ClickMake

Thomas Richter

COLLABORATORS			
	TITLE : ClickMake		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Thomas Richter	April 14, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ClickMake

1.1 ClickMake Guide

ClickMake Guide

Guide Version 1.03 ClickMake Version 1.09

Table of Contents

Thomas Richter

Rühmkorffstraße 10A

12209 Berlin

Germany

EMail: thor@einstein.math.tu-berlin.de

WWW: http://www.math.tu-berlin.de/~thor/thor/index.html

1.2 The THOR-Software Licence

The THOR-Software Licence (v2, 24th June 1998)

This License applies to the computer programs known as "ClickMake" and the "ClickMake.guide". The "Program", below, refers to such program. The "Archive" refers to the package of distribution, as prepared by the author of the Program, Thomas Richter. Each licensee is addressed as "you".

The Program and the data in the archive are freely distributable under the restrictions stated below, but are also Copyright (c) Thomas Richter.

Distribution of the Program, the Archive and the data in the Archive by a commercial organization without written permission from the author to any third party is prohibited if any payment is made in connection with such distribution, whether directly (as in payment for a copy of the Program) or indirectly (as in payment for some service related to the Program, or payment for some product or service that includes a copy of the Program "without charge"; these are only examples, and not an exhaustive enumeration of prohibited activities).

However, the following methods of distribution involving payment shall not in and of themselves be a violation of this restriction:

(i) Posting the Program on a public access information storage and retrieval service for which a fee is received for retrieving information (such as an on-line service), provided that the fee is not content-dependent (i.e., the fee would be the same for retrieving the same volume of information consisting of random data).

(ii) Distributing the Program on a CD-ROM, provided that

a) the Archive is reproduced entirely and verbatim on such CD-ROM, including especially this licence agreement;

b) the CD-ROM is made available to the public for a nominal fee only,

c) a copy of the CD is made available to the author for free except for shipment costs, and

d) provided further that all information on such CD-ROM is redistributable for non-commercial purposes without charge.

Redistribution of a modified version of the Archive, the Program or the contents of the Archive is prohibited in any way, by any organization, regardless whether commercial or non-commercial. Everything must be kept together, in original and unmodified form.

Limitations.

THE PROGRAM IS PROVIDED TO YOU "AS IS", WITHOUT WARRANTY. THERE IS NO WARRANTY FOR THE PROGRAM, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IF YOU DO NOT ACCEPT THIS LICENCE, YOU MUST DELETE THE PROGRAM, THE ARCHIVE AND ALL DATA OF THIS ARCHIVE FROM YOUR STORAGE SYSTEM. YOU ACCEPT THIS LICENCE BY USING OR REDISTRIBUTING THE PROGRAM.

Thomas Richter

1.3 About ClickMake

"ClickMake" is a simple project manager, and can be used to create "single module" projects, i.e. programs that consist only of a single file. It usually does not make much sense to create makefiles for these tiny programs, nor is it useful to type every single command on the shell. The invokation of the editor, compiler, optimizer, assembler and linker is handled completely by "ClickMake".

"ClickMake" is highly configureable - examples for the SAS and the PCQ compiler are supplied in this package. It can be setup to work with every compiler that can be started from the shell, read here .

The easy handling is optimal for beginners in a program language, however the setup procedure takes some time and should be done by experts. To keep thinks as simple as possible, I included example scripts for these popular compilers.

Last but not least: IT'S FREE! (But please read the licence !)

1.4 Installing ClickMake

The installation process depens a bit on the compiler you prefer, but here is the way it works in general:

o) Copy the ClickMake program to a drawer where executeable files can be found. This should be either the directory where your compiler keeps its executeables, or the C: drawer. The last choice should be prefered if you want to use ClickMake with more than one compiler.

o) Locate the directory where your compiler is installed. If you own either the SAS or the PCQ compiler, copy one of the example settings, SAS.cm or PCQ.cm to this drawer.

o) Create a directory in this drawer to keep your source files, if not already present.

o) Locate the startup script of the compiler or create one. This script should contain all stuff necessary to start a compiler session, i.e. it will add the compiler binaries to the path, setup INCLUDE: e.t.c. There is no general rule where this file is kept or how it's called...

