PID Generator

Technical Manual
Version 1.2

Markus Wagner, Jutta Glock, Murat Sariyar, Andreas Borg

Institut für Medizinische Biometrie, Epidemiologie und Informatik

Johannes-Gutenberg-Universität Mainz

PIDservice@gpoh.de

April 10, 2009

Contents

1 Introduction **1**

2 Installation **3**

2.1 Installation on UNIX Systems 3

2.2 Installation on Windows Systems 4

2.3 Database System 4

2.4 Example: Initial setup of the GPOH database 7

2.5 Web Interface 11

3 Configuration **13**

3.1 Database 13

3.2 SQL Syntax 14

3.3 Logging 15

3.4 Request History 16

3.5 Mailing 16

3.6 Encoding 17

3.7 Directory Access 18

3.8 PID Construction 19

3.9 Templates 19

3.10 Messages 21

3.11 Encryption 23

3.12 Health Insurance Code Validation 23

3.13 Example: The GPOH configuration 24

4 Specification **25**

4.1 Data Format 25

4.2 Result Specification 29

4.3 Matching Procedure 31

4.4 Example: The GPOH specification 36

5 Operation **45**

5.1 Database Generation 45

5.2 Encipherment 46

5.3 PID Request 47

5.4 Web Interface 50

5.5 PID Validation 50

5.6 Configuration Information 51

5.7 Version Information 51

5.8 Status Summary 52

A Reference **53**

A.1 Matching Algorithm and Processing of Names 53

A.2 Commands 56

A.3 FAQ 57

1 Introduction

PID Generator is an application designed for the matching of personal data and the assignment of Personal Identifiers (PIDs). It is especially capable of dealing with sensitive data, e. g. in the case of patient data, and with data which may be unsure or incomplete.

The software provides for many different tasks, ranging from simple transformations on single records up to complex operations performed in batch mode. These include decomposition of names, generation of phonetics, and the encipherment with IDEA or AES. To meet cancer registry requirements, the generation of control numbers using the MD5 algorithm has additionally been incorporated. The core facility however is the management of a PID database and the capability to process PID requests.

The PID Generator is highly customisable to meet the requirements of different environments. Adaption to the local database system, details of the logging and mailing activities, and many more settings are determined in the configuration file. Another file, the specification file, defines the structure of the records that are processed, and the overall behaviour of the matching procedure.

Portability The software is implemented in pure C. The source code is designed to be independent of the underlying operating system as much as possible. It should therefore compile on any UNIX system. The code was also compiled on other platforms such as Windows XP without any serious problems. However, the software depends on several libraries that should be installed before the software may be compiled. These include the native libraries either of PostgreSQL or of another database system, whereas the latter again requires the corresponding ODBC libraries. Though the database system may be nearly freely chosen it is left to the system administrator to deal with any specialities of his local installation.

2 Installation

2.1 Installation on UNIX Systems

The software package uses the GNU autotools to increase source code portability.

The first thing to do is to unpack the distributed package `psx-xxx.tar.gz` where `xxx` must be replaced with the actual release number. After that, go to the newly created directory `psx-xxx` and run the `configure` script. The latter will perform a variety of tests on the availability, location, and properties of several libraries installed on the system. Note that the library files for accessing the database system (PostgreSQL or ODBC) must be installed before running the script. For PostgreSQL, it is the header file `libpq-fe.h` that needs to be present.

The `configure` process generates the Makefiles that will be used for compilation and linkage. Use the option `--help` to find out about special configuration options. Finally, the `make` command must be invoked to build the project and to create the executable.

The installation can be tested by executing the command `psx h` in the `src` directory.

```
tar -xzvf psx-xxx.tar.gz    // unpack package
cd psx-xxx                  // change directory
./configure                 // create Makefile
make                        // build project
 ...
cd src                      // change directory
psx h                       // test installation
```

Known pitfalls On some systems one may encounter problems when building with shared libraries enabled. In this case, the `make` command should be called with the option `--enable-shared=no` to prevent libtool from generating a wrapper script rather than building a binary. Refer to the GNU software documentation or to dedicated literature for further information [3].

Another problem has been reported concerning character set incompatibilities using the PostgreSQL database on systems using UNICODE. On those systems, the database tables are also encoded in UNICODE by default. Errors occurred when data was passed from the PID-Generator to the database. One way to solve this problem is to set the system locales in a way that no UTF-8 encoding is used (set `LC_CTYPE` to `de_DE@euro` or `en_US`, for example). The database tables should then be encoded in SQL_ASCII, LATIN1, or LATIN9 respectively. Another solution might be to set the default database language to a non-UNICODE language when creating the database cluster (`initdb --encoding SQL_ASCII`). However, this solution has not been tested, yet.

2.2 Installation on Windows Systems

The release for Windows Systems is contained in a compressed file `psx-xxx.zip` where `xxx` must be replaced with the actual release number. After decompressing the file, the executable `psx.exe` and a sample configuration and specification file are ready to use. The program can be invoked via the commandline.

2.3 Database System

Currently, the only database system that is directly supported with its native interface is PostgreSQL. In addition, the ODBC database standard is supported as a driver. However, each of the existing database access methods requires some preliminary preparation.

The database system must be properly installed in such a way that the software is able to access it. Therefore, the user account under which the PID Generator is intended to run should be provided with the necessary rights to communicate with the database server. Moreover, the database server should be configured to accept connections from the machine the PID Generator will run on.

If the database system of choice is PostgreSQL one should make sure that the postmaster process is running and accepting TCP/IP connection from clients, especially from the machine where the PID Generator is running on. The postmaster service should be started with the `-i` option which allows for remote connections. This also holds if the database system and the PID Generator software are located on the same host. In addition, it is necessary to link to the libpq library. This is automatically done during the configuration process if the header file `libpg-fe.h` is present (usually in `/usr/include/postgresql/`). Refer to the PostgreSQL system manuals, if you encounter any problems [1].

Before setting up the actual database, the data items, the format, the matching strategy, and the possible outcomes must be defined in the specification file (see section 4 *Specification*). The command `psx gen` can then be used to create the PID database and the tables (see section 5.1 *Database Generation*). For database systems other than PostgreSQL, the database has to be created manually before running the `gen` command which will then set up the tables.

The PID database consists of the following tables.

`sch`	schema definitions
`cfg`	configuration
`ctp`	component types
`fld`	field definitions
`rsp`	result specifications
`sts`	state specifications
`rec`	data records
`req`	PID requests

Some of these tables are fixed, i.e. they must not be modified after their creation. These tables are `sch`, `fld`, `ctp`, `rsp`, and `sts`. The tables `fld`, `rsp`, `sts` reflect the data field, result, and status specifications respectively as defined in the specification file.

In contrast to those fixed tables, the table `rec`, `req`, and `cfg` are updated at each PID request. The table `rec` holds the actual data records and is updated each time a record is modified or newly created. The table `req` contains all information on PID requests, thus reflecting the request history if configured to do so (see section 3.4 *Request History*). The information include all data fields of the input record, the request outcome, a timestamp as well as the data fields of the matched record if a match was found. Therefore, this table can also be used for manual or automatic evaluation of the matching algorithm. Finally, the table `cfg` contains the timestamp of the last PID request and the current number which is used in the PID generating algorithm.

The system administrator should assure that backups of the database are made regularly. The current number - along with the PID encryption keys - contains the most important information for the PID Generator. Its loss will most likely cause all previously assigned PIDs to be invalid.

The tables are defined as follows:

Table sch

```
idx     integer     NOT NULL     // schema identifier
sym     text        NOT NULL     // schema name
```

Table cfg

```
atr     text        NOT NULL    // attribute name
val                 NOT NULL    // attribute value
```

Table ctp

```
sym     text        NOT NULL    // identifier of component
```

Table fld

```
idx     integer      NOT NULL   // field identifier
sch     integer                 // corresponding schema
sym     text         NOT NULL   // name of field
dtp     text                    // datatype
lbl     text                    // label
pos     int                     // start position
len     int                     // length
mini    int                     // minimal length
maxi    int                     // maximal length
tfm     text                    // transformation code
cps     text                    // components
equ     text                    // equality code
```

Table rsp

```
idx     integer      NOT NULL   // result identifier
sch     integer                 // corresponding schema
sym     text         NOT NULL   // name of result
prm     character(1)            // PID retrieval mode
rum     character(1)            // record update mode
msg     text                    // message
ntf     text                    // notification
```

Table sts

```
idx     integer      NOT NULL   // status identifier
sch     integer                 // corresponding schema
sym     text         NOT NULL   // name of status
sps     text                    // KSXO specification
tg0     text                    // target - no match
tg1     text                    // target - one match
tgm     text                    // target - multiple matches
```

Table rec

```
pid     character(8) NOT NULL   // PID
sub     character(8)            // substitute PID
sur     integer                 // sureness
f_{field_name}                  // fields according to table fld
...
```

Table req

```
idx     integer     NOT NULL    // request identifier
pid             character(8)    // PID
tsp             text            // timestamp
adr             text            // address
prt             text            // port
hst             text            // host
usr             text            // user
sur             integer         // sureness ([0|1])
result          text            // result identifier
mode            text            // PID retrieval mode ([gen | get])
upd             integer         // update mode ([0|1])
f_{field_name}  text            // fields according to table fld
...
f_{field_name}_o text           // fields of matched record
...
```

