

# Clean System User Manual

©1987-2001

D.F.R. van Arkel

December 19, 2001

Clean is a general purpose, higher order, pure and lazy functional programming language that can be used for the development of real world applications. The Clean System is a special programming environment for Clean.

## Contents

<b>1 Availability</b>	<b>2 17 The Clean Compiler</b>	<b>15</b>
<b>2 More information on the language Clean</b>	<b>2 18 The Code Generator and Linker</b>	<b>15</b>
<b>3 The Clean System</b>	<b>3 19 The File Menu</b>	<b>16</b>
<b>4 Installation</b>	<b>3 20 The Edit Menu</b>	<b>17</b>
<b>5 Uninstalling</b>	<b>3 21 The Search Menu</b>	<b>18</b>
<b>6 Quick Start</b>	<b>3 22 The Project Menu</b>	<b>19</b>
<b>7 Projects</b>	<b>7 23 The Module Menu</b>	<b>20</b>
<b>8 Modules</b>	<b>7 24 The Defaults Menu</b>	<b>20</b>
<b>9 Files</b>	<b>8 25 The Environment Menu</b>	<b>21</b>
<b>10 Windows</b>	<b>8 26 The Help Menu</b>	<b>21</b>
<b>11 Paths</b>	<b>8 27 The Window Menu</b>	<b>22</b>
<b>12 IDE Options</b>	<b>9 A Keymappings</b>	<b>22</b>
<b>13 Environment Options</b>	<b>9 B Command shortcuts</b>	<b>23</b>
<b>14 Project Options</b>	<b>11 C Regular Expressions</b>	<b>23</b>
<b>15 Module Options</b>	<b>14 D Bugs</b>	<b>24</b>
<b>16 The Text Editor</b>	<b>15 E Authors &amp; Credits</b>	<b>24</b>

## Copyright Notice and Disclaimer

Clean and the Clean System ©1987-2001  
Computing Science Institute  
University of Nijmegen  
Nijmegen - The Netherlands  
and  
HILT - High Level Software Tools B.V.  
Nijmegen - The Netherlands  
<mailto:clean@cs.kun.nl>  
We do not make any warranty, either express or implied, with respect to this software.

## 1 Availability

Clean and the Clean System can be used free of charge. They can be obtained

- via the World Wide Web: <http://www.cs.kun.nl/~clean>
- via ftp: <ftp://ftp.cs.kun.nl/pub/Clean>

Clean is available for several platforms. Please check the Clean website for the latest news. If you have any questions, bugs to report, or when you have wishes or comments please send an electronic mail to <mailto:clean@cs.kun.nl>.

## 2 More information on the language Clean

A description of the syntax and semantics of Clean can be found in the following manual which is included in the distribution:

Concurrent Clean Language Report  
Rinus Plasmeijer and Marko van Eekelen  
University of Nijmegen

The ideas behind Clean and its implementation on sequential and parallel machine architectures are explained in detail in the following textbook:

Functional Programming and Parallel Graph Rewriting  
Rinus Plasmeijer and Marko van Eekelen  
Addison-Wesley Publishing Company  
©1993  
ISBN 0-201-41663-8

An introduction to functional programming in Clean for undergraduate students is also available. Special about the book is that it contains non trivial applications such that the book will also be very useful for people who want to develop serious programs:

Functional Programming in Clean  
Paul de Mast (Polytechnical University Brabant)  
Jan-Marten Jansen and Dick Bruin (Polytechnical University Leeuwarden)  
Jeroen Fokker (University of Utrecht)  
Pieter Koopman, Sjaak Smetsers, Marko van Eekelen and Rinus Plasmeijer (all University of Nijmegen)

The books and the latest research papers on Clean can be found via the Clean home page at <http://www.cs.kun.nl/~clean>.

### 3 The Clean System

The Clean System is a programming environment for Clean written in Clean itself. The current version of the system consist of:

- The Project Manager: an automatic make facility for Clean projects
- The Text Editor: a text editor with Clean sensitive search facilities and syntax colouring
- The Clean Compiler: compiler for the platform on which the system is running
- The Code Generator: code generator for the platform on which the system is running
- The Linker: generates stand-alone (Mac or Windows) applications
- Libraries for arithmetic, I/O handling, utility modules for interfacing
- The Time Profiler: for viewing time profiles
- The Heap Profiler: for viewing heap profiles
- Sparkle: a theorem prover for Clean programs, see <http://www.cs.kun.nl/Sparkle>

The project manager and editor are integrated into the Clean Integrated Development Environment (IDE). The remaining tools are separate applications automatically launched by the Clean Ide when this is required.

### 4 Installation

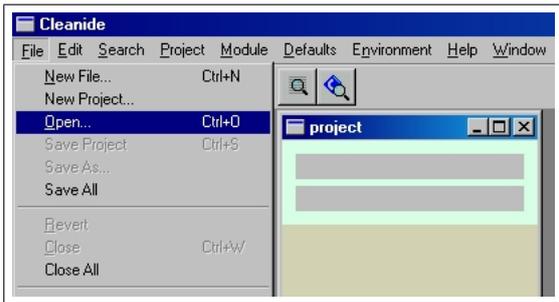
- Make a backup of your old Clean programs, Clean system and I/O libraries.
- Unpack the Clean archive to a suitable directory on your hard disk (i.e. `C:\Clean\`).
- The Clean System generates native applications. If you have virus detectors they may complain about this. Clear the protection for the Clean compiler (`cocl.exe`) and editor (`CleanIDE.exe`). They should be allowed to create, edit and delete files . Make sure that you have an original unaffected Nijmegen version of the Clean System (see [Availability](#), p.2).
- The first time you launch the Clean IDE it will ask if you want to allow it to install the Clean IDE into the Windows registry. This binds the various file types used by the Clean System so that they have nice icons and so that double-clicking them will launch the appropriate tool.

### 5 Uninstalling

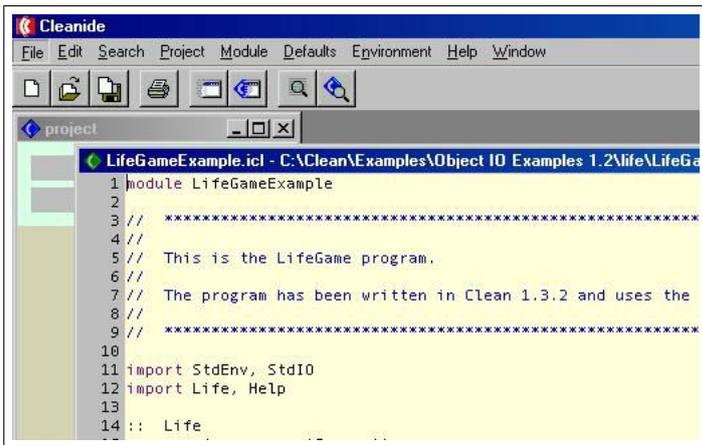
To uninstall Clean: if you allowed the Clean IDE to enter its filetypes and settings into the Windows registry you might want to remove them again. Choose “IDE Options...” from the Defaults menu and press the “Clear Registry” button, then quit the IDE. You can now remove Clean by moving the Clean folder to the Trash.

### 6 Quick Start

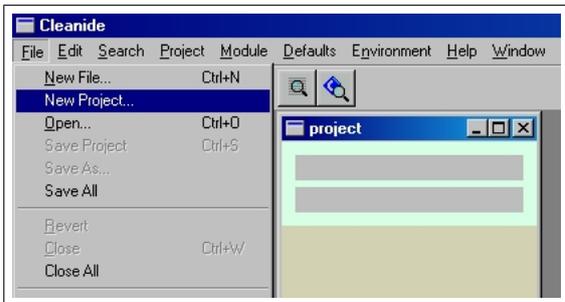
- Start the Clean IDE (`CleanIDE.exe`). You will see an empty project window. First we need to make a new project or open an existing project. A Clean project consists of definition modules (`.dcl` files) and implementation modules (`.icl` files). A definition module indicates which definitions defined in the corresponding implementation module are exported. The topmost root module (the main module) is always an implementation module.



