NAME

Noid − routines to mint and manage nice opaque identifiers

SYNOPSIS

```
use Noid;                           # import routines into a Perl script

$dbreport = Noid::dbcreate(         # create minter database & printable
                $dbdir, $contact,   # report on its properties; $contact
                $template, $term,   # is string identifying the operator
                $naan, $naa,        # (authentication information); the
                $subnaa );          # report is printable

$noid = Noid::dbopen( $dbname, $flags );    # open a minter, optionally
       $flags = 0 | DB_RDONLY;              # in read only mode

Noid::mint( $noid, $contact, $pepper );     # generate an identifier

Noid::dbclose( $noid );                     # close minter when done

Noid::checkchar( $id );      # if id ends in +, replace with new check
                             # char and return full id, else return id
                             # if current check char valid, else return
                             # 'undef'

Noid::validate( $noid,       # check that ids conform to template ("-"
                $template,   # means use minter's template); returns
                @ids );      # array of corresponding strings, errors
                             # beginning with "iderr:"

$n = Noid::bind( $noid, $contact,      # bind data to identifier; set
                $validate, $how,       # $validate to 0 if id. doesn't
                $id, $elem, $value );  # need to conform to a template

Noid::note( $noid, $contact, $key, $value );   # add an internal note

Noid::fetch( $noid, $verbose,          # fetch bound data; set $verbose
                $id, @elems );         # to 1 to return labels

print Noid::dbinfo( $noid,             # get minter information; level
                $level );              # brief (default), full, or dump
Noid::getnoid( $noid, $varname );      # get arbitrary named internal
                                       # variable

Noid::hold( $noid, $contact,           # place or release hold; return
                $on_off, @ids );       # 1 on success, 0 on error
Noid::hold_set( $noid, $id );
Noid::hold_release( $noid, $id );

Noid::parse_template( $template,  # read template for errors, returning
                $prefix, $mask,   # namespace size (NOLIMIT=unbounded)
                $gen_type,        # or 0 on error; $message, $gen_type,
                $message );       # $prefix, & $mask are output params

Noid::queue( $noid, $contact,     # return strings for queue attempts
                $when, @ids );    # (failures start "error:")

Noid::n2xdig( $num, $mask );      # show identifier matching ord. $num

Noid::sample( $noid, $num );      # show random ident. less than $num

Noid::scope( $noid );             # show range of ids inside the minter

print Noid::errmsg( $noid, $reset );   # print message from failed call
       $reset = undef | 1;        # use 1 to clear error message buffer
```

```
Noid::addmsg( $noid, $message );  # add message to error message buffer
Noid::logmsg( $noid, $message );  # write message to minter log
```

DESCRIPTION

This is very brief documentation for the **Noid** Perl module subroutines. For this early version of the software, it is indispensable to have the documentation for the **noid** utility (the primary user of these routines) at hand. Typically that can be viewed with

```
perldoc noid
```

while the present document can be viewed with

```
perldoc Noid
```

The **noid** utility creates minters (identifier generators) and accepts commands that operate them. Once created, a minter can be used to produce persistent, globally unique names for documents, databases, images, vocabulary terms, etc. Properly managed, these identifiers can be used as long term durable information object references within naming schemes such as ARK, PURL, URN, DOI, and LSID. At the same time, alternative minters can be set up to produce short-lived names for transaction identifiers, compact web server session keys (cf. UUIDs), and other ephemera.

In general, a **noid** minter efficiently generates, tracks, and binds unique identifiers, which are produced without replacement in random or sequential order, and with or without a check character that can be used for detecting transcription errors. A minter can bind identifiers to arbitrary element names and element values that are either stored or produced upon retrieval from rule-based transformations of requested identifiers; the latter has application in identifier resolution. Noid minters are very fast, scalable, easy to create and tear down, and have a relatively small footprint. They use BerkeleyDB as the underlying database.

Identifiers generated by a **noid** minter are also known as ‘‘noids’’ (nice opaque identifiers). While a minter can record and bind any identifiers that you bring to its attention, often it is used to generate, bringing to your attention, identifier strings that carry no widely recognizable meaning. This semantic opaqueness reduces their vulnerability to era− and language-specific change, and helps persistence by making for identifiers that can age and travel well.

BUGS

Probably. Please report to jak at ucop dot edu.

COPYRIGHT AND LICENSE

SEE ALSO

dbopen (3), *perl* (1), <http://www.cdlib.org/inside/diglib/ark/>

AUTHOR

John A. Kunze