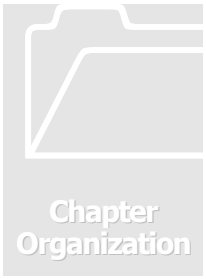
Chapter 1

1. Learning the Basics

Read this chapter if you have never used CM3-IDE before.

This chapter introduces you to CM3-IDE's web-based environment, and to five basic concepts that are central to CM3-IDE: *packages*, *modules*, *interfaces*, *importing*, and *exporting*. By the end of this chapter, you'll be familiar with these concepts and with some of the screens that you'll use often.

If you haven't installed CM3-IDE yet, follow the instructions in the *CM3-IDE Installation Guide* to get started. The rest of this chapter also assumes that you know how to use your web browser and your text editor well.



The chapter is divided into three parts:

Starting CM3-IDE on page 6 outlines how you start the CM3-IDE development environment.

A Quick Walkthrough on page 7 uses the old standby, the "hello world" program to tour the browse and build features of CM3-IDE. In this first tutorial, you'll learn how to:

- create a new package from an existing example
- build a simple "hello world" program
- run the program from within CM3-IDE
- explore how CM3-IDE automatically updates its virtual namespace to keep up with changes to your system

The second tutorial, **Creating a Package from Scratch** on page 15 covers some of the same concepts as the first, but in greater depth. This time, you:

- create your own package
- open and run your text editor from within CM3-IDE
- edit and compile sources and makefiles
- browse one of CM3-IDE's library packages.

1.1 Starting CM3-IDE

You can start CM3-IDE by typing CM3-IDE at the command-prompt, assuming the CM3-IDE program is in your executable path. If you haven't installed CM3-IDE yet, or you are unable to locate the executable program for CM3-IDE, see the *CM3-IDE Installation Guide*.

Once started, CM3-IDE automatically spawns your web browser and points it to CM3-IDE's start screen, as in Figure 1.