o) Edit this script file:

o) Add an assign to the directory of source files. To use the example SAS settings, this drawer should be called LSAVE:, for PCQ it should be called PSAVE:

o) Make shure that you increase the stack size to AT LEAST 8192 bytes which are needed by ClickMake. A usual compiler needs more, so this value should be increased anyways.

o) Make shure that the ClickMake program is in the command path, together with the compiler, the editor, the linker and (propably) the linker. The example scripts that come with ClickMake use either the programs that are part of the compiler distribution, or standard amiga stuff. However, you can change the settings .

o) Add a line like the following to the startup script:

ClickMake <settingsfile.cm>

o) Add a command to create a temporary directory on RAM where to keep the compiler temporaries and intermediate ClickMake output. Add an assign to this directory. For the SAS example settings, this should be called LW:, for PCQ PW:

o) Copy this guide whereever you want.

o) Now edit the settings, to match your personal taste or to match the compiler's requirements. Read on here .

1.5 Contacting the Author

Here's my EMail address:

thor@einstein.math.tu-berlin.de

Thomas Richter

You may also want to visit my web page, latest versions of all my programs (plus more) are available there:

WWW: http://www.math.tu-berlin.de/~thor/thor/index.html

The selection is quickly expanding, check in monthly!

1.6 Using ClickMake

If you have installed and configured ClickMake correctly, the ClickMake window should show up when you start a compiler session.

To start a new project

Enter the name of the project in the project gadget (without any .c or .p), and press the "Edit" button. The editor should be invoked. The project file will be stored in the temporary directory you created in the startup process, read here again if you missed this point.

To load a previously saved work

Click the "Load" gadget. A requester should appear: Select now the Directory where your source is kept. Do NOT select the project file itself, the directory is enough. Do not mind if you can't select a file in this directory, that's usual. Click "OK" in the requester: This should copy the project file, all temporaries and the settings file to the temporary directory. Click now "Edit" to start editing the file.

To save your project

Click the "Save" gadget. This will create (if ClickMake is correctly setup) a drawer in your source code directory, named by the project. ClickMake will copy all project files to this directory.

Do not forget to save! If you leave the editor by "Saving" your source code, this will store the code only in a temporary place. It will be copied to the final directory with the "Save" gadget.

To compile your program

Click the "Make" gadget. This should invoke the compiler, assembler, linker etc automatically, depending on what has changed. If you want to force a remake, click the "Settings" checkmark twice or invoke the editor and save the source code again. Another way is force a recompilation is to select the string gadget with the project name and to press RETURN there, without changing the name itself.

To rename your source code

Enter the new name in the project string gadget.

To debug your program

Click on the "DBug" gadget. This might cause a recompilation, to include debug symbols in the final binary. Finally, it should start the debugger.

The EdAs gadget has some special purpose, provided for specialists. It should, depending on compiler, create or display the intermediate assembler code of your project. This might be useful for debugging your program (at least it used to be useful for me).

To edit the settings, toggle the checkmark gadget called "Settings", then continue to read here .

1.7 Configuring ClickMake

Using ClickMake is quite easy, configuring it, however...

Before you start to configure ClickMake, make shure that...

o) you have enough time. This MIGHT take a while, together with testing.

o) you know your compiler well. You should be clear about,

o) which editor to use

o) what must be done to compile a program (do you need an assembler? Which linker fits?...)

o) how to invoke the compiler, assembler, linker....

It's propaply easier to modify an existing configuration of ClickMake. The SAS configuration is somehow typical for those compilers that work without an additional assembler run, the PCQ configuration on the other hand works with one (actually two)

intermediate assembler stages. Most modern compilers do not need an assembler, most PD compilers will (cause it's easier to generate assembler output than to generate full binaries). Modify the SAS settings for the first type (commercial compilers), modify the PCQ settings for the setting...

To start from the beginning:

o) Open a shell window. Increase the stack size to 8192 bytes (or more).

o) Go to your compiler directory.

o) If you want to modify an existing configuration, type

ClickMake <settingsfile.cm>

o) to create a completely new configuration, start ClickMake without any arguments.

o) Click the "Setttings" checkmark gadget.

The ClickMake window should increase enormous, showing the settings window. You can shrink this window later on with the same gadget when your settings are complete, but for now leave it as big as it is.

Remember this step! Whenever talking about the settings or setup window, you get it like this!

From here on, the configuration is split in several chapters, starting with:

1.8 Choosing the position of the ClickMake Windows

You propably do not like where ClickMake places its window. To setup the window position, you may either:

o) Drag the window where you want it. The changes will be immediatly reflected in the settings window.

