The WebDB WWW Database Interface

A Customizable Database Interface through Word Wide Web
An Application of Persistent Predicates, Pillow and Active Modules
The CIAO System Documentation Series
Technical Report CLIP 11/98.0
RadioWeb (ESPRIT Project 25562) Report D3.1.M1-A3
ECCOSIC (Comision Conjunta Hispano-Norteamericana de Cooperacion Cientifica y Tecnologica Project 98059)
Draft printed on: 8 January 2003
Version 0.5#14 (2002/4/15, 11:27:6 CEST)

J.M. Gomez, D. Cabeza and M. Hermenegildo
clip@dia.fi.upm.es
http://www.clip.dia.fi.upm.es/
The CLIP Group
Facultad de Informática
Universidad Politécnica de Madrid

This document was generated on 8 January 2003 using texi2html 1.56k + clip patches and lpdoc.

Go to the first, previous, next, last section, table of contents.

Summary

WebDB is at the same time:

- A tool which allows browsing and modifying interactively and remotely using any WWW browser many kinds of relational (SQL) databases which may reside in different locations and operating systems.
- A customizable relational (logic) database system with a WWW-based user-interface. In this case the internal databases are kept as WebDB native tables, which are stored as Prolog persistent predicates and can also be consulted/edited from a Prolog program or even by using a text editor.

The user-interface provided by WebDB can be customized for a specific application by performing small changes in its configuration files. Such changes can be made interactively through the interface provided by the system, directly by modifying HTML templates, or through a combination of these methods.

This document also constitutes an *internals* manual, providing information on how the different internal parts of WebDB are connected.

This documentation corresponds to version 0.5#14 (2002/4/15, 11:27:6 CEST).

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

Introduction

`WebDB` provides a way to create and customize a *Relational Database* and/or a *WWW interface* to access it. `WebDB` provides a simple WWW interface based on *HTML* forms which allows the user to access the data stored in any database, local to `WebDB` itself or remote.

In the first case, the tables are designed according to the definition of filebased persistent predicates, using the library `persdb` [GCH98] and are maintained directly by `WebDB`, allowing the system administrator to create, destroy or edit the tables. In the second case, the data is stored in remote SQL databases, so that `WebDB` just provides an interface to them through the primitives espeficied in the library `persdb_sql` [CCG98].

As we will see, the information can be accessed in two different ways. We could say that `WebDB` provides a *single table interface* and a *multitable interface*.

Through the first way, the interface to access a table is created automatically on every access. This is achieved by defining each table field according to an *html_format* type, which is translated by a template into the right HTML input type for the given field, for example, a checkbox, an input box or a textarea.

The system administrator can add new types and define tables according to them. Of course, `WebDB` provides an interface to develope this task in an automatic way.

The multitable interface or, as we will see in further sections, the complex query option provides an interface to make queries over a database view in both, Prolog or SQL syntax, by offering a window where the user can write them.

Settings in `WebDB` are kept in *configuration files*, and are customizable, as well as types and templates. The system administrator can customize them on line, without needing to stop the system, through the interface provided by the application itself.

Data Storage

As mentioned above, the data `WebDB` operates on can be local or located at remote SQL databases. In both cases, `WebDB` provides a WWW interface to access it in a transparent and comfortable way.

These two kinds of data storage are based upon the same concept: Persistent Predicates. The idea is that whatever changes might have been done on one of these predicates survive across executions. This way, no matter whether the system stops, or even crashes, we will always have a consistent state of the predicate, which is quite an useful thing if we want to implement a database. If Prolog is halted and restarted, the predicate the new Prolog process sees is right in the state it was in the very moment the old process was halted, provided that no changes to the external storage were done in the meantime by other processes or the user himself.