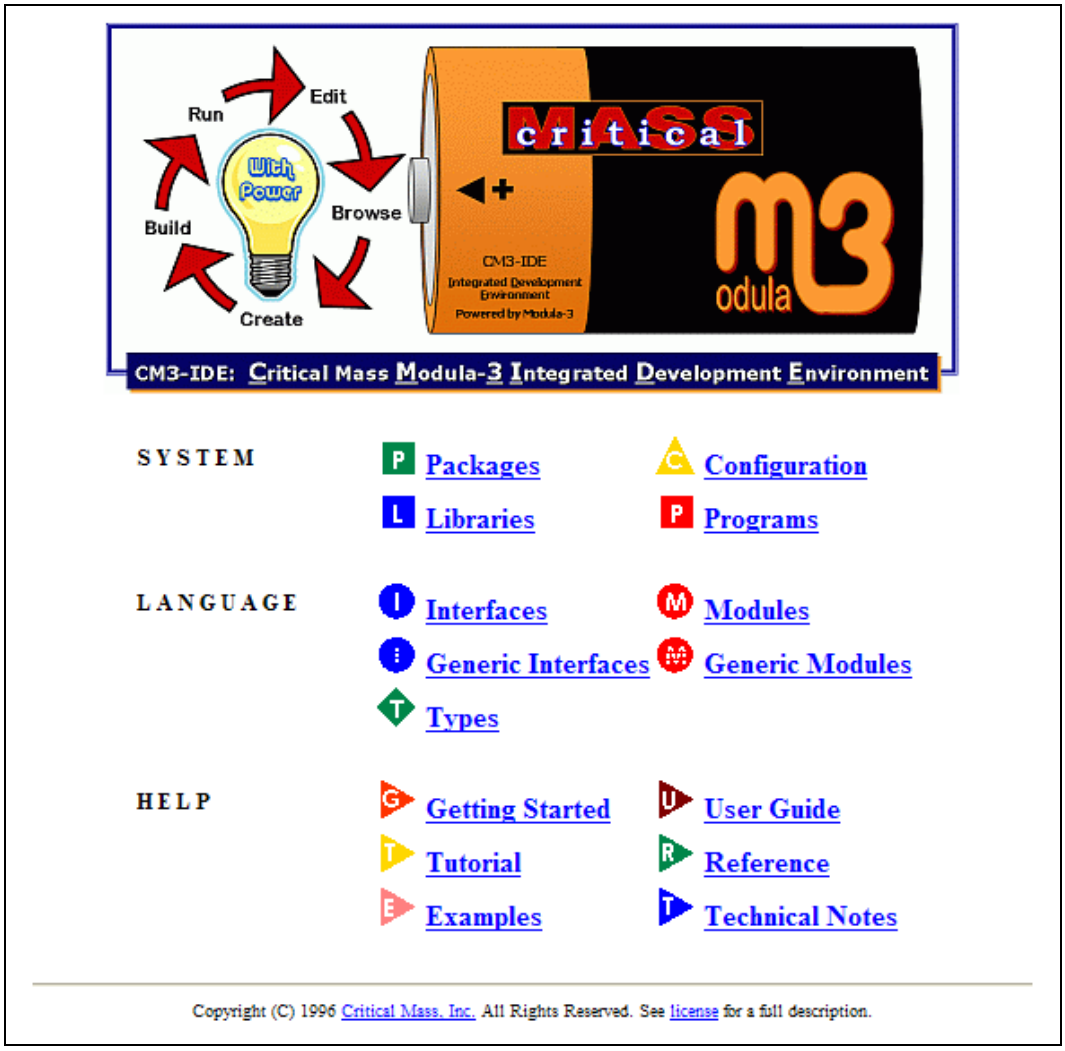


Figure 1. The Start Screen

At the top of the start page is the CM3-IDE logo, and below that a set of icons that represent elements of the CM3-IDE environment. They are divided into three groups: System, Language, and Help.

1.2 A Quick Walkthrough

Having started CM3-IDE, you will see CM3-IDE's start page in your browser's window (Figure 1). Here we quickly walk through the building of a Hello World example program.

Step **The Start Screen.** From the start screen, follow the link to Examples. (It's in the Help category, toward the bottom of the screen.)

Step Click on the item named "Hello World" (Figure 2). CM3-IDE will create a new example program named "`hello`", and will take you to the package summary for the hello package.

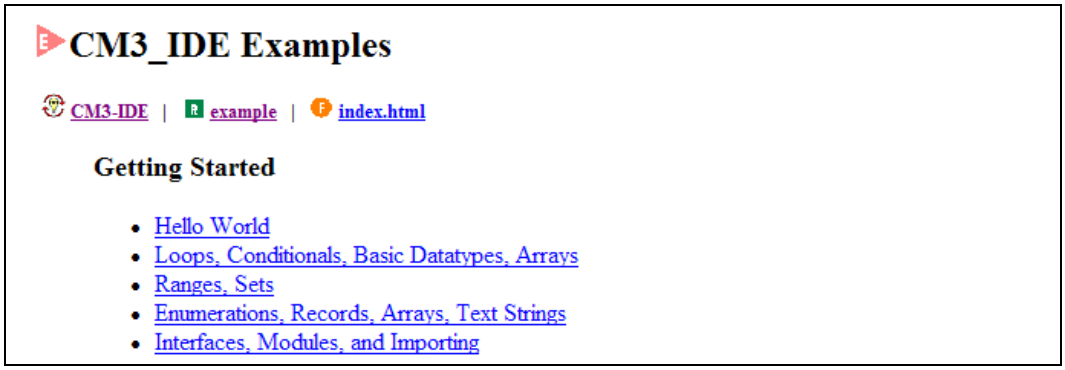


Figure 2. Examples Area

Packages

Using CM3-IDE, you divide your programming projects into *packages*. A package is a unit of ownership in the CM3-IDE system. A project consists of one or more packages. For large projects, different people may "own" different packages.

A CM3-IDE package comprises:

- zero or more modules
- zero or more interfaces
- a makefile (called "`m3makefile`")

The makefile tells the compiler how to put everything together. For simple programs, you may get away without having a makefile.

