Speed of replication

The time in which all nodes receive a new update may greatly depend on the network setup. In the worst case, the update will wait for the time of next sync on each node during node traversal. If this is 5 minutes, and your node is 5 nodes away from a node, it may take half an hour. To avoid this, we suggest a highly connected network topology, where the shortest path to all nodes is reduced to a reasonably small number.

4.5.1.5) Redundancy in updates

A node may get the same update from several neighbours. While this wastes bandwidth, has no other effect to the database, as redundant updates are simply discarded (based on timestamp).

4.5.1.6) Node IDs

There is no mechanism to provide globally unique node ID's. As there will be few nodes, and neighbours need to register them, uniqueness can be easily guaranteed by the cooperative human process of adding a new node.

We suggest a naming scheme for nodes: <*country-code*><*ordinal number*> So the first node in Hungary is HU1, the second is HU2.

4.5.1.7) Getting item details

For each radio show there is a page that describes that radio show and provides "listen" and "download" buttons for the audio, etc. (get.php) As the local node (wrt. To the show) has the most data about a radio show (see Extent of replication), if possible, the local node should render this page, not the node which the user is currently accessing. If the node storing the show is marked as down, any other node may present the detail page.

This does not break the principle of transparent searching/browsing.

4.5.2) XMLRPC

XMLRPC is a http/tcpip based communication model which is defined in RFC 3529. Object or array data is serialzed in an XML-structure. This structure is plainly sent by http (port 80)

Acutally supported RPC calls:

```
sotf.sync         -> node synchronisation
sotf.forward      -> forward new data
sotf.cv.listnames ->
sotf.cv.get       -> used by station to collect controled vocabulary
portal.query      -> execute a query on the node
portal.playlist:  -> collect metadata for shows
portal.events:    -> send portal events to node
```

4.6) Cron Jobs

Cron is the unix-daemon to execute scheduled commands. In the SOTF-Framework, cronjobs are used for synchronisation of metadata and binary exchange. On BSD and Linux server-engines *vixie cron* is per default included.

The following main description for cronjob comes from cron's manpage on debian GNU Linux:

cron searches its spool area (*/var/spool/cron/crontabs*) for crontab files (which are named after accounts in */etc/passwd*); crontabs found are loaded into memory. Note that crontabs in this directory should not be accessed directly - the crontab command should be used to access and update them.

cron also reads */etc/crontab*, which is in a slightly different format. Additionally, cron reads the files in */etc/cron.d*: it treats the files in */etc/cron.d* as extensions to the */etc/crontab* file (they follow the special format of that file, i.e. they include the user field). The intended purpose of this feature is to allow packages that require finer control of their scheduling than the */etc/cron*. {daily,weekly,monthly} directories allow to add a crontab file to */etc/cron.d.* Such files should be named after the package that supplies them. Files must conform to the same naming convention as used by run-parts: they must consist solely of upper- and lower-case letters, digits, under-scores, and hyphens. If the -l option is specified, then they must conform to the LSB namespace specification, exactlyas in the *--lsbsysinit* option in run-parts.

4.6.1) Cron jobs for SOTF

In the SOTF framework is just one cronjob for synchronisation in the connected nodenetwork. This calls cron.php located in *node/www/cron.php*. This script has to be called periodically (e.g. using wget within a cron-job) and it performs all periodic maintenance tasks for the node server:

```
perform expensive updates on all objects: sotf_Object::doUpdates();
synchronise with the network: $neighbour->sync();
Forward messages to remote nodes: $node->forwardObjects();
import arrived XBMF-files:
$id = sotf_Programme::importXBMF($config['xbmfInDir'] . "/$xbmfFile",
$config['publishXbmf']);
```

4.7) Database

SOTF Node is using postgreSQL as database backend. The database schema is designed as RDBMS (relational database modelling system). There is not any object abstraction layer in the schema.

4.7.1) Relational Schema

Since the database schema for SOTF Node is quiet complex the grafics are not in this document. please use the following links to the images to display schema-grafics.

- sotf.streamonthefly.org/db/node_db_schema_circular.jpg
- sotf.streamonthefly.org/db/node_db_schema_hierarchical.jpg
- sotf.streamonthefly.org/db/node_db_schema_orthogonal.jpg

4.7.2) Tableslist and usage

sotf_series	metadata for series related to station
sotf_comments	content comments stored by user, programm and portal
sotf_contacts	user/editors contact data
sotf_deletions	stores ids to delete them in the hole network on synchronisations
sotf_extradata	additional metadata for programms
sotf_genres	list of genres in all available languages
sotf_links	additional links for programms
sotf_media_files	additional media files for programms
sotf_neighbours	list of included nodes in the network
sotf_node_objects	available node objects in the network using node ids
sotf_nodes	metadata for nodes in the network
sotf_object_roles	mapping table for objects, contactdata and role data
sotf_object_status	mapping table for objects and nodes
sotf_other_files	metadata additional files for programmes
sotf_permissions	mapping table for permissions and objects
sotf_playlists	mapping table for users and programms to generate playlists
sotf_portals	metadata for portals according to the user
sotf_prog_rating	rating collection for programmes
sotf_prog_refs	reference table for programms, stations and portals
sotf_prog_stats	statistics for programmes related to stations
sotf_prog_topics	mapping for programmes and topics
sotf_programmes	metadata and mapping for programmes. related to station and series
sotf_ratings	store ratings for programms
sotf_rights	store data for rights related to programms
sotf_role_names	store role names with language and role_id
sotf_roles	roles for programmes related to sotf_role_names
sotf_station_mappings	mapping station and nodes
sotf_stations	metadata for stations per id
sotf_stats	statistic overview for stations and programms
sotf_streams	collected data about available streams
sotf_to_forward	synchronisation mappings
sotf_to_update	updates for existing programms in the node network

sotf_series	metadata for series related to station
sotf_topic_tree_defs	mapping for topic-id, supertopic and tree_id
sotf_topic_trees	metadata for topic trees (parent, tree-matching, name, url and language)
sotf_topics	metadata for topics
sotf_topics_counter	counter for topics
sotf_unique_access	store access for statistics
sotf_user_history	history for user interactions
sotf_user_permissions	mapping for user, object and permission
sotf_user_prefs	stored preferences for users
sotf_user_progs	stored programmes for users
sotf_users	metadata for users
sotf_vars	internal variables

