

Example Programs

[Example Programs](#)
[Hello, World](#)
[Sorting](#)
[What to Do?](#)
[Run the Demo Programs](#)
[Edit Sample Files](#)
[Benchmark](#)
[Read the Manual](#)
[Visit the EuForum](#)
[Trace a Demo](#)
[Run the Tutorial Programs](#)
[Modify the Demo and Tutorial Programs](#)
[Write Your Own](#)
[Creating Euphoria Programs](#)
[Running a Program](#)
[Running Under Windows](#)
[Editing A Program](#)
[Distributing A Program](#)
[Command Line Switches](#)
[Further Notes](#)

Hello, World

The *mandatory* 'Hello World' program is a one-liner in Euphoria.

```
puts(1, "Hello, World\n")
```

The built-in routine puts does the job of displaying text on a screen. It requires two arguments. The first argument, 1, directs the output to STDOUT or the console. The second argument, is a string of text that will be output.

The result is:

```
Hello, World
```

Sorting

The following is an example of a more useful Euphoria program.

```
include std/console.e  
sequence original_list
```

```

function merge_sort(sequence x)
-- put x into ascending order using a recursive merge sort
  integer n, mid
  sequence merged, a, b

  n = length(x)
  if n = 0 or n = 1 then
    return x -- trivial case
  end if

  mid = floor(n/2)
  a = merge_sort(x[1..mid])      -- sort first half of x
  b = merge_sort(x[mid+1..n])    -- sort second half of x

  -- merge the two sorted halves into one
  merged = {}
  while length(a) > 0 and length(b) > 0 do
    if compare(a[1], b[1]) < 0 then
      merged = append(merged, a[1])
      a = a[2..length(a)]
    else
      merged = append(merged, b[1])
      b = b[2..length(b)]
    end if
  end while
  return merged & a & b -- merged data plus leftovers
end function

procedure print_sorted_list()
-- generate sorted_list from original_list
  sequence sorted_list

  original_list = {19, 10, 23, 41, 84, 55, 98, 67, 76, 32}
  sorted_list = merge_sort(original_list)
  for i = 1 to length(sorted_list) do
    display("Number [] was at position [:2], now at [:2]",
      {sorted_list[i], find(sorted_list[i], original_list), i}
    )
  end for
end procedure

print_sorted_list() -- this command starts the program

```

The above example contains a number of statements that are processed in order. include std/console.e This tells Euphoria that this application needs access to the public symbols declared in the file 'std/console.e'. This is referred to as a *library* file. In our case here, the application will be using the display routine from console.e. sequence original_list This

declares a variable that is not public but is accessible from anywhere in this file. The datatype for the variable is a sequence, which is a variable-length "array," and whose symbol name is `original_list`. `function merge_sort(sequence x) ... end function` This declares and defines a function routine. Functions return values when called. This function must be passed a single parameter when called -- a sequence. `procedure print_sorted_list() ... end procedure` This declares and defines a procedure routine. Procedures never return values when called. This procedure must not be passed any parameters when called. `print_sorted_list` This calls the routine called `print_sorted_list`. The output from the program will be:

```
Number 10 was at position 2, now at 1
Number 19 was at position 1, now at 2
Number 23 was at position 3, now at 3
Number 32 was at position 10, now at 4
Number 41 was at position 4, now at 5
Number 55 was at position 6, now at 6
Number 67 was at position 8, now at 7
Number 76 was at position 9, now at 8
Number 84 was at position 5, now at 9
Number 98 was at position 7, now at 10
```

Note that `merge_sort` will just as easily sort any list of data items:

```
{1.5, -9, 1e6, 100}
{"oranges", "apples", "bananas"}
```

This example is stored as `euphoria\tutorial\example.ex`. This is not the fastest way to sort in Euphoria. Go to the `euphoria\demo` directory and type

```
eui allsorts
```

to compare timings on several different sorting algorithms for increasing numbers of objects. For a quick tutorial example of Euphoria programming, see `euphoria\demo\bench\filesort.ex`.

What to Do?

Now that you have installed Euphoria, here are some things you can try:

Run the Demo Programs

Run each of the demo programs in the demo directory. You just type `eui <program name>`. An example of running the demos in a console

```
eui buzz
```

You can also double-click on a `.ex` or `.exw` file from *Windows* as file associations have been setup during the installation process.

Edit Sample Files

You can use any plain text editor for writing Euphoria source-code.

Code-editors provide a syntax highlighter. The wiki lists many editors suitable for Euphoria.

Check the wiki for [more information about Euphoria editors](#).

