Introduction to the off-line version of Yacas

by the Yacas team [1]

Yacas version: 1.3.0
generated on December 8, 2011

This document gives a short introduction to Yacas. Included is a brief tutorial on the syntax and some commands to get you started using Yacas. There are also some examples.

Contents

Chapter 1

Getting started with Yacas off-line

1.1 Introduction

This section describes how to get started with YACAS locally by downloading and compiling the program. However, this is not strictly necessary. You can also go online to our web site and use YACAS in there inside your browser. The web page contains tutorials, example calculations and lots of documentation. This document is only useful if you plan to build and install YACAS yourself.

1.2 Installing Yacas

Read the file `INSTALL` for instructions on how to compile YACAS. YACAS is portable across most Unix-ish platforms and requires only a standard C++ compiler such as `g++`.

The base YACAS application accepts text as input and returns text as output. This makes it rather platform-independent. Apart from Unix-like systems, YACAS has been compiled on Windows and on EPOC32, aka Psion (which doesn't come with a standard C++ library!). The source code to compile YACAS for Windows can be found at the Sourceforge repository (Web URL: `http://sourceforge.net/projects/yacas/`).

For Unix, compilation basically amounts to the standard sequence

```
./configure
make
make install
```

This will install the binaries to `/usr/local/bin` and the library files to `/usr/local/share/yacas/`.

Additionally, LaTeX-formatted documentation in PostScript and PDF formats can be produced by the command

```
make texdocs
```

or, alternatively, by passing `--enable-ps-doc` or `--enable-pdf-doc` to `./configure` when building YACAS. In the latter case, the documentation will be automatically rebuilt every time the documentation changes (which is useful when maintaining the documentation).

In addition, there is also a Java version of the lower-level interpreter. The code for this Java version can be found in the directory "JavaYacas", and can be compiled with the make file "makefile.yacas", by typing in:

```
make -f makefile.yacas
```

The interpreter can then be invoked from the command line with:

```
java -jar yacas.jar
```

or alternatively it can be invoked as an applet, by opening yacasconsole.html.

The binary files that comprise the entire binary release for the Java version are:

1. yacas.jar - the Java class files in one jar file.
2. yacasconsole.html - the file that launches the applet.
3. hints.txt - the hints that are shown in the applet (the grey box with commands that match what you are typing in at that moment).

The Java version has almost all the features the C++ version has. In fact, there is no reason the Java version should not have all the same features. Not all command line arguments are available yet, and the command line prompt does not have the history yet.

1.3 Using the console mode

You can run YACAS in the console mode simply by typing `yacas`. The YACAS command prompt looks like this:

```
In>
```

and YACAS's answers appear after the prompt

```
Out>
```

A YACAS session may be terminated by typing `Exit()` or `quit`. Pressing `^C` will also quit YACAS; however, pressing `^C` while YACAS is busy with a calculation will stop just that calculation. A session can be restarted (forgetting all previous definitions and results) by typing

```
restart
```

Typically, you would enter one statement per line, for example

```
In> Sin(Pi/2);
Out> 1;
```

Statements should end with a semicolon (`;`) although this is not required in interactive sessions (YACAS will append a semicolon at end of line to finish the statement).

Type `Example();` to get some random examples of YACAS calculations.

The command line has a history list, so it should be easy to browse through the expressions you entered previously using the Up and Down arrow keys.

When a few characters have been typed, the command line will use the characters before the cursor as a filter into the history, and allow you to browse through all the commands in the history that start with these characters quickly, instead of browsing through the entire history.

Typing the first few characters of a previous expression and then hitting the TAB key makes YACAS recall the last expression in the history list that matches these first characters.

Commands spanning multiple lines can (and actually have to) be entered by using a trailing backslash
at end of each continued line. For example:

```
In> a:=2+3+
Error on line 1 in file [CommandLine]
Line error occurred on:
>>>
Error parsing expression
```

```
In> a:=2+3+ \
In> 1
Out> 6;
```

The error after our first attempt occurred because YACAS has appended a semicolon at end of the first line and `2+3+;` is not a valid YACAS expression.