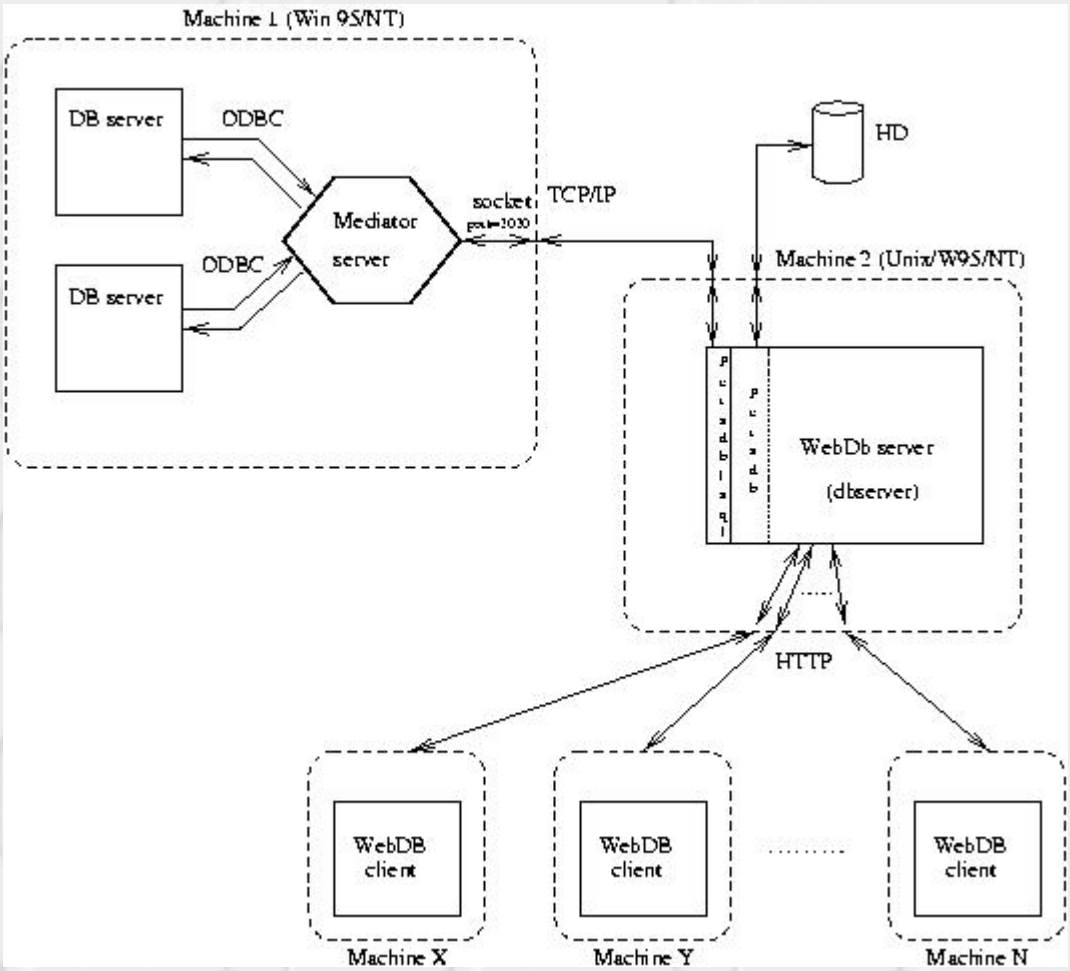
Actually, there are two different implementations of persistent predicates:

- File based persistent predicates, which local tables are based upon, and
- SQL based persistent predicates, which is the way WebDB implements the tables located on remote SQL databases.

The first method can be used for storing smaller amounts of data that we will call *local data*. In WebDB this kind of persistent predicates are also used to manage most of the internal information the system needs to handle the operations the services it provides resemble. In a wider context, e.g. the Prolog world, it seems an easy and efficient way to achieve persistence, saving the state of the predicates through executions, which, up to now, is something Prolog systems where quite lack of.

The second method provides a very high-level natural way to access, from a Prolog program, larger amounts of data in SQL database relations. According to this method, persistent predicates are linked to tables in external databases so that the predicates actually reside in such tables. These tables are seen from the Prolog side as ordinary predicates.

In the next figure, we can appreciate the interaction between WebDB and these two kinds of persistent predicates as well as the system's architecture:



As we can see in the figure, the WebDB server provides its services through the WWW. These services basically consist of an interface to query the database and, in the case of the system administrator, manage the system.

The Generic Database Server: dbserver

This part of the WebDB documentation is aimed to describe its database server, dbserver, which provides the users with services to access a given database. This server has been developed as

a `CIAO prolog` *active module* and is basically an executable which publishes its IP address in order to allow client programs to use its services. Client programs are invoked with the CGI protocol by the web server and connect to the active module to request server actions.

This server also uses the capabilities provided by the filebased persistent predicates to manage the multiple aspects of the database it is bound to, for example, to keep the current state of a search or the current description and type of a given table, or even the text messages which are output as a result of any interaction with the system.

Launching dbserver

In order to make the database accessible from the web, the dbserver is launched through the CGI `startdbserver.cgi` and its standard error output is redirected to a file called `dbserver.log`, located on the `private` directory. This way, any misbehaviour or problem occurred will be registered and, therefore, solved more easily. The steps the dbserver carries out at this point follow:

First of all, the dbserver processes the contents of the file `dbserver_ini.pl`. This file is created during the installation and contains important data relative to the directories of the database and its URLs.

The next step is to read the configuration files for the database. Database customization basically lies on these files, which contain information regarding the html_format *types* and *templates* and their translation into `PiLLoW` terms, as well as information about the searches lifetime, the translation between html_format and SQL types and the remote SQL databases the system is bound to.