You can examine the source-code for the demonstration editor `edx` to see how editors and highlighters work. Notice the use of colors. You can adjust these colors along with the cursor size and many other "user-modifiable" parameters by editing constant declarations in `edx.ex`. Use *Esc q* to quit the editor or *Esc h* for help.

Benchmark

Create some new benchmark tests. See `demo\bench`. Do you get the same speed ratios as we did in comparison with other popular languages? Report your findings on [the forum](#).

Read the Manual

Read the manual in `html\index.html` by double-clicking it. If you have a specific question, type at the console:

```
guru word
```

The `guru` program will search all the `.doc` files, example programs, and other files, and will present you with a *sorted* list of the most relevant chunks of text that might answer your enquiry.

Visit the EuForum

[Euphoria Discussion Forum](#)

Trace a Demo

Try running a Euphoria program with tracing turned on. Add:

```
with trace
trace(1)
```

at the beginning of any Euphoria source file.

Run the Tutorial Programs

Run some of the tutorial programs in `euphoria\tutorial`.
(Read `euphoria\tutorial\tutorial.txt` for more information on the available tutorial files.)

Modify the Demo and Tutorial Programs

Try modifying some of the demo and tutorial programs.
First some *simple* modifications (takes less than a minute):

Simple

What if there were 100 C++ ships in **Language Wars** (`demo\langwar\lw.ex`)? What if `sb.ex` had to move 1000 balls instead of 125? Change some parameters in `polygon.ex`. Can you get prettier pictures to appear? Add some funny phrases to `buzz.ex`.

Harder

Then, some slightly harder ones (takes a few minutes):
Define a new function of x and y in plot3d.ex.

Challenging

Then a challenging one (takes an hour or more):
Set up your own customized database by defining the fields in mydata.ex.

Major

Then a major project (several days or weeks):
Write a *smarter* 3D TicTacToe algorithm.

Write Your Own

Try writing your own program in Euphoria. A program can be as simple as:
? 2+2

Remember that after any error you can simply type: edx to jump into the editor at the offending file and line.

Once you get used to it, you'll be developing programs *much* faster in Euphoria than you could in Perl, Java, C/C++ or any other language that we are aware of.

Creating Euphoria Programs

[Example Programs](#)

[Hello, World](#)

[Sorting](#)

[What to Do?](#)

[Run the Demo Programs](#)

[Edit Sample Files](#)

[Benchmark](#)

[Read the Manual](#)

[Visit the EuForum](#)

[Trace a Demo](#)

[Run the Tutorial Programs](#)

[Modify the Demo and Tutorial Programs](#)

[Write Your Own](#)

[Creating Euphoria Programs](#)

[Running a Program](#)

[Running Under Windows](#)

[Editing A Program](#)

[Distributing A Program](#)
[Command Line Switches](#)
[Further Notes](#)

Euphoria programs can be written with *any* plain text editor. As a convenience, Euphoria comes with `edx`, an editor written in Euphoria, that is handy for editing and executing Euphoria programs. Take a look at `\euphoria\demo` and `euphoria\tutorial` to see many example programs.

Running a Program

To run a Euphoria program, you type the name of the interpreter (`eui`) followed by the filename of the program you want to run. For example:

```
eui example.
```

What you just typed is known as the *command-line*.

Depending on the platform you are using the interpreter could be called:

Executable	Purpose
<code>eui</code>	General interpreter on <i>Windows</i> and <i>Unix</i> variants
<code>euiw</code>	Console-less <i>Windows</i> interpreter

The command-line may contain extra information beside the interpreter and filename. You can add what are called arguments to the command-line that are then passed to your program to customize its behavior. These arguments are read within your program by the built-in function [command_line](#).

Optionally, you may also use [command line switches](#) that are typed between the interpreter name and the program name. Command line switches customize how the interpreter itself behaves.

Unlike many other compilers and interpreters, there is no obligation for any special command-line options for `eui` or `euiw`. Only the name of you Euphoria file is expected, and if you do not supply it, Euphoria will display all the command line options available.

Euphoria doesn't care about your choice of file extensions. By convention, however, console-based applications have an extension of `.ex`, GUI-based applications have an extension of `.exw` and include files have an extension of `.e`. Note that a GUI application is not necessarily a *Windows* program. A GUI application can exist on Linux, OS X, FreeBSD, and so on.