2.4 Example: Initial setup of the GPOH database

sch

```
| idx | sym        |
|-----|------------|
| 1   | Schema     |
```

cfg

```
| atr  | val                  |
|------|--------------------- |
| sch  | 1                    |
| pix  | 00000000             |
| req  |                      |
```

ctp

```
| sym |
|-----|
| C1  |
| C2  |
| C3  |
| PC  |
| PH  |
```

fld

```
| idx | sch |  sym  | dtp |      lbl      | pos | len | mini | maxi | ...
|-----|-----|-------|-----|---------------|-----|-----|------|------|----
|  1  |  1  | lname |  N  | Name          |   1 | 12  |  2   |  0   | ...
|  2  |  1  | aname |  N  | Name (a)      |  14 | 15  |  0   |  0   | ...
|  3  |  1  | fname |  N  | Vorname       |  30 | 15  |  0   |  0   | ...
|  4  |  1  | bd    |  DD | Geburtstag    |  62 | 02  |  0   |  0   | ...
|  5  |  1  | bm    |  DM | Geburtsmonat  |  65 | 02  |  0   |  0   | ...
|  6  |  1  | by    |  DY | Geburtsjahr   |  68 | 04  |  0   |  0   | ...
|  7  |  1  | plz   |  T  | PLZ           |  73 | 08  |  0   |  0   | ...
|  8  |  1  | loc   |  T  | Ort           |  82 | 09  |  0   |  0   | ...
|  9  |  1  | state |  T  | Staat         |  92 | 10  |  0   |  0   | ...
| 10  |  1  | sex   | SEX | Geschlecht    | 103 | 12  |  0   |  0   | ...


... |        tfm         |       cps       | equ |
----|--------------------|-----------------|-----|
... | *:D[N]:P[*]:E[AES] | C1:C2:C3:PC:PH  | +:1 |
... | *:D[N]:P[*]:E[AES] | C1:C2:C3:PC:PH  | -:1 |
... | *:D[F]:P[*]:E[AES] | C1:C2:C3:PC:PH  | +   |
... | *                  |                 | +   |
... | -                  |                 | +   |
... | -                  |                 | +   |
... | -                  |                 | -   |
... | -                  |                 | *   |
... | -                  |                 | *   |
... | -                  |                 | *   |
```

sts

```
| idx | sch |   sym    |     sps     |    tg0     |    tg1     |   tgm    |
|-----|-----|----------|-------------|------------|------------|----------|
|  1  |  1  | T_Start  | I           | S:T_S0_001 | S:T_S1_001 | Z        |
|  2  |  1  | T_S0_001 | K0:S1:X1:01 | S:T_S0_002 | R:POS_NUP  | R:NIR    |
|  3  |  1  | T_S0_002 | K0:S1:X1:00 | S:T_S0_003 | R:POS_NUP  | R:AMB_SUR|
|  4  |  1  | T_S0_003 | K0:S0:X1:01 | S:T_S0_004 | R:POS_WCP  | R:NIR    |
|  5  |  1  | T_S0_004 | K0:S0:X1:00 | S:T_S0_005 | R:POS_WCP  | R:AMB_UNS|
|  6  |  1  | T_S0_005 | K0:S*:X0:01 | S:T_S0_006 | R:POS_TNT  | R:AMB_SIM|
|  7  |  1  | T_S0_006 | K0:S*:X0:00 | R:NEG      | R:NEG_TNT  | R:AMB_SIM|
|  8  |  1  | T_S1_001 | K0:S1:X1:01 | S:T_S1_002 | R:POS_DIR  | R:NIR    |
|  9  |  1  | T_S1_002 | K0:S1:X1:00 | S:T_S1_003 | R:POS_DIR  | R:AMB_SUR|
| 10  |  1  | T_S1_003 | K0:S0:X1:01 | S:T_S1_004 | R:POS_WNG  | R:NIR    |
| 11  |  1  | T_S1_004 | K0:S0:X1:00 | S:T_S1_005 | R:POS_WNG  | R:AMB_UNS|
| 12  |  1  | T_S1_005 | K0:S0:X0:01 | S:T_S1_006 | R:POS_TNT  | R:AMB_SIM|
| 13  |  1  | T_S1_006 | K0:S0:X0:00 | R:NEG      | R:NEG_TNT  | R:AMB_SIM|
```

rsp

```
| idx | sch|   sym   | prm| rum|               msg                    | ...
|-----|----|---------|----|----|--------------------------------------|----
|  1  |  1 | NIR     |  0 |  0 | Es wurden mehrere Fälle mit identi...| ...
|  2  |  1 | POS_DIR |  1 |  1 | Sie können den PID mit Kopieren/Ei...| ...
|  3  |  1 | POS_NUP |  1 |  1 | Sie können den PID mit Kopieren/Ei...| ...
|  4  |  1 | POS_WNG |  1 |  1 | Es wurde ein passender, als > unsi...| ...
|  5  |  1 | POS_NUP |  1 |  0 | Es wurde ein passender Fall gefund...| ...
|  6  |  1 | POS_WCP |  1 |  0 | Es wurde ein passender Fall gefund...| ...
|  7  |  1 | POS_TNT |  1 |  1 | Es wurde ein sehr ähnlicher Fall g...| ...
|  8  |  1 | NEG     |  * |  0 | Sie können den PID mit Kopieren/Ei...| ...
|  9  |  1 | NEG_TNT |  0 |  0 | Es wurden ein oder mehrere ähnlich...| ...
| 10  |  1 | AMB_SUR |  0 |  0 | Es wurden mehrere Fälle mit identi...| ...
| 11  |  1 | AMB_UNS |  0 |  0 | Es wurden mehrere Fälle mit identi...| ...
| 12  |  1 | AMB_SIM |  0 |  0 | Es wurden ein oder mehrere ähnlich...| ...


... |          ntf               |
----|----------------------------|
... | Identische Fälle gefunden. |
... | Direct Match.              |
... |                            |
... |                            |
... |                            |
... |                            |
... |                            |
... |                            |
... |                            |
... |                            |
... |                            |
... |                            |
```

rec

```
| pid | sub | sur | f_lname_c1 | f_lname_c2 | f_lname_c3 | f_lname_pc |...
|-----|-----|-----|------------|------------|------------|------------|---


...| f_lname_ph | f_aname_c1 | f_aname_c2 | f_aname_c3 | f_aname_pc |...
---|------------|------------|------------|------------|------------|----


...| f_aname_ph | f_fname_c1 | f_fname_c2 | f_fname_c3 | f_fname_pc |...
---|------------|------------|------------|------------|------------|----


...| f_fname_ph | f_bd | f_bm | f_by | f_plz | f_loc | f_state | f_sex |
---|------------|------|------|------|-------|-------|---------|-------|
```

req

```
| idx | pid | tsp | adr | prt | hst | usr | sur | result | mode | upd | ...
|-----|-----|-----|-----|-----|-----|-----|-----|--------|------|-----|----

...| f_lname_c1 | f_lname_c2 | f_lname_c3 | f_lname_pc | f_lname_ph | ...
---|------------|------------|------------|------------|------------|----

...| f_aname_c1 | f_aname_c2 | f_aname_c3 | f_aname_pc | f_aname_ph | ...
---|------------|------------|------------|------------|------------|----

...| f_fname_c1 | f_fname_c2 | f_fname_c3 | f_fname_pc | f_fname_ph | ...
---|------------|------------|------------|------------|------------|----

...| f_bd | f_bm | f_by | f_plz | f_loc | f_state | f_sex |...
---|------|------|------|-------|-------|---------|-------|----

...|f_lname_c1_o|f_lname_c2_o|f_lname_c3_o|f_lname_pc_o|f_lname_ph_o| ...
---|------------|------------|------------|------------|------------|----

...|f_aname_c1_o|f_aname_c2_o|f_aname_c3_o|f_aname_pc_o|f_aname_ph_o| ...
---|------------|------------|------------|------------|------------|----

...|f_fname_c1_o|f_fname_c2_o|f_fname_c3_o|f_fname_pc_o|f_fname_ph_o| ...
---|------------|------------|------------|------------|------------|----

...| f_bd_o | f_bm_o | f_by_o | f_plz_o | f_loc_o | f_state_o | f_sex_o |
---|--------|--------|--------|---------|---------|-----------|---------|
```

2.5 Web Interface

The PID software provides a special command (`gui`) which enables a PID request to be invoked via a web browser. It indicates that the program is executed in CGI mode and thus shows the appropriate behaviour. Note that this graphical user interface requires the installation of a Web Server.

The general installation method of the graphical user web interface is to wrap this invocation in a script (`psx.cgi`) that will be called by the web server. The script should be placed into the `cgi-bin` directory of the web server and provided with the corresponding rights. The following shell script provides for the corresponding functionality.

psx.cgi

```
#!/bin/sh

dir="/sys/svr/psx"
prg="${dir}/psx"
cfg="${dir}/psx.cfg"
cmd="${prg} gui -c:${cfg}"

${cmd}
```

Note that the `dir` variable should be adapted to the local installation. For a quick test, one may call the script from a web browser without arguments. Any problems arising during execution will be written to the log file, if it is configured properly (see section 3.3 *Logging*).

To employ self-defined html documents instead of the automatically generated web pages one can configure the use of templates in the configuration file (see section 3.9 *Templates*).

3 Configuration

The configuration of the software is located within a special file that should be accessible at execution time. The default file name is `psx.cfg` and the default location is the current directory. Especially in the context of web server applications using CGI one should not rely on these defaults and one should make sure that the location of the configuration file is explicitly given on the commandline using the `-c` option. The configuration file is organized as a set of attribute-value-assignments. Double-slashes introduce comments that consume the rest of the current line. The most important sections relate to the database connection, the logging mechanism and the default character set for the encoding of encrypted data.