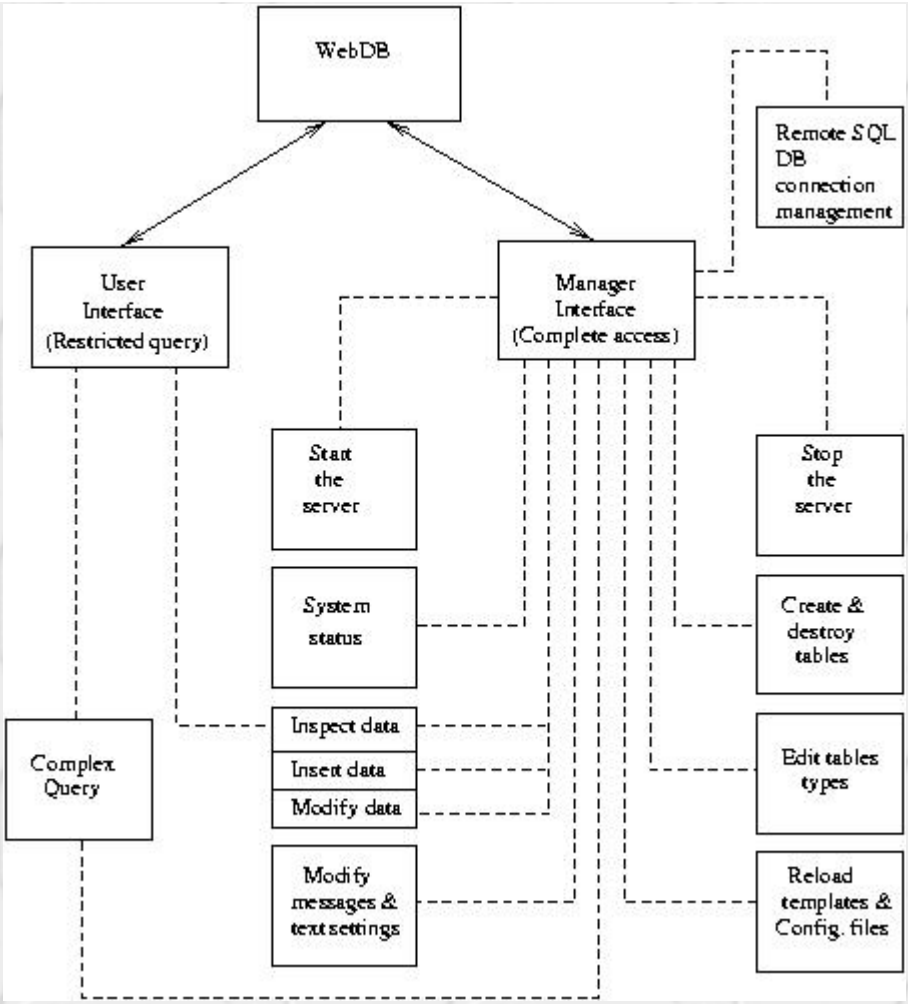
Once all this information has been processed, dbserver can start the processing of data regarding the database tables. So, dbserver enables for future queries both kinds of tables, local tables based upon filebased persistent predicates and remote tables based upon SQL-based persistent predicates.

The next step is to start the statistics procedure by resetting the date of initialization for the dbserver. This information can be accessed through the service provided by the option `status` lying on the maintenance front page .

Finally, if necessary, dbserver generates a `maintenance front page` which is the first part of the manager interface, and initializes all those other persistent predicates embedded in the system which are used for inner tasks such as keeping a trace of the current state of queries and the text of the user interface messages.

WebDB Interface

The `WebDB` interface is aimed to be an easy and straightforward way to query and manage a database. As we can see in the figure, this interface is composed by several stages. Each one offers the right interface for a given service, many of them only allowed to the user administrator.



Manager Interface

Let's concentrate on the manager interface as the common user interface is quite simple and consists of a restricted version of the manager search interface.

The first stage of the manager or administrator interface is a maintenance front page where the following options are provided:

- *Start the System*, obviously used to start the dbserver.
- *System Status*, which provides information about the dbserver status and statistics about the system's usage.
- *Remote SQL Databases Connection Management* is the door to an interface which enables the system administrator to connect `WebDB` to remote SQL databases and unlink it from a previously added one. Through this interface, it is possible to collect the information `WebDB` needs in order to accessing the tables of a given database. Besides, it provides information about the remote SQL databases the system is already linked to.
- *Inspect/Insert/Modify Data*, provides an interface where the user can operate on each table of the database individually.
- *Complex Query*, provides another interface to operate on the database. This interface is quite similar to a Prolog top level, an environment where the user can make his own queries according to Prolog and SQL syntax standards.
- *Modify Messages and Text Settings*, provides a list with all the system messages and the

text settings and allows the system administrator to edit them.
- *Create/Destroy Tables*, allows the system administrator to create new filebased tables for the current database or destroy any of the already existing as well as unlink the system from remote SQL tables.
- *Edit Tables Types*, allows the system administrator to edit the type of any field of a filebased table or to add or delete them.
- *Reload Templates and Update DB Types*. Once the configuration files have been changed, this option reloads them and updates the system with the new configuration.
- *Stop the System*, halts the system and updates the information stored upon filebased persistent predicates.