4.8) Codestructure

4.8.1) Files

showStation.php	show connected stations
addFiles.php	add files to the node (audio, binary, text)
addToSeries.php	adds a programm to a series
admin.php	main for administrative interface
advsearch.php	advanced search engine
advsearchresults.php	results for advanced search engine
changeStation.php	change the connected station
closeAndRefresh.php	close and refresh a certain page
config.inc.php	main configuration
config.inc.php.template	template for configurations
convert.php	converts audio files
createContact.php	generate contact data
createNeighbour.php	create a neighbour node
createSeries.php	create a new serie
createStation.php	create a new station
cron.php	called by unix-cronjob for synchronisation
editFiles.php	edit file data
editLink.php	edit link data
editMeta.php	edit meta data
editNeighbour.php	edit neighbour nodes
editor.php	main for editor interface
editPermissions.php	edit programms permissions

showStation.php	show connected stations
editRight.php	edit programms rights
editRole.php	edit producers roles
editSeries.php	edit series
editStation.php	edit station
export.php	export data
functions.inc.php	cupple of useful functions used in all code parts
getFile.php	get file informations (getID3)
getIcon.php	get icon for programm, station or series
getJingle.php	get jingle audio file
get.php	get Audio files via ID
getUserFile.php	get related user files
help.php	display help content for users
index.php	application cockpit ;-)
init.inc.php	initialisation of objects and constants
listen.php	call audio file for streaming
login.php	login and set session
logout.php	destroy session and logout
log.php	write log files
manageFiles.php	manage files
phpinfo.php	just a testcall for phpinfo()
playlist.php	display playlists
playlistPopup.php	display playlists in popup
portal_upload.php	upload data to portal
register.php	register new users
rss.php	write rss feeds
search.php	easy search engine
showContact.php	show contact data
showContactProgs.php	show programms related to contact
showSeries.php	show series
startStream.php	start an audio stream
stations.php	manage station data
topicSearch.php	manage topics
topicTree.php	display topicTree
viewConfig.php	view node config
xmlrpcServer.php	xmlrpc php-helper

4.8.2) Classes / Objects

sotf_ParamCache.class.php	cache parameters for advanced search
sotf_Metadata.class.php	handle metadata
sotf_PlayList.class.php	handle playlists
sotf_UserPlaylist.class.php	handle userPlaylists
sotf_AudioCheck.class.php	check audio
sotf_Neighbour.class.php	check and manage neighbour nodes
sotf_Programme.class.php	check and manage programms
sotf_UserPrefs.class.php	manage userpreferences
sotf_AudioFile.class.php	handle audio files
sotf_Node.class.php	mainhandler for node
sotf_Rating.class.php	handles content ratings
sotf_Utils.class.php	useful methods collected in an object
sotf_Blob.class.php	blob extension for NodeObject
sotf_NodeObject.class.php	extension for Object (sotf.Object.class.php)
sotf_Repository.class.php	handles audio repositories
sotf_Vars.class.php	variables and topics
sotf_ComplexNodeObject.class.php	extension for the node object
sotf_Object.class.php	the main node object itself
sotf_Series.class.php	handles series
sotf_Vocabularies.class.php	handles vocabulary
sotf_Contact.class.php	handles contact data
sotf_Page.class.php	handles page (smarty)
sotf_Station.class.php	station extension for complex_node_object
sotf_File.class.php	general file and path handling
sotf_Statistics.class.php	read/write of statistics
sotf_FileList.class.php	handle and create file lists
sotf_Permission.class.php	handle programm permissions
sotf_User.class.php	general user object

4.9) NODE Interface

SOTF templates are designed as smarty-templates since the hole framework is using this template-engine to generate the interfaces. Permissions and a cupple of other functions are defined within the templates. For more information about smarty visit the smarty section in this document or read the docs on smarty.php.net

4.9.1) Interface HTML Files

about.htm	displays about information for the node
addFiles.htm	adds files to programms
addToSeries.htm	add programms to series
admin.htm	main admin interface
advsearch.htm	interface for advanced search
advsearchresults.htm	displays search results for advanced search engine
changeStation.htm	select stations to change
createContact.htm	interface to create contact data
createNeighbour.htm	interface to create neighbours in the node network
createSeries.htm	interface to create new series
createStation.htm	interface to create new stations
debug.tpl	interface for debugging and error data
editContact.htm	interface to edit contact data for users
editFiles.htm	interface to edit files related to a programm
editLink.htm	interface to edit links related to a programm
editMeta.htm	interface to edit metadata for programms
editNeighbour.htm	interface to edit neighbourhood nodes in the network
editor.htm	editors console
editPermissions.htm	edit permissions for users/editors related to programms
editRight.htm	edit rights for programms
editRole.htm	edit roles relation programms/user/editors
editSeries.htm	edit series in a station
editStation.htm	edit stations itself
error.htm	display all error data / called if SOTF error-handler is active
get.htm	?????
help.htm	displays help for node interface
index.htm	cockpit of interface
install.htm	edit database connection for the node
login.htm	login interface

about.htm	displays about information for the node
main_frame_left.htm	frameset left for topics
main_frame_right.htm	frameset right if topics are displayed
main.htm	main interface
manageFiles.htm	manage files for programms
main_popup.htm	main html framework for popup windows
playlist.htm	display playlists for users
playlistPopup.htm	display playlists for users in a popup window
portal_upload.htm	interface to upload data to a portal for users
register.htm	register new users
rssContributors.htm	rss rows for contact data and roles
rssListen.htm	rss lists for audio files
rssMeta.htm	rss lists for metadata
rssRating.htm	rss lists for ratings
search.htm	search interface
showContact.htm	display contact data for users
showContactProgs.htm	display contact data for programms
showStation.htm	display station data
splitList.htm	splits a resultlist in various pages
stations.htm	displays available stations
test.htm	just for testing purposes
topicSearch.htm	dislpay search for topics
topics.htm	displays all topics
topicTree.htm	displays all topics as tree

4.9.2) Interface language files

SOTF is translated in 3 languages. This is handled by language configs using the smarty function *'config_load'.*

```
{config_load file="eng.conf"} -> reads configuration for english wording
```

The files are located under node/code/configs

```
# general translations
ok=Ok
cancel = Cancel
```

.....

```
admin = Owner
```

to include wording in the interface follow the smarty definitions:

```
<h3>{#admin#}</h3>
```

5) The Portal

The portal in the StreamOnTheFly architecture is a personaliced website for users which provides content created by search queries against the node-network. The actual version included in the cvs on sourceforge.net is still a alpha pre-release. So keep in mind that you are using a software which is not yet finished and **not** released as a stable version.

5.1) Principles

The Portal-Software in the StreamOnTheFly framework offers the possibility to customize pages for authentificated users. One can collect content with stored search queries against the node network and. The way to go is very easy. On a running portal-engine create a new user and portal. The system sends you a confirmation message with a code-number included. return to the portal engine and enter login-data and the confirmation code to start the new created portal. In the new portal you can edit the homepage, the style and upload search-queries from the node itself. This helps you to create a personalized website using content from the hole SOTF-network.