- Open the main module of the “Game of Life”, `LifeGameExample.icl` (in the `Examples\ObjectIO Examples\Life\` directory). The source text of this Clean implementation module will be displayed in a text window and can be modified with the built-in editor.

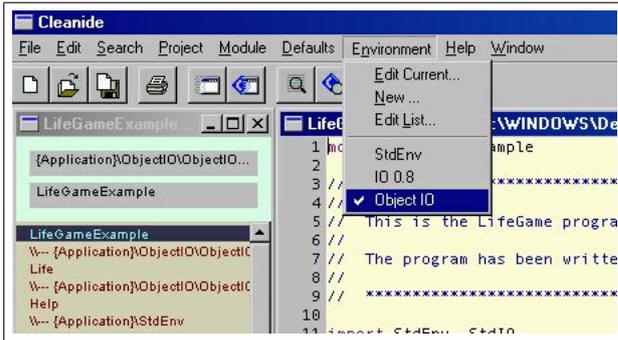


- We now make a new Clean project which will have `LifeGameExample` as its main module by selecting `New Project` from the `File` menu. This presents a dialogue allowing you to set the project name and location. It is easiest to simply use the suggested defaults. As the examples come with pre-setup projects the IDE will ask whether you want to replace the existing version. Selecting `yes` is safe as you will exactly recreate the existing project in the following steps.

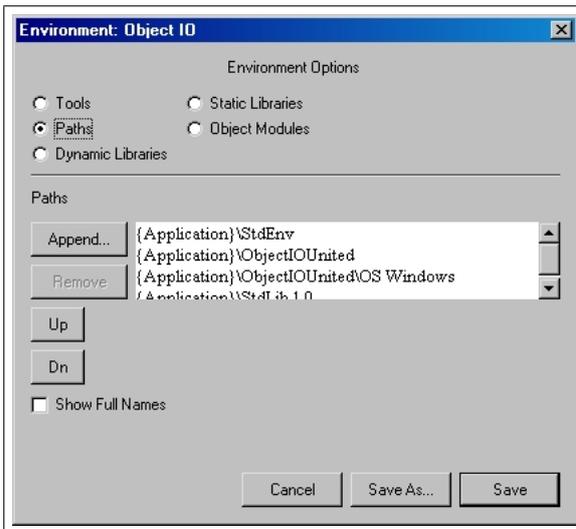


The Clean IDE now knows where to save the project administration and the name of the main module (`LifeGameExample`) will appear in the project window.

- The `LifeGameExample` uses other Clean modules, namely modules from the `StdEnv` library and modules from the `Object I/O` library. Each project has an environment in which it is specified which libraries are being used in the project.

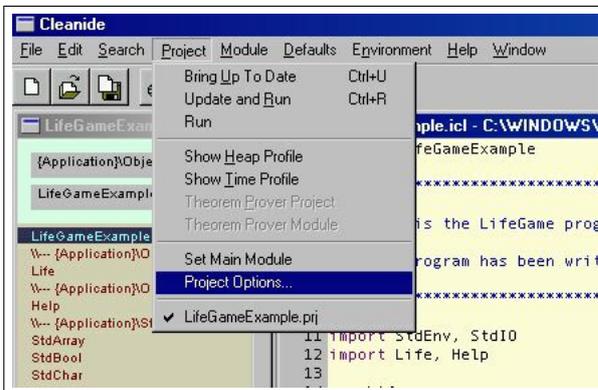


The Clean IDE has a number of popular environments predefined. You have to choose which one you would like to use in this particular project. From the Environment menu choose Object IO. Now the IDE knows which libraries to use in this project.



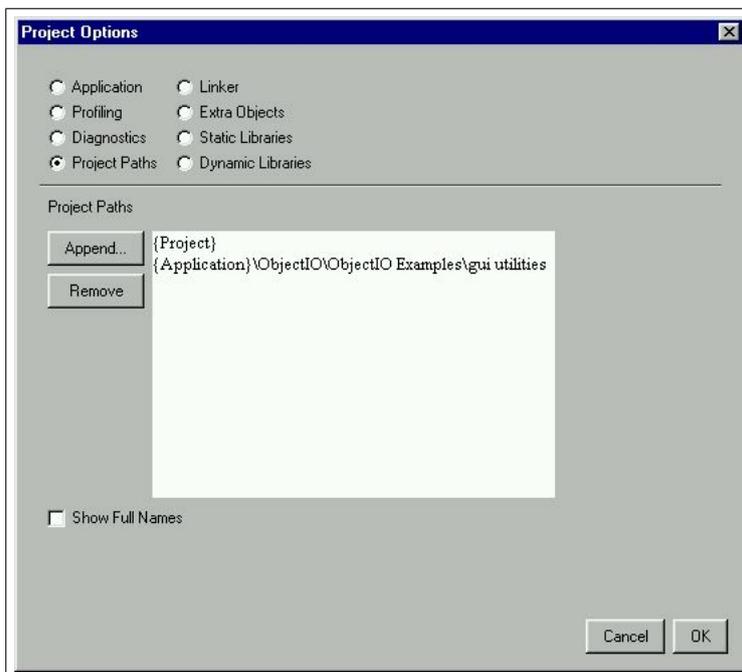
By the way, one can define new environments (“New Environment...”) or change existing ones (“Edit Current...”). One can set the libraries to be used, by adding the paths of the folders in which the corresponding Clean source modules can be found. In the environments one can also include “real” libraries containing object code (Static Libraries and Dynamic Libraries). One can even define which IDE components (Tools) to use (which Clean compiler, e.g. version 1.3 or version 2.0, which linker, and so on). In this way you can easily assign your own preferences to a project or switch between settings just by choosing another environment setting. The environments will be remembered by the IDE, the chosen environment will be remembered in the project administration file.

In addition to the environment you might also want to use more project specific libraries which you might have created to structure your projects, but, which are too specific to put in an environment. For example, the Life Game has a help facility. The corresponding Clean modules are put in a separate folder to be shared by other examples.

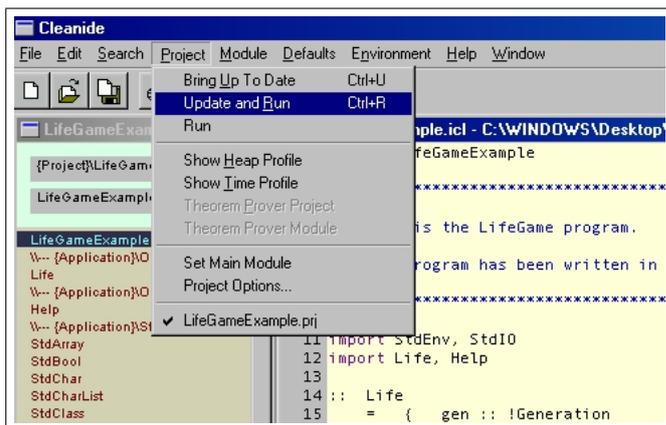


- Choose “Project Options...” from the “Project” menu, select the “Paths” panel and add the library

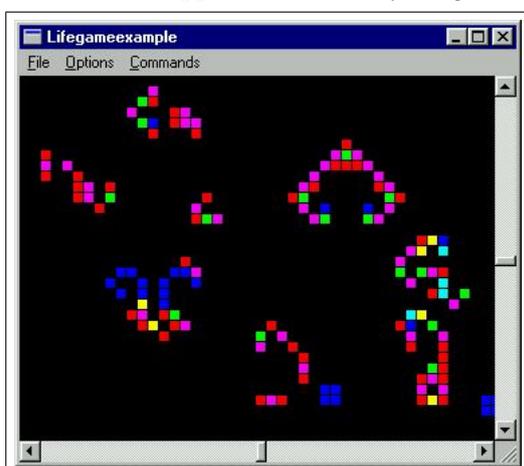
ObjectIO Examples\gui utilities to the Life Game project paths.