3.1 Database

The database connection is specified with the attribute `dbs`, as a single string of tokens, separated by colons. Note that, in general, the syntax of this string is DBMS-specific, i. e. depending on the database access method there will be different representations of the corresponding access specification. A connection string always begins with a prefix, which identifies a unique database driver, e. g. a specific system such as PostgreSQL or a specific standard such as ODBC. The rest of the connection string will be specific to the selected driver. However, in general, the syntax of the database connection string is defined as follows.

Syntax		Connection Specification
Connection Specification	::=	`dbs =` [`pgs` : *pgs access* | `odbc` : *odbc access*]
pgs access	::=	*host* : *port* : *name* : *user* : *password*
odbc access	::=	*datasource name* : *user* : *password*

Example

```
dbs = pgs:localhost::pdb:foo:bar
```

This expression specifies a database connection on the actual machine to the database named ‘pdb’, accessed as the user ‘foo’ with password ‘bar’, using the native database driver for PostgreSQL.

```
dbs = odbc:psx:foo:bar
```

This expression is an example of a connect string for a database system using ODBC. The datasource name is specified as ‘psx’ which will be accessed by the user ‘foo’ using the password ‘bar’.

3.2 SQL Syntax

This section is only applicable for database systems using the ODBC driver.

Different database systems require slightly different SQL syntaxes. To make the PID Generator software as flexible as possible, the attribute `sql` my be used to adapt the syntax of certain statements to the respective database system.

Syntax SQL Syntax

SQL syntax	::=	*begin syntax*
		commit syntax
		abort syntax
		isnull syntax
		semicolon
begin syntax	::=	`sql.begin =` *begin statement*
commit syntax	::=	`sql.commit =` *commit statement*
abort syntax	::=	`sql.abort =` *abort statement*
isnull syntax	::=	`sql.null = [’IS NULL’|’ISNULL’]`
semicolon	::=	`sql.sem = [0|1]`

For `sql` attributes not specified in the configuration file the default values are assumed. The default values for the begin, commit, and abort statements are `BEGIN TRANSACTION`, `COMMIT TRANSACTION`, and `ABORT TRANSACTION` respectively. The isnull syntax defaults to `IS NULL` and no semicolon is appended to the statements. These settings have successfully been tested with MS SQL Server.

Example

```
sql.begin = ‘START TRANSCATION’
sql.commit = ‘COMMIT’
sql.abort = ‘ROLLBACK’
```

The settings above have been tested with a MySQL database. Since the isnull syntax equals the default value and no semicolon should be appended, it suffices to specify these three statements that differ from the default syntax.

3.3 Logging

The location of the log file and the activities that are logged are specified with two attributes.

The attribute `log.file` specifies the path to the file where logs are written to. This should be an absolute path and the file should be accessible under the user account the software is run. Note that this file may grow very rapidly, depending on the log mask.

The attribute `log.mask` specifies a set of modules, the activities of which are logged. Each module is represented with a mnenomic symbol, and the set of modules is specified as a sequence of symbols, separated by colons. The symbol ‘`-`’ disables all logging activities while the symbol ‘`*`’ enables logging for all modules. Note that the modules `cpi`, `cni`, and `dbi` may log data items in cleartext which may be undesirable for privacy reasons.

Syntax Logging Specification

Logging Specification	::=	*file mask*
file	::=	`log.file =` *filename*
mask	::=	`log.mask = [ - `| `*` | *module*{`:`*module*}`* ]`

You can specify the following modules to be logged.

`sys`	System Interface
`cmd`	Command Interface
`cpi`	Cipher Interface
`aes`	AES Encryption Interface
`idea`	IDEA Encryption Interface
`cni`	Control Number Interface
`dbi`	Database Interface
`dbi/pgsql`	PostgreSQL Database Interface
`dbi/odbc`	ODBC Driver Interface
`pdi`	PID Database Interface
`psi`	PID Service Interface
`mti`	Matching Interface

Example A log mask such as `sys:dbi:mti` would cause the logging of the activities of the system interface, the database interface and the matching interface. The activities of `sys` include the basic startup and finalization steps. Enabling `dbi` will cause any database commands to be written to the log file, while enabling `mti` will cause the whole matching procedure to be documented.

3.4 Request History

Each request for a PID will be logged in the database table `req` which is thus reflecting the request history. The stored data comprises at least a timestamp and the assigned PID.

If configured to do so, it will additionally store the input data record in its original form (before encipherment), the match result, and - if applicable - the data items of the matching record found in the database. Since this may be undesirable in some cases for data privacy reasons, this option can be disabled by setting the value to ‘`0`’.

Syntax Request History

Request History Specification ::= `History = [0|1]`

3.5 Mailing

The mailing interface allows for the individual notification of PID request events. Whenever something happens the administrator would like to know, the PID Generator will automatically send a notification mail to a predefined address, if configured to do so. Such a notification mail will contain some obligatory information such as a timestamp, the name and address of the requesting person, if available, as well as a special message that relates to a special outcome of the matching algorithm.

The mailing specification involves five attributes that define the names and addresses of the sender and the receiver of a notification mail, as well as the subject. These attributes are specified under a node named `mail`, involving subordinated nodes for the source and the destination of the mailings.

Syntax Mailing

Mailing Specification	::=	*source*
		destination
		subject
source	::=	`mail.src.name` = *name*
		`mail.src.adr` = *address*
destination	::=	`mail.dst.name` = *name*
		`mail.dst.adr` = *address*
subject	::=	`mail.sub` = *string*

Example

```
mail.src.name  = ‘PSX’
mail.src.adr   = ‘admin@domain.de’
mail.dst.name  = ‘Admin’
mail.dst.adr   = ‘admin@domain.de’
mail.sub       = ‘PSX Notification’
```

The configuration above causes notifications to be sent to ‘Admin’. The emails will contain the subject line ‘PSX Notification’. The sender’s and the recipient’s addresses are set to ‘admin@domain.de’ and the sender name to ‘PSX’.

3.6 Encoding

The encoding of some or all data items may be necessary in some settings - like cancer registry, for example. Control numbers are generated as MD5 hash codes, which are traditionally represented as printable ASCII characters , ranging from `‘!’` to `‘u’` [2]. However, using this character set, which contains special characters such as apostrophes or backslash, may cause additional conversion overhead with most database systems. Therefore, an optional hexadecimal representation was introduced, which comes along with codes that are somewhat longer though, but much more simpler when dealing with database systems.

The encoding of data items may be specified at the commandline but will be retrieved from the configuration if not given at the commanline. The default encoding may be specified with the attribute `ctn.enc`, the value of which can be `raw`, for the raw encoding, or `hex`, for hexadecimal encoding, respectively. Note that commandline arguments will always override default specifications.

For ordinary control numbers, the raw encoding of a single data item will always result in a string with a length of 23 characters. The hexadecimal encoding of a data item will result in a string with a length of 32 characters. In any case, the actual output format will specify the positions and lengths of the data items.

Syntax		Default encoding
Encoding Specification	::=	`ctn.enc =` *encoding*
encoding	::=	`[ raw` | `hex ]`

3.7 Directory Access

Directory service access provides for restrictions on the users that may submit requests to the PID service. If enabled, an LDAP server will be consulted whenever a user tries to submit a request. The model of the server's directories has to support a special attribute `pid` that states wether a user is allowed to do so. If the lookup is successful, the request will be processed as usual. In any other cases, the request will be rejected.

Directory service access is enabled with the attribute `dir.lookup`. If this attribute is set to `0`, the directory configuration has no effect at all. The LDAP server used for access validation is specified with the attribute `dir.server`. The base DN is specified with the attribute `dir.base`.

For now, directory service access is implemented as an experimental extension. One should not consider this feature without being able to setup a dedicated LDAP server with a customised schema.

Syntax		Directory Access Specification
Directory Access Specification	::=	*lookup server base*
lookup	::=	`dir.lookup = [` 0 | 1 `]`
server	::=	`dir.server =` *host*
base	::=	`dir.base =` *base DN*

3.8 PID Construction

PID construction is performed in two steps. First, an internal counter is incremented. Secondly, this counter is mapped into a string that does not give insight on the contents of the database. This string is the well-known PID.

The mapping function depends on some parameters that are retrieved from the configuration file. **These parameters must never change** - a change would make all formerly generated PIDs invalid.

The three *encryption keys* (k_1, k_2, k_3) are internally represented as unsigned 32-bit integers and can thus contain any number between 0 and 4,294,967,295. The *random width* determines the number of bits used for randomization and can be between 0 and 12. Randomisation makes it more difficult to attack the encryption system with known plain text. On the other hand, it reduces the number of PIDs that can potentially be generated.

Syntax PID Construction Specification

PID Construction Specification	::=	*k1 k2 k3 rw*
k1	::=	`pid.k1 =` *integer*
k2	::=	`pid.k2 =` *integer*
k3	::=	`pid.k3 =` *integer*
rw	::=	`pid.rw =` *integer*

3.9 Templates

Templates are user-defined HTML documents for the web interface. They are meant to replace the HTML output that is created automatically by the PID Generator if no templates are supplied.

There are three different situations considered by the web interface, namely the PID request (`req`), the PID return (`ret`) if the request was successfully executed, and the message (`msg`) in case the request could not be satisfied. These correspond to three different templates that can be specified within the configuration with the attributes `tpl.req`, `tpl.ret` and `tpl.msg`. Each of these attributes should be assigned an absolute path to a template file that should be readable by the system.

`req`	PID request panel
`ret`	PID return panel
`msg`	message panel

When the user accesses the service, he usually gets a PID request form, where he enters the patient data and submits the request. If the request is processed successfully, a result page containing the PID along with other information is returned. If the request is not processed successfully, a message page is displayed containing information about the reasons for the request failure.