Additional information must be given about the fourth and fifth options:

The Inspect/Insert/Modify Data option

Once the user has decided on which table to operate on, this option displays the system consult interface. Through this interface the user can choose which operation to carry out.

Initially, the system administrator, can choose among the following operations:

- *Search*: Sequential search. The results of the query are displayed one by one as the user requests them.
- *Overview*: All the tuples that instantiate the query are displayed in a table, but just as prolog facts, without the additional information supplied by the fields types.
- *Full search*: As in the case of the *Overview* operation, all the tuples that instantiate the query are displayed, but in this case with the additional information of its field types. Besides, through this option, the user can edit the results too.
- *Insert element*: Allows the user to insert a new element, which has to be completely specified, e.g. no field of the form can remain empty. Besides, the system tests that the element to be inserted doesn't exist in the table already.
- *Delete element*: Deletes all the tuples that instantiate the query.
- *Modify element*: Allows the user to edit a given tuple of the table.

If the user is not an authorized client, the operations allowed will only be *Search*, *Overview* and *Full search* which, in this case, will not allow the user to edit the results of a previous query.

Note that when any query is made over a database table, the information written into the form fields is taken as prolog terms. This way, any empty field of the form is translated into a free variable and therefore will unify all the corresponding arguments of the given table.

Besides, `WebDB` provides three different search modes:

- *Normal search mode*: This is the most restrictive search mode. The results of the query will be just those tuples which exactly match the fields specified.
- *Case insensitive search mode*: In this mode upper and lower case is indistinct.
- *Wildcards search mode*: Implements a search where wildcards can be used. Special characters are:

  *****
  : Matches any string, including the null string.

  ?
  : Matches any single character.

  [...]
  : Matches any one of the enclosed characters. A pair of characters separated by a minus sign denotes a range; any character lexically between those two characters, inclusive, is matched. If the first character following the [is a ^ then any character not

enclosed is matched. No other character is special inside this construct. To include a] in a character set, you must make it the first character. To include a `-', you must use it in a context where it cannot possibly indicate a range: that is, as the first character, or immediately after a range.

|

specifies an alternative. Two regular expressions A and B with | in between form an expression that matches anything that either A or B will match.

{...}

groups alternatives inside larger patterns.

Quotes a special character (including itself).

The Complex Query option

This option is fully available for both authorised and not authorised clients. As mentioned above, it provides an interface to query the database in Prolog and SQL syntax. It is necessary to select the view the tables involved in the query lay on.

When only local tables are involved in the query, the predicates allowed in this environment are restricted in order to avoid any eventual misuse. This set of allowed predicates can be classified in the next categories: arithmetic and relational operators, connectors (i.e. ',' and ';') and the tables of the database itself.

On the other hand, in the case of tables belonging to remote SQL databases, it is quite recommendable to control which views are to be accessible for the system users and with which rights in order to prevent the SQL tables from being manipulated unduly.

As an on-line guide for the user, this option displays a list of the tables the database consists of. This includes information about the arity of the tables, the physical database where they are located and the types of their fields.

There are three possible answers for a complex query:

- *No elements found* if there were no matches for the query.
- *Unsuccesful query* if the query tried to attempt against the system integrity or any predicate involved didn't exist.
- *Complex Query Response*. In this case the output is a table with the name of every variable written in the query and their values.

Clients of the Generic WebDB Distribution

This part of the `WebDB` documentation is aimed to describe the CGI clients which are included by default with the distribution of `WebDB` and form the interface described in the previous section.

These clients use the services provided by the server to query the database or manage the system. From this point on, the number of clients added only depends on the system administrator and the number and nature of the services required by any specific case.

Each client is firmly related to the services provided by the server. In most cases, this relation is an injective application, as we can see:

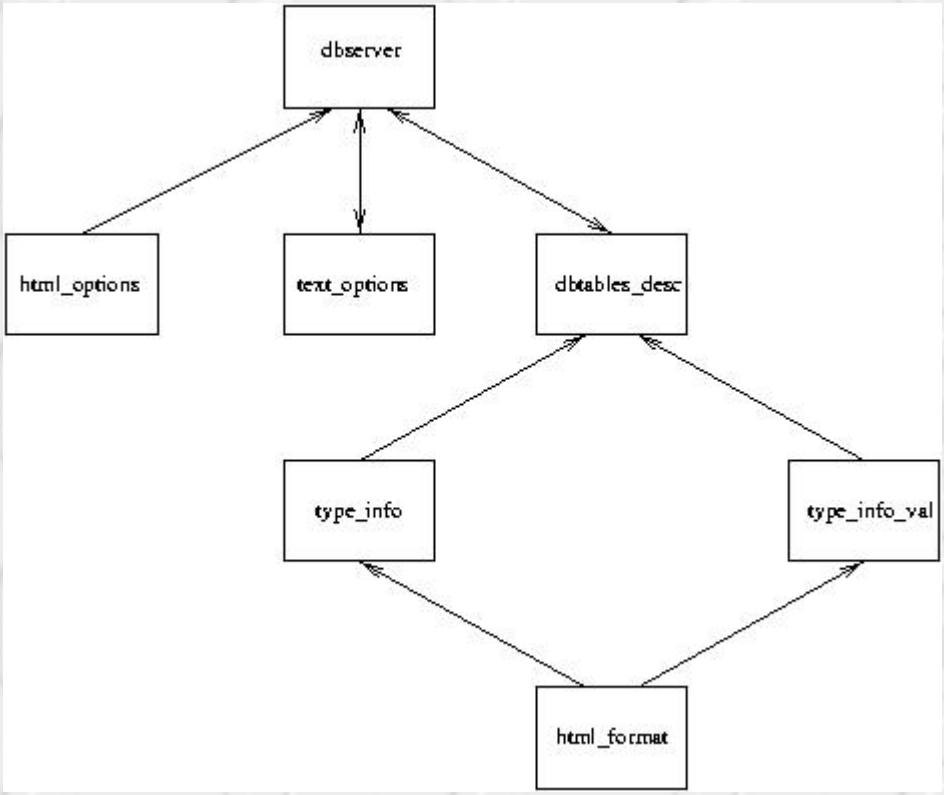
- *dbcommonselection* <-> `get_relationsdb`
- *dbcommonclient* <-> `consultdb`
- *dbselection* <-> `get_relationsdb`
- *dbclient* <-> `consultdb`
- *dbcomplexquery* <-> `complex_query`

- *dbcreatetable* <-> `create_destroy_table`
- *dbeditselection* <-> `select_table_to_edit`
- *dbeditclient* <-> `edit_table`
- *dboptions* <-> `consultdboptions`
- *dbtemplates* <-> `dbtemplates`
- *dbserverstatus* <-> `status`
- *dbconnect_DB* <-> `connect_to_DB`
- *dbserverstop* <-> `dying`, `die`

Going back to the first figure, we can see that, in fact, every component of `WebDB` can be located on distant hosts on the WWW, as they get through with one another using the HTTP protocol. This leaves an open door for distributed databases due to its interconnectivity.

WebDB Types and Templates

`WebDB`'s customizability lies on a series of files, file based persistent predicates and predicates which constitute a system of types and templates in order to generate the HTML representation of every table. In the nex figure we can appreciate the interaction of each of these components with one another and with dbserver.



Let's describe these components:

- *html_options* are a set of .html files processed by the dbserver both when it is started and when the system administrator reloads the templates of the system through the operation *Reload Templates and Update DB Types*, available at the maintenance front page. These templates are parametric. So, the user can set them to the right values and customize the database. It is important to bear in mind that although these files are parametric, their structure, as well as their name, have to be respected. Moreover, the user must not delete any of them. So, in order to customize the settings, from the background color to the representation of the html tables, the user only needs to change the parameters of these files and reload them, using the previously mentioned operation.
- *text_options* is one of the persistent predicates embedded in the system mentioned in previous sections. It is used to store all the messages and text the system uses when

interacting with a client. As we can see in the figure, this component is linked to dbserver in both ways. This means that, although dbserver uses it to produce the right text and messages, it is also able to edit them. To do so, the system administrator must choose the *Modify Messages and Text Settings* operation.

- *dbtables_desc* is the next of these system embedded persistent predicates. It stores the definition of all the tables the database consists of, both remote and local. To define a table, the system administrator has two options:
  - Create it through the interface provided by `WebDB` using the operations *Create/Destroy Tables* and *Edit Tables Types* of the maintenance front page.
  - Create it by adding an entry in the file `DBDATADESC/db_desc/dbserver:dbtables_desc-2.pl`. This entry must be written in the following format:

```
'dbserver:dbtables_desc'(
<<Table_name>>(
        <<Type 1>>(<<Field description 1>>, <<Length 1>>),
        <<Type 2>>(<<Field description 2>>, <<Length 2>>),

        <<Type n>>(<<Field description n>>, <<Length n>>)),
        <<Situation>>,
        <<Creation Date>>).
```

    Besides, it is possible to link the tables laying in an SQL remote database through the operation *Remote SQL Databases Connection Management*. `WebDB` distribution includes an example of these types and templates for a given database and surely the best way to make a new one is by pattern matching. Each of these types need to have their corresponding entry in the file `html_format` located on the directory `DBDATADESC/db_types` which will be described next. In every case, the second argument of `dbserver:dbtables_desc/3` points out whether the table is local, by setting that field to the value `local', or remote, by writing the identifier of the view for the SQL database. That kind of identifier is defined in the filebased persistent predicate `sql_location/2` which, just like the `DBDATADESC/db_types/html2sql.pl`, is described in the next section (Defining SQL remote Databases).