5.2) Installation

5.2.1) Requisits:

The SOTF-Portal has the same requirements then the NODE itself. It needs Apache as a running webserver, PHP4, and PostgreSQL as database backend. As helpers you have to install PEAR (bundled with PHP / see section PEAR in this document) and the Smarty template engine (see section smarty in this document)

An operational StreamOnTheFly node server.

Note: On Apache 2 you have to set 'AcceptPathInfo on'! Portal will not work if register_globals is *ON* in PHP.

5.2.2) Preinstall:

You can get the Portal software from sourceforge.net cvs or as a tarball. please keep in mind, that the latest version is allways located in the cvs.

Check that the PHP scripts in portal/www directory can be run by your web server.

Make some directories under portal writeable by the web server process (e.g. logs, code/templates_c)

```
chown www-data:www-data logs
chown www-data:www-data code/templates_c
```

Create a new database in Postgres, and feed it with portal/code/doc/db.sql

as user postgres (or pgsql on UNIX-like systems):

```
createdb sotf_portal
psql sotf_portal < portal/code/doc/db.sql
```

5.3) Configuration

Copy portal/www/config.inc.php.template into portal/www/config.inc.php, and edit it according to your local settings.

The most importantend settings are:

```
// your sql connection to the portal database
$nodeDbUser = 'user';
$nodeDbHost = 'localhost';
$nodeDbPort = '5432';
$nodeDbPasswd = '';
$nodeDbName = 'portal';

// Your nearest SOTF node
$sotfSite = 'http://sotf2.dsd.sztaki.hu/node/www/';
```

Please make sure that path-related settings in the config.inc.php file are valid! You will find comments to all configuration-values in config.inc.php.

5.4) Code Structure

Code is structured as usual in SOTF in two sections: www/ and code/

```
code/
  classes/      <- all logical object-classes
  configs/      <- language configurations
  doc/          <- documents / actualy just a database dump
  templates/    <- all smarty template-files
  templates_c/  <- compiled templates (writeable by apache)
  xmlrpc/       <- xmlrpc client and server

www/    <- executeable files
  admin/
  static/
```

5.4.1) Classes

db_Wrap.class.php	General database wrapper
Portal.Rating.php	Ratings and callback to the nodenetwork
rpc_Utils.class.php	Utils for communication between node and portal
sotf_Page.class.php	General page class (generates frontend)
sotf_Portal.class.php	General portal class
sotf_Utils.class.php	Sotf-related utilities
sotf_Vars.class.php	Handles sotf-variables

5.4.2) Executable Files

portal_upload.php	Upload files to your portal
config.inc.php	General configuration (see section configuration)

portal_upload.php	Upload files to your portal
edit_text.php	Edit text and metadata
functions.inc.php	Useful functions to include in the code
index.php	Index / general eventhandling
init.inc.php	Initialisation of objects and settings
log.php	Write log files
phpinfo.php	Just displays the local php-settings
portal_login.php	Login handling
portal.php	Portal core page
portal_popup.php	Portal core popup-page
portal_template.php	Create a new portal from template
viewConfig.php	View configuration in the webbrowser

5.5) Database Strucutre

The Portal in SOTF uses it's own database in postgreSQL. This database is created within the installation of the portal. In this section some of the most important tables are described:

5.5.1) List of tables

You find here a list of the tables included in the portal-database. On the instructions on the right you see the

portal_cache	Caches some content coming from the nodenetwork for faster access
portal_events	
portal_files	Stores additional files for programms
portal_prglist	Programmlist for portals
portal_prog_rating	Cache table for actual ratings
portal_programmes	Table for matchings between prglist and portals
portal_queries	Stores queries to run against the nodenetwork
portal_ratings	Store rating results after syncronisation
portal_settings	Save personal settings for portals
portal_statistics	Save statistics after synchronisation
portal_templates	Templates for personal portals
portal_users	User data for authentification and contact
portal_vars	Save portal variables
programmes_comments	Store comments on programms
programmes_description	Store description for programms

5.5.2) Table description

In the following section you find a full description of the most important tables in the portal database. Please use tools like phppgadmin or dbvisualizer (as described in this document) to display the hole database structure.

portal_files

In this table the portal stores all additional files for a programm related to a personal portal.

```
   Column     |         Type          | Modifiers
--------------+-----------------------------------------------------
 id           | integer               | not null default nextval ...
 portal_id    | integer               | not null
 progid       | character varying(20) |
 file_location | character varying    | not null
 filename     | character varying     |
```

portal_queries

The portal uses this table to store queries to run against the nodenetwork. The queries are related to the personaliced portals using portal_id and save the queries using a given 'name':

```
  Column   |       Type        |Modifiers
-----------+-------------------------------------------------------
 id        | integer           | not null default nextval ...
 portal_id | integer           | not null
 name      | character varying | not null
 query     | character varying |
```

portal_prglist

Lists all programms for a given portal using portal_id to identify the user-portal.

```
  Column   |       Type        |Modifiers
-----------+-------------------------------------------------------
 id        | integer           | not null default nextval
 portal_id | integer           | not null
 name      | character varying |
```

portal_prog_rating

Stores ratings on programmes using prog_id. The rating does not depend to the portal. It's used as rating content for the hole network after synchronisation.

```
      Column       |         Type          | Modifiers
-------------------+-----------------------------------------------------
 id                | integer               | not null default nextval
 prog_id           | character varying(12) | not null
 rating_value      | double precision      |
 alt_value         | double precision      |
 rating_count      | integer               | default 0
 rating_count_reg  | integer               | default 0
 rating_count_anon | integer               | default 0
```

```
rating_sum_reg    | integer               | default 0
rating_sum_anon   | integer               | default 0
detail            | text                  |
```

portal_programmes

Stores all programmes for a portal. It uses the values progid and portal_id to match the content-items.

```
   Column   |         Type          | Modifiers
------------------------------------------------------------
id          | integer               | not null default nextval
portal_id   | integer               | not null
progid      | character varying(20) | not null
prglist_id | integer               |
```

portal_queries

Here we store all queries running against the node. Each query is stored by a given name and related to a portal (using portal_id for identifying)

```
  Column   |       Type         | Modifiers
-----------------------------------------------------------
id         | integer            | not null default nextval ...
portal_id | integer            | not null
name       | character varying | not null
query      | character varying |
```

portal_ratings

In this table direct response from authentificated users to programms is stored.

```
  Column   |           Type           | Modifiers
-----------+--------------------------------------------------------
id         | integer                  | not null default nextval
prog_id    | character varying(12)    | not null
user_id    | integer                  |
rate       | smallint                 | not null default 0::smallint
host       | character varying(100)   | not null
entered    | timestamp with time zone | not null default
auth_key   | character varying(50)    |
problem    | character varying(50)    |
```