Now all libraries used by the “LifeGameExample” are included and we can generate the application. Of course, next time, when you open the Life Game project, everything is remembered.



- From the “Project” menu choose “Update and Run” . All Clean modules used by “LifeGameExample” will appear in the project window. Those modules which are not up to date will be compiled to abc code. If no errors are found, the code generator will translate all abc-files to object code. Finally, the linker will build the application. If everything is fine, the Clean IDE will launch the application automatically.



Building the application might not succeed when you have a virus detector active (see [Installation](#), p.3). Launching will not succeed when the generated application does not have enough memory to run. You can change the amount of allocated memory in the Project Options dialogue (see [Project Options](#), p.11).

## 7 Projects

The project manager in the Clean IDE checks and controls the consistency of all the files, windows and options of the system such that the creation and execution of Clean applications can be invoked with only one command.

All information about a project is remembered such that a second compilation goes much faster than the first one. The information about a project is stored in a file (`name.prj`) such that projects can be continued in the same status they are left.

A Clean program consists of modules that can be compiled separately. There are implementation modules (`name.icl`) and definition modules (`name.dcl`). Only implementation modules can be compiled explicitly, the corresponding definition modules are checked automatically.

Modules are hierarchical. The topmost implementation module is variously called the main, start, or root module. It should contain a Start rule (see the Clean Language Report).

The project manager needs to know what the main module is. When this module is known all other modules (and the corresponding intermediate files) in the hierarchy are examined by the manager. All these files together are called a project (see [Files](#), p.8). The default name of the project is the name of the main module. The settings of a particular project are stored in the project file.

The project manager will scan your disk to find the files it needs. In order to restrict the search time the search paths should be set as accurately as possible (see [Paths](#), p.8).

A new project can be created by selecting **New Project...** in the **File** menu. The default name of the project is the name of the module in the active editor window and the default main module is also that module. Any implementation module can be assigned to be the main module via **Set Main Module** in the **Project** menu.

A window can be opened on a file containing a Clean module. The contents can be changed with the text editor (see [The Text Editor](#), p.15). When you generate an application, the project manager may warn you when windows and files have different contents. The Clean compiler always uses the version on disk, so don't forget to save your changes before you bring your project up-to-date.

## 8 Modules

### Implementation modules

An implementation module (`name.icl`) consists of a sequence of type, function, graph, class, macro and other definitions. These definitions have in principle only a local scope: they only have meaning in the implementation module they are defined in. In the corresponding definition module one can specify which types and rules (defined in the corresponding implementation module) are exported, i.e. are made visible to the outside world.

### The main module

The main module is always an implementation module. In its module header the keyword **implementation** can be left out. In that case the main module does not need a corresponding definition module. Furthermore, a main module should always contain an explicit or implicit declaration of the Start rule, i.e. this rule should either be defined in the main module or imported from another module. Any implementation module can become a main module by inserting a Start rule. This is helpful for the debugging of modules.

### Definition modules

In a definition module (`name.dcl`) one can specify which definitions (defined in the corresponding implementation module) are exported, i.e. are made visible to the outside world. All other definitions given in the implementation module remain hidden.

## System modules

System modules are special definition modules indicated by the keyword **system** instead of **definition**. This indicates that the corresponding implementation modules are special system modules. These modules are predefined and can only be made by system programmers.

## 9 Files

Files used by the Clean System have certain extensions that indicate their use. The following kinds of files are recognized by the Clean System:

**name.icl** A Clean implementation module with the name 'name'.

**name.dcl** A Clean definition module. It is assumed that a `.icl` file and its corresponding `.dcl` file are always kept in one and the same folder.

**name.abc** An abc file generated by the Clean compiler.

**name.o** An object file generated by the Clean code generator.

**name.s** An assembly file generated by the Clean code generator.

The generated abc and object files are stored in the Clean System Files folder that will be generated in the folder where the corresponding `.icl` (and `.dcl`) files are stored.

**name.prj** The project file.

**name.exe** An application generated by the linker. The generated application will be stored in the location given in the Project Options dialogue reachable from the **Project** menu. By default it uses the same name as the project with `.exe` extension and is stored in the same folder as the project file.

The `.icl`, `.dcl`, `.abc`, `.s` files are text files which can be edited with the Clean editor.

## 10 Windows

Several kinds of windows can appear on your screen.

**.icl** An editor window on an implementation module.

**.dcl** An editor window on a definition module.

**Types** A window that displays derived and checked type information as well as the derived and assumed strictness information.

**Errors & Warnings** A window that displays warnings and error messages generated by the system.

**Project** A window that displays information on the current project.

**Search Results** A window displaying the results of the last search command.

**Clipboard** A window displaying the current contents of the clipboard.

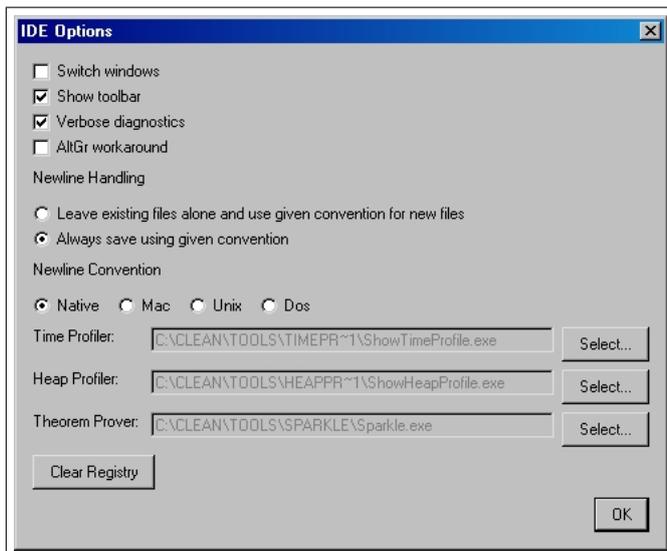
**Console** A window showing the input and output of a Clean console application (see [Environment Options](#), p.9).

Any other text file can be opened in an editor window as well.

## 11 Paths

In a number of locations paths can be set for the IDE. These determine where the project manager looks for Clean modules and intermediate files. For optimal performance these should reflect as accurately as possible where modules can be found. In general the project manager first searches the paths given in the **Project Options** dialogue in the order that they are listed. If the module is not found there then the Environment paths of the current environment are searched.

## 12 IDE Options



The IDE has a number of global options which can be set from the **IDE Options** dialogue available from the **Defaults** menu.

**Switch windows** This setting influences the way project switches are handled. If it is on then when switching between projects the IDE closes the windows belonging to the old project and remembers their settings and then opens the windows belonging to the new project. If this setting is off then all open windows remain open and no new ones are opened.

**Show toolbar** This setting determines whether or not the IDE toolbar is displayed. If it is on and the Clean IDE can find its bitmaps in the bitmaps folder then a toolbar is displayed the next time the IDE starts up.