If you drag the window while the "Settings" gadget is selected, you setup the position of the complete settings window.

If you drag the window without the "Settings" gadget beeing selected, you'll adjust the position of the tiny window.

o) Directly enter the coordinates in the gadgets "Normal Left" to "Full Top" in the upper right corner of the settings window. The "Normal" gadgets keep the position of the tiny window, the "Full" gadgets are related to the x and y coordiates of the settings window.

1.9 Run ClickMake in Background?

If you want ClickMake to launch itself in background, check the "Background" gadget in the settings window located on the right edge under the coordinates gadgets; this might save one "RUN" command for invokation of ClickMake.

Please note that changing the state of this gadget will not launch ClickMake right now in background, it affects only its behaivour when the settings are loaded later on.

It's usually best to keep this gadget checked....

1.10 Choosing the directory for sources

As part of the installation process , you should have created a subdirectory of your compiler directory, where you like to store your source codes. Now you have to tell ClickMake this directory, since it will save sources there:

Enter the path of the directory (or the assign name, if you created one) into the string gadget near the "SaveDir" button on the left edge of the settings window. You may also select this directory by hand thru a requester by pressing this button itself.

1.11 Setup the screen

Usually ClickMake will open its window on the workbench screen. However, you might choose a different public screen for ClickMake. Just enter the Pubscreen name in the "Screen" gadget on the left hand side of the Settings window. If you prefer the workbench, just enter

Workbench

which is also the default.

1.12 Theory of Operation

To compile a program, you'll usually need to invoke more than one shell command - in almost all systems at least the compiler and the linker. In this setting the compiler output file, beeing an object code file, is the linker input file, and if this file gets changed, the linker must be rerun.

On the other hand, changing the source code must result in recompilation, which changes the object code. This must again be followed by a linker run to create the final output.

As in this example, the creation process from source code to the final program is a process involving certain types of intermediate files (the object code in our example), one depending on another, and shell commands to create one file from the other. Here we have:

source -> object code -> final code

^ ^

| |

compiler linker

This need not to be as simple as in this example. For example, compiling a pascal program with the PD compiler PCQ and optimizing it takes more generation steps:

source -> assembler code -> optimized assembler -> object code -> final code

^ ^ ^ ^

| | | |

compiler peephole optimizer assembler linker

It would be very anoying if you had to type all commands each time you made a change to your source.

The whole generation process from the source to the final code is called a "generation pipeline" or just a "pipe" in ClickMake terminology, cause you "pump in" your source at the one end and the final program will "pop out" at the other.

The shell commands that generate one file from another (the sort of "pumps" in this pipe) are called "stages" in the ClickMake terminology. Thus the example above just described the creation "pipe" of PCQ with the "stages" compiler, optimizer, assembler and linker.

Each one of the files involved in this "pipe" has by tradition a unique ending name (which, of course, can be setup in ClickMake). In this case, we have the endings

.p for pascal input code

.asm for compiler output code

.s for the optimized assembler code

.o for the assembler object code output

(no . extension) for the final file.

For example, the ".s" file will depend on the ".asm" file, in the sense if the later is changed the former must be rebuild by the optimizer stage. All these dependencies must be told to ClickMake, together with the rules and shell commands how to "pump" the files thru the pipe by the stages. Lucky enough, this must be done only once per compiler, or never if you use the compilers supported by the example settings. If you want to change later on some compiler flags, you'll only have to modify the setting of the compiler stage a little, with no need to setup it completely new. Of course, the setting for your project will be saved together with the source code and the final program in subdirectory of the save directory .

The Chapter continues here with more stuff about pipes and stages.

1.13 More theory about pipes

All ClickMake buttons are linked to one "pipe", the example above is invoked by the "Make" gadget in the ClickMake window. Every action that is performed by ClickMake is thought in pipes and stages, and each pipe is linked to a button of its own. For example, if you press the "Edit" button, the "Edit" pipeline is started. It looks usually very trivial:

source code -> source code

^

|

editor

where as a special case, the left hand side may be left empty, i.e. the source code need not exist to make the pipe working.

