GCI
short documentation

This is a very short document, but should contain enough information on how to use the most important features of GCI.
This document is only about GCI itself, nor its modules, whose specifications are not reported here.

Index:

1) GCI

1.1) Introduction

GCI, which stands for 'GCI C++ Interface', was born to help c++ developers code their software.
GCI is a software which interfaces with GCC adding features to c++, such as the possibility of declaring command line arguments from within source files and a module system.
I will talk more about these features on their chapters.

2) Modules

2.1) Why modules?

Someone may ask why did GCI implements a module system.
It is mainly for 2 reasons:
-a module system can help developers giving more modularity and library developers;
-giving modules standard function names could let developers use a lot of libraries studying only a few standard commands.
The second point is the most important one.
Using modules, developers would only need to study the standard functions' name a time, and then they could be able to use hundreds of libraryies with the same commands.

2.2) How to use modules

Using modules is very easy.
You only need to use a new preprocessor directive, "#using", whose syntax is the following:

#using basedir.modname

Where basedir is one of the directories within the ../modules/ folder of GCI, and modname is the name of one of the folders within basedir.
NB: if modname is equal to '*', all the modules within basedir will be imported.

Some examples are the following:

#using basic.string
imports the string module

#using sdl.*
imports all the sdl modules

#using basic.io
imports basic i/o functions

3) Command line

3.1) A new feature

This chapter is about another interesting feature GCI have.
GCI is able to interpreter another new preprocessor directive, named "#addcompilerargs" ("#addmoduleargs" for usage within modules), which let the developer specify options which will be sent to gcc at compiling time.

3.2) How to add arguments

This feature has many interesting capabilities, which i will show with the help of examples.

#addcompilerargs linux:-llibname -L[$$$]libfolder

'linux:' is an optional specifyier, which states that those options will be told to gcc if and only if the system on which the program is being compiled is a 32-bit linux platform.
Other directives of this kind are lin32:(which does the same as linux) and win32.
For more options read the long documentation.
[$$$] is a special option, resumed in this 'table':

[$] it's the current code directory(it cannot be used when compiling modules)
[$$] it's the basedir/modname/ folder
[$$$] it's the basedir/ folder
[$$$$] it's the modules/ folder

#addcompilerargs first:something

'first:' is another directive. It states that the options following it will be passed to gcc before other linking options