**Verbose diagnostics** This activates more verbose diagnostics from the Clean project manager when updating a project. If you are having difficulties convincing a project to build this can be a useful option to activate.

**AltGr workaround** This activates a dirty hack for some users with non-english keyboards. It disables all menu shortcuts using symbols instead of letters. French and German users might want to activate this if they are currently having trouble accessing special symbols on their keyboard. It only has effect when launching the IDE so to see the result you need to change the flag and then quit and restart the IDE.

**Newline Handling / Newline Convention** These settings together determine the newline handling for editor windows. There are two possible behaviours: one is to maintain the existing newline convention for existing files, the other is to always use the preferred newline convention. This preferred newline convention is settable, native refers to the platform native newline convention e.g. Dos when using Windows.

**Time Profiler/Heap Profiler/Theorem Prover** These settings allow customisation of the profiler and prover locations as new versions appear. Note that this also automatically adjusts the Windows registry settings for these tools so that double-clicking on profile files activates the correct version of the profiler. Also note that the current implementation of the file browser dialogue on selecting an application asks whether you want to replace it. Just answer yes, the application will not actually be replaced. This will be fixed in a future version.

**Clear Registry** This button clears all Clean entries from the Windows registry. Use this to remove all traces of Clean from your system.

## 13 Environment Options

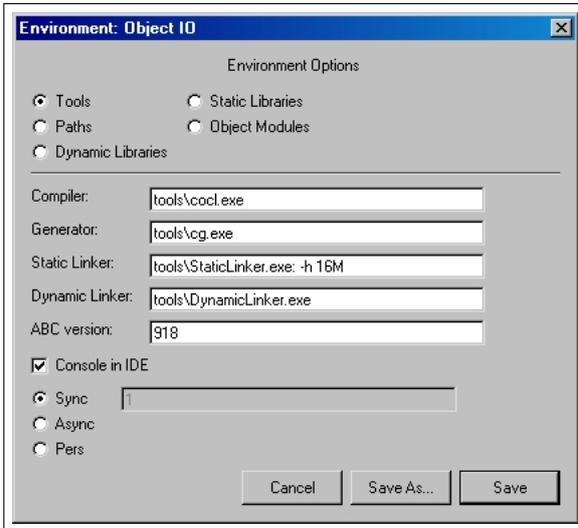
Environments are used to gather commonly used settings. The active environment for a project is the environment that is flagged in the **Environment** menu when that project is active. You can select a new active

environment by choosing it in the **Environment** menu. The environment options are split into a number of groups. These options can be changed by selecting **Edit Current...** from the **Environment** menu.

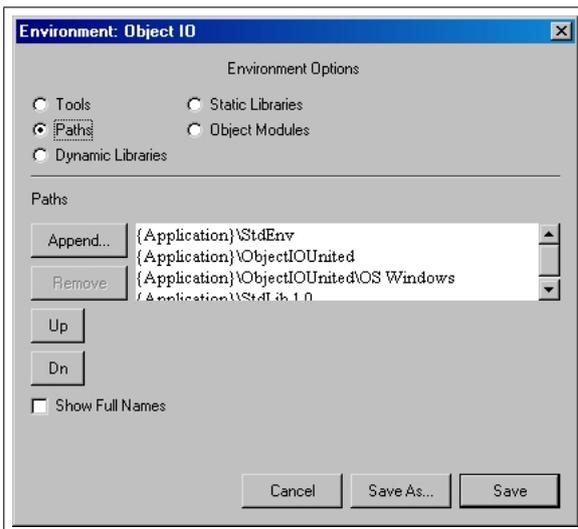
**Tools** Here the compiler, code generator, and linkers can be set that are used within an environment. Also the associated version of the abc-code has to be given so that the Project Manager can check that compatible versions of these tools are being used. Most programmers will only be using a single version of the Clean compiler and will never need to change these settings. For people mixing different versions of the Clean compiler the necessary details will be given separately.

The “Console in IDE” flag activates an experimental feature which redirects the console i/o for a Clean Console application into a special IDE window.

The “Sync/Async/Pers” radiocontrol with associated editbox control the settings for the Clean make facility. These, together with the other tools options, will be subsumed by a system of tool descriptors in a future version of the IDE. For now use ABC version 916 together with the Pers flag when using the Clean 2.0 tools, and ABC version 918 together with the Sync flag when using the Clean 1.3.x tools.



**Paths** Here search paths can be specified which are common to all projects using this environment. See [Paths](#), p.8. Up/Dn allows reordering the paths to influence where modules are found.



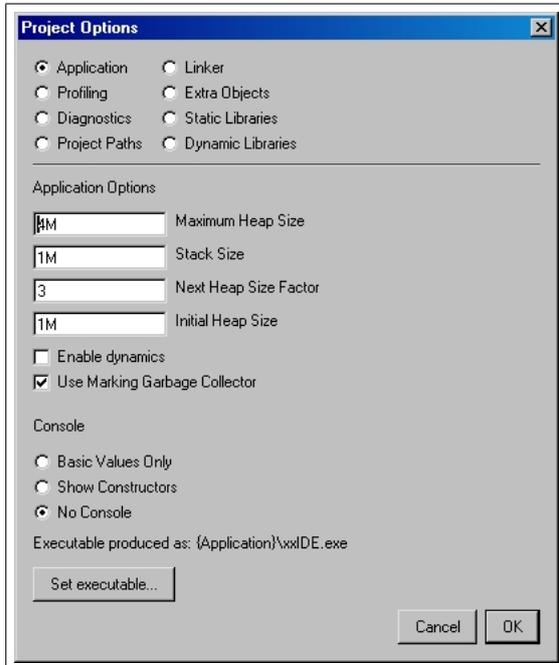
**Dynamic Libraries** Here dynamic libraries can be specified which are not already implicitly included but are required for a project.

**Static Libraries** Here static libraries can be specified, upcoming versions of the Clean System will be able to generate libraries of Clean modules.

**Object Modules** Here extra object modules can be specified. These last three options are generally only needed by Clean wizards.

## 14 Project Options

**Application** In this dialogue the run-time options for an application can be set. The options are used when a new application is generated.



- Application Options

Set the heap space and the maximal amount of space available for the stacks. The stack space is created outside the heap space. Initially not all of the heap is used (the Initial Heap Size). If a garbage collection does not free enough memory, the heap is expanded (with the factor Next Heap Size Factor) up to a maximum (the Maximum Heap Size).

The application will stop with a run-time error message when a heap turns out to be too small during execution. After a stack overflow the program may crash or behave unpredictably.

Increase the space sizes when the application needs more memory. It is also possible that the application requires less memory. For an optimal tuning look at the run-time information that the application can produce (see the section on Profiling options).

- Enable dynamics

Activates support for the new Clean 2.0 language feature of dynamics. Essentially it makes sure that all the tools know that a project using dynamics is being built so that all the extra steps needed to support them are being taken. As dynamics support matures this option will probably develop into its own project type.

- Marking Garbage Collector

When fine-tuning your application performance you can experiment with an alternative garbage collector, the marking garbage collector, again check the available run-time information to see which setting is optimal.