The dynamics of request processing are realized with special placeholders, which are embedded within the templates and replaced during execution. These variables begin with a dollar sign ($) and may be placed within the documents. Those variables that represent the fields of the input records begin with the prefix `FLD_` to avoid naming conflicts.

Note the following on fields that represent human sex. Implementation specific templates should present such a field as a set of option buttons with the values 'M', 'F', or 'N', indicating male, female, or unknown sex, respectively.

If a template is not specified or not loadable at execution time for some reason, a default document will be generated automatically. This document may look somewhat rigid, but it is a good starting point for the installation. The following table shows the set of placeholders that may be used within template files, as well as their applicability for the different template types.

symbol	description	req	ret	msg
FLD_*	input fields corresponding to specification	+		
FLD_SUR	sureness (0 or 1)	+		
PID	the assigned PID		+	
CTN	control numbers generated for the input		+	
MSG	descriptive message for the user		+	+
USR	user name, if processed in protected mode		+	
TSP	time stamp of PID request		+	
DATE	date of PID request		+	
TIME	time of PID request		+	
ADR	IP address of the requesting host		+	
HST	name of the requesting host		+	
PRT	port of the requesting host		+	
SSL_CLI	SSL: client certificate		+	
SSL_CLI_C	SSL: country		+	
SSL_CLI_N	SSL: name		+	
SSL_CLI_E	SSL: email		+	
SSL_CLI_L	SSL: location		+	
SSL_CLI_O	SSL:organization		+	
SSL_CLI_U	SSL:organizational unit		+	
SSL_CLI_S	SSL: state		+	

3.10 Messages

The message items may be used to specify messages presented to the user in case of erroneous data input. Four different types can be distinguished: Missing values, too short or too long input, or otherwise invalid values. In addition, result messages of the matching process can be replaced without changing them in the database. That means, if a result message for a particular result is defined in the configuration file, it takes precedence over the one defined in the specification file and stored in the database[1].

Syntax Message Specification

Message Specification	::=	*incomplete min max invalid result*
incomplete	::=	`msg.req.inc` = *string*
min	::=	`msg.req.min` = *string*
max	::=	`msg.req.max` = *string*
invalid	::=	`msg.req.inv` = *string*
result	::=	`msg.req.`*id* = *string*

[1]See Section 4.2 for information on result specifications.

id has to be replaced by a valid result id (see Section 4.2). Note that the first four message strings may contain the placeholders `$FLD` and `$LEN` that will be replaced by the corresponding values if an incorrect item is found.

Example

```
msg.req.inc = ‘Fill in the field ‘$FLD’, please.’
msg.req.min = ‘The length of ‘$FLD’ must at least amount to $LEN.’
msg.req.max = ‘The length of ‘$FLD’ may not exceed $LEN.’
msg.req.inv = ‘The content of field ‘$FLD’ is invalid.’
```

3.11 Encryption

Another configuration option allows to protect sensitive data by encryption before storing them in the PID database. This is only applicable though if the generation of control numbers has been activated for these items as is done in cancer registration. The encryption specification contains the password used for AES or IDEA encryption. The items themselves have to be defined in the specification file (see section 4.1 *Transformation*).

Syntax		Encryption Specification
Encryption Specification	::=	`[` *aes specification* | *idea specification* `]`
aes specification	::=	`aes.key =` *string*
idea specification	::=	`idea.key =` *string*

3.12 Health Insurance Code Validation

The PID Generator software includes some functions to verify the health insurance codes for a variety of health insurance organizations. A code identified as erroneous will cause the processing of the input record to stop which may be undesirable in some cases. Therefore an option is provided to control this feature. If not set, the validation will be disabled by default.

Syntax		Health Insurance Code Validation
Health Insurance Code Validation	::=	`ValidateHIC = [0|1]`

3.13 Example: The GPOH configuration

GPOH Configuration

```
//////////////////////////////////////////////////////////////////////////
// GPOH Configuration                                                   //
//////////////////////////////////////////////////////////////////////////

// database connection
dbs          = pgs:localhost::psx_db:psx_user:

// logging
log.file     = /sys/log/psx.log
log.mask     = sys:dbi:pdi

// encoding
ctn.enc      = hex

// mailing
mail.src.name = ‘PSX’
mail.src.adr  = ‘PIDservice@gpoh.de’
mail.dst.name = ‘Admin’
mail.dst.adr  = ‘PIDService@gpoh.de’
mail.sub      = ‘PSX Notification’

// templates
tpl.req      = /sys/dvl/prj/psx/web/psx-req.html
tpl.ret      = /sys/dvl/prj/psx/web/psx-ret.htm
tpl.msg      = /sys/dvl/prj/psx/web/psx-msg.html

// messages
msg.req.inc = ‘Das Feld ‘$FLD’ darf nicht leer sein.’
msg.req.min = ‘Die Länge des Feldes ‘$FLD’ muss mindestens $LEN sein.’
msg.req.max = ‘Die Länge des Feldes ‘$FLD’ darf höchstens $LEN sein.’
msg.req.inv = ‘Der Inhalt des Feldes ‘$FLD’ ist ungültig.’

// PID construction
pid.k1       = 1030420120      // key 1
pid.k2       = 237344121       // key 2
pid.k3       = 365421576       // key 3
pid.rw       = 0               // random width

// Encryption
aes.key      = ‘Insert your favourite passphrase here’
```

4 Specification

A configurable specification file allows to customize the PID Generator to the local requirements. It consists of three sections, the *data format* section that describes the contents of a data record, the *result* section that defines the set of possible results the matching engine may produce, and the *matching procedure* section that implements the core of the matching strategy.

The specification file is read once when setting up the PID database. At the same time, all the information is stored within the tables `fld` (data fields), `sts` (states of the matching procedure), and `rsp` (results of the matching procedure). Thus, after creating the database, the specification file is not needed anymore but should probably be kept for future reference. The only case in which a specification file might be needed later on, is when using data input files with fixed item boundaries which do not comply to the boundaries specified in the `fld` table. In this case, one can specify a shortened specification file containing only a data format section which is then used by the PID-Generator instead of the settings stored within the database.

4.1 Data Format

The *data format specification* defines the structure of data records. It consists of a sequence of field definitions whereas each field represents one item of the input record. The sequence implicitly defines the order of the items when processed by the different commands. This affects the order of plausibility checks and the automatic generation of data entry forms, for example. However, it does not affect the outcome of the matching procedure at all.

A field definition is introduced by the keyword `Field`, followed by a unique identifier and the assignment block in curly brackets. The field identifier must obey the typical conventions of popular programming languages, i. e., $letter([letter|digit|'\_'])^*$. Every assignment is of the form *attribute* = *value*, separated by white space. The following table shows the set of field attributes allowed within the assignment block.

`type`	data type
`label`	quoted label for user interaction
`start`	start position for reading in ASCII line mode
`length`	length for reading in ASCII line mode
`min`	minimum length for field data, 0 = ignore
`max`	maximum length for field data, 0 = ignore
`transformation`	transformation code
`equ`	equality specification

Syntax Field Specification

Field Specification	::=	`Field` *identifier* { { *assignment* }$^+$ }
assignment	::=	[`type` = *type* |
		`label` = *string* |
		`start` = *integer* |
		`length` = *integer* |
		`min` = *integer* |
		`max` = *integer* |
		`transformation` = *transformation* |
		`equ` = *equality*]

The data `type` defines the valid contents a field may have. The arbitrary `label` is used for external representation to the user. The `start` position and the `length` define the location of a data item within a ASCII file with fixed column boundaries. These attributes, however, are not required if one uses delimiter separated data files instead. `min` and `max` further define the number of characters an item must at least and at most consist of. Finally, the `transformation` code and `equ` (equality) define how the data items are being processed and how they are used within the matching process. See the following sections *Data Types, Transformation Code, and Equality Specification* for more detailed information on these attributes.

Example Figure 4.1 shows an example of a field specification. It defines a field named `lname` for the representation of last names. The type is set to `N` (name) and the label is set to a natural language description ('Last Name'). The start position for this field within a data file is set to 14 and the length of the field is set to 15 (only applicable if the data file uses fixed boundaries instead of a delimiter to separate items). The minimum field length is set to 1, while there is no limitation for the maximum field length. The transformation code specifies that control numbers are to be generated, after name decomposition and generation of phonetics, and that the result is to be encrypted with AES. The equality specifies that the contents of two records for this field must match. In addition they may be exchanged with any other fields holding the same exchange id which is 1 in this example.

```
Field lname
{
  type            = N
  label           = "Last Name"
  start           = 14
  length          = 15
  min             = 1
  max             = 0
  transformation  = "*:D[N]:P:E[AES]"
  equality        = "+:1"
}
```

Figure 4.1: Example field specification

There is no need to put every allowed assignment within such a definition, since missing assignments will always cause default values to be assigned to the corresponding attributes. However, in case of doubt one should not rely on default values.

Data Types

Data types are used to characterize the contents of data fields. This allows suitable normalization and plausibility checks as well as data type specific transformation on input records.

There are types for simple text data, names of persons, dates and date components, sex, and for health insurance organizations and health insurance codes. The following table shows the set of data types and specifies the range of valid values.

Data Type	Description	Values
`T`	text	*alphanumerical string*
`N`	name	*alphabetical string*
`DATE`	date	*date in format YYYY-MM-DD*
`DY`	year of date	*4-digit-number*
`DM`	month of date	`[0-12]`
`DD`	day of date	`[0-31]`
`SEX`	human sex	`[f | m | n]`
`HIO`	health insurance organization	*string*
`HIC`	health insurance code	*string*

Transformation Code

A set of transformation functions can be incorporated to modify the original data items of an input record. These modifications include decomposition of names into components, generation of phonetic codes using different algorithms, generation of control numbers using MD5 hash, as well as final encryption with AES or IDEA. The latter is only applicable though if control numbers are generated. Decomposition of names is described in detail in section A.1. Transformation takes place at the very beginning of any operation, so that the internal input will always be an already transformed record.