You can redirect standard input and standard output when you run a Euphoria program, for example:

```
eui filesort.
```

or simply,

```
eui filesort < raw.
```

For frequently-used programs under *Windows*, you might want to make a small `.bat` (batch) file, perhaps called `myprog.bat`, containing two statements like:

```
@echo off  
eui myprog.
```

The first statement turns off echoing of commands to the screen. The second runs `eui`

myprog.ex with up to 9 command-line arguments. See [command_line](#) for an example of how to read these arguments. Having a .bat file will save you the minor inconvenience of typing eui all the time; for example you can just type:

```
myprog
```

instead of:

```
eui myprog
```

Under modern *Unix* variants, you can use `#!/usr/bin/env eui` as the first line of your script file. On older *Unix* variants, you may need to use the full path to eui, `#!/usr/local/bin/eui`.

If your program is called foo.ex:

```
#!/usr/bin/env eui
```

```
procedure foo()  
  ? 2+2  
end procedure
```

```
foo()
```

Then if you make your file executable:

```
chmod +x foo.
```

You can just type:

```
foo.
```

to run your program. You could even shorten the name to simply "foo". Euphoria ignores the first line when it starts with `#!`. Be careful though that your first line ends with the *Unix*-style `\n`, and not the *Windows*-style `\r\n`, or the unix shell might get confused. If your file is shrouded, you must give the path to eub, not eui.

You can also run bind to combine your Euphoria program with the eui interpreter, to make a stand-alone executable file. With a stand-alone executable, you *can* redirect standard input and output. Binding is discussed further in [Distributing a Program](#) and [Shrouding and Binding](#).

Using the [Euphoria To C Translator](#), you can also make a stand-alone executable file, and it will normally run much faster than a bound program.

Running Under Windows

You can run Euphoria programs directly from the *Windows* environment, or from a console shell that you have opened from *Windows*. By "associating" .ex files with eui.exe and .exw files with euiw.exe. You will then be able to double click a Euphoria source file to run it. The installer will perform this operation for you, if you wish.

Editing A Program

You can use any text editor to edit a Euphoria program. However, Euphoria comes with its own special editor that is written entirely in Euphoria. Type: edx followed by the complete name of the file you wish to edit. You can use this editor to edit any kind of text file. When you edit a

Euphoria file, some extra features, such as color syntax highlighting and auto-completion of certain statements, are available to make your job easier.

Whenever you run a Euphoria program and get an error message during compilation or execution, you can simply type `edx` with no file name and you will be automatically positioned in the file containing the error, at the correct line and column, and with the error message displayed at the top of the screen.

Under *Windows* you can associate `edx.bat` with various kinds of text files that you want to edit. Color syntax highlighting is provided for `.ex`, `.exw`, `.exd`, `.e` and `.pro` ([profile files](#)).

Most keys that you type are inserted into the file at the cursor position. Hit the Esc key once to get a menu bar of special commands. The arrow keys, and the Insert/Delete/Home/End/PageUp/PageDown keys are also active. Under *Linux/FreeBSD* some keys may not be available, and alternate keys are provided. See [Edx Demonstration Code Editor](#) for a complete description of the editing commands.

If you need to understand or modify any detail of the editor's operation, you can edit the file `edx.ex` in `euphoria\bin` (be sure to make a backup copy so you don't lose your ability to edit). If the name `edx` conflicts with some other command on your system, simply rename the file `euphoria\bin\edx.bat` to something else. Because this editor is written in Euphoria, it is remarkably concise and easy to understand. The same functionality implemented in a language like C, would take far more lines of code.

`edx` is a simple text-mode editor that runs on all platforms and is distributed with Euphoria. There is a list of other editors at the [OpenEuphoria web site](#), many of which include extra features such as syntax highlighting.

Distributing A Program

Euphoria provides you with 4 distinct ways of distributing a program.

- Interpreted
- Binding
- Shrouding
- Translate/Compile

The Interpreted method requires you to simply ship your users the Euphoria interpreter along with your Euphoria program source files, including any Euphoria includes that may be necessary from the `euphoria/include` directory. If the Euphoria source files and the interpreter are placed together in one directory, then your user can run your program by typing `eui` followed by the path of your main executable source file. You might also provide a small `.bat` file so people will not actually have to type the interpreter name. This way assumes that you are willing to share your Euphoria source code with your users.

Binding and Shrouding are two additional ways of distributing your program. You can shroud your program, or you can bind your program.

Shrouding combines all of the Euphoria source code that your program needs to create a single `.il` file.

Binding combines your shrouded program with the Euphoria backend (`eub` or `eubw` on Windows) to create a single, stand-alone executable file.

For example, if your program is called `"myprog.ex"` you can create `"myprog.exe,"` which will run identically to the interpreted version. For more information about shrouding and binding, see [Shrouding and Binding](#).

Finally, with the [Euphoria To C Translator](#), you can translate your Euphoria program into C and

then compile it with a C compiler to get an executable program.