Incidentally, any text YACAS prints without a prompt is either messages printed by functions as their side-effect, or error messages. Resulting values of expressions are always printed after an `Out>` prompt.

Chapter 2

Examples

This is a small tour of the capabilities YACAS currently offers. Note that this list of examples is far from complete. YACAS contains a few hundred commands, of which only a few are shown here.

Additional example calculations including the results can be found here:

- A selection of calculations from the *Wester benchmark*, in *Essays on Yacas, Chapter 2*.
- Some additional example calculations (Web URL: `mybench2.html`) that YACAS can currently perform.

2.1 Using Yacas from the console

Command-line options

The default operation of YACAS is to run in the interactive console mode. YACAS accepts several options that modify its operation. Here is a summary of options:

- *filename* ... (read and execute a file or several files)
- `-c` (omit line prompts)
- `-d` (print default directory)
- `-v` (print version information)
- `-f` (execute standard input as one statement)
- `-p` (do not use terminal capabilities)
- `-t` (enable extra history features)
- `--archive` *filename* (use a given library archive file)
- `--init` *filename* (use a given initial file)
- `--patchload` (use `PatchLoad` to load files)
- `--read-eval-print` *expression* (call this expression for the read-eval-print loop)
- `--rootdir` *directory* (specify default directory for scripts)
- `--server` *port* (start YACAS as a network server on given port)
- `--single-user-server` (If in server mode, start it in single-user mode)
- `--verbose-debug` (turn on showing some additional debugging information on screen)
- `--disable-compiled-plugins` (disable loading of compiled plugins, loading the script versions instead)
- `--stacksize` *size* (change size of stack arguments are stored on)
- `--execute` *expression* (run expression from the command line)

Options can be combined, for example

```
yacas -pc filename
```

will read and execute the file `filename` non-interactively without using terminal capabilities and without printing prompts.

Here is a more detailed description of the command-line options.

```
yacas -c
```

Inhibit printing of prompts `In>` and `Out>`. Useful for non-interactive sessions.

```
yacas -f
```

Reads standard input as one file, but executes only the first statement in it. (You may want to use a statement block to have several statements executed.)

```
yacas -p
```

Does not use terminal capabilities, no fancy editing on the command line and no escape sequences printed. Useful for non-interactive sessions.

```
yacas -t
```

Enable some extra history recall functionality in console mode: after executing a command from the history list, the next unmodified command from the history list will be automatically entered on the command line.

```
yacas [options] {filename}
```

Reads and executes commands in the filename and exits. Equivalent to `Load()`.

```
yacas -v
```

Prints version information and exits. (This is the same information as returned by `Version()`.)

```
yacas -d
```

Prints the path to the YACAS default library directory (this information is compiled into the YACAS executable) and exits.

```
yacas --patchload
```

Will load every file on the command line with the `PatchLoad` command instead of the normal `Load` command, This is useful for generating HTML pages for a web site using the YACAS scripting language, much like you can do with the PHP scripting language.

```
yacas --init [file]
```

Tells the system to load `file` as the initialization file. By default it loads the file `yacasinit.ys` from the scripts directory. Thus for customization one has two options: write a `~/.yacasrc` file with initialization code (as it is loaded after the initialization script is loaded), or write a custom initialization script that first uses `yacasinit.ys` and adds some extra custom code.

```
yacas --read-eval-print [expression]
```

Call `expression` for the read-eval-print loop. The default read-eval-print loop is implemented in the initialization script `yacasinit.ys` as the function `REP`. The default behavior is therefore equivalent to `--read-eval-print REP()`.

There is also a fallback read-eval-print loop in the kernel; it can be invoked by passing an empty string to this command line option, as `--read-eval-print ""`.

An alternative way to replace the default read-eval-print loop is to write a custom initialization script that implements the read-eval-print loop function `REP()` instead of `yacasinit.ys`.