- `type_info/3` and `type_val_info/4` are used to obtain the translation between a type defined for a field in *dbtables_desc* and its corresponding template of *html_format*, which is the component to be described next. `type_info` is used to get the *html_format* template of a table field and its description before making any query and, on the other hand, `type_val_info` gets the contents of a table field after the query has been made.
- html_format is a *macro* which provides an interface to use the primitives provided by the library `PiLLoW` in a transparent way. An entry of *html_format* can be equivalent to a large piece of `PiLLoW` code and, therefore, it contributes to make things simpler and more customizable.

Defining SQL remote Databases

This is fundamentally achieved thanks to the library `persdb_sql` which provides and interface to remote SQL databases located on Win95/NT machines. In order to use the facilities provided by that library, `WebDB` needs the predicate `sql_location/2` and the file `html2sql.pl`, described next:

- `sql_location.pl` is an SQL based persistent predicate that contains entries `WebDB` uses to define the fact `sql_persistent_location/2` (see the documentation of `persdb_sql` [CCG98]) for each remote SQL database view linked to `WebDB`.
- `html2sql.pl` contains instances of the predicate html2sql/2 which provide the translation between the macros defined in *html_format* and their equivalent SQL types.

- `sql2html.pl`, in the opposite direction, allows to find the right html_dormat representation type for a given SQL type.

Version/Change Log (`WebDB`)

Version 0.5#12 (1999/11/6, 22:2:26 MET)
Updated documentation to lpdoc-1.9 formats. (Manuel Hermenegildo)
Version 0.5#11 (1999/9/16, 12:52:5 MEST)
Added lock file to prevent the system from being started several times (Jose Manuel Gomez Perez)
Version 0.5#10 (1999/9/3, 10:58:16 MEST)
Generic client interface added (Jose Manuel Gomez Perez)
Version 0.5#9 (1999/6/1, 19:35:14 MEST)
Automatic distribution procedure completed. (Manuel Hermenegildo)
Version 0.5#8 (1999/5/31, 17:11:11 MEST)
Readmes now automatically generated from documentation. (Manuel Hermenegildo)
Version 0.5#7 (1999/4/23, 13:14:44 MEST)
Naive graphical statistics added (Jose Manuel Gomez Perez)
Version 0.5#6 (1999/4/21, 11:58:39 MEST)
Text statistics added (Jose Manuel Gomez Perez)
Version 0.5#5 (1999/4/20, 13:9:27 MEST)
Infrastructure for system status info added (Jose Manuel Gomez Perez)
Version 0.5#4 (1999/3/29, 19:57:47 MEST)
HTML templates revised (Jose Manuel Gomez Perez)
Version 0.5#3 (1999/3/23, 18:58:21 MET)
Automatic link to remote SQL databases through ODBC added (Jose Manuel Gomez Perez)
Version 0.5#2 (1999/2/10, 12:29:33 MET)
Corrected lack of generality at the edit tables interface (Jose Manuel Gomez Perez)
Version 0.5#1 (1999/2/10, 12:28:51 MET)
Complex query with restricted preds. completed (Jose Manuel Gomez Perez)
Version 0.5 (1998/7/23, 13:26:0 MET DST)
Started autodocumentation. (Manuel Hermenegildo)
Version 0.1#1 (1998/3/18)
Updated maintenance interface. (Jose Manuel Gomez Perez)
Version 0.1 (1998/3/18)
Added recursive treatment of relations (base case). (Jose Manuel Gomez Perez)
Version 0.1 (1998/2/20)
Started automatic documentation. (Jose Manuel Gomez Perez)

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

An active module based dbserver with interface to SQL remote DBs

Version: 0.5#14 (2002/4/15, 11:27:6 CEST)

Usage and interface (dbserver)

- **Library usage:** This module is typically compiled as an *active module* (i.e., a *server*). Clients can access this server by using the active modules usage convention:

```
:- use_active_module(dbserver).
```

- **Exports:**
  - *Predicates:* `db_handler/3`, `db_maintenance/3`, `dying/2`, `die/1`.
  - *Multifiles:* `$is_persistent/2`, `persistent_dir/2`, `$is_sql_persistent/3`, `sql_persistent_location/2`, `relation/3`, `attribute/4`.
- **Other modules used:**
  - *System library modules:* `operators`, `pillow/http`, `pillow/html`, `persdb/persdbrt`, `persdb_mysql/persdbrt_mysql`, `det_hook/det_hook_rt`, `file_utils`, `lists`, `system`, `aggregates`, `dynamic`, `streams`, `read`, `write`, `terms`, `patterns`, `errhandle`.
  - *Internal (engine) modules:* `internals`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

Documentation on exports (dbserver)

PREDICATE: db_handler/3:

Usage: `db_handler(+Input,+Passwd,-Response)`

- *Description:* Used to provide an interface to the service requested by the generic client. Once this predicate identifies the required service, it calls the particular predicate that implements it.
- *Call and exit should be compatible with:* `+Input` is a Pillow dictionary. (`dic/1`) `+Passwd` is an identifier for the client type. (`passwd/1`) `-Response` is an HTML page. (`output_page/1`)

PREDICATE: db_maintenance/3:

Usage: `db_maintenance(+Input,+Passwd,-Response)`

- *Description:* According to the kind of user that calls this predicate, the system generates an interface which provides access to the whole of its services (in the administrator case) or to a restricted set of them (in the case of a common user). Typically used in installation time to automatically generate both common user and administrator front pages.
- *Call and exit should be compatible with:* `+Input` is a Pillow dictionary. (`dic/1`) `+Passwd` is an identifier for the client type. (`passwd/1`) `-Response` is an HTML page. (`output_page/1`)

PREDICATE: dying/2:

Usage: `dying(+ClientType,-Response)`

- *Description:* Implements with `die` the stop the system service. Before having halted the database server through `die/1`, this predicate is used to produce a message, pointing out that the system is about to be halted. Only authorised clients can use this service.
- *Call and exit should be compatible with:* `+ClientType` is an identifier for the client type. (`passwd/1`) `-Response` is an HTML page. (`output_page/1`)

PREDICATE: die/1:

Once the database has been updated in order to keep all the information in a consistant state, it halts the database server. Only authorised clients can use this service.

Usage: `die(ClientType)`

- *Description:* Used to kill the server (active module) after updating the database.
- *The following properties should hold at call time:* `ClientType` is an identifier for the client type. (`passwd/1`)

Documentation on multifiles (`dbserver`)

PREDICATE: $is_persistent/2:

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

PREDICATE: persistent_dir/2:

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

PREDICATE: $is_sql_persistent/3:

No further documentation available for this predicate.

The predicate is *multifile*.

PREDICATE: sql_persistent_location/2:

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

PREDICATE: relation/3:

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

PREDICATE: attribute/4:

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

Documentation on internals (`dbserver`)

REGTYPE: passwd/1:

Usage: `passwd(X)`

- *Description:* `X` is an identifier for the client type.

REGTYPE: dic/1:

Usage: `dic(X)`

- *Description:* `X` is a Pillow dictionary.

REGTYPE: output_page/1:

Usage: `output_page(X)`

- *Description:* `X` is an HTML page.

Version/Change Log (`dbserver`)

Version 0.5#14 (2002/4/15, 11:27:6 CEST)
Corrected bug at numbers representation (Jose Manuel Gomez Perez)
Version 0.5#13 (2001/10/15, 17:37:55 CEST)
Installation process and documentation updated (Jose Manuel Gomez Perez)
Version 0.5#12 (2001/3/23, 17:48:27 CET)
Distribution process revised (Jose Manuel Gomez Perez)
Version 0.5#11 (1999/9/16, 12:52:5 MEST)
Added lock file to prevent the system from being started several times (Jose Manuel Gomez Perez)
Version 0.5#10 (1999/9/3, 10:58:16 MEST)
Generic client interface added (Jose Manuel Gomez Perez)
Version 0.5#8 (1999/5/31, 17:11:11 MEST)
Readmes now automatically generated from documentation. (Manuel Hermenegildo)
Version 0.5#7 (1999/4/23, 13:14:44 MEST)
Naive graphical statistics added (Jose Manuel Gomez Perez)
Version 0.5#6 (1999/4/21, 11:58:39 MEST)
Text statistics added (Jose Manuel Gomez Perez)
Version 0.5#5 (1999/4/20, 13:9:27 MEST)
Infrastructure for system status info added (Jose Manuel Gomez Perez)
Version 0.5#4 (1999/3/29, 19:57:47 MEST)
HTML templates revised (Jose Manuel Gomez Perez)
Version 0.5#3 (1999/3/23, 18:58:21 MET)
Automatic link to remote SQL databases through ODBC added (Jose Manuel Gomez Perez)
Version 0.5#2 (1999/2/10, 12:29:33 MET)
Corrected lack of generality at the edit tables interface (Jose Manuel Gomez Perez)
Version 0.5#1 (1999/2/10, 12:28:51 MET)
Complex query with restricted preds. completed (Jose Manuel Gomez Perez)
Version 0.5 (2001/2/14, 15:44:4 CET)
Due to changes in persdb, static persistent predicates also need to be declared dynamically. (Jose Manuel Gomez Perez)
Version 0.5 (1998/7/23, 13:26:0 MET DST)
Started autodocumentation. (Manuel Hermenegildo)
Version 0.1#1 (1998/3/18)
Updated maintenance interface. (Jose Manuel Gomez Perez)
Version 0.1 (1998/3/18)
Added recursive treatment of relations (base case). (Jose Manuel Gomez Perez)
Version 0.1 (1998/2/20)
Started automatic documentation. (Jose Manuel Gomez Perez)
Version 0.0 (1998/2/20)
Started automatic documentation. (Jose Manuel Gomez Perez)

Go to the **first**, **previous**, **next**, **last** section, **table of contents**.