portal_settings

The main table for customized portals. It stores the tempaltes, different settings and the admin-password for the portal.

```
   Column    |       Type         | Modifiers
-------------------------------------------------------------------
id           | integer            | not null default nextval
name         | character varying | not null
template_id | integer            |
admin_id     | integer            |
settings     | character varying |
password     | character varying | not null
```

portal_users

The portal users are able to login in the portal engine. This table is used for authentification and the activate key sent to the user on creating a new portal.

```
  Column   |            Type             | Modifiers
-----------+-----------------------------+-----------------------------------------------------
 id        | integer                     | not null default nextval ...
 portal_id | integer                     | not null
 name      | character varying           | not null
 password  | character varying           | not null
 email     | character varying           |
 activate  | integer                     |
 timestamp | timestamp without time zone | not null default date ...
```

portal_vars

The portal variables are stored in this table. The administrators can create and edit values identified by there name and holding content stored in the field value

```
 Column |          Type          | Modifiers
--------+------------------------+-----------------------------------------------------
 id     | integer                | not null default nextval ...
 name   | character varying(32)  |
 value  | character varying(255) | not null
```

programmes_comments

This table is used to store text-comments to each programm. They are also related to a portal using portal_id.

```
  Column   |            Type             | Modifiers
-----------+-----------------------------+-----------------------------------------------------
 id        | integer                     | not null default nextval ...
 portal_id | integer                     | not null
 progid    | character varying(20)       |
 user_id   | integer                     |
 reply_to  | integer                     |
 path      | character varying           |
 timestamp | timestamp without time zone | not null default date ...
 ipaddr    | character varying(24)       | not null
 email     | character varying(100)      |
 title     | character varying           |
 comment   | character varying           |
 level     | smallint                    |
```

programmes_description

This is the content description for programs it's additional to the metadata coming from the nodenetwork on imported programms.

```
  Column   |         Type          |  Modifiers
-----------+-----------------------+-----------------------------------------------------
 id        | integer               | not null default nextval ...
 portal_id | integer               | not null
 progid    | character varying(20) | not null
 teaser    | character varying     |
 text      | character varying     |
```

6) UNIX Daemons

6.1) Apache 1.3.x/PHP4.x

Apache is a HTTP server written as "a patchy server" (hence its name) enhancement to the 1995 EOL'ed NCSA server. Due to its flexible modular concept, rubustness and easy configurability Apache has become the most popular web server on the net (as of January 2005 68% worldwide according to http://netcraft.com). Apache supports keep-alive persistent connections, virtual hosts, rewrite abilities, dynamic shared object, piped logs, POSIX threads and IPv6 – just to name a few.
PHP, which stands for "PHP: Hypertext Preprocessor" is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. Its syntax draws upon C, Java, and Perl, and is easy to learn. The main goal of the language is to allow web developers to write dynamically generated webpages quickly, but you can do much more with PHP.
You need a basic installation of Apache. SOTF is mainly tested on Apache 1.3.x. It works well under Apache 2.0.x. Since there are no specials used in the code it should run under PHP5 as well. Till now it's untested.

Installation for Debian/GNU Linux

```
apt-get install apache php4 php4-domxml php4-gd php4-imagemagick php4-pgsql
php4-xslt
```

Installation for FreeBSD

```
cd /usr/ports/www/apache13
make install clean
cd /usr/ports/lang/php4
make install clean
cd /usr/ports/textproc/php4-domxml
make install clean
cd /usr/ports/graphics/php4-gd
make install clean
cd /usr/ports/databases/php4-pgsql
make install clean
cd /usr/ports/textproc/php4-xslt
make install clean
cd /usr/ports/graphics/ImageMagick
make install clean
```

6.2) PostgreSQL

PostgreSQL is an object-relational database management system, supporting almost all SQL constructs, including subselects, transactions, and user-defined types and functions. It is the most advanced open-source database available anywhere. PostgreSQL is based on POSTGRES Version 4.2 developed at the University of California at Berkeley Computer Science Department. It supports SQL92 and SQL99 and offers many modern features: complex queries; foreign keys; triggers; views; transactional integrity; multiversion concurrency control.

SOTF framework is tested on 6.8 up to 7.4. It's not tested on 8.0.x. Because of several specials, offered by postgreSQL (such as sequenzing), you can not easily switch to another database backend like mySQL. The use of postgreSQL provides us a high performance and quiet advanced backend.

Installation for Debian/GNU Linux

```
apt-get install postgresql
```

Installation for FreeBSD

```
cd /usr/ports/databases/postgresql74-server
make install clean
```

6.3) ProFTPd (optional)

ProFTPd is a highly configurable File Transfer Protocol daemon which supports hidden directories, virtual hosts, and per directory ".ftpaccess" files. It uses a single main configuration file, with a syntax similar to Apache. It has support for chrooting, maximum numbers of clients, denyfilters and many more. Various authentication backends can be used: PAM, MySQL, PostgreSQL and LDAP.

Installation for Debian/GNU Linux

```
apt-get install proftpd
```

Installation for FreeBSD

```
cd /usr/ports/ftp/proftpd
make install clean
```

7) UNIX Helpers

7.1) OGG – Tools

OGG Vorbis is a fully open general-purpose audio and music encoding format similar to MPEG-4's AAC and TwinVQ, the next generation beyond MPEG audio layer 3. The compressed audio format allows mid to high quality (8kHz-48.0kHz, 16+ bit, polyphonic) audio and music at fixed and variable bitrates from 16 to 128 kbps/channel. The vorbis-tools contains oggenc (an encoder), ogg123 (a playback tool), ogginfo (displays ogg information), vcut (ogg file splitter), and vorbiscomment (ogg comment editor)

Installation for Debian/GNU Linux

```
apt-get install vorbis-tools
```

Installation for FreeBSD

```
cd /usr/ports/audio/vorbis-tools
make install clean
```

7.2) transcode

(optional for video encoding)

Transcode is a text-console based utility for video stream processing. Decoding and encoding is

done by loading modules that are responsible for feeding transcode with raw video/audio streams (import modules) and encoding the frames (export modules). It supports elementary video and audio frame transformations, including de-interlacing or fast resizing of video frames and loading of external filters. A number of modules are included to enable import of DVDs on-the-fly, MPEG video, DivX 4.xx – just to name some few. Additional export modules to write single frames (PPM) or YUV4MPEG streams are available, as well as an interface import module to the avifile library.

