

Nim Standard Library nimversion

Andreas Rumpf

July 26, 2019

Contents

1	Pure libraries	2
1.1	Core	2
1.2	Collections and algorithms	2
1.3	String handling	3
1.4	Generic Operating System Services	3
1.5	Math libraries	4
1.6	Internet Protocols and Support	4
1.7	Parsers	4
1.8	XML Processing	5
1.9	Cryptography and Hashing	5
1.10	Multimedia support	5
1.11	Miscellaneous	5
1.12	Modules for JS backend	6
2	Impure libraries	6
2.1	Regular expressions	6
2.2	Database support	6
2.3	Other	6
3	Wrappers	6
3.1	Windows specific	6
3.2	UNIX specific	6
3.3	Regular expressions	6
3.4	GUI libraries	6
3.5	Database support	7
3.6	Network Programming and Internet Protocols	7
4	Nimble	7

"The good thing about reinventing the wheel is that you can get a round one."

Though the Nim Standard Library is still evolving, it is already quite usable. It is divided into *pure libraries*, *impure libraries* and *wrappers*.

Pure libraries do not depend on any external `*.dll` or `lib*.so` binary while impure libraries do. A wrapper is an impure library that is a very low-level interface to a C library.

Read this document for a quick overview of the API design.

In addition to the modules in the standard library, third-party packages created by the Nim community can be used via Nimble, Nim's package manager.

1 Pure libraries

1.1 Core

- `system` Basic procs and operators that every program needs. It also provides IO facilities for reading and writing text and binary files. It is imported implicitly by the compiler. Do not import it directly. It relies on compiler magic to work.
- `threads` Nim thread support. **Note:** This is part of the `system` module. Do not import it explicitly.
- `channels` Nim message passing support for threads. **Note:** This is part of the `system` module. Do not import it explicitly.
- `locks` Locks and condition variables for Nim.
- `rlocks` Reentrant locks for Nim.
- `macros` Contains the AST API and documentation of Nim for writing macros.
- `typeinfo` Provides (unsafe) access to Nim's run time type information.
- `typetraits` This module defines compile-time reflection procs for working with types.
- `threadpool` Implements Nim's `spawn`.
- `cpuinfo` This module implements procs to determine the number of CPUs / cores.
- `lenientops` Provides binary operators for mixed integer/float expressions for convenience.
- `bitops` Provides a series of low level methods for bit manipulation.

1.2 Collections and algorithms

- `algorithm` Implements some common generic algorithms like `sort` or `binary search`.
- `tables` Nim hash table support. Contains `tables`, `ordered tables` and `count tables`.
- `sets` Nim hash and bit set support.
- `lists` Nim linked list support. Contains `singly` and `doubly linked lists` and `circular lists` ("rings").
- `deques` Implementation of a double-ended queue. The underlying implementation uses a `seq`.
- `heapqueue` This module implements `Heap queue` (a.k.a. `priority queue`) algorithm.
- `intsets` Efficient implementation of a set of ints as a sparse bit set.
- `critbits` This module implements a *crit bit tree* which is an efficient container for a sorted set of strings, or for a sorted mapping of strings.
- `sequtils` This module implements operations for the built-in `seq` type which were inspired by functional programming languages.
- `sharedtables` Nim shared hash table support. Contains `shared tables`.
- `sharedlist` Nim shared linked list support. Contains `shared singly linked list`.

1.3 String handling

- `strutils` This module contains common string handling operations like changing case of a string, splitting a string into substrings, searching for substrings, replacing substrings.
- `strformat` Macro based standard string interpolation / formatting. Inspired by Python's `f`-strings.
- `strmisc` This module contains uncommon string handling operations that do not fit with the commonly used operations in `strutils`.
- `parseutils` This module contains helpers for parsing tokens, numbers, identifiers, etc.
- `strscans` This module contains a `scanf` macro for convenient parsing of mini languages.
- `strtabs` The `strtabs` module implements an efficient hash table that is a mapping from strings to strings. Supports a case-sensitive, case-insensitive and style-insensitive mode. An efficient string substitution operator `%` for the string table is also provided.
- `unicode` This module provides support to handle the Unicode UTF-8 encoding.
- `unidecode` It provides a single proc that does Unicode to ASCII transliterations. Based on Python's `Unidecode` module.
- `punycode` Implements a representation of Unicode with the limited ASCII character subset.
- `encodings` Converts between different character encodings. On UNIX, this uses the `iconv` library, on Windows the Windows API.
- `pegs` This module contains procedures and operators for handling PEGs.
- `ropes` This module contains support for a *rope* data type. Ropes can represent very long strings efficiently; especially concatenation is done in $O(1)$ instead of $O(n)$.
- `matchers` This module contains various string matchers for email addresses, etc.
- `subexes` This module implements advanced string substitution operations.