Or another example: Consider you want to save your project. This will call the "Save" pipe. It should, if setup correctly, look like the following:

project files -> project files & setup -> got folder -> saved project

^ ^ ^

| | |

save ClickMake setup create dest dir copy all the stuff

in save folder from tmp to final

In this case, the certain stages are only loosely bounded and cannot be connected with certain temporary files, and there are no dependecies coming from these files. However, before doing one stage all the stage before must have been done.

A historical note:

All this type of dependency stuff is tradionally done by so called "makefiles", which have been invented for unix. These makefiles use a strange syntax, for example you need to type TABs in the correct way, and are not setup in a easy way. It would be a complete waste of time to type in a makefile for a tiny single module project. This can be done better by ClickMake. However, if you plan a larger project, I recommend you to learn the "make" syntax....

1.14 All ClickMake pipelines

Before setting up a pipeline , you must know what must be done in it. In principle, you may invoke the compiler with the "Edit" button, but this *might* be a bit misleading. I want to give you a guide which pipelines exist and when they are invoked, how they are actually setup is written elsewhere :

The "Edit" pipeline

This pipe is invoked by the "Edit" button. It should only call the editor to modify your source code.

The "Make" pipeline

Invoked by the "Make" button. This is usually the most complex and does the heavy work of creating a runable program out of your source code. In the easiest version, a simple compiler call might suffer, but most compilers need at least a compiler and a linker run - two stages are standard here. I can't give a general rule what must be done in this pipe, since it does depend on the compiler you use.

The "Edit Assembler" pipeline

This one is for specialists only, and gets started by the "EdAs" button. It should do whatever must be done in order to invoke the editor with an assembly language version of the source code. This is useful to see wether the compiler had done its job properly.

This stage may either need a compiler run to create assembly output - together with some special flags to tell the compiler to generate no object code - and the invokation of the editor, or

Creation of the object code output, together with a run of an utility that creates readable assembler output from the object code.

This again depends on your compiler!

As this stage is only for experts, you may want to leave it blank.

The "Debug" pipeline

Almost like the "Make" pipeline, but invoked by the "DBug" button and used to debug a program. This pipeline should create a debug-able version of the compiler output - you may need to set some flags for the compiler - together with an invokation of the debugger.

The "Load" pipeline

This is invoked by the gadget of the same name and is used to copy your source code together with the settings from the save diretory to the temporary working directory. It should consist of three stages:

o) Show a requester and let the user choose a project to work on.

o) Copy the source code together with the settings for this project from the selected directory to the working directory.

o) Load the copied settings into ClickMake to make shure that the compiler flags match the requirements of this program.

The "Save" pipeline

This one does the reverse of the "Load" pipeline. It's invoked by "Save", and should do the following:

o) Save the current settings of ClickMake to keep the compiler flags and more stuff.

o) Create a project directory in the save folder, if not already present. This is done best by the build-in command "MakeDir" with the special option "NOWARN".

o) Copy the source code and the settings, maybe together with some additional files to this directory.

The "Rename" pipeline

Unlike the others, this one is not called by a button, but if you type a different name into the "Project" string gadget and this gadget has been non-empty before. It should rename all project relevant files to the new name:

o) Rename the source code

o) Rename the settings file

Optionally:

o) Rename the output file.

The "Load Settings" pipeline

This pipeline is invoked by the "Load Settings" gadget which is normally hidden unless you checked the "Settings" button. It is used to reload some ClickMake settings.

The "Save Settings" pipeline

Invoked by the usually hidden "Save Settings" gadget in the setup window this pipeline should save the current settings of ViNCEd to disk.

I finally decided not to add a Go like button to start your program since I found it is usually *better* for your system's consitency not to start a new project by accident without a debugger running :-). Use either the "DBug" button or call the program by hand from the shell.

1.15 Setup the stages of a pipeline

Here we're going to discuss how you actually edited the stages, provided you've read the chapters before and you now what each pipe should do .

First, open the setup window by checking the "Settings" gadget. You'll then find two lists right in the middle of the window:

The left one contains all pipes ClickMake knows. This list is fixed, you cannot add new pipelines to it. Each of them is engaged by a special action, read here .

Once you have selected one pipeline, the right list shows all stages that are involved in this pipe, hence the different "pumps" in the pipe or the different actions that need to be performed to execute the pipe. If you start with a blank ClickMake session without having loaded a settings file by a command line argument, this list will be empty.