Installation for Debian/GNU Linux

First of all you have to use nerim.net as an additional apt source:

```
echo "deb ftp://ftp.nerim.net/debian-marillat/ unstable main" \
>> /etc/apt/sources.list
```

Then checkout and install the transcode package:

```
apt-get update
apt-get install transcode
```

Installation for FreeBSD

```
cd /usr/ports/multimedia/transcode
make install clean
```

7.3) Lame

LAME (Lame aint MP3 encoder) is an highly evolved LGPL MP3 encoder. It's most important features are: MPEG1,2 and 2.5 layer III encoding, psycho acoustic and noise shaping, CBR (constant bitrate) and two types of variable bitrate.

Installation for Debian/GNU Linux

Also for this package you need nerim.net apt sources (see 6.2 transcode)

```
apt-get install lame
```

Installation for FreeBSD

```
cd /usr/ports/audio/lame
make install clean
```

7.4) Sox

SoX (also known as Sound eXchange) is a general purpose sound converter/player/recorder. SoX translates sound samples between different file formats, and optionally applies various sound effects (like chorus, fade in or out, swap stereo channels, just to name a few). SoX is able to handle formats like .ogg (vorbis), mp3, wav, aiff, voc, snd, au, gsm and several more.

Installation for Debian/GNU Linux

```
apt-get install sox
```

Installation for FreeBSD

```
cd /usr/ports/audio/sox
make install clean
```

7.5) rsync

rsync is a program that allows files to be copied to and from remote machines in much the same

way as rcp. The rsync remote-update protocol allows rsync to transfer just the differences between two sets of files across the network connection, using an efficient checksum-search algorithm described in the technical report that accompanies this package. rsync needs to be installed both on the client as on the server. It is able to use ssh as transport agent to encrypt the data stream, supports copying links, devices, owners, groups and permissions and can use exclude modes just like tar and cvs.

Installation for Debian/GNU Linux

```
apt-get install rsync
```

Installation for FreeBSD

```
cd /usr/ports/net/rsync
make install clean
```

7.6) ImageMagick

ImageMagick is a package for display and interactive manipulation of images. The package includes tools for image conversion, annotation, compositing, animation, and creating montages. ImageMagick can read and write many of the more popular image formats (e.g. JPEG, TIFF, PNM, XPM, Photo CD, etc.). All manipulations can be achieved through shell commands. Possible effects: colormap manipulation, channel operations, thumbnail creation, image distortion, image scaling, image rotation, color reduction, merging of images and many more.

Installation for Debian/GNU Linux

```
apt-get install imagemagick
```

Installation for FreeBSD

```
cd /usr/ports/graphics/ImageMagick
make install clean
```

8) PHP Libraries

8.1) PEAR // The Database Interface

PEAR is short for "PHP Extension and Application Repository". It offers a structured library of open-sourced code written in PHP. SOTF is using the Pear-Class *DB* as a general database interface.

There are several ways to get the latest Pear-Scripts:

You can easily download the scripts from http://pear.php.net. In the packaging system of most of the Linux/Unix distributions you can use the installer to get pear. Systems like freeBSD or Debian provide you a small shellscript to get and install pear-php scripts on your system.

The code in PEAR is partitioned in "packages". Each package is a separate project with its own development team, version number, release cycle, documentation and a defined relation to other packages (including dependencies). Packages are distributed as gzipped tar files with a description file inside, and installed on your local system using the PEAR installer.

Principally there are two types of packages: source packages (containing source files only), and binary packages (containing platform-specific binary files, and possible source files). Installing

source packages with C code obviously requires a C build environment.

The used DB API works on the top of PHP Database functions. It's compatible to the following SQL-servers: dbase, fbsql, interbase, informix, msql, mssql, mysql, mysqli, oci8, odbc, pgsql, sqlite and sybase.

8.1.1) Installation

Installation for Debian GNU/Linux:

```
apt-get install php4-pear
```

Installation for FreeBSD:

```
cd /usr/ports/devel/pear-PEAR/
make install clean
cd /usr/ports/databases/pear-DB
make install clean
cd /usr/ports/devel/pear-XML_Parser
make install clean
```

8.1.2) Mode of operation

PEAR - DB is working as a database abstraction layer. PEAR Classes are included in the application in the same way as other PHP Classes written for SOTF. It is required in the init.inc.php script which reads the database configuration from config.inc.php and initialize the PEAR Database Object.

7. Figure: General Workflow of PEAR DB-Interface

This is the main filestructure to invoke PEAR using the PostgreSQL interface.

```
/pear/DB.php
/pear/DB/common.php
/pear/DB/psql.php
```

DB.php is invoked by the application and reads the defined Constant 'DB_TYPE'. According to this value it requries the adequate DB interface. In the case of SOTF (using postgreSQL) this is DB/common.php and DB/psql.php. Take a look in the PEAR filestructure to see the provided interfaces for other SQL-servers.

8.1.3) PEAR DB-Object for PostgreSQL

The DB_DataObject is a SQL Builder and Data Modeling Layer built on top of PEAR::DB. Its main purpose is to build and execute SQL statements against the given database server, based on the object variables. The object provides us a simple consistent API to access and manipulate that data.

So what does that mean in english? Well, if you look around at some of the better written PHP applications and frameworks out there, you will notice a common approach to using classes to wrap access to database tables or groups. In PEAR this wrapper is done as a cupple of methods which access the database and returns you a result set.

8.1.3.1) Provided Methods

You can find a list of all methods on http://pear.php.net/manual/en/package.database.php

Basicaly SOTF uses $db as a global object within the sources. It is defined in methods as *global $db* or inside of classes as *$this->db = & $db.*

`$db->query()`	Executes all given SQL statements. Returns row data or error.
`$db->getOne()`	Executes given select statement. Returns one single row. If more rows/columns are affected it returns the first value as scalar.
`$db->getAssoc()`	Executes given select statement and returns data as an associative array.
`$db->getAll()`	Executes select statement and returns data as an associative array or an object depending on given parameter DB_FETCH_MODE
`$db->getCol()`	Executes select statement and returns a single colomn. If more colomns are affected it returns the first.
`$db->getRow()`	Executes select statement and returns a single row.
`$db->begin()`	Starts postgreSQL transaction control
`$db->rollback()`	Executes a rollback since the last transaction started without commit
`$db->commit()`	Executes commit to the active transaction control
`$db->limitQuery()`	Limits the result set of a query by a given number

For Methods returning constructed results sets of data one can decide to use fetch modes to create the needed result set as an associative array or an object. There are two values for this option: DB_FETCHMODE_ASSOC and DB_FETCHMODE_OBJECT.

```
$db->getAll("SELECT foo BROM baa", DB_FETCHMODE_OBJECT)
```

8.1.3.2) Error Object

PEAR DB provides us an error object using: DB::isError(). It returns TRUE on error and FALSE if now error occurred. The method getMessage() would return you the active error code from the object:

```
if(DB::isError($db)) die ($db->getMessage());
```

8.2) Smarty // the Template Engine

http://smarty.php.net

Smarty comes with the node software (cvs or tar-ball from sourceforge). There is no installation required.