Each package has its own directory on your system, where all its source files are stored together.

Package Summary. A package summary page outlines different elements that comprise a package (Figure 3). You can follow the links on this page to view any of its components.

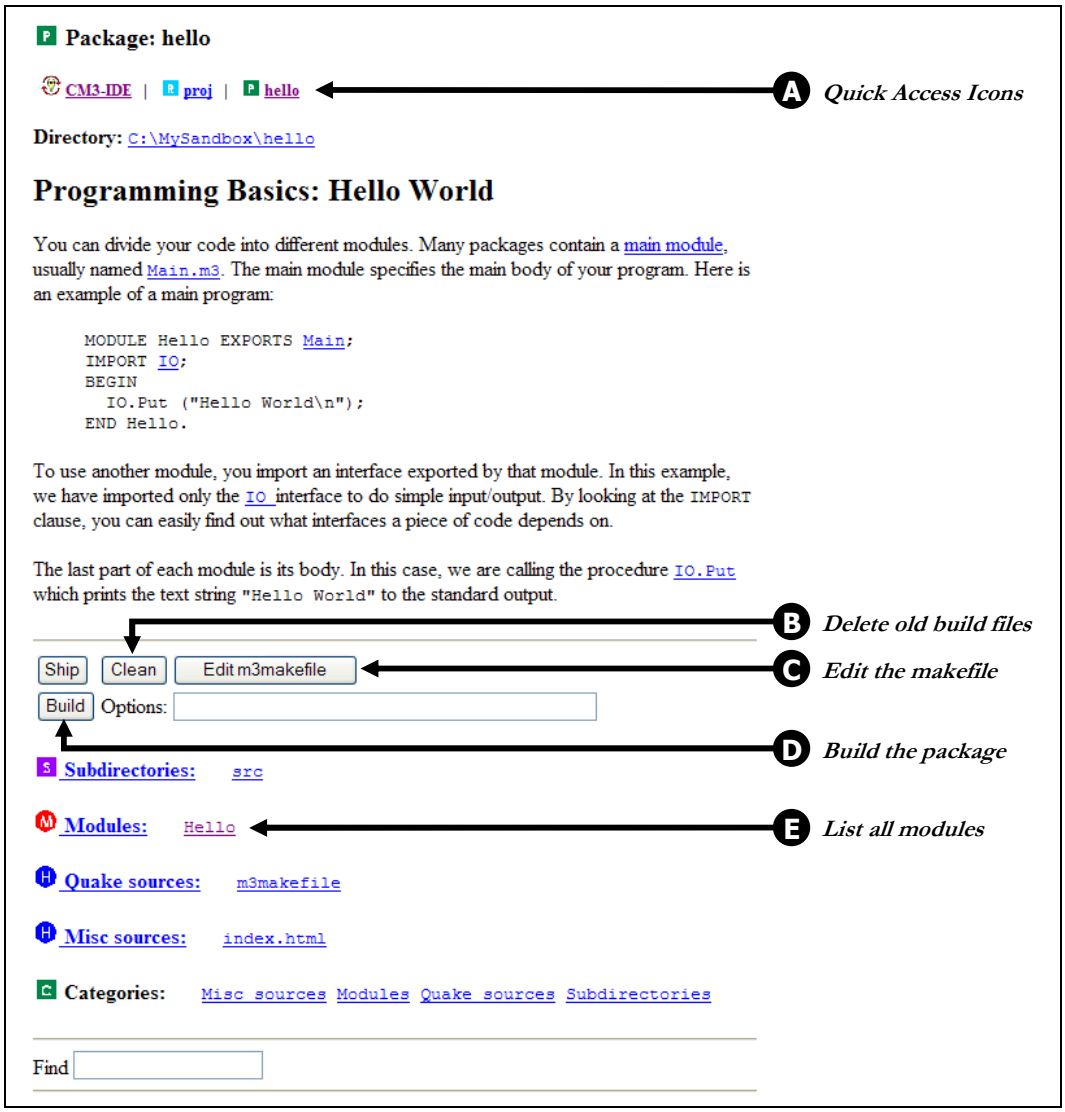


Figure 3. A Package Summary

At the top of the page, you'll see a row of Quick Access Icons. Below that, a button labeled Build, and below that, package components, such as Subdirectories and Modules.