- Console

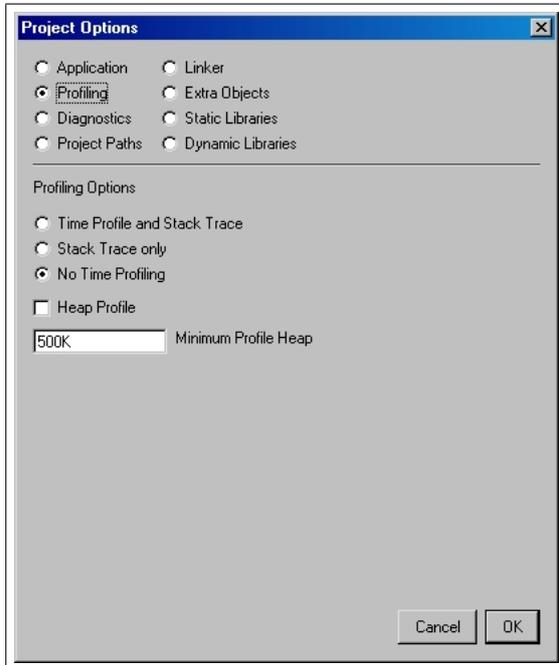
With the Console options one has control over the way the result of the Start function is displayed. Choose Show Constructors and all constructors and brackets will be shown. Choose Basic Values Only if you only want to display basic values only (integers, booleans, strings, characters, reals).

Choose No Console if the I/O of the program is handled via the I/O libraries. When this option is chosen no console (output window) for the program will be provided and the result of the Start function will not be displayed. When the Clean program explicitly tries to write to the console a

console window will be opened. So, when your program terminates unexpectedly probably a run-time error has been detected. Choose one of the other Output options and run the program again to be able to read the error message.

- Executable

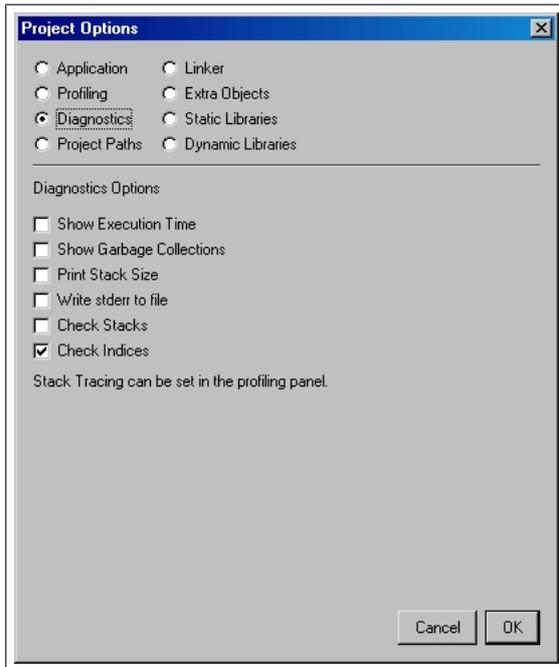
Here you can select a different location for the generated executable. You will know when you want to do this.



**Profiling** The profiling options can be used to measure the performance of your application. With Time profiling enabled, a Clean program will write a profile file after execution. This file (**Profile** on the PowerMac, **ApplicationName Time Profile.pcl** on Windows) can be inspected with the ShowTime-Profile tool. The time profile file is overwritten every time you run the profiled application. For the PowerMac you also have to specify the target processor. You will get a run-time error message if you choose the wrong processor.

Time profiling can also be useful for debugging. With Time Profiling a Clean application that unexpectedly halts will print a stack trace of the functions that lead to the error. This can be used track down stack overflows, out of bounds array indices and 'rule does not match' bugs. If you only want stack traces this can also be selected here.

With Heap profiling the application writes a number of heap files (**ApplicationName Heap Profile0..9**). These files are generated when there is a garbage collection and the heap use is above a certain limit (the Minimum Profile Heap). The heap files can be inspected with the ShowHeapProfile application.



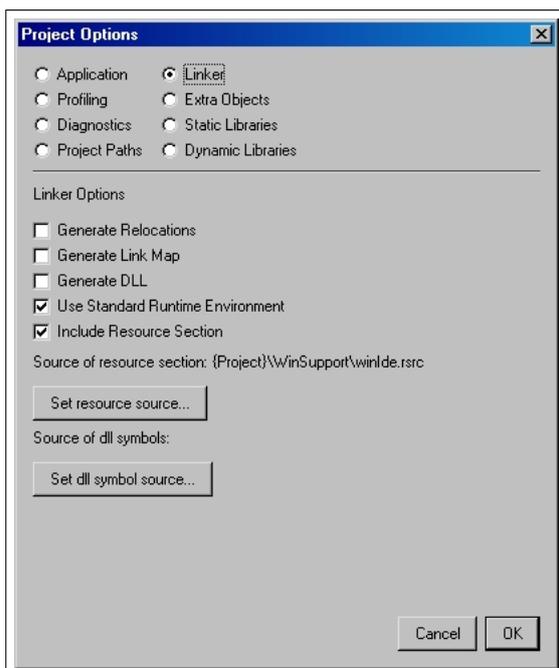
**Diagnostics** An application can produce all kinds of information to make an optimal tuning of the memory size to be assigned to the application possible.

The execution time will be displayed when the application stops and Show Execution Time is chosen.

Each time the garbage collector is invoked this is displayed when Show Garbage Collections is chosen. The run-time system uses both a copying as well as a mark-scan garbage collection algorithm. It automatically switches between these two garbage collection strategies to obtain an optimal time/space behaviour. This option gives you information about how well the heap size was chosen.

In addition, also the sizes of the different stacks are printed when Print Stack Size is chosen. This helps to tune the sizes of the stacks.

When Write stderr to file is enabled, the messages in the errors window are copied to a file with name 'Messages' in the current directory.



**Linker Options** Here you can select the link method and various link options.

**Generate Relocations:** this is an option for generating relocateable code. It provides early support for making Clean DLL's. Only to be used by Clean wizards!

**Generate Link Map:** this writes out a text file `name.map` containing the location of various Clean symbols in the generated application. Used by Clean wizards to analyze low-level application crashes.

**Generate DLL:** another option under development similar to the Generate Relocations option.

**Use Standard Runtime Environment:** an option for turning of the inclusion of the Clean Runtime Environment. Only used by Clean wizards when developing new versions of the runtime environment.

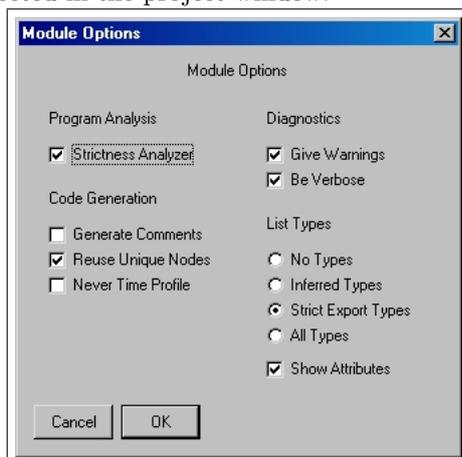
**Include Resource Section:** a currently Windows only option for copying the resource section from another application into the freshly linked application. Used to include icons and whatnot in Windows GUI applications. Note that the current implementation of the file browser dialogue on selecting an application asks wether you want to replace it. Just answer yes, the application will not actually be replaced. This will be fixed in a future version.

**Project Paths** Here the project search paths can be set, see [Paths](#), p.8 and [Environment Options](#), p.9.

**Extra Objects / Dynamic Libraries / Static Libraries** These are for extra items to be included when linking, for more information see [Environment Options](#), p.9.

## 15 Module Options

There are also a number of options which can be set per module. These can be set with the `Module Options` entry in the `Module` menu. This displays the module options dialogue for the active window in the editor. If this entry is selected when the project window is active, then the settings are applied to all modules currently selected in the project window.



**Strictness analyzer** When this option is set the strictness analyser will try to determine whether or not the defined functions are strict in their arguments. Strict arguments will be marked with '!' in the type definition.

In general, much better code will be generated when this option is set. The penalty is that the compilation takes more time.

