PID Generator

Manual

Markus Wagner, Jutta Moormann

Institute for Medical Biometry, Epidemiology and Informatics

University of Mainz

moormann@imbei.uni-mainz.de

January 7, 2005

1 Introduction

The *PID Generator* is an application designed for the transformation, encipherment, and matching of personal data as well as for the generation and assignment of Personal Identifiers (PIDs) to individuals that belongs to a bounded population. It is especially capable to deal 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 transformations performed in batch mode. These include decomposition of names, generation of phonetics, the encipherment with IDEA or AES, and the generation of control numbers using the MD5 algorithm. The core facility however is the management of a PID database and the capability to process PID requests.

The PID Generator is highly customizable to meet the requirements of different environments. This is due to extensive configuration options in the configuration and specification files. The latter includes the format specification which defines the structure of the records that are processed, the result specification which defines the set of possible match results and arbitrary messages to be presented to the end user, and the matching specification describing the overall algorithmic behavior 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 NT 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 of PostgreSQL and the OpenLDAP 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 one should change into 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 in order to generate the final Makefiles that will be used for compilation and linkage. One can 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.

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 [**?**].

The software uses some third-party algorithms, which were not modified and left in their original version. However, there are problems with the code for the generation of phonetics that may cause a segmentation fault on some systems or with some compilers. The problem is caused by a write access to strings that are defined as constant. The simple solution for the GNU compiler uses the option `-fwritable-strings`. The configuration files contain all of the necessary instructions. If you want to build a new project from scratch by simply adding the source files, you are on your own.

```
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
```

2.2 Installation on Windows Systems

to be written

2.3 Database System

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

In general, the database system should 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. Refer to the PostgreSQL system manuals, if you have any problems [**?**].

To create and set up a PID database with PostgreSQL one should use the command `psx gen` (see section 5.1 *Database Generation*). For database systems other than PostgreSQL one has to create the database before running the `psx gen` command.

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 should not be modified after their creation. These tables are `sch`, `fld`, `ctp`, `rsp`, and `sts`. The tables `fld`, `rsp`, `sts` mirror the data field, result and status specifications as defined in the specification file (see section 4 *Specification*) so be sure to carefully specify the data format and matching strategy before setting up these tables.

In contrast to those fixed tables the table `req` will be updated at each PID request thus mirroring the request history if configured to do so (see section 3.3 *Request History*). The table `rec` holds the (transformed) data records and will be updated each time a new record is created or a data record is updated by newer information. The table `cfg` contains the current number used to create the PID and the timestamp of the last PID request.

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
f_{field_name}                  // fields according to table fld
...
```

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	...
1	1	fname	N	Vorname	30	15	0	0	...
1	1	bd	DD	Geburtstag	62	02	0	0	...
1	1	bm	DM	Geburtsmonat	65	02	0	0	...
1	1	by	DY	Geburtsjahr	68	04	0	0	...
1	1	plz	T	PLZ	73	08	0	0	...
1	1	loc	T	Ort	82	09	0	0	...
1	1	state	T	Staat	92	10	0	0	...
1	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	+
...	*		+
...	-		+
...	-		+
...	-		-
...	-		*
...	-		*
...	-		*

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.
...	
...	
...	
...	
...	
...	
...	
...	
...	

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:O1	S:T_S0_002	R:POS_NUP	R:NIR
3	1	T_S0_002	K0:S1:X1:O0	S:T_S0_003	R:POS_NUP	R:AMB_SUR
4	1	T_S0_003	K0:S0:X1:O1	S:T_S0_004	R:POS_WCP	R:NIR
5	1	T_S0_004	K0:S0:X1:O0	S:T_S0_005	R:POS_WCP	R:AMB_UNS
6	1	T_S0_005	K0:S*:X0:O1	S:T_S0_006	R:POS_TNT	R:AMB_SIM
7	1	T_S0_006	K0:S*:X0:O0	R:NEG	R:NEG_TNT	R:AMB_SIM
8	1	T_S1_001	K0:S1:X1:O1	S:T_S1_002	R:POS_DIR	R:NIR
9	1	T_S1_002	K0:S1:X1:O0	S:T_S1_003	R:POS_DIR	R:AMB_SUR
10	1	T_S1_003	K0:S0:X1:O1	S:T_S1_004	R:POS_WNG	R:NIR
11	1	T_S1_004	K0:S0:X1:O0	S:T_S1_005	R:POS_WNG	R:AMB_UNS
12	1	T_S1_005	K0:S0:X0:O1	S:T_S1_006	R:POS_TNT	R:AMB_SIM
13	1	T_S1_006	K0:S0:X0:O0	R:NEG	R:NEG_TNT	R:AMB_SIM

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 | 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 |
---|------|-------|-------|---------|-------|
```