Command Line Switches

[Example Programs](#)

[Hello, World](#)

[Sorting](#)

[What to Do?](#)

[Run the Demo Programs](#)

[Edit Sample Files](#)

[Benchmark](#)

[Read the Manual](#)

[Visit the EuForum](#)

[Trace a Demo](#)

[Run the Tutorial Programs](#)

[Modify the Demo and Tutorial Programs](#)

[Write Your Own](#)

[Creating Euphoria Programs](#)

[Running a Program](#)

[Running Under Windows](#)

[Editing A Program](#)

[Distributing A Program](#)

[Command Line Switches](#)

[Further Notes](#)

You can launch Euphoria with some extra command line switches in order to add or change configuration elements. When running a GUI, there is always some way to open a prompt and enter any text with options, arguments and whatever the program being launched may need for proper, expected operation. Under *Windows*, this is achieved by clicking the Start button and selecting Run..., or hitting Windows-R.

Command line switches may be changed or added, one at a time.

In the table below, (all) indicates that the given switch applies to the Interpreter, Translator and Binder. Use of (interpreter), (translator) and/or (binder) indicates that the referenced switch applies only to that execution mode.

-BATCH (all) Executes the program but if any error occurs, the "Press Enter" prompt is not presented. The exit code will be set to 1 on error, 0 on success. This option can also be set via the [with batch](#) directive. -COM dir (translator) Specifies the include directory for the C compiler once EUPHORIA code is translated. This should be set such that dir/include/euphoria.h exists. -COPYRIGHT (all) Displays the copyright banner for euphoria. -C config_file (all) Specifies either a file name or the path for where the default file called eu.cfg exists. The configuration file which holds a set of additional command line switches. See Also [Configuration file format](#) -CON (translator) Windows only. Specifies that the translated program should be a console

application. The default is to build a windowed application. -D word (all) Defines a word as being set. Words are processed by the [ifdef statement](#). Words can also be defined via the [with / without define](#) directive. -DEBUG (translator) Enable debug mode for the generated code. (interpreter) An external debugger translated into a .DLL or .so library to be used instead of the built-in debugger. -DLL, -SO (translator) Compiles and links the translated euphoria code into a DLL, SO or DYLIB (depending on the platform). -EUDIR dir (all) This overrides the environment variable EUDIR. -H, (all) Displays the list of available command line options. -I include_path (all) Specifies an extra include path. -LIB file (translator) Specifies the run-time library to use when translating euphoria programs. -LIB-PIC file (translator) Specifies the run-time library to use when translating euphoria programs as shared objects. The library should be built using the -fPIC (position independent code) flag. This is meant to be used in a eu.cfg file to be able to specify both a non-PIC (using the -lib option) and a PIC option in the same eu.cfg file. -PLAT word (translator) Specify the target platform for translation. This allows euphoria code to be translated for any supported platform from any other supported platform. Supported platforms: FREEBSD, LINUX, NETBSD, OPENBSD, OSX, WINDOWS -STRICT (all) This turns on all warnings, overriding any with/without warning statement found in the source. This option can also be set via the [with/without warning](#) directive. -TEST (all) Parses the code only and issues any warnings or errors to STDOUT. On error the exit code will be 1, otherwise 0. If an error was found, the normal "Press Enter" prompt will not be presented when using the -TEST parameter which enables many editor/IDE programs to test the syntax of your Euphoria source in real time. -TRACE-LINES n (all) Changes the number of lines that will be used in ctrace.out for lines traced under trace(3). The default is 500. -VERSION (all) Displays the version of euphoria that is running. -W warning_name (all) Resets, or adds to, the current list of warnings that may be emitted. The list of known names is to be found in the subsection [with/without warning](#). A name should appear without quotes. If the warning_name begins with a plus symbol '+', this warning is added to the current set of warnings checked for, otherwise the first usage resets the list to the warning being introduced, and each subsequent -W warning_name adds to the list. -WF file_name (all) Sets the file where the warnings should go instead of the standard error. Warnings are written to that file regardless of whether or not there are errors in the source. If there are no warnings, the -wf file is not created. If the -wf file cannot be created, a suitable message is displayed on STDERR and written to ex.err. -X; Resets, or adds to, the list of warnings that will not be issued. This is opposite of the -W switch. The case of the switches is ignored, so -I and -i are equivalent.

Further Notes

- Included files are searched for in all included paths, in the following order:
 1. The current path
 2. Paths specified in a -I command line switch, which can also come from any configuration files found.
 3. Paths listed in the EUINC environment variable, in the order in which they appear
 4. Paths listed in the EUDIR environment variable, in the order in which they appear
 5. The interpreter's path