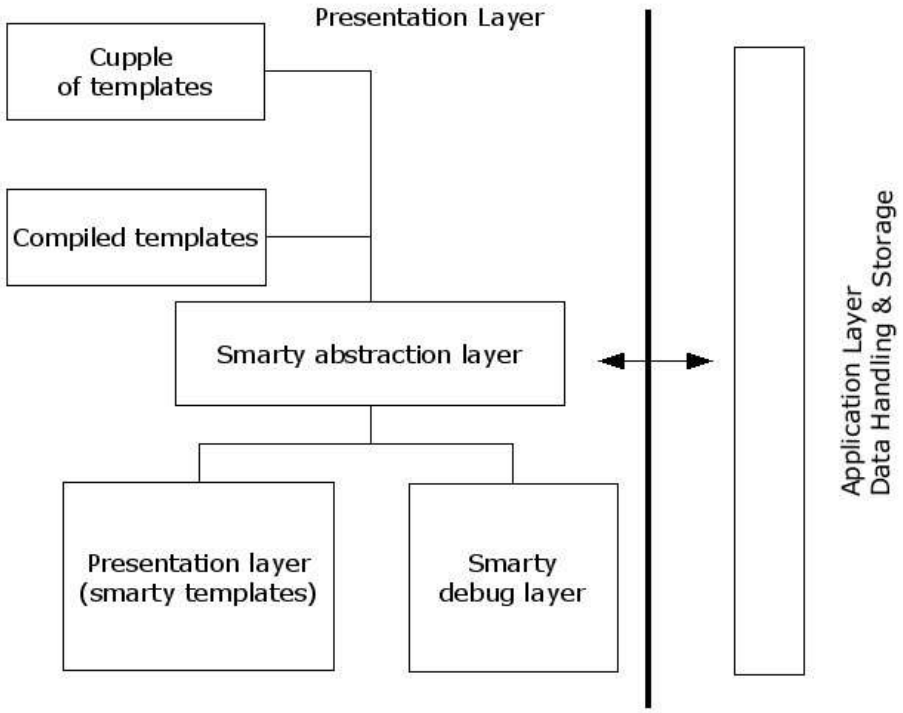
Template/Presentation Framework. It provides the programmer and template designer with a wealth of tools to automate tasks commonly dealt with at the presentation layer of an application. In the latest Versions Smarty provides you an API to write plugins used as extensions in the HTML/CSS templates. You get it from http://smarty.php.net

One of Smartys primary design goals is to facilitate the separation of application code from presentation. In SOTF this provides us the possibility to easily design new skins for the presentation layer of the application. At its most basic function, the application code collects content, assigns it to the template engine and displays it. Typically, the application code contains the business logic of an application, written and maintained in PHP code. This code is maintained by programmers. The presentation is the way your content is presented to the end user, which is written and maintained in template files. The templates are maintained by template designers.

In this way all data needed for the presentation layer in a specified case the business logic of SOTF collects and handles all data requests and feeds the smarty template object. The

- Designers can't break application code. They can mess with the templates all they want, but the code stays intact. The code will be tighter, more secure and easier to maintain.
- Errors in the templates are confined to the Smartys error handling routines, making them as simple and intuitive as possible for the designer.
- With presentation on its own layer, designers can modify or completely redesign it from scratch, all without intervention from the programmer.

Although application code is separated from presentation, this does not necessarily mean that *logic* is separated. In the case of SOTF there are several easy logical elements in the template code. This will simplify your application and keep your templates flexible. Smarty supplies the tools to handle this kind of situation.



8. Figure: Smarty principle way of operation

8.2.1) Code and structure

Smarty reads templates from a given template directory and compiles from smarty-tags to html-files with php-code included. On calling a special file the engine checks if this template is the same version like the stored compilation of it. Compilations are stored in a directory (must be writeable by the webbrowser) called templates_c/.

```
/template/template_files.tpl
/templates_c/compiled_versions
/smarty/Smarty.class.php
/smarty/Smarty.addons.php
/smarty/Smarty_Compiler.class.php
/smarty/plugins/functions...
/smarty/plugins/modifiers...
```

8.2.2) general Methods and way of operation

You will find an entire list of provided methods by the smarty object online:

http://smarty.php.net/manual/en/

SOTF creates in init.inc.php a smarty object which is used later on do assign values to the smarty object. We show in this document just the main methods. Please see smarty documentation

Create a smarty object:

```
$smarty = new mySmarty();
```

This object can be used as global in SOTF methods using *global $smarty;* or assigned to the hole class using *$this->smarty = & $smarty;*

To assign values to this object you use the provided method assign:

```
$this->smarty->assign("FOO", $baa);
```

This inserts you the content of $baa in the variable FOO. Smarty is using scalar values. This means if you assign an object to FOO it will create an object FOO within the smarty object. If you assign an assoc array you get an array FOO in smarty. Easy and smart ;-)

To use this values in the presentation layer you need to set smarty-tags in the templates:

Just print a single value to the presentation:

```
{$FOO}
```

To print an array or an object you need to use a foreach-loop to access all data. In this example it's an array. The foreach-loop would walk through the array and print in each loop the value baa.

```
{foreach FROM=$FOO item=baa}
  {$baa}
{/foreach}
```

To acces data from an array or an object directly use the variable direct access methods:

```
{$FOO.0.value}     (for an array)
{$FOO->0->value}   (for an object)
```

To keep parts of the logical system of an application flexible in the templates smarty provides several logical components like *if-else-elseif* structures assigning methods and value calculation

if-elseif-else structure:

```
{if $PERMISSION == 2}   (permission to edit and view)
  <a href="your view link">view</a>
  <a href="your edit link">edit</a>
{elseif $PERMISSON == 1}
  <a href="your view link">view</a>
{else}
  sorry you are not allowed to view or edit this content
{/if}
```

assign values:

```
{assign var=FOO value="this value"}
```

mathematical operations:

```
{math equation="c / ((a + b + c) / 100)" a = 11 b = 22 c = 33}
```

8.2.3) config files and variables

Config vars are special variables which are described in the configuration files. Classical use for this feature is the wording in your interface. You can set a cupple of wording-values for a special language and easily change the language of your interface to others. In SOTF actualy 3 languages are described

node/code/configs/eng.conf:

```
ok=Ok
cancel = Cancel
Save = Save
```

node/code/configs/ger.conf:

```
ok = Ok
cancel = Abbrechen
Save = Sichern
```

To use a value in the template you easily do:

```
{#cancel#}
```

This will write the content of 'cancel' as defined in the language config to the interface. You can use this smarty-feature for languages and other values you need for your interface. Such as numbers, and styles.