To add a new stage to the pipeline, select the stage behind which you want to insert a new stage and press "Add New", or, alternatively, press the "Add 1st" button to add a new stage at the beginning.

Then type its name in the string gadget below the list. This title is only kept to remind you later on what this stage will do, and is never used interally - type whatever you think is descriptive here.

To remove a stage you added by accident, mark it first and click the "Remove" button beneath.

The set of gadgets below both list now describes what must be done in this stage, together with the dependecy rules: These rules are encoded in the two gadgets on top of them, and specify the input & output files of this stage - or short: which type of file is converted to what. More about these two is here .

Now lets come to the next gadgets below them:

Cmd

Put here the pure command to run this stage without any options, e.g. the name of your compiler. Do not add options to this string.

The way how these settings form a complete call will be specified in another gadget.

Opt

Put options for the command here, typically compiler options how to generate code should go here.

Pre

Put special arguments that must go between the options and the command itself here. For a linker call, this might be the name of the startup module.

Post

This gadget specifies arguments that must be placed between the options and the "Exts" below.

Exts

Put special arguments that must go last here. Again, for the linker example this will be the linker libraries you'd like to link with your program.

In principle, you're free how to use the gadgets descriped above since you may alter the way how a complete command line is build from them. However, I recommend that you use them in the way I said above to avoid confusion. Plus, you only have to change very few of them to configure ClickMake for a given project, i.e. modify the "Opt" field in the compiler stage to change the compiler options. I think this is more clearly than to edit the whole command line or to remember the syntax of how to invoke the compiler.

Format

Now, this one is important. It tells ClickMake how to construct the command line out of all gadgets above. The complete line is then send to the shell for execution. The string in this gadget has a C-style syntax, where all settings of ClickMake, like the gadgets on top, are specified by a percent "%" sign plus an additional character dependent on the setting. Cause there are quite a lot of them, I put the description here .

1.16 Encoding the Dependency Rules

A stage of a pipeline is always thought to take a input file of a certain type to generate a corresponding output type, i.e. a C compiler would take a file which ends with ".c" and would generate an object module of the same name, but ending with ".o".

The ".o" file depends now on the ".c" file in the sense that if the later is changed the object file must be regenerated. After adding a new stage , ClickMake must be told about the rules how the files do depend, and which file types a stage takes as input and generates as output.

The input and output file extensions of a stage - the file part behind the dot - have to be typed into the "Input" and "Output" gadgets below the two lists in the setup window. In the example above, enter the strings

c into the "Input" gadget and

o into "Output". Do NOT enter a ".", since it is generated by ClickMake.

The "%" character

Like in AmigaDOS, if a file does not have an extension, enter a "%" to indicate this. Usually the file without any extension will be the final executeable binary, which is generated by the linker. Thus a typical linker stage would have

o as "Input", i.e. it takes object modules as input, and

% as "Output", i.e. it generates files without any extension, beeing the final executeable binary.

The "@" character

The "at" sign is a special character you may enter as output meaning a "non-existing" file. Thus this stage will be run in any case, independent of wether its output is already present or not. This is also useful for stages that do not produce output, like the invocation of the editor in a "Edit Assembler" stage.

A stage with a "@" output should always be the last stage in a pipeline, since it is assumed by ClickMake that its output will be "dumped", i.e. is meaningless to the generation process itself. This does NOT mean that this stage does eventually produce some output, but this output is ignored by ClickMake and always considered as "non-existing". Thus "@" is the analogon to the "NIL:" file name in terms of ClickMake.

The "|" character

The vertical bar character is used to indicate that a stage needs more than one input file or produces more than one output. This is usually not true for the "common" compiler pipeline; but in the save pipeline, for example, the stage that copies the project from the temporary storage to the save directory will have more than one output file, namely the source file, the settings file and the final executeable file. Thus

clcm|% as input and output. "c" is a C-source code in this example, "cm" is the "ClickMake" settings file and "%" is used to indicate a file without an extension - the final executable.

Independent stages

Stages that take the same files as input and as output are called "independent", since they obviously do not need a run of another stage before they can be executed. Unlike other stages with different input and output, which are only run if the output is needed, the independent stages will be always started by ClickMake, and before any stage which has dependencies in the order of their appearence in the stage list.