A Use the **Quick Access Icons** to navigate to other locations in the CM3-IDE Environment. For more information, see **The CM3-IDE Environment** on page 29.

B Clean The **Clean** button tells CM3-IDE to delete files from previous builds. Clean does not remove sources of your program.

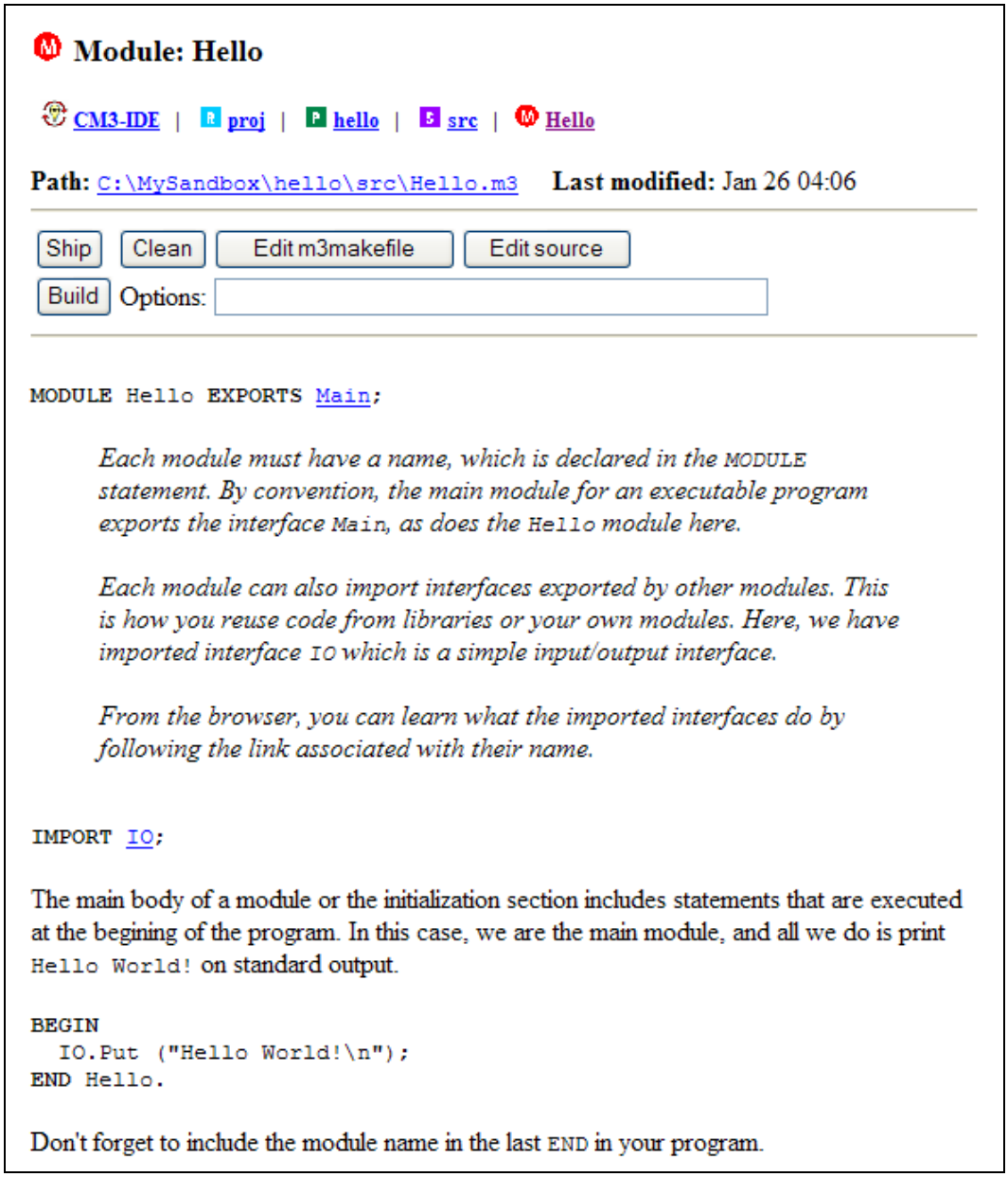
C Edit m3makefile The **Edit m3makefile** button starts your text editor and opens the makefile for this package.