2.5 Graphical User 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 causes the appropriate behavior. 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 3.2 *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 3.8 *Templates*).

2.6 XML Web Interface

to be written

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 would specify 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:root:lgsym
```

This expression would be suitable for another database system using ODBC. The datasource name is specified as 'psx' which will be accessed by the user 'root' using the password 'lgsym'.

3.2 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.

3.3 Request History

Each request for a PID will be logged in the database table `req` which is thus mirroring 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). 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	::=	`Request History = [0|1]`	

3.4 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*	

3.5 Encoding

The encoding of data items may be configured to make use of different character sets. Control numbers are generated as MD5 hash codes, which are traditionally represented as printable ASCII characters [**?**]. However, using this character set, which contains special characters such as apostrophes or backslash, may cause additional conversion overhead with most database management 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 for virtually any task the software provides. If not given at the commandline, the encoding will be retrieved from the configuration. 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 ]`	

3.6 Directory Access

Directory service access provides for restrictions on the users that may submit requests to the PID service. If enabled, a 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 customized 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*	

3.7 PID Construction

PID construction is performed in two steps. First, an internal number is incremented, representing a logical counter that corresponds to the number of records stored in the PID database. Secondly, this number 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 is dependent on some parameters that are retrieved from the configuration file. These include three *encryption keys* (k_1, k_2, k_3), as well as the *random width*, which determines the number of bits used for randomization. All four parameters are of type integer.

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*	

3.8 Templates

Templates are user-defined HTML documents serving as a flexible framework for the graphical user interface. There is a predefined set of document types, each of which is used in a predefined set of unique situations to perform some interaction with the user. Whenever a CGI request comes in, some document type is selected depending on the outcome of the request, and the corresponding template file is processed. The template may contain placeholders for many properties of the actual situation, which are replaced by their actual values, resulting in a context specific document, which is sent as a response to the user.

There are three different situations considered by the software, namely the request (`req`), the response (`ret`) and the message (`msg`). These correspond to three different templates that may be activated within the configuration with the attributes `tpl.req`, `tpl.ret` and `tpl.msg`, for the presentation of requests, returns and arbitrary messages, respectively. Each of these attributes should be assigned an absolute path to a template file that should be readable by the system.

The overall process looks as follows. When the user accesses the service, he usually gets a request form, where he enters the patient data and submits the request. If the request is processed successfully, he is returned a result page containing the PID along with other information. If the request is not processed successfully, he gets a message page that informs 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 as desired 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 pair of option buttons, which should be specified with a value of 'M' or 'W', which indicates male or female sex, respectively. The automatically generated user interface will present this field as a text field, for now. In general, if one wishes to limit user input to a predefined set of values, the templates will provide for the appropriate means.

Template configuration is designed to be a feature, rather than an enforcement. 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, because it was generated automatically, 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 existence for the different template types.

`ptl`	portal
`sys`	system administration panel
`req`	PID request panel
`ret`	PID return panel
`msg`	message panel

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		+	

1	type	data type (see data types)
2	symbol	unique regular identifier
3	label	quoted field description
4	start	start position within a row of a data file
5	length	fixed length
6	min	minimum size for field data, 0: ignore
7	max	maximum size for field data, 0: ignore
8	transformation	transformation code
8	match	match code

3.9 Messages

The message items may be used to specify messages to present to the user in case of erroneous data input. Four different types can be distinguished: Missing values, to short or to long input, or otherwise invalid values.