1.4 Generic Operating System Services

- `os` Basic operating system facilities like retrieving environment variables, reading command line arguments, working with directories, running shell commands, etc.
- `osproc` Module for process communication beyond `os.execShellCmd`.
- `times` The `times` module contains basic support for working with time.
- `dynlib` This module implements the ability to access symbols from shared libraries.
- `streams` This module provides a stream interface and two implementations thereof: the *FileStream* and the *StringStream* which implement the stream interface for Nim file objects (*File*) and strings. Other modules may provide other implementations for this standard stream interface.
- `marshal` Contains procs for serialization and deserialization of arbitrary Nim data structures.
- `terminal` This module contains a few procedures to control the *terminal* (also called *console*). The implementation simply uses ANSI escape sequences and does not depend on any other module.
- `memfiles` This module provides support for memory mapped files (Posix's `mmap`) on the different operating systems.
- `asyncfile` This module implements asynchronous file reading and writing using `asyncdispatch`.
- `distros` This module implements the basics for OS distribution ("distro") detection and the OS's native package manager. Its primary purpose is to produce output for Nimble packages, but it also contains the widely used **Distribution** enum that is useful for writing platform specific code.
- `volatile` This module contains code for generating volatile loads and stores, which are useful in embedded and systems programming.

1.5 Math libraries

- `math` Mathematical operations like cosine, square root.
- `complex` This module implements complex numbers and their mathematical operations.
- `rational` This module implements rational numbers and their mathematical operations.
- `fenv` Floating-point environment. Handling of floating-point rounding and exceptions (overflow, zero-divide, etc.).
- `mersenne` Mersenne twister random number generator.
- `random` Fast and tiny random number generator.
- `stats` Statistical analysis

1.6 Internet Protocols and Support

- `cgi` This module implements helpers for CGI applications.
- `scgi` This module implements helpers for SCGI applications.
- `browsers` This module implements procs for opening URLs with the user's default browser.
- `httpclient` This module implements a simple HTTP client which supports both synchronous and asynchronous retrieval of web pages.
- `smtp` This module implement a simple SMTP client.
- `cookies` This module contains helper procs for parsing and generating cookies.
- `mimetypes` This module implements a mimetypes database.
- `uri` This module provides functions for working with URIs.
- `asyncdispatch` This module implements an asynchronous dispatcher for IO operations.
- `asyncnet` This module implements asynchronous sockets based on the `asyncdispatch` module.
- `asyncttpserver` This module implements an asynchronous HTTP server using the `asyncnet` module.
- `asynctftpclient` This module implements an asynchronous FTP client using the `asyncnet` module.
- `net` This module implements a high-level sockets API. It will replace the `sockets` module in the future.
- `nativesockets` This module implements a low-level sockets API.
- `selectors` This module implements a selector API with backends specific to each OS. Currently `epoll` on Linux and `select` on other operating systems.

1.7 Parsers

- `parseopt` The `parseopt` module implements a command line option parser.
- `parsecfg` The `parsecfg` module implements a high performance configuration file parser. The configuration file's syntax is similar to the Windows `.ini` format, but much more powerful, as it is not a line based parser. String literals, raw string literals and triple quote string literals are supported as in the Nim programming language.
- `parsexml` The `parsexml` module implements a simple high performance XML/HTML parser. The only encoding that is supported is UTF-8. The parser has been designed to be somewhat error correcting, so that even some "wild HTML" found on the Web can be parsed with it.