The transformation is specified with a *transformation code*, a single string composed of substrings separated by colons. Each substring represents some special transformation option whereas some of them require additional parameters which further specify the algorithms. These parameters are specified in square brackets, right behind the option symbol. Except the symbol for the generation of control numbers (`*` or `-` respectively) which is required and must be specified first, transformation options may be specified in an arbitrary order. The following table shows the set of transformation options.

`-`	do not generate control numbers
`*`	generate control numbers using MD5 hash
`D[N]`	decompose (sur)name
`D[F]`	decompose first name
`P[C]`	generate phonetics using the system of Cologne
`P[H]`	generate phonetics using the system of Hannover
`P[*]`	generate both phonetic codes
`E[IDEA]`	encrypt the resulting components with IDEA
`E[AES]`	encrypt the resulting components with AES

Some guidelines should be respected for choosing phonetic codes:

- Using phonetic code only makes sense for fields of type `name`. Phonetic codes for other types are still stored in the database, but are not considered for matching as of version 1.2.
- The same phonetic codes should be used for all fields in one exchange group. If one, for example, specifies phonetics using the system of Cologne for surnames and phonetics using the system of Hannover for alternative surnames, a phonetic comparision between these fields is not possible, potentially affecting matching success.

Example

```
transformation = "*:D[N]:P[C]:E[AES]"
```

The transformation code above specifies that control numbers should be generated, after decomposition of names and generation of phonetics using the algorithm of Cologne, and that the final result components should be encrypted with AES, using the key specified in the configuration file.

Equality Specification

Whenever two records are compared, it is in fact the field contents that are used to tell whether the records match, i. e. they represent the same individual, or not. The equality specification describes which fields are to be considered and also includes a simple mechanism to describe exchangeability. For example, the fields `name` (surname) and `aname` (alternative, former surname) should be defined as exchangeable.

The equality specification is given as a single string composed of one or two substrings. The first part defines an equality code. The optional second part is separated from the equality code by a colon and specifies an exchange identifier which must be identical for the exchangeable fields. The following table shows the set of equality codes.

`-`	no equality condition; field is not considered in the matching procedure
`+`	mandatory equality; field values must be equal for two records to be considered a match
`*`	optional equality; matching field values increase similarity between records
`#`	key equality; two records are considered to be equal if these field values match

4.2 Result Specification

The *result specification* defines the set of possible results. A result represents a unique situation at the end of the matching process. Beside the identity each result has additional properties. The syntax of a result definition involves a block of attribute-value pairs, which define the properties of the result. This is introduced by the keyword `Result`, followed by a unique symbol and the assignment block in curly brackets. Every assignment is of the form *attribute* = *value*, separated by white space.

Syntax Result Specification

Result Specification	::=	`Result` *id* { { *assignment* }$^+$ }
assignment	::=	[`pid` = [0 | 1 | ∗] |
		`update` = [0 | 1] |
		`message` = *string*]

As in the case of the format specification, the result identifier must be a valid symbol. The `pid` attribute specifies the pid retrieval mode and should always be present. Note that a retrieval definition for existing PIDs requires that a unique record is identified by the matching procedure. The following table summarizes the set of possible values for the `pid` attribute.

`0`	no PID is returned at all
`1`	the PID of an existing record is returned
`*`	a new PID is generated and returned

The `update` attribute specifies, whether an existing record should be completed with the fields of the input record. Note that a positive update definition requires that a unique record is identified by the matching procedure. The following table summarizes the set of possible values for the `update` attribute.

`0`	the existing record will not be modified
`1`	the existing record will be completed

Since the results are produced as identifiers for unique situations in the course of the matching procedure, their true meanings are relative to the context, in which they are selected. This means that one should take care that only suitable results are produced by the matching procedure, because not every result makes sense in some situation. Obviously, it would generally not make much sense to have a PID generated and an existing record updated.

Example Figure 4.2 shows an example of a result specification. It defines a result named `POS_DIR` for a direct, positive match. The pid retrieval mode is set to 1, which means that the pid of the matched record is to be returned. The `update` attribute is set to 1, which means that the matched record is to be updated with the appropriate contents of the input record. There also is a message that should be returned to the user, whenever this result is computed.

```
Result POS_DIR    // positive, direct match
{
  pid             = 1
  update          = 1
  message         = "PID Retrieval successful"
}
```

Figure 4.2: Example result specification

4.3 Matching Procedure

The *matching procedure specification* defines the overall algorithmic strategy for the matching of input records against the PID database. It characterizes the sequence of tests carried out defining a network of states, results and transitions. The arrangement of these analysis steps and their linkage with the final results are the core of each individual matching strategy and should therefore carefully be developed for each project.

Upon activation, some starting state is declared to be the active state. The proceeding from the active state follows one of its defined exits which are links to either other states or results. Results define the end of the matching procedure.

There are two types of states, namely *input states* which require some sort of user input and *query states* in which a distinct database query is executed on the PID database. The input state normally occurs exactly once at the beginning of a matching schema and refers to the sureness of the input record. The proceeding from an input state follows one of two links, which refer to an input record marked as unsure (0) or marked as sure (1). In contrast to that, a query state delivers an numeric result representing the number of matched records that fulfill the query condition. The proceeding from a query state follows one of three exits, which are associated with the query delivering no record (0), exactly one record (1) or more than one record ($*$).

The specification of a state always consists of a unique symbolic identifier introduced by the keyword `Test`, a prologue section, and an epilogue section. The identifier allows other states to reference the state in their exit definitions, while the epilogue section connects the exits of the actual state to other states or results. The prologue section of an input state simply consists of the literal `input sure ?`. The prologue section of a query state, however, defines a database query depending on the parameters `key`, `sure`, `exact`, and `optional`. The meaning of the parameters is as follows. Key relevance selects only records with equal key fields (1) if these are present in the input record, or it has no effect at all (0). Sureness selects all records that are marked as unsure (0), sure (1) or both ($*$). Exactness causes fields to be compared either exactly (1) or by phonetic similarity

(0) if phonetic codes have been generated[1]. Optionality causes the comparison to include optional values (1) or to ignore their existence (0). If optionality is turned on, *all* of the optional variables have to be equal for a match. If a single optional variable differs from the input record, the corresponding database record is considered to be different. In combination, these parameters define some kind of filter on the records stored in the database. It may be more or less strict in the different states of the matching procedure. Figure 4.3 illustrates the parameters using the so called KSXO notation.

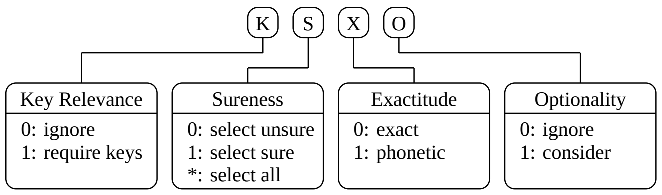

Figure 4.3: KSXO notation

The exit specification defines the connections of the actual state to other states or results. The syntax requires an exit identifier (`[0|1|*]`), followed by a colon, followed by the identifier of another state to proceed to another query, or of a result terminating the test sequence. For an input query state, only the exits 0 and 1 are defined, corresponding to an unsure or sure input record. For a data query state, the exits 0, 1 and $*$ are defined, corresponding to the different outcomes of the query. The most important integrity constraint is that there never are references to undefined objects.

[1]see section 4.1 for some guidelines concerning phonetic codes

Syntax Test Specification

Test Specification	::=	`Test` *id* { *prologue epilogue* }
prologue	::=	`Prologue` { *query* }
epilogue	::=	`Epilogue` { *exit* }
query	::=	[*input query* | *data query*]
input query	::=	`input sure ?`
data query	::=	`key` = *key*
		`sure` = *sureness*
		`exact` = *exactness*
		`optional` = *optionality*
sureness	::=	[0 | 1 | $*$]
exactness	::=	[0 | 1]
key	::=	[0 | 1]
optionality	::=	[0 | 1]
exit	::=	0 : *target*
		1 : *target*
		$*$: *target*
target	::=	[*state identifier* | *result identifier*]

Example Figure 4.4 shows an example of a test specification. The definition refers to a query state, which selects all records that are marked as unsure and the fields of which are exactly equal to the fields of the input record, considering optional fields and not considering key fields. The corresponding KSXO notation is `K0:S0:X1:O1`. There are three exits to the state `T_S1_004`, the result `POS_WNG`, and the result `NIR`, which are to be selected, when the query returns 0, 1 or more records, respectively. The result `POS_WNG` represents a situation, where a record is matched positively but the corresponding query is too weak to lead to a direct match, and thus, the result is associated with a warning message. The result `NIR` is associated with multiple database hits so that the ambiguity prevents a direct match.

```
Test T_S1_003
{
  Prologue
  {
    key            = 0
    sure           = 0
    exact          = 1
    optional       = 1
  }
  Epilogue
  {
    0 : T_S1_004
    1 : POS_WNG
    * : NIR
  }
}
```

Figure 4.4: Example test specification

Flow notation

The flow of analysis may be represented with a state-chart diagram. States are represented as diamonds, while results are represented as rounded rectangles. Any object has an entry point, which connects it to an exit of another state. An input query state has two exits, while a data query state has three exits, corresponding to an outcome of zero, one or more records, labeled with 0, 1 and $*$, respectively. A result object has no exits, since it constitutes a final state of analysis. In general, any path the analysis process will form leads from an input query state to a final result.