When you are exporting a function with strict arguments, it is for efficiency reasons better to export the strictness information also. You have to do this by hand. Define the type specification including the strictness information in both the definition and the implementation module.

Strictness annotations defined by the programmer are not checked but taken to be true.

**No Types** No type or strictness information is displayed when this option is chosen.

**Inferred Types** The inferred types of functions that are not typed by the programmer are displayed in the Types window. Strictness information is not displayed.

**Strict Export Types** The types of functions that are exported are displayed including inferred strictness information, but only when the type in the definition module does not contain the same strictness information as was found by the strictness analyser. This way it is easy to check for all functions if all strictness information is exported.

**All types** The types of all functions including strictness information are displayed in the Types window.

**Show Attributes** Check this box if you want uniqueness attributes displayed in the listed types.

**Be Verbose** If set additional information about the compilation process is given. Turn it off to speed up the compilation.

**Give warnings** When this option is set also warnings are produced in the Error window. Turn it on when you are an inexperienced Clean programmer or when you want to be informed about unusual but not erroneous situations.

**Generate comments** Set this option if you want to be able to read the generated .abc file. Turn it off to speed up the compilation process.

**Reuse unique nodes** Enables compile-time garbage collection for certain unique nodes. See section 4.5.7 in the Clean Language Reference manual for more details.

**Never time profile** Don't generate time profiling information for this module, even if time profiling is enabled in the Project Options dialogue.

## 16 The Text Editor

All text files (see [Files](#), p.8) can be edited with the built-in editor using the commands available in the **Edit** and **Search** menus. The editor has an auto indent facility. When this option is on the editor will remember the first non-tab, non-space position at the beginning of the line. When a return is typed the cursor will move to the remembered position in the next (new) line and the needed amount of tabs and spaces will be added. The auto indent option can be switched on and off in the Editor Settings dialogue (see **Editor Settings...** under the **Defaults** menu).

The editor also contains two key and mouse combinations with which definitions of imported Clean symbols (functions, types and constructors) can be found: **Control + Double-Click** on a Clean symbol locates the definition of the symbol in its .dcl file. **Control-Shift + Double-Click** on a Clean symbol locates the definition of the symbol in its .icl file.

The same effect can be achieved by selecting a symbol name and then choosing **Search Definition** (for the .dcl file) or **Search Implementation** (for the .icl file) in the **Search** menu. Warning: Functions that have no type specification may not be found. When your Clean modules have a (very) unusual layout the search algorithm might also not be able to find the definitions of Clean symbols.

## 17 The Clean Compiler

The Clean compiler compiles an implementation module to abc-code. It performs syntactical and semantical checks (see the Clean Language Report). The corresponding definition module is also checked, as well as the consistency between definition and implementation module. Errors are reported in the Errors & Warnings window.

(Uniqueness) type information is inferred and checked. Strictness information is inferred with help of the builtin strictness analyser. On demand, the type and strictness information is displayed in the Types window.

If no errors are found, abc-code is generated. For each .icl module a file .abc will be created in the corresponding Clean System Files folder. This code is input for the Code Generator to build native object files.

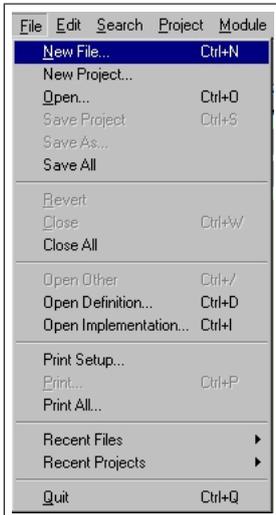
## 18 The Code Generator and Linker

The Code Generator takes sequential abc-code as input (an .abc file) and produces an object file in the Clean System Files folder. The Linker links all object files of the project together and builds an application.

By default, the Code Generator generates code for the machine that you are working on. When you are developing your program you should use the Check Indices option. This will add checks for out-of-bound array operations. After you have convinced yourself that your program doesn't violate this condition, you can remove the checks for optimal speed.

Heap and stack space must be allocated to the application. Also other options can be set (see **Project Options** in the **Project** menu). These settings take effect the next time the application is linked. On demand, the Code Generator can produce assembly files. To produce the assembly code choose **Generate Assembly** in the **Module** menu. The assembly file is generated in the same directory as the Clean `.icl` module with the `.s` extension and is automatically opened in the editor.

## 19 The File Menu



**New File...** A standard New File dialogue is displayed allowing you to specify which file is to be created. A window is opened on this new file which can then be edited with the text editor.

**New Project...** A standard New File dialogue is displayed allowing you to specify where the new project is to be created. Initially the associated main module is set to be the module displayed in the active window when **New Project** was selected. If no module was active the project does not have a main module yet and one needs to be set by activating a module window and selecting **Set Main Module** from the **Project** menu.

**Open...** Open a window on an existing file or project.

**Save** Save the contents of the active (front) window to the corresponding file.

**Save As...** Save the contents of the active (front) window to the indicated file. A standard New File dialogue appears, asking you to specify the file's name and location.

**Save All** This command saves the contents of all open editor windows.

**Revert** Revert to the last saved version of the file corresponding to the active (front) window.

**Close** Closes the active (front) window. A dialogue will appear when a text window is closed with contents that need to be saved.

**Close All** This command closes all open windows.

**Open Other** This command will open the corresponding `.dcl` file if the front window is a `.icl` file or vice versa.

**Open Definition** When a portion of text is selected (for example the text 'selection') the command is enabled and changes to **Open selection.dcl**. When this command is chosen it will try to open the corresponding `.dcl` file. It can be used to quickly open imported modules: just select the name of the file in the **import** statement and choose this command. For an easy way to locate the definitions of imported functions and types see **Find Definition** and **Find Implementation** under the **Search** menu. When no module name is selected, you can specify the module name in a dialogue box.

**Open Implementation** This command works the same as the Open Definition command, but now the implementation module (.ic1 file) will be opened.

**Print Setup** Display the standard Print Setup dialogue for setting default printer options.

**Print...** Display the standard Print dialogue for printing the contents of the currently active editor window.

**Print All** Prints all open editor windows.

**Recent Files** This is a submenu containing the twelve most recently opened files. Selecting one of these reopens the corresponding file if it was closed or otherwise makes it the active window.

**Recent Projects** This is a submenu containing the twelve most recently opened projects. Selecting one of these reopens the corresponding project after saving the currently active project.

**Quit** Quit the Clean Integrated Development Environment.

## 20 The Edit Menu



**Undo** Undo the last change made in the text of the front window.

**Cut** Cuts the text currently selected in the front window to the clipboard.

**Copy** Copies the text currently selected in the front window to the clipboard.

**Paste** Insert the contents of the clipboard at the current cursor location or replace the currently selected text in the front window by the contents of the clipboard.

**Clear** Clear the text currently selected in the front window.

**Shift Left / Shift Right** With these commands you can (un)indent the selected text. *Warning: undo is not implemented for these commands.*

**Balance** This commands selects the smallest balanced piece of text that surrounds the current cursor position or currently selected text. This feature can be used to find corresponding open and close symbols in Clean programs. The following symbols are balanced: Parentheses: '(' and ')' to check expressions, Brackets: '[' and ']' to check lists, Curly braces: '{' and '}' to check annotations or record definitions.