D Build The **Build** button activates CM3-IDE's builder and uses the instructions in your makefile to build the package. If there is no makefile, CM3-IDE's builder will scan your package's directory tree and attempts to build a program based on that information.

E The **modules** available in this package are listed under the Modules heading. This page has only one entry—it's called Hello.

> **Modules**
>
> A *module* is a named collection of declarations, including constants, types, variables, procedures, and their associated bodies.

Step **Module Summary.** Next, follow the link Hello under the heading Modules to go to the summary page for the Hello module. Your browser will display a page titled "Module: Hello" (Figure 4).



Module: Hello

CM3-IDE | proj | hello | src | Hello

Path: C:\MySandbox\hello\src\Hello.m3 **Last modified:** Jan 26 04:06

Ship | Clean | Edit m3makefile | Edit source

Build Options:

```
MODULE Hello EXPORTS Main;
```

Each module must have a name, which is declared in the `MODULE` statement. By convention, the main module for an executable program exports the interface `Main`, as does the `Hello` module here.

Each module can also import interfaces exported by other modules. This is how you reuse code from libraries or your own modules. Here, we have imported interface `IO` which is a simple input/output interface.

From the browser, you can learn what the imported interfaces do by following the link associated with their name.

```
IMPORT IO;
```

The main body of a module or the initialization section includes statements that are executed at the begining of the program. In this case, we are the main module, and all we do is print `Hello World!` on standard output.

```
BEGIN
  IO.Put ("Hello World!\n");
END Hello.
```

Don't forget to include the module name in the last `END` in your program.

Figure 4. A Module Summary

Viewing Code of a Module. On this page, you can view the code for `Hello.m3`, the file containing the Hello module.

CM3-IDE is case-sensitive.

```
MODULE Hello EXPORTS Main;
IMPORT IO;
BEGIN
   IO.Put ("Hello World\n");
END Hello.
```

Note that CM3-IDE is case-sensitive. Keywords are always in upper-case.

Module Statement. The first line reads:

```
MODULE Hello EXPORTS Main;
```

This is the *module statement*. Each module must have a name; in this case the name is `Hello`. By including "`EXPORTS Main`" in the module statement, this module is considered to be the main module, i.e., the module containing the main body of the program.

Main Module of a Program

Every program must have a single main module, specifying its main body. The main module for your program exports the `Main` interface. This can be done either by naming your main module `Main`, or by including `EXPORTS Main` in the module statement for the main module.

Import Statement. The next line reads:

```
IMPORT IO;
```

This is an *import statement*. To use items defined in another module, you *import* an interface *exported* by that module. You do that by listing it here, in the import statements for your module. In this example, we have imported only the IO interface for doing simple output (Figure 5). By looking at the import statements for a module, you can easily find out what interfaces it depends on.