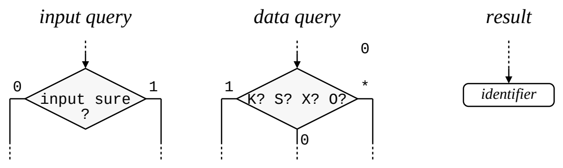

Figure 4.5: Procedure notation

Example Figure 4.6 shows the GPOH matching diagram.

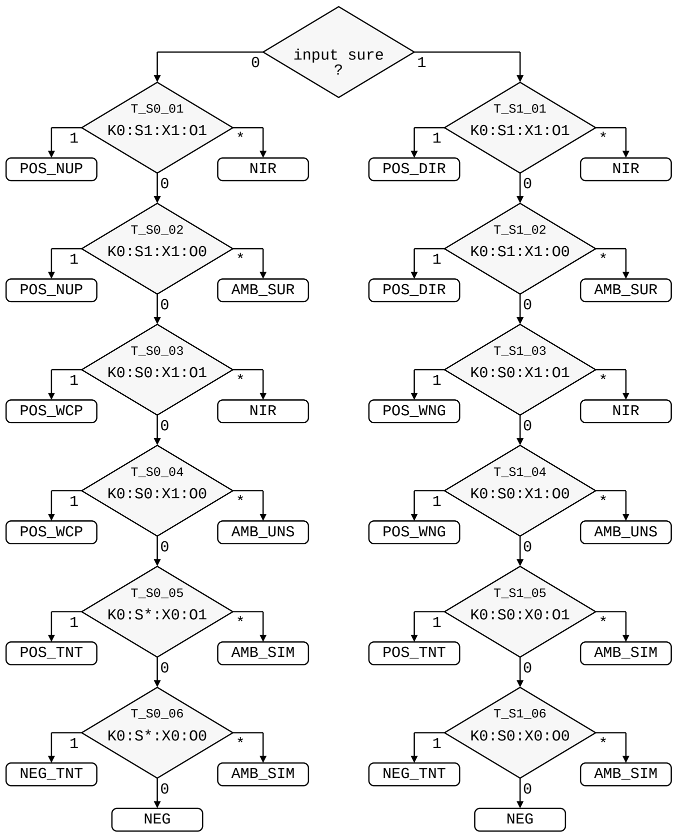

Figure 4.6: The GPOH matching diagram

4.4 Example: The GPOH specification

GPOH Schema Specification

```
//////////////////////////////////////////////////////////////////////////
// Field Specification                                                  //
//////////////////////////////////////////////////////////////////////////


Field lname {
 type           = N
 label          = "Name"
 start          = 01
 length         = 12
 min            = 2
 max            = 0
 transformation = "*:D[N]:P[*]:E[AES]"
 equ            = "+:1"
}

Field aname {
 type           = N
 label          = "Name (a)"
 start          = 14
 length         = 15
 min            = 0
 max            = 0
 transformation = "*:D[N]:P[*]:E[AES]"
 equ            = "-:1"
}

Field fname {
 type           = N
 label          = "Vorname"
 start          = 30
 length         = 15
 min            = 0
 max            = 0
 transformation = "*:D[F]:P[*]:E[AES]"
 equ            = "+"
}

Field bd {
 type           = DD
 label          = "Geburtstag"
 start          = 62
 length         = 02
 min            = 0
 max            = 0
 transformation = "*"
 equ            = "+"
}
```

```
Field bm {
 type           = DM
 label          = "Geburtsmonat"
 start          = 65
 length         = 02
 min            = 0
 max            = 0
 transformation = "-"
 equ            = "+"
}

Field by {
 type           = DY
 label          = "Geburtsjahr"
 start          = 68
 length         = 04
 min            = 0
 max            = 0
 transformation = "-"
 equ            = "+"
}

Field plz {
 type           = T
 label          = "PLZ"
 start          = 73
 length         = 08
 min            = 0
 max            = 0
 transformation = "-"
 equ            = "-"
}

Field loc {
 type           = T
 label          = "Ort"
 start          = 82
 length         = 09
 min            = 0
 max            = 0
 transformation = "-"
 equ            = "*"
}

Field state {
 type           = T
 label          = "Staat"
 start          = 92
 length         = 10
 min            = 0
 max            = 0
 transformation = "-"
 equ            = "*"
}

Field sex {
 type           = SEX
 label          = "Geschlecht"
 start          = 103
 length         = 09
 min            = 0
 max            = 0
 transformation = "-"
 equ            = "*"
}

//////////////////////////////////////////////////////////////////////////
// Result Specification                                                 //
//////////////////////////////////////////////////////////////////////////

Result NIR {
 pid            = 0
 update         = 0
 message        = "Es wurden mehrere Fälle mit identischen Daten gefunden."
                    "Eine PID-Vergabe ist daher nicht möglich."
}

Result POS_DIR {
 pid            = 1
 update         = 1
 message        = "Sie können den PID mit Kopieren/Einfügen in"
                    "andere Dokumente einfügen."
}

Result POS_NOP {
 pid            = 1
 update         = 1
 message        = "Sie können den PID mit Kopieren/Einfügen in"
          "andere Dokumente einfügen."
        "Bemerkung: Die ergänzenden Angaben waren"
        "unvollständig oder die Adresse hat sich geändert."
}

Result POS_WNG {
 pid            = 1
 update         = 1
 message        = "Es wurde ein passender, als 'unsicher' markierter Fall"
          "gefunden. Eine Fehlzuordnung ist nicht mit absoluter"
            "Sicherheit auszuschließen."
        "Sie können den PID mit Kopieren/Einfügen in"
            "andere Dokumente einfügen."
```

```
}

Result POS_NUP {
 pid           = 1
 update        = 1
 message       = "Es wurde ein passender Fall gefunden. "
                 "Da die Eingabe als 'unsicher' markiert war, ist eine"
            "Fehlzuordnung ist nicht mit absoluter Sicherheit"
            "auszuschließen."
            "Sie können den PID mit Kopieren/Einfügen in"
            "andere Dokumente einfügen."
}

Result POS_WCP {
 pid           = 1
 update        = 1
 message       = "Es wurde ein passender Fall gefunden. "
                 "Da die Eingabe als 'unsicher' markiert war, ist eine"
            "Fehlzuordnung ist nicht mit absoluter Sicherheit"
            "auszuschließen."
            "Sie können den PID mit Kopieren/Einfügen in"
            "andere Dokumente einfügen."
}

Result POS_TNT {
 pid           = 1
 update        = 0
 message       = "Es wurde ein sehr ähnlicher Fall gefunden. "
                 "Dessen PID wird mit Vorbehalt ausgegeben."
            "Sie können den PID mit Kopieren/Einfügen in"
            "andere Dokumente einfügen."
            "Bitte kennzeichnen Sie bis auf weiteres den PID bei"
            "jeder Verwendung mit einem Fragezeichen."
}

Result NEG {
 pid           = *
 update        = 0
 message       = "Sie können den PID mit Kopieren/Einfügen in"
                   "andere Dokumente einfügen."
}

Result NEG_TNT {
 pid           = 0
 update        = 0
 message       = "Es wurden ein oder mehrere ähnliche Fälle gefunden."
                   "Die Wahrscheinlichkeit reicht für eine hinreichend "
        "sichere Zuordnung nicht aus."
}

Result AMB_SUR {
 pid            = 0
 update         = 0
 message        = "Es wurden mehrere Fälle mit identischen Daten gefunden."
                  "Eine PID-Vergabe ist daher nicht möglich."
}

Result AMB_UNS {
 pid            = 0
 update         = 0
 message        = "Es wurden mehrere Fälle mit identischen Daten gefunden."
                  "Eine PID-Vergabe ist daher nicht möglich."
}

Result AMB_SIM {
 pid            = 0
 update         = 0
 message        = "Es wurden ein oder mehrere ähnliche Fälle gefunden."
                  "Die Wahrscheinlichkeit reicht für eine hinreichend "
          sichere Zuordnung nicht aus."
}

//////////////////////////////////////////////////////////////////////////////
// Procedure Specification                                                  //
//////////////////////////////////////////////////////////////////////////////

Test T_Start {
 Prologue
 {
  input sure ?
 }

 Epilogue
 {
  0: T_S0_001
  1: T_S1_001
 }
}

Test T_S0_001 {
 Prologue
 {
  key      = 0
  sure     = 1
  exact    = 1
  optional = 1
 }
 Epilogue
 {
  0: T_S0_002
  1: POS_NUP
  *: NIR
 }
}

Test T_S0_002 {
 Prologue
 {
  key      = 0
  sure     = 1
  exact    = 1
  optional = 0
 }
 Epilogue
 {
  0: T_S0_003
  1: POS_NUP
  *: AMB_SUR
 }
}

Test T_S0_003 {
 Prologue
 {
  key      = 0
  sure     = 0
  exact    = 1
  optional = 1
 }
 Epilogue
 {
  0: T_S0_004
  1: POS_WCP
  *: NIR
 }
}

Test T_S0_004 {
 Prologue
 {
  key      = 0
  sure     = 0
  exact    = 1
  optional = 0
 }
 Epilogue
 {
  0: T_S0_005
  1: POS_WCP
  *: AMB_UNS
 }
}
```

```
Test T_S0_005 {
 Prologue
 {
  key      = 0
  sure     = *
  exact    = 0
  optional = 1
 }
 Epilogue
 {
  0: T_S0_006
  1: POS_TNT
  *: AMB_SIM
 }
}

Test T_S0_006 {
 Prologue
 {
  key      = 0
  sure     = *
  exact    = 0
  optional = 0
 }
 Epilogue
 {
  0: NEG
  1: NEG_TNT
  *: AMB_SIM
 }
}

Test T_S1_001 {
 Prologue
 {
  key      = 0
  sure     = 1
  exact    = 1
  optional = 1
 }
 Epilogue
 {
  0: T_S1_002
  1: POS_DIR
  *: NIR
 }
}

Test T_S1_002 {
 Prologue
 {
   key      = 0
   sure     = 1
   exact    = 1
   optional = 0
  }
  Epilogue
  {
   0: T_S1_003
   1: POS_DIR
   *: AMB_SUR
  }
 }

 Test T_S1_003 {
  Prologue
  {
   key      = 0
   sure     = 0
   exact    = 1
   optional = 1
  }
  Epilogue
  {
   0: T_S1_004
   1: POS_WNG
   *: NIR
  }
 }

 Test T_S1_004 {
  Prologue
  {
   key      = 0
   sure     = 0
   exact    = 1
   optional = 0
  }
  Epilogue
  {
   0: T_S1_005
   1: POS_WNG
   *: AMB_UNS
  }
 }

 Test T_S1_005 {
  Prologue
  {
   key      = 0
   sure     = 0
   exact    = 0
```

```
    optional = 1
  }
  Epilogue
  {
    0: T_S1_006
    1: POS_TNT
    *: AMB_SIM
  }
}

Test T_S1_006 {
  Prologue
  {
    key      = 0
    sure     = 0
    exact    = 0
    optional = 0
  }
  Epilogue
  {
    0: NEG
    1: NEG_TNT
    *: AMB_SIM
  }
}
```