Go to the first, previous, next, last section, table of contents.

Installing WebDB

Installation steps

The following provides the different steps which are necessary to take to make a complete installation of `WebDB`

- Uncompress (using `gunzip`) and unpackage (using `tar -xpf`) the distribution in a suitable directory.
- Edit the `SETTINGS.pl` file and change things to suit your installation. Note that some directories must have certain permissions.
- In a shell, cd to the WebDB root and run `lpmake install`. This creates the CGI executables, creates the target directories, and copies several `files` to their destinations in these directories. In order to ease this process it is highly recommendable both ciaoc and lpmake are reachable through the PATH environment variable.
- cd to the `DBDESC` directory defined in `SETTINGS.pl`. Using the file `dbserver:dbtables_desc.pl` as a template, create a new one in this directory defining the relations/fields of your database. Note that Prolog syntax must be respected in this file. This can also be done through WebDB's administration interface.
- PERMISSIONS. The installation processoffers a default configuration but these are some hints you should bear in mind anyway:
    - The directory `DBHOME/private` (`DBHOME` is defined in `SETTINGS.pl`) will contain the scripts which allow the querying and management of the database. You will probably want to protect it with an `HTTP password`. In the distribution directory `utils` you can find the source of a program for manipulating password files for `NCSA httpd`. Ideally, `DBHOME/private` should not be readable by anyone (but needs to be readable by the httpd, of course). Besides, note that, in `SETTINGS.pl`, to prevent undesired downloading of the `DBDATA`, `DBDESC` and `DBTYPES` files it's quite advisable to set `DBDESCDATA` to some different directory from `DBHOME` which is not under the public_html directory tree.
    - Check that your system's httpd configuration file allows the cgi executables (all ending in .cgi) located in `DBHOME` and `DBHOME/private` to be run from a browser. Also, the user under which the httpd is run needs to have write permission in `DBADD` and in directories `DBHOME/private`, `DBDESC` and `DBDATA`.
- You are done! If you want to delete temporary files created during the installation, run `make clean` in the source directory.

Usage Summary

A very brief summary of usage (you should consult the relevant sections of the manual for a full explanation):

- To perform administration duties (i.e., modification/management of the database) open `DBURL/private/maintenance.html` (`DBURL` is defined in `SETTINGS.pl`). If the database server has not been started yet (which can be checked by seeing if a correct response is received following the link *System Status*) start one by selecting the link *Start the System*.
- To consult the database as a common user (no modification allowed), open `DBURL/dbcommonselection.cgi`.

Customization

Extensive customization of the appearance of the database pages is possible:

- The basic text messages used in the database can be changed from a browser, by following the corresponding link from `DBURL/private/maintenance.html` (or edit manually the file `DBDESC/dbserver:text_option-3.pl`).
- The layout of the pages is determined by the HTML files in the `DBHOME` directory. You can change these files (between the body tags) to change the appearance of the HTML pages. The title of each template file explains what the template is used for, and lists the template variables between parenthesis. In these files, the template variables are written like this `<V>varname</V>` if they appear in normal text or like `_varname`, if they appear as an attribute or attribute value of a tag. Note that all URLs must be relative to the `DBHOME` directory. You can change these templates by hand or using an HTML editor. Beware, however, of WYSIWYG HTML editors! Always check that they have not arbitrarily changed attributes, tag ordering, etc. After changing some templates, you can load the new files into the running db server by selecting the appropriate link in the `maintenance page`. This will also recreate the maintenance page (remember to reload if the browser cache is activated).

Troubleshooting

If an unexpected response is obtained, you may check the database server log file `DBHOME/private/dbserver.log`. For further, persistent problems subscribe to ciao-users@clip.dia.fi.upm.es{}

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

References

[CCG98]
I. Caballero, D. Cabeza, S. Genaim, J.M. Gomez, and M. Hermenegildo.
persdb_sql: {SQL} {P}ersistent {D}atabase {I}nterface.
{T}echnical {R}eport D3.1.M2-A2 CLIP10/98.0, RADIOWEB Project, December 1998.

[GCH98]
J.M. Gomez, D. Cabeza, and M. Hermenegildo.
persdb: {P}ersistent {D}atabase {I}nterface.
{T}echnical {R}eport D3.1.M2-A1 CLIP9/98.0, RADIOWEB Project, December 1998.

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

Predicate/Method Definition Index

Jump to: $ - a - d - p - r - s

$

a

d

p

r

s

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

Property Definition Index

Jump to:

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

Regular Type Definition Index

Jump to: d - o - p

d

- dic/1

o

- output_page/1

p

- passwd/1

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

Concept Definition Index

Jump to: a - c - h - l - m - r - s - t - w

Go to the first, previous, next, last section, table of contents.