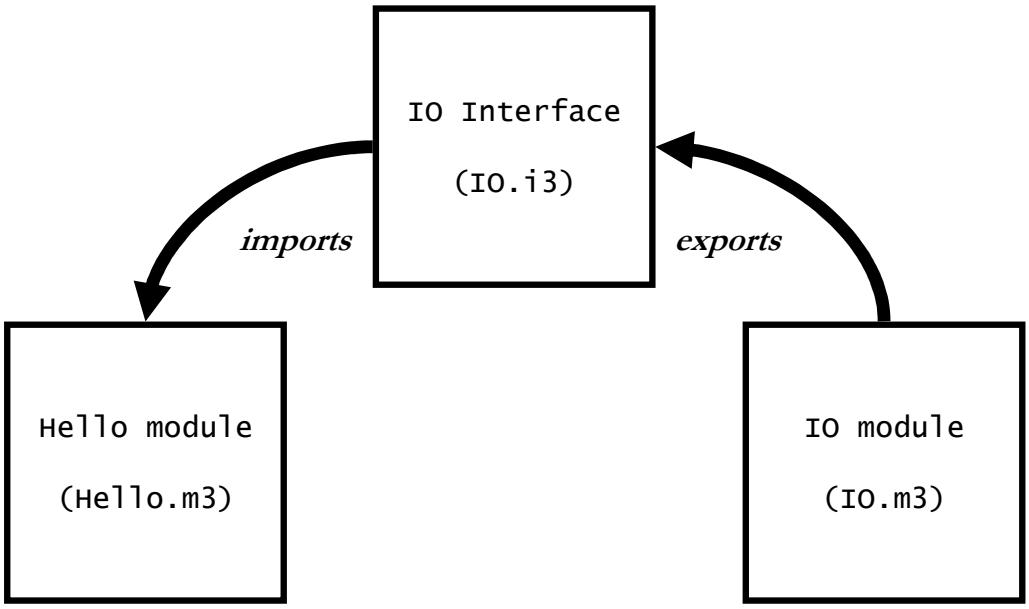


Figure 5. A Schematic of the IO Interface

The module body simply calls the procedure in the `IO` interface, denoted by `IO.Put`. `IO.Put` prints out the text string "`Hello World`", followed by a new line (`\n`) to the standard output.

```
BEGIN
   IO.Put ("Hello World\n");
END Hello.
```

Interfaces

An *interface* defines what parts of a module are visible to its clients. An interface can include declarations for types, procedures, constants, and variables.

Usually, the name for an interface matches that of the module that exports it; for example, the `IO` module exports the `IO` interface. (This does not have to be the case at all times.)

A useful way to think about an interface is as a window into the module that exports it.

Building a Package. This next step will produce an executable program by building the sources for the hello package.

Step Build Click the Build button from the module summary. This will start the builder, taking you to a Build Results page (Figure 6). From this page, you can view the output from your build. Errors will appear here as hypertext links to the line of code that generated them. With this example, you should not see any errors. If you do, retrace your steps up to this point.

Figure 6. Building a package.

Step **From Building to Running.** Once built, follow the hello link in the Quick Access Icons on top of the Build Results page. This returns you to the package summary for the hello package. (You can also use the "Back" button on your browser.)

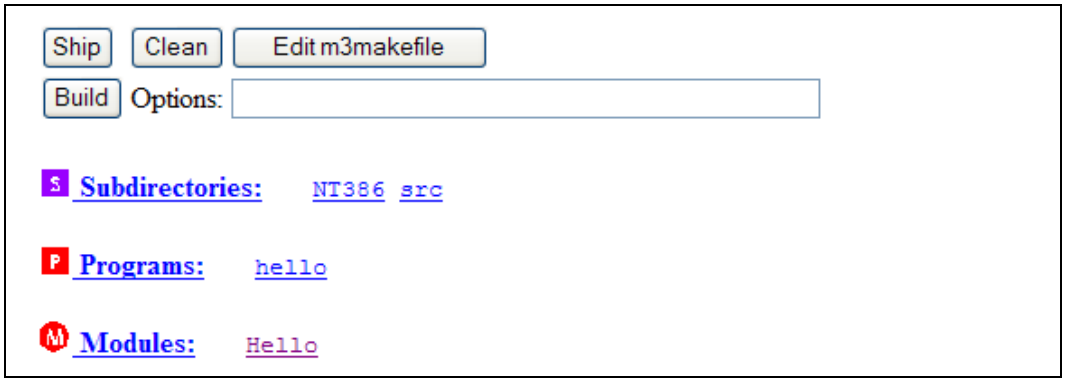


Figure 7. A Package Summary containing a Built Program

Step **Program Summary.** If you look at the bottom of the page, you'll notice a change: You should see a new category labeled Programs. Next to the icon you'll see the word "`hello`." This is the program you just built. Click on the word "`hello`" to navigate to the Hello program summary.