Care has to be taken with this option because a Yacas session may become unusable if the read-eval-print expression doesn't function correctly.

```
yacas --server <port>
```

On some platforms server mode can be enabled at build time by passing the flag `--enable-server` to the `./configure` script. YACAS then allows you to pass the flag `--server` with a port number behind it, and the YACAS executable will listen to the socket behind that port instead of waiting for user input on the console.

Commands can be sent to the server by sending a text line as one block of data, and the server will respond back with another text block.

One can test this function by using `telnet`. First, set up the server by calling

```
yacas --server 9734
```

and then invoke `telnet` in another window, for example:

```
telnet 127.0.0.1 9734
```

Then type a line of Yacas input and hit Enter. The result will be one line that you will get back from the Yacas server.

Some security measures and resource management measures have been taken. No more than 10 connections can be alive at any time, a calculation cannot take more than 30 seconds, and YACAS operates in the *secure* mode, much like calling an expression by passing it as an argument to the `Secure` function. This means that no system calls are allowed, and no writing to local files, amongst other things. Something that has not been taken care of yet is memory use. A calculation could take up all memory, but not for longer than 30 seconds.

The server is single-threaded, but has persistent sessions for at most 10 users at a time, from which it can service requests in a sequential order. To make the service multi-threaded, a solution might be to have a proxy in front of the service listening to the port, redirecting it to different processes which get started up for users (this has not been tried yet).

The flag –single-user-server can be passed on to instruct `yacas` to start in single-user mode. In this mode, unsecure operations can be performed (like reading from and writing to files), and the calculation may take more than 30 seconds. The `yacas` process will automatically be shut down when the last session is closed or when "Exit();" is sent.

```
yacas --rootdir [directory]
```

Tells the system where to find the library scripts. Here, `directory` is a path that is passed to `DefaultDirectory`. It is also possible to give a list of directories, separated by a colon, e.g. `yacas --rootdir scripts/:morescripts/`. Note that it is not necessary to append a trailing slash to the directory names.

```
yacas --archive [file]
```

Use a compressed archive instead of the script library.

YACAS has an experimental system where files can be compressed into one file, and accessed through this command line option. The advantages are:

1. Smaller disk/memory use (useful if YACAS is used on small hand-held computers).
2. No problems with directory path separators: "`path/file`" will always resolve to the right file, no matter what platform (read: Windows) it runs on.
3. The start-up time of the program might improve a little, since a smaller file is loaded from disk (disk access being slow), and then decompressed in memory, which might be a lot faster than loading from disk.

An additional savings is due to the fact that the script files are stripped from white spaces and comments, making them smaller and faster loading.

To prepare the compressed library archive, run `./configure` with the command line option `--enable-archive`.

The result should be the archive file `scripts.dat`. Then launch YACAS with the command line option `--archive scripts.dat`, with the file `scripts.dat` in the current directory.

The reason that the `scripts.dat` file is not built automatically is that it is not tested, at this time, that the build process works on all platforms. (Right now it works on Unix, MacOSX, and Win32.)

Alternatively, configure Yacas with

```
./configure --enable-archive
```

and the archive file `scripts.dat` will be created in the `ramscripts/` subdirectory.

When an archive is present, Yacas will try to load it before it looks for scripts from the library directories. Typing

```
make archivetest -f makefile.compressor
```

in the `ramscripts/` directory runs all the test scripts using the archived files.

The currently supported compression schemes are uncompressed and compressed with `minilzo`. Script file stripping (removing whitespace and comments) may be disabled by editing `compressor.cpp` (variable `strip_script`).

```
yacas --disable-compiled-plugins
```

Disable loading of compiled scripts, in favor of scripts themselves. This is useful when developing the scripts that need to be compiled in the end, or when the scripts have not been compiled yet.

```
yacas --stacksize <size>
```

Yacas maintains an internal stack for arguments. For nested function calls, all arguments currently used are on this stack. The size of this stack is 50000 be default.

For a function that would take 4 arguments and has one return value, there would be 5 places reserved on this stack, and the function could call itself recursively 10000 steps deep.