A typical pipeline that uses independent stages is the "Edit" pipe, which only invokes the editor. It takes both a "c" file as input and as output, thus beeing "independent". For this reason the editor is always run, unlike an ordinary stage which is only run if the source has changed. As you see, it would be rather stupid to run the editor only if the source has changed! The whole point of the editor is to change the source, hence it must be run in any time, and independent of wether its input has changed or not.

1.17 The control strings in the format gadget

To setup a stage it is of course not enough to specify the input and output, and all the options - somehow the command line must be constructed from all these settings. This is done by the format gadget, which is located in the lower right corner of the ClickMake setup window.

The string you put here is parsed like a C-style printf format argument, with "%" plus a control character beeing expanded in a string provided by ClickMake. Here the list of known control characters, they are not case sensitive:

%% : The percent sign itself.

%c : The contents of the "Cmd" gadget of this stage.

%d : The contents of the "SaveDir" gadget on top of the lists. Thus "%d" will expand into the directory, where the project should be copied by save.

%p : The contents of "Pre" gadget. For this reason, "%p" should go directly behind "%c".

%q : The contents of the "Post" gadget. This should go behind "%b" or "%n".

%o : The contents of the "Opt" gadget.

%e : The contents of the "Exts" gadget.

%n or %b: The contents of the "Project" gadget on top of the window. Thus "%n" will expand into the base name of your project. "%b" is the same.

%r : The file part of a new project name. Only useful in the "Rename" or "Load" stage. In this case "%n" contains the old project name and "%r" the new one or the name of the project to be loaded.

%R : The "%R" control sequence is the only case sensitive sequence. "%R" is the path name (ending with "/" or ":") of a new project name, used in the "Load" or "Rename" stage.

Examples:

To invoke the editor in the "Edit" stage in the PCQ settings, use:

%c "%n.p"

This will expand into the editor's name, plus the project name with extension ".p", enclosed in double quotes to make project names with spaces work.

To invoke the PCQ compiler, use

%c %o "%n.p" "%n.asm"

This will build a command line consisting of the compiler's name, followed by the options, the project name with extension ".p" and the project name with extension ".asm".

Let's come to the most complex example, the linker invocation:

%c FROM %p "%n.o" %q TO "%n" %e %o

This will be again the linker name with the options "FROM pre sourcefile.o", using the "Pre" gadget as the name of the startup code and the "Post" gadget as a list of additional modules to link with. The destination of the link process is again given with "TO %n" by the project name, but this time without any extension. "%e" is used as a list of libraries, thus the "LIB" keyword has to be typed into the "Exts" gadget. Last but not least the "Opt" gadget is inserted.

In principle, you could use the gadgets as you like, i.e. use the "Exts" gadget for the options etc., but this would be rather confusing. On the other hand, there's no need to use really ALL gadgets.

1.18 Build-In Commands

Not all the work that need to be done can be done by standard shell commands, like loading and saving the ClickMake setup file. This is done by build-in commands, which are started like every other shell command. To distinguish them from "ordinary" external commands, their names start with an @ ("at") sign. Except that, they behaive like usual shell commands.

Here's the list of all build-in commands together with their options:

@load FILE/A,NOPROJECTNAME/S,NOWARN/S,IGNORE/M

This command reads in a clickmake settings file of given name "FILE".

If you add the option "NOPROJECTNAME", the project name which is part of the settings file will not be loaded. This is most useful when you want to load a new compiler setup, without starting a new project.

If "NOWARN" is given, you won't get a warning if the settings file cannot be found. This is usual if you want to start a new project which does not yet have a settings file. In this case, the old settings will remain active.

The arguments to "IGNORE" are a list of pipelines that are NOT to be loaded from the settings file; instead, the old settings will be used.

This is useful if you want to change some of the pipelines or settings later on and don't want to use the ones saved with the project.

The following strings are accepted as arguments to "INGORE":

Edit Ignore the "Edit" pipe

Make Ignore the "Make" pipe

"Edit Assembler" Ignore the "Edit Assembler" pipe. Note that due to the way how the DOS parses these strings, you MUST enclose this in double quotes.

Debug Ignore the "Debug" pipe.

Load Do not load the "Load" pipe.

Save Do not load the "Save" pipe.

Rename Do not load the "Rename" pipe.

"Load Settings" Ignore the "Load Settings" pipe. Please note again the double quotes!

"Save Settings" Ignore the "Save Settings" pipe. Again, double quotes are needed!

Additionally, the following ones are legal:

Screen Do not load the name of the public screen, use old one.