Syntax Message Specification

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

Note that the message strings may contain the placeholders `$FLD` and `$LEN` that will be replaced by the corresponding values if an incorrect item is found.

3.10 Encryption

The encryption specification contains the password used for AES or IDEA encryption.

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

3.11 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]`

3.12 Example: The GPOH configuration

GPOH Configuration

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

// database connection
dbs         = pgsql:localhost::pdb_test:wagner:

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

// directory access
dir.lookup  = 1
dir.server  = 'gpoh.imsd.uni-mainz.de'
dir.base    = 'o=University of Mainz,c=DE'

// encoding
ctn.enc     = hex

// mailing
mail.src.name = 'PSX'
mail.src.adr  = 'magnus@gmx.de'
mail.dst.name = 'Admin'
mail.dst.adr  = 'magnus@gmx.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, as well as the *matching procedure* section that implements the core of the matching strategy. For this purpose a formal language (KSXO) was designed as described at page 24. The specification file is read once when setting up the PID database. The data format section can furthermore be used to process input data that differs from the initial format specification settings.

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 an item of the record. The sequence implicitly defines the order of the items when processed by the different commands. This affects the order of plausibility checks, the automatic generation of data entry forms, and the presentation of output records and output format specifications. However, it does not affect the outcome of the matching procedure at all.

4.1.1 Basic Definitions

The specification relies on some basic definitions, namely data types, equality and transformation codes. Data types are used to classify the items with respect to plausibility checks and range limits. Equality codes define when two data items are considered to be equal and how the fields of two records may be exchanged without affecting equality. Transformation codes define how data items are to be modified upon acquisition before any further processing is done.

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, 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	*string*
N	last name (surname)	*string*
F	first name (given name)	*string*
DY	year of date	*4-digit-number*
DM	month of date	[0-12]
DD	day of date	[0-12]
SEX	human sex	[f | m]
HIO	health insurance organization	*string*
HIC	health insurance code	*string*

Transformation

A set of transformation functions can flexibly be incorporated to modify the original data items of an input record. These modifications include the decomposition of names into components, the generation of phonetic codes using different algorithms, the generation of control numbers using MD5 hash as well as the final encryption with AES or IDEA. The latter is only applicable though if control numbers are generated. 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

Example The transformation code *:D[N]:P[C]:E[AES] 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

Whenever two records are compared, the *equality specifications* of the fields are used to tell wether the field contents match or not. The specification includes a simple mechanism to describe exchangeability. For example, the fields name (surname) and aname (alternative, former surname) should be defined as exchangeable, which allows a married person to be still matched correctly.

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 respective 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

4.1.2 Field Definitions

A field definition is introduced by the keyword `Field`, followed by a unique identifier and the assignment block in curly brackets. The assignments include any further information on the properties a field may have. Every assignment is of the form *attribute* = *value*, separated by white space. A data type defines the valid contents, a field may have. The arbitrary label is used for external representation to the user. A start position and a length define the location of a data item within a row-oriented ASCII file. A minimum and a maximum length further define limitations on the set of valid contents. Finally, the transformation code and the equality code may prepare a field for consideration within the matching process. This is defined as follows.

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 field identifier must obey the typical conventions of popular programming languages, i. e., *letter*([*letter*|*digit*|'_'])*. 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. The following table shows the set of field attributes recognized by the schema parser.

`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

Example Figure 4.1 shows an example of a field specification. It defines a field named `lname` for the representation of person last names. The type is set to `N` (name) and the label is set to a natural language description. The start position for ASCII file representation is set to 14 and the length of the field is set to 15. 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 or may be exchanged with any other fields, the exchange id of which are equal to 1.