This differs from the MaxEvalDepth mechanism. The MaxEvalDepth mechanism allows one to specify the number of separate stack frames (number of calls, nested), instead of the number of arguments pushed on the stack. MaxEvalDepth was introduced to protect the normal `C++` stack.

```
yacas --execute <expression>
```

This instructs Yacas to run a certain expression, passed in over the command line, before dropping to the read-eval-print loop. This can be used to load a file before dropping to the command line without exiting (if there are files to run specified on the command line, Yacas will exit after running these scripts). Alternatively, the expression can exit the interpreter immediately by calling `Exit();`. When used in combination with `-pc`, the Yacas interpreter can be used to calculate something and print the result to standard output. Example:

```
user% ./yacas -pc --execute '[Echo("answer ",D(x)Sin(x));Exit();]'
answer Cos(x)
user%
```

Chapter 3

Running Yacas off-line

3.1 Interactive session commands

This section describes the special commands for the Yacas interactive sessions (for example, to restart or to exit the interpreter). These commands are not functions but special directives that only apply while running Yacas interactively. They should not be used in scripts.

`quit` — stop Yacas from running, from the command line

`restart` — restart Yacas (to start with a clean slate)

(YACAS internal)

Calling format:

```
quit
restart
```

Description:

Type `quit` or `restart` at the Yacas prompt to exit or to restart the interpreter.

The directives `quit` and `restart` are *not* reserved words or variable names. They take effect only when typed as first characters at a prompt.

Pressing `Ctrl-C` will stop the currently running calculation. If there is no currently running calculation, `Ctrl-C` will quit the interpreter.

When the interpreter quits, it saves the command history (so quitting by `Ctrl-C` does not mean a "crash").

Examples:

To be effective, the directive must be typed immediately after the prompt:

```
In> quit
Quitting...
```

We can use variables named `quit`:

```
In> 1+quit
Out> quit+1;
```

There is no effect if we type some spaces first:

```
In>       restart
Out> restart;
```

See also: `Exit`

3.2 Command-line options

The default operation of YACAS is to run in the interactive console mode. YACAS accepts several options that modify its operation. Here is a summary of options:

- *filename* ... (read and execute a file or several files)
- `-c` (omit line prompts)
- `-d` (print default directory)
- `-v` (print version information)
- `-f` (execute standard input as one statement)
- `-p` (do not use terminal capabilities)
- `-t` (enable extra history features)
- `--archive` *filename* (use a given library archive file)
- `--init` *filename* (use a given initial file)
- `--patchload` (use `PatchLoad` to load files)
- `--read-eval-print` *expression* (call this expression for the read-eval-print loop)
- `--rootdir` *directory* (specify default directory for scripts)
- `--server` *port* (start YACAS as a network server on given port)
- `--single-user-server` (If in server mode, start it in single-user mode)
- `--verbose-debug` (turn on showing some additional debugging information on screen)
- `--disable-compiled-plugins` (disable loading of compiled plugins, loading the script versions instead)
- `--stacksize` *size* (change size of stack arguments are stored on)
- `--execute` *expression* (run expression from the command line)

Options can be combined, for example

```
yacas -pc filename
```

will read and execute the file `filename` non-interactively without using terminal capabilities and without printing prompts.

Here is a more detailed description of the command-line options.

```
yacas -c
```

Inhibit printing of prompts `In>` and `Out>`. Useful for non-interactive sessions.

```
yacas -f
```

Reads standard input as one file, but executes only the first statement in it. (You may want to use a statement block to have several statements executed.)

```
yacas -p
```

Does not use terminal capabilities, no fancy editing on the command line and no escape sequences printed. Useful for non-interactive sessions.

```
yacas -t
```

Enable some extra history recall functionality in console mode: after executing a command from the history list, the next unmodified command from the history list will be automatically entered on the command line.

```
yacas [options] {filename}
```

Reads and executes commands in the filename and exits. Equivalent to `Load()`.

```
yacas -v
```

Prints version information and exits. (This is the same information as returned by `Version()`.)

```
yacas -d
```