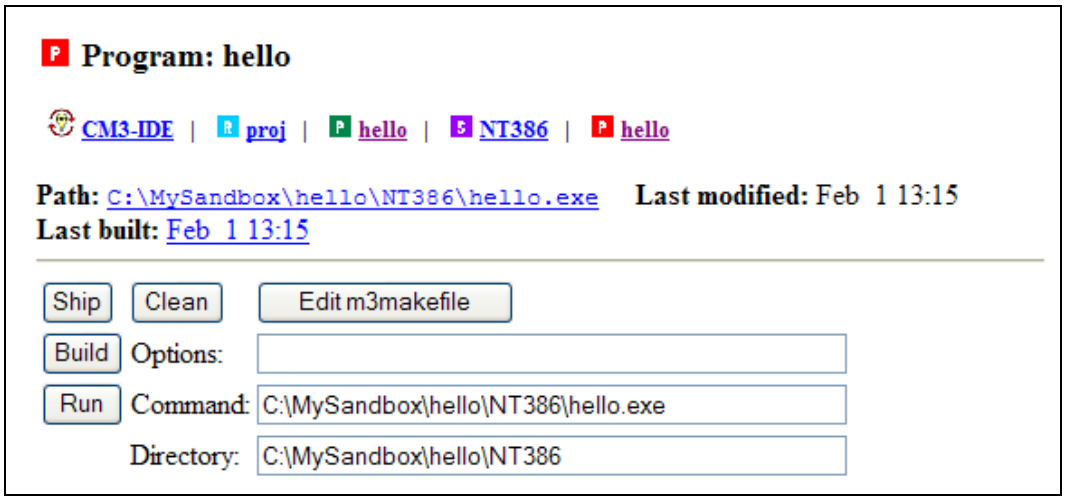


Figure 8. A Program Summary with a Run Button

Missing Program Icons in a Package Summary. If you don't see a program icon in your package summary, you may have to reload the page or click on the Rescan button (when available) to update CM3-IDE's browser view with the package contents on your file system.

If you still can't see a program icon, you probably did not build the package properly. (Perhaps you encountered a compilation or installation error.) Retrace your steps up to this point, making sure you followed the instructions correctly, and compare their results with the user guide.

Running a Program. Similar to other pages for a package, from the program summary page, you can navigate to the package top, or to any package components, or rebuild your program.

More importantly, you can run your program from this screen. (See Figure 8.) The Run button is directly underneath the Build button. Next to the Run button is a type-in field where you can enter the text as you would on a command line. (CM3-IDE should have already done this for you.) Beneath that is a text box containing the path to the directory in which your package resides.

Step Click the Run button now. Your program will run, and your browser will display the result of the execution of the program. In this case, you should see the text "`Hello World`" appear in the program results page. (See Figure 9.)

Figure 9. Running "Hello World".

You've just built and run your first CM3-IDE program.

You may use the CM3-IDE icon at the top left of the page you are on to return to the CM3-IDE start screen.

1.3 Creating a Package From Scratch

In the first tutorial you used a ready-made package that comes with CM3-IDE. All you had to do was navigate to it, build and run it. This time, you'll create a new package, open your text editor from CM3-IDE, add some code, and take a look at a very basic makefile.

1.3.1 List of All Packages

Step

From the CM3-IDE Start Page, follow the link to Packages to see a list of all available packages that are currently available within CM3-IDE. (See Figure 10.)