8.2.4) plugins (functions and modifiers)

Smarty offers a cupple of functions and modifiers which can be used in the template-scripts.

8.2.4.1) Modifiers

Modifiers are called in the template to modifiy a given variable and print the results to the page. This example would truncate a string to 100 Characters.

```
{$FOO|truncate:100}
```

The | (pipe) calls the modifier called truncate. This modifier has to reside in the plugins directory of smarty. The function has to have the name according to the smarty interface: *smarty_modifier_trundcate().* Below is an example of an easy modifier script. You are able to write new modifier plugins if you need. Please see the smarty page and your smarty directory/plugins to checkout which modifiers already exist. The API is easily to understand:
If the given string is empty, it returns an empty string. Otherwise it would truncate the string to a number of 80 chars. If you call it like above it would override the variable $length and truncate the string to 100 chars. At the end it adds '...' to the string to make visible in the presentation layer that this string is truncated. Easy and nice ;-)

```
function smarty_modifier_truncate($string, $length = 80, $etc = '...',
                                  $break_words = false)
{
    if ($length == 0)
        return '';
    if (strlen($string) > $length) {
        $length -= strlen($etc);
        $fragment = substr($string, 0, $length+1);
        if ($break_words)
            $fragment = substr($fragment, 0, -1);
        else
            $fragment = preg_replace('/\s+(\S+)?$/', '', $fragment);
        return $fragment.$etc;
    } else
        return $string;
}
```

8.2.4.2) Functions

Smarty-Functions can be understood as a logical extension in the templates of an application. Using this standard, developers and designers can add simple logic-parts to an application without touching the main php-files and classes. You can run each script you want using this function call. You can assign smarty-vars in this script and use the values for your templates.

```
{smartyfun var=FUN id = $ID}
```

This calls the function *smartyfun* and assignes the variables *var* and *id.* The assigned values are passed to the function in an array *$params.* In the function we get this array and the global smarty -object.

```
function smarty_function_smartyfun($params, &$smarty)  {
  global $db;
  $FOO = $db->getRow("SELECT foo FROM baa WHERE id = ".$params['id']);
  .... logic // do something useful with the data in $FOO
  $smarty->assign($params['var'], $FOO);
  return;
} // function
```

the data is assigned to the smarty variable $FUN.

/**

```
* Short note: i love smarty a lot ;-)
```

8.3) GetID3

getID3() is a PHP4 script that extracts useful information from MP3s and other multimedia file formats. It supports reading various tag, audio-only, audio-video, graphics and data formats.

Installation for Debian/GNU Linux and FreeBSD

Download the latest stable sources from http://getid3.sourceforge.net and unpack them via unzip to your favorite location. You can then use the features of getiD3() by including it to your PHP-Sources, for example:

```
<?php
require_once('./getid3.php');
$filename = "path/to/file.mp3";
$getID3 = new getID3;
$fileinfo = $getID3->analyze($filename);
print_r ($fileinfo);
?>
```

9) Debugging and Tools

9.1) error messages

On error sotf displays an interface with an error message. all errormessages are logged in:

```
/node/logs/log   -> general log file
```

9. Figure: Error Display

debug function:

The debug function is defined in sotf_Object.class.php. It is called on errors to log the loging data (defined in $this->data) to the log file.

Error Message in /node/logs/log:

```
[07-Jan-2005 15:26:17] 192.168.0.6: DB:node: Query: BEGIN TRANSACTION
[07-Jan-2005 15:26:17] 192.168.0.6: DB:node: Query: SELECT * FRO
sotf_programmes WHERE id = '666pr441'
[07-Jan-2005 15:26:17] 192.168.0.6: ERROR: SQL error!. ERROR:  syntax error
at or near "FRO" at character 10 in
 SELECT * FRO sotf_programmes WHERE id = '666pr441'
[07-Jan-2005 15:26:17] 192.168.0.6: page halted:
[07-Jan-2005 15:26:17] 192.168.0.6: sending error page:
[07-Jan-2005 15:26:17] 192.168.0.6: 0.5289 ms, /node/get.php?id=666pr441
```

[07-Jan-2005 15:26:17] ----------

9.2) psql / commandline interface

psql is a terminal-based front-end to PostgreSQL. It enables you to type in queries interactively, issue them to PostgreSQL, and see the query results. Alternatively, input can be from a file. In addition, it provides a number of meta-commands and various shell-like features to facilitate writing scripts and automating a wide variety of tasks.

you can find a full list of options on http://www.postgresql.org/docs/current/static/app-psql.html

here are some examples about connecting, using of the interface and the displayed results