Prints the path to the YACAS default library directory (this information is compiled into the YACAS executable) and exits.

```
yacas --patchload
```

Will load every file on the command line with the `PatchLoad` command instead of the normal `Load` command, This is useful for generating HTML pages for a web site using the YACAS scripting language, much like you can do with the PHP scripting language.

```
yacas --init [file]
```

Tells the system to load `file` as the initialization file. By default it loads the file `yacasinit.ys` from the scripts directory. Thus for customization one has two options: write a `~/.yacasrc` file with initialization code (as it is loaded after the initialization script is loaded), or write a custom initialization script that first uses `yacasinit.ys` and adds some extra custom code.

```
yacas --read-eval-print [expression]
```

Call `expression` for the read-eval-print loop. The default read-eval-print loop is implemented in the initialization script `yacasinit.ys` as the function `REP`. The default behavior is therefore equivalent to `--read-eval-print REP()`.

There is also a fallback read-eval-print loop in the kernel; it can be invoked by passing an empty string to this command line option, as `--read-eval-print ""`.

An alternative way to replace the default read-eval-print loop is to write a custom initialization script that implements the read-eval-print loop function `REP()` instead of `yacasinit.ys`.

Care has to be taken with this option because a Yacas session may become unusable if the read-eval-print expression doesn't function correctly.

```
yacas --server <port>
```

On some platforms server mode can be enabled at build time by passing the flag `--enable-server` to the `./configure` script. YACAS then allows you to pass the flag `--server` with a port number behind it, and the YACAS executable will listen to the socket behind that port instead of waiting for user input on the console.

Commands can be sent to the server by sending a text line as one block of data, and the server will respond back with another text block.

One can test this function by using `telnet`. First, set up the server by calling

```
yacas --server 9734
```

and then invoke `telnet` in another window, for example:

```
telnet 127.0.0.1 9734
```

Then type a line of Yacas input and hit Enter. The result will be one line that you will get back from the Yacas server.

Some security measures and resource management measures have been taken. No more than 10 connections can be alive at any time, a calculation cannot take more than 30 seconds, and YACAS operates in the *secure* mode, much like calling an expression by passing it as an argument to the `Secure` function. This means that no system calls are allowed, and no writing to local files, amongst other things. Something that has not been taken care of yet is memory use. A calculation could take up all memory, but not for longer than 30 seconds.

The server is single-threaded, but has persistent sessions for at most 10 users at a time, from which it can service requests in a sequential order. To make the service multi-threaded, a solution might be to have a proxy in front of the service listening to the port, redirecting it to different processes which get started up for users (this has not been tried yet).

The flag –single-user-server can be passed on to instruct `yacas` to start in single-user mode. In this mode, unsecure operations can be performed (like reading from and writing to files), and the calculation may take more than 30 seconds. The `yacas` process will automatically be shut down when the last session is closed or when "Exit();" is sent.

```
yacas --rootdir [directory]
```

Tells the system where to find the library scripts. Here, `directory` is a path that is passed to `DefaultDirectory`. It is also possible to give a list of directories, separated by a colon, e.g. `yacas --rootdir scripts/:morescripts/`. Note that it is not necessary to append a trailing slash to the directory names.

```
yacas --archive [file]
```

Use a compressed archive instead of the script library.

YACAS has an experimental system where files can be compressed into one file, and accessed through this command line option. The advantages are:

1. Smaller disk/memory use (useful if YACAS is used on small hand-held computers).
2. No problems with directory path separators: "`path/file`" will always resolve to the right file, no matter what platform (read: Windows) it runs on.
3. The start-up time of the program might improve a little, since a smaller file is loaded from disk (disk access being slow), and then decompressed in memory, which might be a lot faster than loading from disk.

An additional savings is due to the fact that the script files are stripped from white spaces and comments, making them smaller and faster loading.

To prepare the compressed library archive, run `./configure` with the command line option `--enable-archive`.