- `parsecsv` The `parsecsv` module implements a simple high performance CSV parser.
- `parsesql` The `parsesql` module implements a simple high performance SQL parser.
- `json` High performance JSON parser.
- `lexbase` This is a low level module that implements an extremely efficient buffering scheme for lexers and parsers. This is used by the diverse parsing modules.
- `highlite` Source highlighter for programming or markup languages. Currently only few languages are supported, other languages may be added. The interface supports one language nested in another.
- `rst` This module implements a `reStructuredText` parser. A large subset is implemented. Some features of the markdown wiki syntax are also supported.
- `rstast` This module implements an AST for the `reStructuredText` parser.
- `rstgen` This module implements a generator of HTML/Latex from `reStructuredText`.
- `sexp` High performance `sexp` parser and generator, mainly for communication with emacs.

1.8 XML Processing

- `xmldom` This module implements the XML DOM Level 2.
- `xmldomparser` This module parses an XML Document into a XML DOM Document representation.
- `xmmtree` A simple XML tree. More efficient and simpler than the DOM. It also contains a macro for XML/HTML code generation.
- `xmlparser` This module parses an XML document and creates its XML tree representation.
- `htmlparser` This module parses an HTML document and creates its XML tree representation.
- `htmlgen` This module implements a simple XML and HTML code generator. Each commonly used HTML tag has a corresponding macro that generates a string with its HTML representation.

1.9 Cryptography and Hashing

- `hashes` This module implements efficient computations of hash values for diverse Nim types.
- `md5` This module implements the MD5 checksum algorithm.
- `base64` This module implements a base64 encoder and decoder.
- `sha1` This module implements a sha1 encoder and decoder.

1.10 Multimedia support

- `colors` This module implements color handling for Nim. It is used by the `graphics` module.

1.11 Miscellaneous

- `oids` An OID is a global ID that consists of a timestamp, a unique counter and a random value. This combination should suffice to produce a globally distributed unique ID. This implementation was extracted from the MongoDB interface and it thus binary compatible with a Mongo OID.
- `endians` This module contains helpers that deal with different byte orders.
- `logging` This module implements a simple logger.
- `options` Types which encapsulate an optional value.
- `sugar` This module implements nice syntactic sugar based on Nim's macro system.

- `coro` This module implements experimental coroutines in Nim.
- `unittest` Implements a Unit testing DSL.
- `segfaults` Turns access violations or segfaults into a `NilAccessError` exception.

1.12 Modules for JS backend

- `dom` Declaration of the Document Object Model for the JS backend.
- `jsffi` Types and macros for easier interaction with JavaScript.
- `asyncjs` Types and macros for writing asynchronous procedures in JavaScript.
- `jscore` Wrapper of core JavaScript functions. For most purposes you should be using the `math`, `json`, and `times` `stdlib` modules instead of this module.

2 Impure libraries

2.1 Regular expressions

- `re` This module contains procedures and operators for handling regular expressions. The current implementation uses PCRE.

2.2 Database support

- `db_postgres` A higher level PostgreSQL database wrapper. The same interface is implemented for other databases too.
- `db_mysql` A higher level MySQL database wrapper. The same interface is implemented for other databases too.
- `db_sqlite` A higher level SQLite database wrapper. The same interface is implemented for other databases too.

2.3 Other

- `ssl` This module provides an easy to use sockets-style Nim interface to the OpenSSL library.

3 Wrappers

The generated HTML for some of these wrappers is so huge that it is not contained in the distribution. You can then find them on the website.

3.1 Windows specific

- `winlean` Contains a wrapper for a small subset of the Win32 API.

3.2 UNIX specific

- `posix` Contains a wrapper for the POSIX standard.

3.3 Regular expressions

- `pcre` Wrapper for the PCRE library.

3.4 GUI libraries

- `iup` Wrapper of the IUP GUI library.

3.5 Database support

- postgres Contains a wrapper for the PostgreSQL API.
- mysql Contains a wrapper for the MySQL API.
- sqlite3 Contains a wrapper for SQLite 3 API.
- odbcsql interface to the ODBC driver.

3.6 Network Programming and Internet Protocols

- openssl Wrapper for OpenSSL.

4 Nimble

Nimble is a package manager for the Nim programming language. For instructions on how to install Nimble packages see its README.

To see a list of Nimble's packages, check out <https://nimble.directory/> or the packages repo on GitHub.