**Select All** Select all text in the front window.

**Detab** Replaces all tabs in the front window by the correct number of spaces.

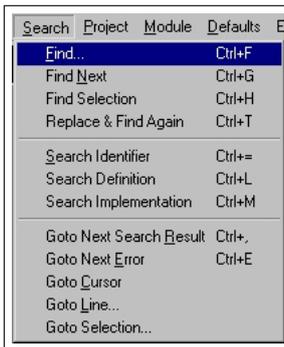
**Add Prefix / Remove Prefix / Change Prefix...** These three together are a generalisation of the **Shift Left** and **Shift Right** commands. They allow the adding and removing of any prefix to the lines in the current selection. Useful for example for block commenting some code. Note that the **Change Prefix** dialogue performs some mangling on the prefix string, that is, it interprets \\ as \ and \t as a tab. Other \x combinations are currently just taken as literals. *Warning: undo is not implemented for these commands.*

**Next Window** Activates the window directly under the active window and brings it to the front. The active window is moved to the bottom of the stack of windows that are currently open in the Clean Integrated Development Environment.

**Show Clipboard / Hide Clipboard** This command opens a window on the Clipboard or closes this window when it is already open.

**List Definitions...** This opens a popup menu with the global definitions found in the currently active editor window. The list is orted by definition type and alphabetically. Selecting an entry moves the cursor to the appropriate line.

## 21 The Search Menu



**Find...** A dialogue appears in which one can define a text to search for (in the Find box) and the text to replace the found text with (in the Replace box).

By default the Find and Replace starts from the current cursor position or currently selected text to the end of the text. On demand the Find and Replace goes backwards to the beginning of the text. When Wrap Around is chosen the Find and Replace does not stop at the beginning and end of the text in the active window. When Ignore Case is chosen, no distinction is made between upper and lower case characters in the text to search for. When the Match Words option is checked only whole words will be selected. When the text to search for appears as a part of another word it will be ignored. Whole words are delimited by outline characters (spaces, tabs, (semi)colons, quotes etcetera). When Use Reg Exp is chosen the Find and Replace strings will be interpreted as regular expression as detailed in [Regular Expressions](#), p.23.

**Find Next** This will find the next piece of text as defined in the Find dialogue. When the **Shift** key is held down the command finds the previous piece of text as defined in the Find dialogue.

**Find Selection** The selected piece of text will be entered into the Find box and the next occurrence of the selection will be found. When **Shift** is pressed the previous occurrence will be found.

**Replace & Find Again** Replace selected text with the text as defined in the Replace box and find the next piece of text as defined in the Find box. When the **Shift** key is being held down the previous occurrence will be found.

**Search Identifier** If there is a selection in the active window then this command wil attempt to find this selection in all files in the current project. If there is no selection or the **Shift** key was pressed when selecting this command a dialogue appears allowing you to enter the search string and set the search method.

Find Identifiers locates all occurrences of an identifier. You can use this for example to change the name of an identifier in all the modules of a program. The search results are displayed in the Search Results window. You can use **Goto Next Search Result** (**Ctrl-,**) to jump to the next occurrence.

If Be Verbose is chosen the file currently being searched is listed.

**Search Definition** When a Clean symbol (function, type or constructor) is selected this command will try to find the definition of this symbol in its definition module. Pressing **Ctrl** and double-clicking on the name has the same effect. Macros and functions that have no type specification may not be found. When

your Clean modules have a (very) unusual layout the search algorithm may also not be able to find the definitions of Clean symbols.

**Search Implementation** This command does the same as **Find Definition**, but now the implementation modules will be searched for the symbol. Holding down **Shift** and **Ctrl** and double-clicking on the name has the same effect as this command.

**Goto Next Search Result** Move to the location of the next entry in the Search Results window.

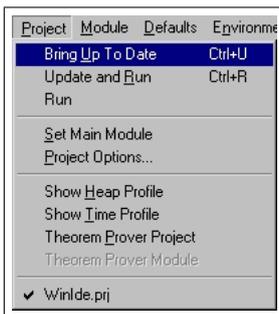
**Goto Next Error** This command selects the next error message in the Errors window and also selects the corresponding text line in which the error occurred. When an error occurs in a file which is not opened a window on this file will be opened automatically. This command only has an effect when the Errors & Warnings window is open.

**Goto Cursor** Scroll front window to show the text at the current cursor position.

**Goto Line...** Select the line with the indicated line number. Define the desired line number in dialogue box. The lines in a text window are numbered starting from 1. The line with the indicated line number is selected. For other ways of moving the cursor around and selecting lines see [The Text Editor](#), p.15.

**Goto Selection...** Define the desired selection in the dialogue box. The given selection will be highlighted and displayed in the text window.

## 22 The Project Menu



**Bring Up to Date** This command will bring the project up-to-date.

This means that, starting from the main module all implementations that are inconsistent will be recompiled. Also the inconsistent object files are regenerated and a new application is built.

Inconsistencies can arise when implementation modules are modified or when implementation modules have to be recompiled because they are depending on modified modules.

After an update all files are consistent. Launching of an application can now be done without recompilations.

**Update and Run** This command first brings a Clean program up-to-date and then launches it.

The **Update and Run** command will perform the same actions as the **Bring Up To Date** command explained above. In addition the generated application is launched. The system will warn you when there is not enough free memory left to launch the generated application.

**Run** If you are convinced that the application already exists and is up-to-date then you can use the **Run** command to immediately launch it without first performing the consistency checks performed by the **Update and Run** command described above.

**Set Main Module** Set the main module of the currently active project to the implementation module displayed in the front window. Note that this clears all the found dependency information for the current project and all the associated compiler and editor settings.

**Project Options** Here the project options for the current project can be set, see [Project Options](#), p.11.

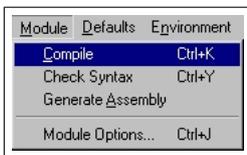
**Show Heap Profile** This launches the `Show Heap Profile` application and attempts to open the heap profile for the current project. This will only succeed if a heap profile has been generated for the current project. That is if heap profiling has been selected in the `Profiling` pane of the `Project Options` of the current project, and then the generated application has been executed.

**Show Time Profile** This launches the `Show Time Profile` application and attempts to open the time profile for the current project. This will only succeed if a time profile has been generated for the current project. That is if time profiling has been selected in the `Profiling` pane of the `Project Options` of the current project, and then the generated application has been executed.

**Theorem Prover Project** This launches the Clean Theorem Prover and attempts to open a proof project for the currently active project in the IDE.

**Theorem Prover Module** This command is currently disabled.

## 23 The Module Menu



**Compile** Compile the Clean implementation module (`name.icl`) displayed in the active window. Error messages will appear when inconsistencies are found. With `Find Error` in the `Search` menu the line in which the error was found is selected. When no errors are found, type information is displayed in the `Types` window when this option was chosen (see [Module Options](#), p.14).

A file `name.abc` will be created in the folder `Clean System Files`. This folder will also be created if it does not yet exist in the same folder where `name.icl` and `name.dcl` are located.

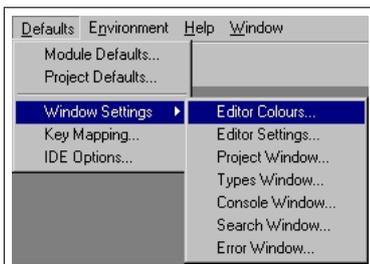
**Check Syntax** Check the Clean implementation module (`name.icl`) displayed in the active window. `Check Syntax` does the same as `compile`, but now no `.abc` file is generated.

**Generate Assembly** Generate an assembly file corresponding to the Clean implementation module (`name.icl`) displayed in the active window. The assembly file is placed in the same folder as the `icl` file and has the extension `.s` (for example `name.s`).

The assembly file is generated by the Code Generator taking a `name.abc` file as input. This file is generated by the Clean compiler. If the `.abc` file is not present or inconsistent the `Generate Assembly` command will invoke a compilation of the implementation module.