```
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

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 PID retrieval mode (prm) defines, if a given PID is returned, a new PID is generated or nothing is returned at all. The record update mode (rum) defines, wether the contents of a successfully identified record are updated or completed with the contents of the input record. The message returned to the user is also defined in this context.

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. This is defined as follows.

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, i. e. a query such as `KOS1X1OO` yields exactly one result record (see section 4.3 *KSXO notation*). 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 *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 depending on the respective previous test result. The proceeding is represented as a finite state machine which will be considered by the matching interpreter whenever a request is processed. Upon activation, some starting state is declared to be the active state, and repetitive state processing begins.

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 proceeding from a state follows one of its defined exits which are links to either other states or results. The latter will define the end of the sequence of tests causing the result identity to be returned to the superordinated procedure.

The input state normally occurs exactly once within a matching schema and refers to the sureness of the input record, thus delivering a binary result. 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 arbitrary 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 evaluation delivering no record (0), exactly one record (1) or more than one record (∗).

The arrangement of analysis steps and their linkage with the final results define the core of individual PID generation strategies and should be developed independently for each pseudonymization project. Therefore, it has been isolated from the source code. The dedicated specification language supported by the software should provide enough scope for an in-depth local adaptation, without modifying any line of code.

KSXO notation

The KSXO-notation serves as a simplified query definition language. The KSXO-expression of each state will be transformed into the correspoding complex SQL-query, thus specifying a subset of the records in the PID database. It consists of four parameters which affect the relevance of key fields (K), the sureness of records(S), the exactness of field comparison(X), as well as the consideration of optional values(O).

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) (since it makes no sense to select non-equal keys). 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) by the phonetic algorithm selected in the configuration. 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. Figure 4.3 illustrates the parameters of the KSXO-notation.

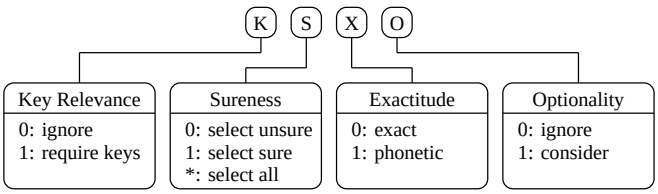

Figure 4.3: KSXO notation

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. In general, one wants to find a filter, which delivers exactly one record because only in this case a match is produced.

Syntax

The specification of a state is divided into a prologue section that defines the database query, and an epilogue section that connects the exits of the actual state to other objects within the procedure specification. Besides that, a state definition always includes a unique symbolic identifier introduced by the keyword `Test`, which allows other states to reference its identity through some of their defined exits.

Thus, a state definition consists of an identifier, a prologue block and an epilogue block.

For an input state the prologue section simply consists of the literal `input sure ?`. The prologue section of a query state however becomes decomposed into the fields `key`, `sure`, `exact` and `optional`, which correspond to the four parameters of the KSXO-notation.

The exit specification defines the connections of the actual state to other objects of the current schema. The syntax requires the exit identifier, followed by a colon, followed by the identifier of another state (to proceed to another query) or a result (thus terminating the test sequence). For an input query state, the exits 0 and 1 are defined, corresponding to a sure or unsure input record. For a data query state, the exits 0, 1 and $*$ are defined, corresponding to the different outcomes of the query. In any case, the exit definitions will have to reference existing objects.

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*]

The procedure specification defines a network of states, results and transitions. The most important integrity constraint is that there never are references to undefined objects.

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-query would be `K0S0X1O1`. 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.