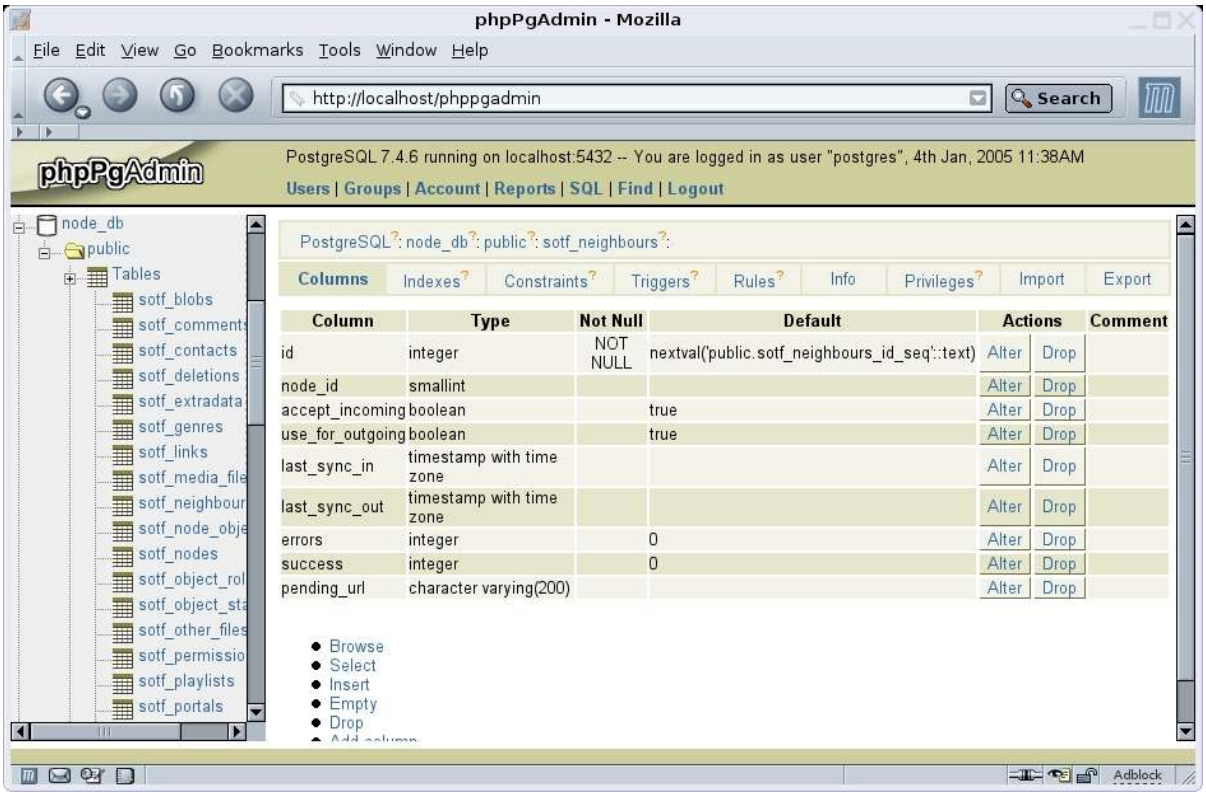
```
postgres@x77:~$ psql node_db;
Welcome to psql 7.4.6, the PostgreSQL interactive terminal.
Type:  \copyright for distribution terms
```

```
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit
node_db=# \d   -> list all database objects
node_db=# \d sotf_stations;
Table "public.sotf_stations"
 Column      |          Type          | Modifiers
-------------+------------------------+---------------------------
 id          | character varying(12)  | not null
 id2         | character varying(50)  |
 name        | character varying(60)  | not null
 description | text                   |
 url         | character varying(100) |
 language    | character varying(40)  |
 entry_date  | date                   | default ('now'::text)::date
Indexes:
    "sotf_stations_pkey" primary key, btree (id)
    "sotf_stations_name_index" btree (name)
Foreign-key constraints:
    "$1" FOREIGN KEY (id) REFERENCES sotf_node_objects(id) ON DELETE
CASCADE
```

you can do all database calls using the interface such as select, update or delete statements.

9.3) phpPgAdmin

phpPgAdmin is a fully functional PostgreSQL administration utility written in PHP. You can use it to create and maintain multiple databases and even multiple servers via a web frontend. Features include: create and drop databases; create, copy, drop and alter tables/views/sequences/functions/indicies/triggers; edit and add fields (to the extent Postgres allows); execute any SQL-statement, even batch-queries; manage primary and unique keys; create and read dumps of tables; administer one single database; administer multiple servers; administer postgres users and groups.



10.Figure: Screenshot of phpPgAdmin while accessing the node database.

Installation for Debian/GNU Linux

```
apt-get install phppgadmin
```

Installation for FreeBSD

```
cd /usr/ports/databases/phppgadmin
make install clean
```

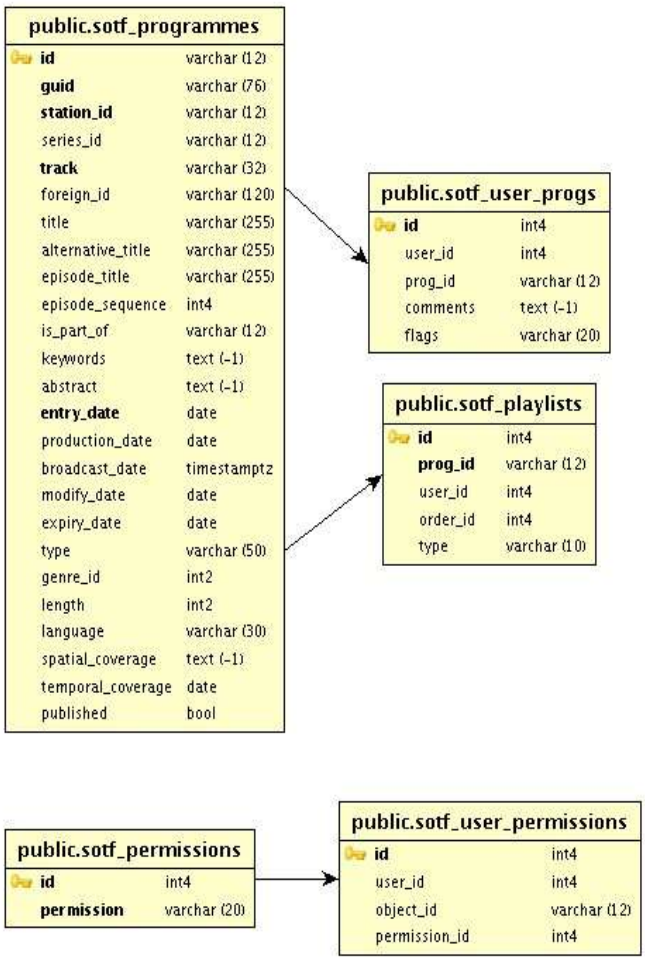
9.4) DbVisualizer / The Universal Database Tool

DbVisualizer is a cross-platform database tool for all major relational databases. DbVisualizer enables simultaneous connections to many different databases through JDBC drivers. Just point and click to browse the database structure, view detailed characteristics of database objects, edit table data graphically, execute arbitrary SQL statements or SQL scripts, reverse engineer primary/foreign key mappings graphically or why not let DbVisualizer chart your database with its advanced charting options. The user friendly graphical interface in combination with the unique collection of features makes DbVisualizer the ideal choice for database administrators and developers.

On postgreSQL since we have defined indexes and relations this tool is able to graph the database scheme.

DbVisualizer is not licensed under GPL, its propietary software. There are free downloads on www.minq.se with a number of limitations. actualy the limited editions are useful enough to graph database schemes. editing data and scheme is not possible in the free demo version.

The software is written in java and uses jdbc connectors to the database backend.



11.Graphical view of relations

9.5) phpdocumentor

phpDocumentor, sometimes referred to as phpdoc or phpdocu, is the current standard auto-documentation tool for the php language. Similar to Javadoc, and written in php, phpDocumentor can be used from the command line or a web interface to create professional documentation from php source code. phpDocumentor has support for linking between documentation, incorporating user level documents like tutorials and creation of highlighted source code with cross referencing to php general documentation. A complete list of features is available.

phpDocumentor uses an extensive templating system to change your source code comments into human readable, and hence useful, formats. This system allows the creation of easy to read documentation in 15 different pre-designed HTML versions, PDF format, Windows Helpfile CHM format, and in Docbook XML. You can also create your own templates to match the look and feel of your project.

- http://phpdoc.org
- http://manual.phpdoc.org/