5 Operation

The PID Generator usually is invoked via the commandline. In the case of a PID request however, a web interface can be used to run the `gui` (graphical user interface) command. This requires some specifications in a cgi-script described in section 2.5 *Web Interface*.

The overall commandline syntax requires the name of the executable, followed by a command identifier, followed by a set of parameters and options. Options occur in two flavors. Some options are boolean and do not require any further input. Other options require a parameter, which has to be appended to the option symbol, separated by a colon (`:`). Options may be arranged in arbitrary order. Note that the command `psx h` provides a summary of the most frequently used commands and options. For all other commands the configuration file is required. If it is not located in the current directory its path should be provided by the option `-c:`.

Syntax Commandline Interface

`psx` *command* { *parameter* }* { *option* }*

Input Records The commands `enc` and `req` require some input records either provided via an input data file or in some cases via the commandline dialogue. If a data file is used it may either be formatted according to the ISO 8857-1 (Latin 1) or the DIN 66003 standard - the latter is used for German health insurance cards, for example. This file may either contain a single record (one item per line) or multiple records (one record per line). In the latter case the data items can be separated by a delimiter, otherwise the data format must conform to the input format specification (see section 4.1 *Data format*).

5.1 Database Generation

Note: For database systems other than PostgreSQL you have to create the database before running the database generation command.

The database generation command is used to create the PID database (as of now for PostgreSQL only) and the necessary database tables. First, the schema specification (see section 4 *Specification*) provided by the option `-s` is parsed and evaluated. If any errors occur execution terminates. Otherwise, a new empty database is created, using the name and access options specified in the configuration file. If the database already exists execution aborts unless the `-d` option is specified, which forces the deletion of the existing database if necessary. After successful database creation, the specified schema is registered. This means that a set of tables is created which will correspond to the schema specification. After initialization the database will contain the complete schema specification as well as an empty record table, the fields of which are based on the field specification and the transformation options. See section 2.3 *Database System* for details on the database structure.

Syntax Database Generation

`psx gen -c:  -s:  -d`

`-c:` *file*	configuration file
`-s:` *file*	schema specification
`-d`	delete existing database if necessary

5.2 Encipherment

The encipherment operation parses records according to the input format specification and transforms each item as defined in the corresponding transformation code. This includes decomposition of names, phonetic code generation, as well as the encryption using one of several encryption or hash algorithms, including MD5, AES, and IDEA.

The encipherment operation may be performed in two different modes. The single processing mode will handle one record at a time while the batch processing mode will handle large files containing many records. Batch mode is activated with the `-b` option. The encipherment operation requires an input format specification to be provided by the `-f` option. Both, single and batch mode, may be configured to use raw or hexadecimal encoding (see section 3.6 *Encoding*), using the `-e` option. Input records are assumed to be formatted according to ISO 8857-1 (Latin 1) but may also conform to the DIN 66003 standard if specified by the option `-a`.

In single mode the input record may be specified as a file containing one item per line using the `-i` option. If this option is present at the command line, the file will be read in and its lines will be processed as attributes of the input record. Otherwise, the interactive shell dialog is started which will ask for the attributes specified in the format specification.

In batch mode the input records must be specified in a file containing one record per line. It's path is passed using the `-i` option. The records may be presented as a delimiter separated file, whereas the single-character delimiter can be (almost) freely chosen by the option `-d`. (Note that this character must not be contained within the actual data items.) If no delimiter is set, the items of the records are identified by column boundaries as specified in the format specification stored within the PID database (see section 4.1 *Data Format*).
An output data file may be specified using the `-o` option. If the `-u` option is provided, a new specification file, conforming to the generated output records, will be generated. The option `-p` causes the encipher operation to skip the first *num* lines and the `h` option causes a heading line to be inserted into to the output data file to enhance readability.

Syntax Encipherment

`psx enc -c: -f: -i: -a: -u: -o: -d: -p: -e: -b -h`

`-c:` *file*	configuration file
`-f:` *file*	input format specification
`-i:` *file*	input data file
`-a:[din | iso]`	input character set
`-u:` *file*	output format specification
`-o:` *file*	output data file
`-d:` *character*	record item delimiter
`-p:` *num*	start position
`-e:[raw | hex]`	encoding
`-b`	batch mode
`-h`	headings

5.3 PID Request

A PID request command requires an existing PID database which has to be registered with a valid schema specification.

The command `req` processes a set of input records and tries to match them against the records in the PID database. Each record item is normalized, checked for plausibility, and transformed according to the corresponding data type and transformation specification. Normalization steps include the removal of non-alphanumeric characters (except apostrophe), the dissolving of umlauts, and the conversion of accented characters into the corresponding characters without accent.

The resulting record is then passed to the matching interpreter which will execute the matching procedure as defined in the schema resulting in one of three possible outcomes: If matched, an existing PID will be assigned. If not matched, a new PID will be generated. In any other case, ambiguous results, for example, no PID is assigned.

The request command may be executed in two different modes. In single mode one record is processed at a time. This input record may be specified as a file containing one data item per line using the `-i` option. Otherwise, the interactive shell dialog is started which will ask for the attributes specified in the format specification. Note that a special CGI interface for a single PID request is provided by the `gui` command (see section 5.4 *Web Interface*).

Batch mode is activated with the `-b` option. The input data file - specified with the `-i` option - contains one record per line. The records may be presented as a delimiter separated file, whereas the single-character delimiter can be (almost) freely chosen by the option `-d`. (Of course, this character must not be contained within the actual data items.) If no delimiter is set, the items of the records are identified by column boundaries as specified in the format specification stored within the PID database (table `fld`) (see section 4.1 *Data Format*). This means that the input file must be formatted in such a way that each item starts at a certain position and does not exceed a fixed field length. If these specifications stored in the database are not sufficient, it is also possible to use different settings. A shortened specification file, containing only a data section, can be defined differently from the database entries and passed to the program by the `-f` option. In batch mode, it is also possible to overwrite the default sureness value (see below), since not all records can be assumed to be associated with the same sureness. The sureness of each input record can simply be set by appending a '`+`' or a '`-`' to the end of each line (separated by a delimiter, if appropriate). One may use the `-t` option to specify an output trace file into which the results are written. The software can be directed to begin at a certain record position using the `-p` option causing the tool to skip the first p-1 lines.

The request operation understands some general arguments which are independent of the selected processing mode. The sureness of the input record(s) may be specified with the `-s` option. A '`+`' indicates reliable data (e. g. originating from health insurance cards) whereas a '`-`' should be specified when less trustworthy data sources are used. The generation of a new PID may be forced using the `-F` option. Input records are assumed to be formatted according to ISO 8857-1 (Latin 1) but may also conform to the DIN 66003 standard if specified by the option `-a`.

Syntax PID Request

```
psx req -c:  -f:  -i:  -a:  -t:  -s:  -i:  -p:  -b -F
```

`-c:` *file*	configuration file
`-f:` *file*	input format specification
`-i:` *file*	input data file
`-a:[din	iso]`	input character set
`-t:` *file*	output trace file
`-s:` [+ | −]	sureness specification
`-d:` *character*	record item delimiter
`-b`	batch mode
`-F`	force PID generation
`-p:` *num*	start position

The result of batch mode processing is written into the output trace file. Each line consists of a consecutive number specifying the position of the record in the corresponding input file. It is followed by a colon, the result specifier, another colon, and finally the PID itself. The result specifier is either 'PID' - indicating that a PID is returned and ready to be used - or '???'. The three question marks indicate either a very weak or an ambiguous matching result. In the former case, a PID is returned but one should rather try to retrieve more reliable data to assert the result. In the latter case, no definite PID could be retrieved. An error is indicated by a question mark returned instead of a PID. In this case, you should consult the logfile. An example of a batch run, which consists of the batch command and excerpts of the input and output file, is given below:

Batch Command Example psx.spc.short contains only the field specification

```
psx req -b -d:, -c:psx.cfg -f:psx.spc.short -i:input.txt
-t:output.txt -s:-
```

Excerpt of the input.txt

```
Bork,,Andre,18,07,1980,55124,Mainz,,m
Mustermann,,Gabriele,29,4,1962,54391,Neustadt,,f
Müller,Maier,Heinz,20,11,1950,34253,Altenburg,,
```

Excerpt of the output.txt The last column is not part of the output.txt

```
0001: PID:  7QHG8VGA     PID ready to use
0002: PID:  DKJVXP9B     PID ready to use
0003: ???:               ambigous match result
0004: ???:  AK8ZWHF4     PID should not be used; weak similarity
0005: ???:  ?            an error occured
```

5.4 Web Interface

The command `gui` provides an access point for a cgi-script thus enabling the invocation of a PID request via a web server. See details about the cgi script in section 2.5 *Web Interface*.