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

```
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

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.

The graphical notation simplifies the view on the principal flow of analysis steps, while neglecting the details of the results and the contents of the actual database fields. This abstraction will help thinking about the matching procedure, when talking about something like a strategy.

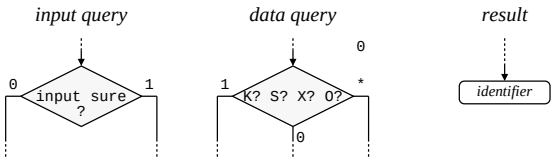

Figure 4.5: Procedure notation

4.4 Example: The GPOH matching diagram

Figure 4.6 shows the GPOH matching diagram.

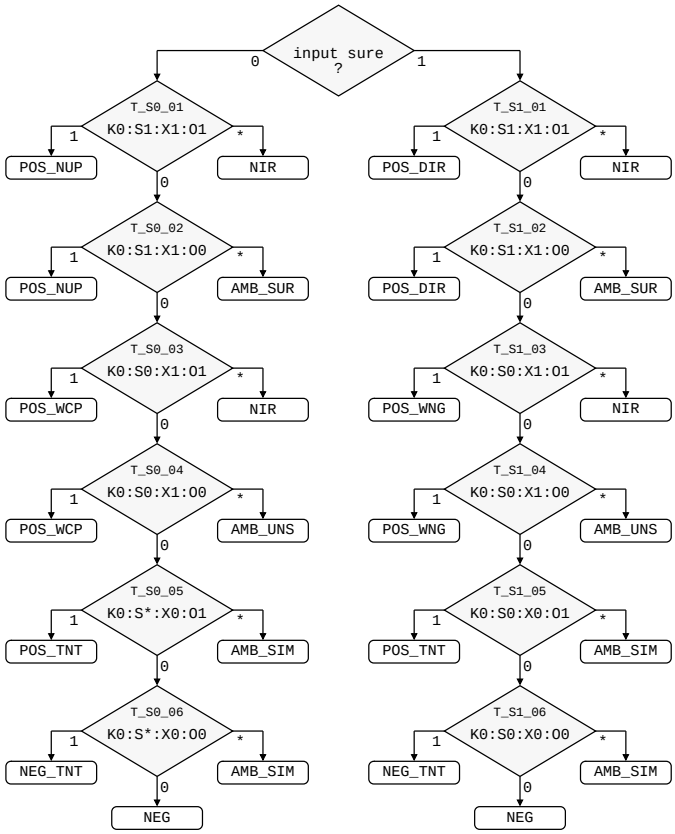

Figure 4.6: The GPOH matching diagram

4.5 Example: The GPOH specification

GPOH Schema Specification

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

%Field kvn
%{
% type           = HIO
% label          = "Krankenversicherung"
% start          = 0
% length         = 0
% min            = 0
% max            = 0
% transformation = "-"
% equ            = "#"
%}

%Field kvc
%{
% type           = HIC
% label          = "Krankenversicherungscode"
% start          = 01
% length         = 12
% min            = 0
% max            = 0
% transformation = "-"
% equ            = "#"
%}

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.<br><br>"
          "Bitte melden Sie dieses Ereignis unmittelbat an den "
          "<a href='mailto:webmaster@gpoh.imsd.uni-mainz.de?"
          "subject=Homonym'> Webmaster</A>!<br><br>Bitte arbeiten Sie"
          "zunächst ohne PID."
}

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

Result POS_NOP {
 pid            = 1
 update         = 1
 message        = "Sie können den PID mit Kopieren/Einfügen (Copy & Paste) in"
                  "andere Dokumente einfügen.<br><br>"
          "<em>Bemerkung</em>: 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.<br><br>"
          "Sie können den PID mit Kopieren/Einfügen (Copy & Paste) 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.<br><br>"
          "Sie können den PID mit Kopieren/Einfügen (Copy & Paste) 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.<br><br>"
          "Sie können den PID mit Kopieren/Einfügen (Copy & Paste) 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.<br>"
          "Sie können den PID mit Kopieren/Einfügen (Copy & Paste) in"
          "andere Dokumente einfügen.<br>"
          "<strong>bitte kennzeichnen Sie bis auf weiteres den PID bei"
          "jeder Verwendung mit einem Fragezeichen.</strong>"
}

Result NEG {
 pid            = *
 update         = 0
 message        = "Sie können den PID mit Kopieren/Einfügen (Copy & Paste) 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.<br><br>"
          "Bitte arbeiten Sie zunächst ohne PID und versuchen Sie,"
          "weitere odere genauere Daten zu erhalten."
}

Result AMB_SUR {
 pid            = 0
 update         = 0
 message        = "Es wurden mehrere Fälle mit identischen Daten gefunden."
                  "Eine PID-Vergabe ist daher nicht möglich.<br><br>"
          "Bitte melden Sie dieses Ereignis unmittelbar an den "
          "<a href='mailto:webmaster@gpoh.imsd.uni-mainz.de?"
          "subject=Homonym'>Webmaster</A>!<br><br>"
          "Bitte arbeiten Sie zunächst ohne PID."
}

Result AMB_UNS {
 pid            = 0
 update         = 0
 message        = "Es wurden mehrere Fälle mit identischen Daten gefunden."
                  "Eine PID-Vergabe ist daher nicht möglich.<br><br>"
          "Bitte melden Sie dieses Ereignis unmittelbar an den "
          "<a href='mailto:webmaster@gpoh.imsd.uni-mainz.de?"
          "subject=Homonym'>Webmaster</A>!<br><br>"
          "Bitte arbeiten Sie zunächst ohne PID."
}

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.<br><br>"
          "Bitte arbeiten Sie zunächst ohne PID und versuchen Sie,"
          "weitere odere genauere Daten zu erhalten."
}