**Module Options** This displays a dialogue where a number of options can be set for the currently active module, see [Module Options](#), p.14.

## 24 The Defaults Menu



**Module Defaults** A dialogue appears to set the default module options for new modules added to a project, see [Module Options](#), p.14.

**Project Defaults** In this dialogue the default `Project Options` for new projects can be set, note that paths and extra link items cannot be set by default. If this is required then add them to an environment. See [Project Options](#), p.11 and [Environment Options](#), p.9.

**Window Settings** The following menu entries are placed under the submenu Window Settings:

**Editor Colours...** Here you can set the colours used for syntax colouring the editor windows.

**Editor Settings...** This dialogue allows you to set a new text font and text size for editor windows. Also the tab width can be redefined. With the Auto Indent checkbox the auto indent facility of the text editor can be switched on and off.

**Project Window** Here you can set some font and colour information for the project window.

**Types Window** This allows you to set the font and syntax colouring information for the Types window.

**Console Window** This allows you to set the font and syntax colouring information for the Console window.

**Search Window** Here you can set the font and colours used in the Search Results window.

**Error Window** Here you can set the font and colours used in the Errors & Warnings window.

**Key Mapping...** In this dialogue the key mappings for the editor can be set, see [Key mappings](#), p.22.

**IDE Options...** In this dialogue some global preferences for the Clean IDE can be set. See [IDE Options](#), p.9.

## 25 The Environment Menu



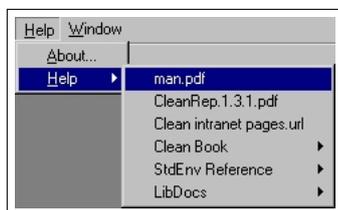
**Edit Current...** Allows you to edit the settings for the currently active environment. See [Environment Options](#), p.9.

**New...** Displays a dialogue asking for the name of the environment to be added, and then displays the options dialogue.

**Edit List...** Allows you to edit the list of known environments.

Below these comes a list with the currently available environments with the currently active environment indicated by a check mark. You can change the active environment by selecting the desired environment from the list.

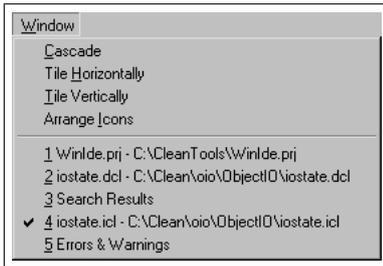
## 26 The Help Menu



**About...** Displays an about window for the Clean IDE. The current version number can be found here, please mention this when reporting bugs.

**Help** This submenu displays the documents found in the `Help` directory. Selecting a help item opens the associated document in an appropriate application. E.g. Acrobat Reader for this manual.

## 27 The Window Menu



The Window menu displays a platform dependant number of options for arranging the open windows and switching between them.

## A Keymappings

The editor also supports (user modifiable) key mappings. The preinstalled default mapping is as follows:

SHIFT + ALT + LeftKey	Select	WordLeft
CTRL + SHIFT + LeftKey	Select	StartOfLine
SHIFT + LeftKey	Select	CharLeft
ALT + LeftKey	Move	WordLeft
CTRL + LeftKey	Move	StartOfLine
LeftKey	Move	CharLeft
ALT + SHIFT + RightKey	Select	WordRight
SHIFT + CTRL + RightKey	Select	EndOfLine
SHIFT + RightKey	Select	CharRight
ALT + RightKey	Move	WordRight
CTRL + RightKey	Move	EndOfLine
RightKey	Move	CharRight
ALT + SHIFT + UpKey	Select	PageUp
CTRL + SHIFT + UpKey	Select	StartOfText
SHIFT + UpKey	Select	LineUp
ALT + UpKey	Move	PageUp
CTRL + UpKey	Move	StartOfText
UpKey	Move	LineUp
ALT + SHIFT + DownKey	Select	PageDown
CTRL + SHIFT + DownKey	Select	EndOfText
SHIFT + DownKey	Select	LineDown
ALT + DownKey	Move	PageDown
CTRL + DownKey	Move	EndOfText
DownKey	Move	LineDown
BeginKey	Scroll	StartOfText
EndKey	Scroll	EndOfText
PgUpKey	Scroll	PageUp
PgDownKey	Scroll	PageDown
CTRL + DeleteKey	Remove	StartOfLine
ALT + DeleteKey	Remove	WordRight
DeleteKey	Remove	CharRight
CTRL + BackSpaceKey	Remove	EndOfLine
ALT + BackSpaceKey	Remove	WordLeft
BackSpaceKey	Remove	CharLeft

A customized mapping can be made by defining a mapping using the `Key Mapping` dialogue in the `Defaults` menu and then saving it. Any saved keymapping can later be reloaded from the same dialogue. If the keymapping is then installed in the “Config” directory under the name `default.km` then in future the IDE will use that as its default key mapping rather than the preinstalled one given above. In the distribution an alternative (Windows) keymapping is provided under the name `windows.km`



RE	= unit	: search for unit
	unit *	: search for zero or more occurrences of unit
	unit +	: search for one or more occurrences of unit
unit	= c	: search for the indicated character
	(exprs)	: turn exprs into unit
char	= c	: search for the specified character, if it is not used as special RE character
	#c	: search for the specified character, even if it is a special RE character
	.	: search for any character
	@	: search for any letter
	~	: search for any non-letter
	[alt]	: search for any of the specified characters
alt	= c	: this character
	c1c2	: or any character in this sequence
	c1..c2	: or any character in this interval

where the special RE characters are: { } \* + ( ) # . @ ~ [ ]

And the replacement expression can be specified as follows:

rexpr	= exprs	: replace string found by exprs
	exprs rexpr	: replace string found by sequence of exprs
exprs	= c	: replace string found by the indicated character, if no control character
	#c	: replace string found by the specified character, even if it is a control character.
	&	: replace string found by itself
	\$n	: replace string found by the $n$ -th substring indicated in the search expression, $0 \leq n \leq 9$
		: replaced by empty string if the $n$ -th substring does not exist

where the replacement expression control characters are: & \$ #

## D Bugs

If you have bugs to report, please send us a short description with a tiny example via electronic mail (<mailto:clean@cs.kun.nl>) and we will try to fix it as soon as possible.

## E Authors & Credits

Clean and the Clean System are a spinoff of the research performed by the research group on functional programming languages, Computing Science Institute, at the University of Nijmegen under the supervision of Rinus Plasmeijer. To see who is currently responsible for which part of Clean and the Clean System have a look at the Clean web pages.

Special thanks to the following people: Christ Aarts, Steffen van Bakel, Erik Barendsen, Henk Barendregt, Pieter Hartel, Hans Koetsier, Pieter Koopman, Ken McDonald, Ronan Sleep and all the Clean users who helped us to get a better system.

Many thanks to the following sponsors:

- the Dutch Foundation for Applied Technical Sciences (STW);
- the Dutch Foundation for Scientific Research (NWO);
- the International Academic Centre for Informatics (IACI);
- Kropman B.V., Installation Techniques, Nijmegen, The Netherlands.
- Hitachi Advanced Research Laboratories, Japan;
- the Dutch Ministry of Science and Education (the Parallel Reduction Machine project (1984-1987)) who initiated the Clean research;
- Esprit Basic Research Action (project 3074, SemaGraph: the Semantics and Pragmatics of Graph Rewriting (1989-1991));

- Esprit Basic Research Action (SemaGraph II working group 3646 (1992- 1995));
- Esprit Parallel Computing Action (project 4106, (1990-1991));
- Esprit II (TIP-M project area II.3.2, Tropics: TRansparent Object-oriented Parallel Information Computing System (1989-1990)).