The result should be the archive file `scripts.dat`. Then launch YACAS with the command line option `--archive scripts.dat`, with the file `scripts.dat` in the current directory.

The reason that the `scripts.dat` file is not built automatically is that it is not tested, at this time, that the build process works on all platforms. (Right now it works on Unix, MacOSX, and Win32.)

Alternatively, configure Yacas with

```
./configure --enable-archive
```

and the archive file `scripts.dat` will be created in the `ramscripts/` subdirectory.

When an archive is present, Yacas will try to load it before it looks for scripts from the library directories. Typing

```
make archivetest -f makefile.compressor
```

in the `ramscripts/` directory runs all the test scripts using the archived files.

The currently supported compression schemes are uncompressed and compressed with `minilzo`. Script file stripping (removing whitespace and comments) may be disabled by editing `compressor.cpp` (variable `strip_script`).

```
yacas --disable-compiled-plugins
```

Disable loading of compiled scripts, in favor of scripts themselves. This is useful when developing the scripts that need to be compiled in the end, or when the scripts have not been compiled yet.

```
yacas --stacksize <size>
```

Yacas maintains an internal stack for arguments. For nested function calls, all arguments currently used are on this stack. The size of this stack is 50000 be default.

For a function that would take 4 arguments and has one return value, there would be 5 places reserved on this stack, and the function could call itself recursively 10000 steps deep.

This differs from the MaxEvalDepth mechanism. The MaxEvalDepth mechanism allows one to specify the number of separate stack frames (number of calls, nested), instead of the number of arguments pushed on the stack. MaxEvalDepth was introduced to protect the normal `C++` stack.

```
yacas --execute <expression>
```

This instructs Yacas to run a certain expression, passed in over the command line, before dropping to the read-eval-print loop. This can be used to load a file before dropping to the command line without exiting (if there are files to run specified on the command line, Yacas will exit after running these scripts). Alternatively, the expression can exit the interpreter immediately by calling `Exit();`. When used in combination with `-pc`, the Yacas interpreter can be used to calculate something and print the result to standard output. Example:

```
user% ./yacas -pc --execute '[Echo("answer ",D(x)Sin(x));Exit();]'
answer Cos(x)
user%
```

Chapter 4

Startup configuration

Yacas allows you to configure a few things at startup. The file `~.yacasrc` is written in the Yacas language and will be executed when Yacas is run. The following functions can be useful in the `~.yacasrc` file.

DefaultDirectory — add directory to path for Yacas scripts

(YACAS internal)

Calling format:

```
DefaultDirectory(path)
```

Parameters:

`path` – a string containing a full path where yacas script files reside

Description:

When loading files, yacas is also allowed to look in the folder "path". `path` will be prepended to the file name before trying to load the file. This means that "path" should end with a forward slash (under Unix-like operating systems).

Yacas first tries to load a file from the current directory, and otherwise it tries to load from directories defined with this function, in the order they are defined. Note there will be at least one directory specified at start-up time, defined during compilation. This is the directory Yacas searches for the initialization scripts and standard scripts.

Examples:

```
In> DefaultDirectory("/home/user/myscripts/");
Out> True;
```

See also: `Load`, `Use`, `DefLoad`, `FindFile`

PrettyPrinter'Set — set routine to use as pretty-printer

PrettyPrinter'Get — get routine to use as pretty-printer

(standard library)

Calling format:

```
PrettyPrinter'Set(printer)
PrettyPrinter'Set()
PrettyPrinter'Get()
```

Parameters:

`printer` – a string containing the name of a function that can "pretty-print" an expression.

Description:

This function sets up the function printer to print out the results on the command line. This can be reset to the internal printer with `PrettyPrinter'Set()` (when no argument is given, the system returns to the default).

Currently implemented prettyprinters are: `PrettyForm`, `TeXForm`, `Print`, `OMForm`, `CForm` and `DefaultPrint`.

`PrettyPrinter'Get()` returns the current pretty printer, or it returns an empty string if the default pretty printer is used.