////////////////////////////////////////////////////////////////////////////
// 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 **??** *Graphical User 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 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 ?? *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 3.5 *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 `-d` 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 as a file containing one record per line using the `-d` option. In this case the items of the records are identified by column boundaries as specified in the format specification. 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 `p-1` 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:  -d:  -a:  -u:  -o:  -p:  -e:  -b -h
 -c: file          configuration file
 -f: file          input format specification
 -d: file          input data file
 -a:[din | iso]    input character set
 -u: file          output format specification
 -o: file          output data file
 -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 plausability, 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 either be specified as a file containing one data item per line using the `-d` option or interactively at the command prompt. In the latter case the PID Generator will ask for every data item defined 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 *Graphical User Interface*).

In batch mode, the input data file must contain one record per line according to the format specification. It is specified with the `-d` option. The format specification may be specified using the `-f` option, if the tabular organization of the input data file differs from the format found in the PID database (see section 4.1 *Data Format*). Optionally, one may use the `-t` option to specify an output trace file where the results are written to. On may direct the software 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 records 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: -d: -a: -t: -s: -p: -b -F`

`-c:` *file*	configuration file
`-f:` *file*	input format specification
`-d:` *file*	input data file
`-a:[din | iso]`	input character set
`-t:` *file*	output trace file
`-s:` [+ | −]	sureness specification
`-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 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 '???'. Three question marks indicate either a very weak match or an ambiguous matching result. In the former case a PID is returned but one should try to retrieve more reliable data to assert the result. In the latter case no definite PID could be retrieved. If instead of a PID a question mark is returned an error has occurred. In this case you should consult the logfile.

`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 Graphical User 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 ?? *Graphical User 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 an arbitrary message. The automatically generated HTML output can be replaced by customized HTML documents as described in section 3.8 *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 `-d`. The 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
`-d:` *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 Commands

`gen`	generate database	
	`-c:`	configuration file
	`-s:`	schema specification
	`-d`	delete existing database
`enc`	encipher	
	`-c:`	configuration file
	`-f:`	input format specification
	`-d:`	input data file
	`-a:`	input character set
	`-o:`	output data file
	`-u:`	output format specification
	`-p:`	start position
	`-e:`	encoding specification
	`-b`	batch mode processing
	`-h`	print headings
`req`	request	
	`-c:`	configuration file
	`-f:`	input format specification
	`-d:`	input data file
	`-a:`	input character set
	`-t:`	output trace file
	`-s:`	sureness
	`-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
	`-d:`	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	