Depending on the context the PID Generator generates an HTML document presenting the user either a data entry form or a result page containing the match result or a context specific message. The automatically generated HTML output can be replaced by customized HTML documents as described in section 3.9 *Templates*.

Syntax Graphical User Web Interface

`psx gui -c: -s -d`

`-c:` *file*	configuration file
`-s`	enable system panel
`-d`	enable debugging

5.5 PID Validation

A PID consists of eight letters and digits, the last two characters being check characters. The PID Generator offers a validation command to run plausibility checks on PIDs whose correctness is not assured. It is capable of recognizing errors affecting two positions within a PID and of correcting errors that affect only one position or two neighbouring digits if these have interchanged. Note that there is no check to ensure that the given PID has been assigned at all.

The command `chk` can be used either in single mode or in batch mode by specifying the `-b` option. Single mode requires the specification of the PID using the option `-P` while in batch mode an input data file must be presented using `-i`. This input file contains one PID per line.

Syntax PID Validation

`psx chk -c: -d: -o: -t: -p: -b -P:`

`-c:` *file*	configuration file
`-P:` *PID*	PID
`-i:` *file*	input data file
`-o:` *file*	output data file
`-b`	batch mode
`-p:` *num*	start position

The following table shows examples for possible outcomes of a PID validation. The line numbers only appear in batch mode though.

`0001: VAL:  7QHG8VGA`	PID is valid
`0002: INV:`	PID is invalid, no correction possible
`0003: COR:  7QHG8VGA`	PID is invalid, corrected as listed

5.6 Configuration Information

Configuration information encloses tasks that provide some insight into the settings of the local implementation. This especially includes details on the internal schema specification as stored in the PID database.

A summary of the data format, result and state specification can be obtained using the `-s` option. To get the details about the matching procedure of a single state, the option `-m` must be provided followed by the `-q` option and the KSXO notation of the respective state. (Example: `-q:K0:S1:X1:O0`.)

Syntax Configuration Information

`psx cfg -c:  -s -m -q:`

`-c:` *file*	configuration file
`-s`	dump schema
`-m`	dump matching
`-q:` *string*	KSXO filter

5.7 Version Information

Version information should report the global version numbers of the compiled binary, a unique time stamp that reflects the time of the last build, as well as a build number, which is the absolute number of times when the binary was built, since the first time.

Syntax Version Information

`psx ver -c:`

`-c:` *file*	configuration file

5.8 Status Summary

The summary command will provide general status information on the contents of the PID database. For now, only the current number of records is displayed.

Syntax Status Summary

```
psx sts -c:
 -c: file   configuration file
```

A Reference

A.1 Matching Algorithm and Processing of Names

The matching of input data records against the PID database is executed according to the definitions in the specification file. A good starting point for creating these complex procedure definitions is a state-chart diagram reflecting the order in which the query states will be processed.

As mentioned in section 4.3 *Matching procedure*, each query state represents a filter, i. e. a set of conditions, applied to the set of records stored in the PID database. One wants to find a filter which delivers exactly one record because only in this case a match is produced. Therefore, the matching procedure should begin with more strict conditions and only continue with less strict filters if the former do not deliver a result.

The definition of the filter is described in section 4.3. In summary, the search for matching records can be restricted to data records that are marked as 'sure', or expanded to those marked as 'unsure', optional data fields can be taken into consideration or alternatively be ignored, and key fields can be required to be identical or not. A match is assigned, if the values of the data fields of interest are equal, with an exception concerning fields of data type `N` (first and last names). For this category, a special matching strategy is supplied. It is more complex due to the fact that names can consist of several components. The filter can furthermore be expanded so that data records can be matched where the names do not exactly match but show a phonetic similarity.

A special matching strategy is applied for data items representing names. Since these items are prone to spelling mistakes, a restriction of the filter to exact matches would most likely lead to the generation of synonyms within the PID database. To attend this characteristic, it is possible to define a filter which can not only detect exact agreement in these fields, but also find data records which show some similarity. Data items are considered to be similar if their phonetic codes are the same. Note that similarity should only be considered if at least one of the data records is marked as ‘unsure’. If both records are marked as ‘sure’, one could expect exact agreement in the names if both records belonged to the same individual.

Data items of type `N` can furthermore be decomposed into maximal three components whereas spaces and characters like ‘-’, ‘/’, etc. define separators for the respective components. If more than three components are found, the surplus items will be added to the third component which is not considered in the matching strategy described below. For last names, name pre- and suffixes such as ‘von’, ‘de’, ‘zu’, etc. will also be stored in the third component. *Note that the decomposition of names is not taken into account for generating the phonetic codes, which are always generated for the complete input value.*

For equality checks only the first two components are considered. A comparison of names yields the status ‘equal’ in following cases:

Name 1 (Input)		Name 2 (In Database)
(Smith, Jones)	and	(Smith, Jones)
(Smith, NULL)	and	(Smith, NULL)
(Smith, NULL)	and	(Smith, Jones)
(Smith, Jones)	and	(Smith, NULL)
(Smith, Jones)	and	(Jones, Smith)
(Smith, Jones)	and	(Jones, NULL)
(Smith, NULL)	and	(Smith, NULL)
(Smith, NULL)	and	(Smith, Jones)
(Smith, NULL)	and	(Jones, Smith)

A.2 Commands

`gen`	generate database	
	`-c:`	configuration file
	`-s:`	schema specification
	`-d`	delete existing database
`enc`	encipher	
	`-c:`	configuration file
	`-f:`	input format specification
	`-i:`	input data file
	`-a:`	input character set
	`-o:`	output data file
	`-u:`	output format specification
	`-d:`	record item delimiter
	`-p:`	start position
	`-e:`	encoding specification
	`-b`	batch mode processing
	`-h`	print headings
`req`	request	
	`-c:`	configuration file
	`-f:`	input format specification
	`-i:`	input data file
	`-a:`	input character set
	`-t:`	output trace file
	`-s:`	sureness
	`-d:`	record item delimiter
	`-b`	batch mode processing
	`-F`	force PID generation
	`-p:`	start position
`gui`	graphical user web interface	
	`-c:`	configuration file
	`-s`	enable system panel
`chk`	check	
	`-c:`	configuration file
	`-i:`	input data file
	`-o:`	output data file
	`-P:`	PID
	`-b`	batch mode processing
	`-p:`	start position
`cfg`	configuration	
	`-c:`	configuration file
	`-s`	dump schema
	`-m`	dump matching
	`-q:`	filter (KSXO)
`ver`	version information	
	`-c:`	configuration file
`sts`	print status summary	
	`-c:`	configuration file
`hlp`	show help	

A.3 FAQ

Disposer of on instance of the PID generator should establish a technical service for user problems. For questions concerning general difficulties of the PID generator when operating with encrypted data this FAQ has introductory value and contains suggestions.

1. Does the PID have a specific format?

   - The PID consists of eight alpha-numerical literals. The characters B I O S are not part of this PID because of the similarity with the digits 8 1 0 2. The PID could consist of only characters, e.g. RWETGSDE, or only digits, e.g. 54358790.

2. Do I have to type only capitals for checking or evaluating the PID?

   - The PID generator is case insensitive, so vegunr9j is equal to VEGUNR9J.

3. What do I have to do if I realize that a PID is generated with incorrect entries, like false forename?

   - Contact the administrator of the PID generator who will give order to type the correct data. The old PID will be marked as invalid (filling the `sub`-field of the old record with the new PID in table `rec`) and you can proceed with the new PID.

4. What can I do if I discover a real homonym?

   - Contact the administrator of the PID generator who enters the non-entered case with a modification in one free field. This modification should be used as additional information to distinguish between the two cases.

5. What happens if two PID's for one case exists?

   - Contact the administrator of the PID generator. You both have to try to discover the PID which corresponds to correct entries. Then the PID corresponding to false entries will be marked as invalid. Known facilities using the invalid PID will be informed about the correct PID.

6. What can I do if I can't obtain a PID?

- Contact the administrator of the PID generator. This is mostly a case of phonetic equity (context: `unsure`), which is considered as too weak. You both have to search for the correct data or further information.

7. What happens if the forename of one patient is entered once as 'Max' and another time as 'Jan-Max'?

- You will get a match because of the provided cross comparison of name fields.

8. What happens when I enter once 'Schmidt' and another time 'Schmitt'?

- It will be recognized as phonetically equal. If both cases are marked as sure two PID's will be generated.

Bibliography

[1] PostgreSQL, Homepage of the database management system. http://www.postgresql.org.

[2] I. Schmidtmann, H.-J. Appelrath, J. Michaelis, and W. Thoben. Empfehlungen an die Bundesländer zur technischen Unsetzung der Verfahrensweisen gemäß Gesetz über Krebsregister (KRG). *Informatik, Biometrie und Epidemiologie in Medizin und Biologie*, 27:101–110, 1996.

[3] Gary Vaughn, Ben Ellison, Tom Tromey, and Ian Taylor. *GNU Autoconf, Automake, and Libtool.* New Riders, October 2000. 'The Goat Book', ISBN: 1578701902, http://sources.redhat.com/autobook.