Examples:

```
In> Taylor(x,0,5)Sin(x)
Out> x-x^3/6+x^5/120;
In> PrettyPrinter'Set("PrettyForm");

True

In> Taylor(x,0,5)Sin(x)

     3    5
    x    x
x - -- + ---
    6    120

In> PrettyPrinter'Set();
Out> True;
In> Taylor(x,0,5)Sin(x)
Out> x-x^3/6+x^5/120;
```

See also: `PrettyForm`, `Write`, `TeXForm`, `CForm`, `OMForm`, `PrettyReader'Set`, `PrettyReader'Get`

PrettyReader'Set — set routine to use as pretty-reader

PrettyReader'Get — get routine that is currently used as pretty-reader

(standard library)

Calling format:

```
PrettyReader'Set(reader)
PrettyReader'Set()
PrettyReader'Get()
```

Parameters:

`reader` – a string containing the name of a function that can read an expression from current input.

Description:

This function sets up the function reader to read in the input on the command line. This can be reset to the internal reader with `PrettyReader'Set()` (when no argument is given, the system returns to the default).

Currently implemented PrettyReaders are: `LispRead`, `OMRead`.

`PrettyReader'Get()` returns the current reader, or it returns an empty string if the default pretty printer is used.

Examples:

```
In> Taylor(x,0,5)Sin(x)
Out> x-x^3/6+x^5/120
In> PrettyReader'Set("LispRead")
Out> True
In> (Taylor x 0 5 (Sin x))
Out> x-x^3/6+x^5/120
```

See also: `Read`, `LispRead`, `OMRead`, `PrettyPrinter'Set`, `PrettyPrinter'Get`

`MaxEvalDepth` — set depth of recursion stack

(YACAS internal)

Calling format:

```
MaxEvalDepth(n)
```

Parameters:

`n` – integer

Description:

Sets the maximum depth of recursive function call. An error message is printed when too many recursive calls are executed, and this function can be used to increase or decrease the limit as necessary.

`HistorySize` — set size of history file

(YACAS internal)

Calling format:

```
HistorySize(n)
```

Parameters:

`n` – number of lines to store in history file

Description:

When exiting, yacas saves the command line history to a file `~/.yacas_history`. By default it will save the last 1024 lines. The default can be overridden with this function. Passing -1 tells the system to save *all* lines.

Examples:

```
In> HistorySize(200)
Out> True;
In> quit
```

See also: `quit`

Chapter 5

Platform-dependent packages

Certain facilities have been developed for use on Unix-like platforms, which is currently the main development target for Yacas. Other facilities have limited support on the Windows platform as well. These functions are described in this chapter.

`Version` — show version of Yacas

(YACAS internal)

Calling format:

```
Version()
```

Description:

The function `Version()` returns a string representing the version of the currently running Yacas interpreter.

Examples:

```
In> Version()
Out> "1.0.48rev3";
In> LessThan(Version(), "1.0.47")
Out> False;
In> GreaterThan(Version(), "1.0.47")
Out> True;
```

The last two calls show that the `LessThan` and `GreaterThan` functions can be used for comparing version numbers. This method is only guaranteed, however, if the version is always expressed in the form `d.d.dd` as above.

See also: `LessThan`, `GreaterThan`

Chapter 6

GNU Free Documentation License

Version 1.1, March 2000

59 Temple Place, Suite 330
Boston, MA, 02111-1307
USA

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, `LaTeX` input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
3. State on the Title page the name of the publisher of the Modified Version, as the publisher.
4. Preserve all the copyright notices of the Document.
5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
8. Include an unaltered copy of this License.
9. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
11. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
13. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
14. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above

for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C) YEAR  YOUR NAME. Permission is
granted to copy, distribute and/or modify this
document under the terms of the GNU Free
Documentation License, Version 1.1 or any later
version published by the Free Software Foundation;
with the Invariant Sections being LIST THEIR
TITLES, with the Front-Cover Texts being LIST, and
with the Back-Cover Texts being LIST. A copy of
the license is included in the section entitled
``GNU Free Documentation License''.
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.