Project Do not load the name of the project (Well, this one is a little useless)

SaveDir Do not load the name of the destination directory.

NormalPosition Do not load the position where to place the small (regular) window.

FullPosition Ignore the position of the Setup window saved in the settings and use the old one instead.

RunBack Ignore the setting of the "RunBack" checkmark.

Activate Ignore the setting of the "Activate" checkmark.

You may also enter names of stages you want to ignore, provided that they are present in the current setting. As example, it is legal to specify

"Invoke Assembler" as option. In this case, the assembly stage won't be taken from the new file, instead the old one will be used if present.

@save FILE/A

Save the current settings to the file of given name. As a standard, I would recommend file names with the extension ".cm".

@makedir NAME/A,NOWARN/S

Create a new directory of the given name. Unlike the shell equivalent, you may ask ClickMake not to warn you if this directory already exists.

This is most useful in the "Save" stage to create the destination directory where to store your project.

@request DRAWER,FILE,SAVEMODE/S,DRAWERSONLY/S

Display a file requester and put the result into "%r" (file part) and "%R" (path part). Read also here for understanding the format strings.

DRAWER is the name of the initial drawer to display.

FILE is the initial file name.

SAVEMODE display the requester in save mode.

DRAWERSONLY Allow only the selection of drawers. Most useful to force the user to pick the name of the project's destination drawer.

1.19 Save your Settings

If you finished the setup procedure, you should save your work to disk. Like every action that can be performed by ClickMake, even saving the settings gets activated by a press of a gadget, and the action is again done by a pipeline .

The gadget I was talking about is of course the "Save Settings" gadget in the setup window, and the pipeline it activates is also called like this.

To make your live a bit easier and to avoid that you run into trouble saving your work, I give here my suggestions how the "Save Settings" pipeline should look like:

o) If you see only the tiny window, click the "Settings" checkmark to display the settings window.

o) Select the "Save Settings" pipeline from the left hand side list of the settings window.

o) Add a new stage to the stage list of this pipeline by pressing the "Add 1st" button the right side. A new stage should now show up in the stage list, called "Untitled".

o) Rename this stage to something useful by entering the name in the gadget at the bottom of the list. "Save Settings" should be a good choice...

o) Enter "@SAVE" as command into the "Cmd" gadget, at the left bottom side of the window. The "at" sign is not a typo, it is used to indicate a build-in command!

o) Enter

%c \"%n.cm\" %o

as format line. This will invoke the internal "@SAVE" command with the project name + ".cm" as file name.

o) Insert the name of the file you want to save your settings to into the project icon. A ".cm" will be added.

o) Now press the "Save Settings" buttons. This should do the save - if it fails, please follow the steps above carefully!

1.20 Shell Arguments

Before attempting to start ClickMake from the shell, make shure your stacksize is at least 8192 bytes - this program requires a large stack for keeping all dependency rules. This shouldn't be a drawback, since most compilers need an even larger stack anyways.

Here's the argument template:

ClickMake FROM,HELP/S,NORUNBACK/S

FROM: This argument specifies a file name of a settings file to load from.

HELP: Print a tiny help text.

NORUNBACK: Do not start ClickMake in background, even if the settings tell to do so.

1.21 Thank you, folks

Special thanks go to....

Ernst Besser, for using this program with the PCQ pascal compiler.

Special thanks does NOT go to...

Commodore Amiga for their @^&%! - computer (adverb censored). This ugly think still fails to work correctly. Now my Chip Mem starts to get "creative" (bits start flipping). It's a CREATIVE computer, you know...)-:

Viscorp for their GREAT support.... my computer is still to big for a set-top box (-;

1.22 The history of ClickMake

Version 1.02: First working release.

Version 1.03: Added special characters "|" and "@" for dependency.

Version 1.04: Removed a bug in the argument parsing of internal commands that sometimes caused a crash.

Version 1.05: Added the IGNORE argument to "@\LOAD"

Version 1.06: Set console correctly to support ViNCEd fork command.

Version 1.07: Removed requester when user hit "Cancel" in the requester. The project is now re-made if RETURN is pressed in the project name gadget without changing the name. The window is only re-activated if the active window did not change during the execution of a pipeline.

Version 1.08: Removed one Enforcer hit, rewrote the startup code.

Version 1.09: Fixed one Enforcer hit when parsing messages.