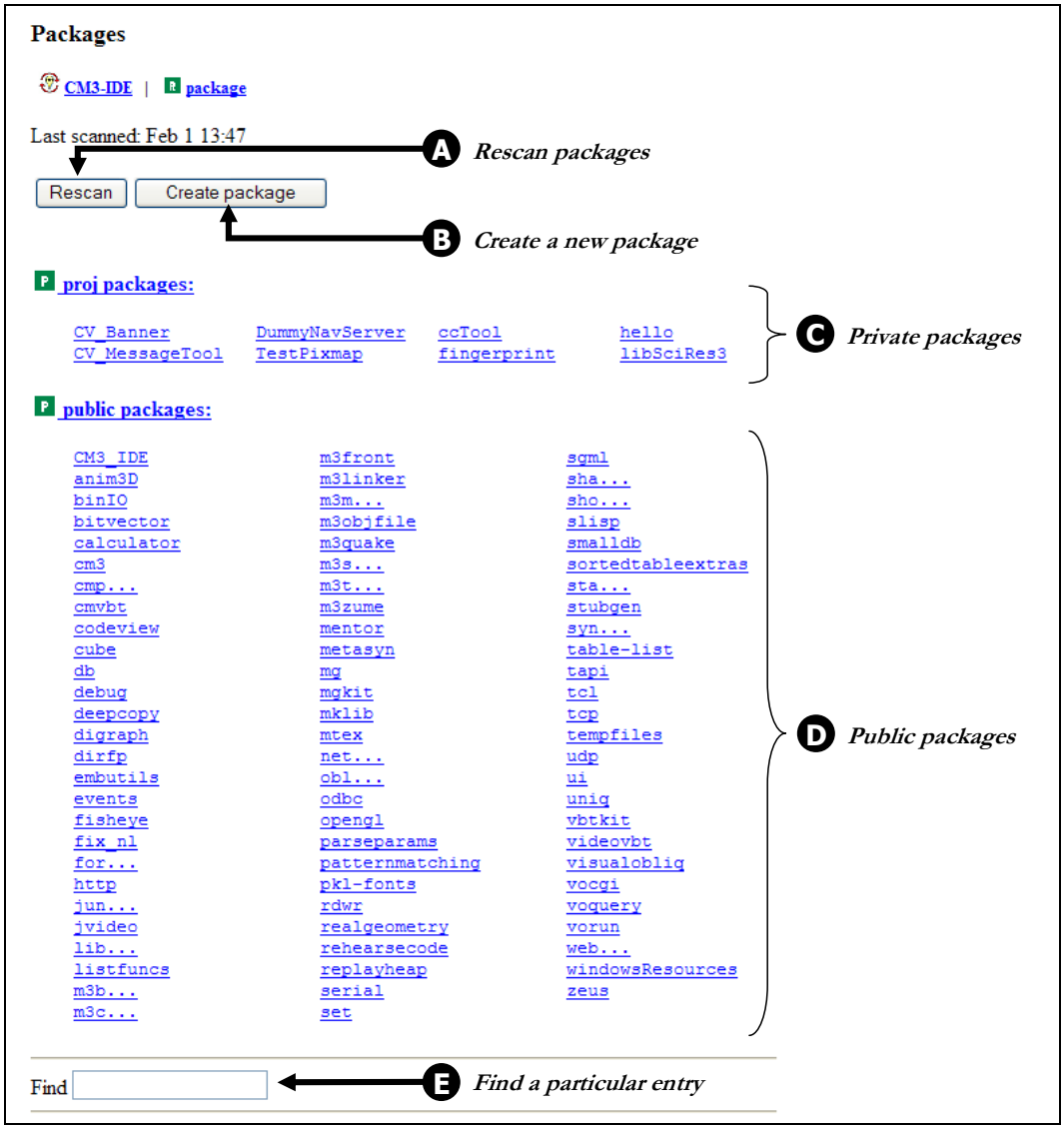


Figure 10. Packages Page: Listing of All Packages in CM3-IDE

Each (highlighted) package name represents a link to the summary page of a particular package. Some of the functions available on this page are:

A Rescan tells CM3-IDE to update its database from the files in your filesystem.

B Create New Package takes you to the New Package dialog, where you can create a new package.

C Your private packages are normally filed under "`proj`."

D Public packages are normally filed under "`public`."

E The Find type-in field instructs CM3-IDE to only list entries that match a particular regular expression, such as "`m*`".

1.3.2 Creating New Packages

Step

Navigating to the Create Package dialog. Near the top of the page, you should see two buttons. (See Figure 11.) Click the right button, "Create package". CM3-IDE will take you to the Create Package dialog.

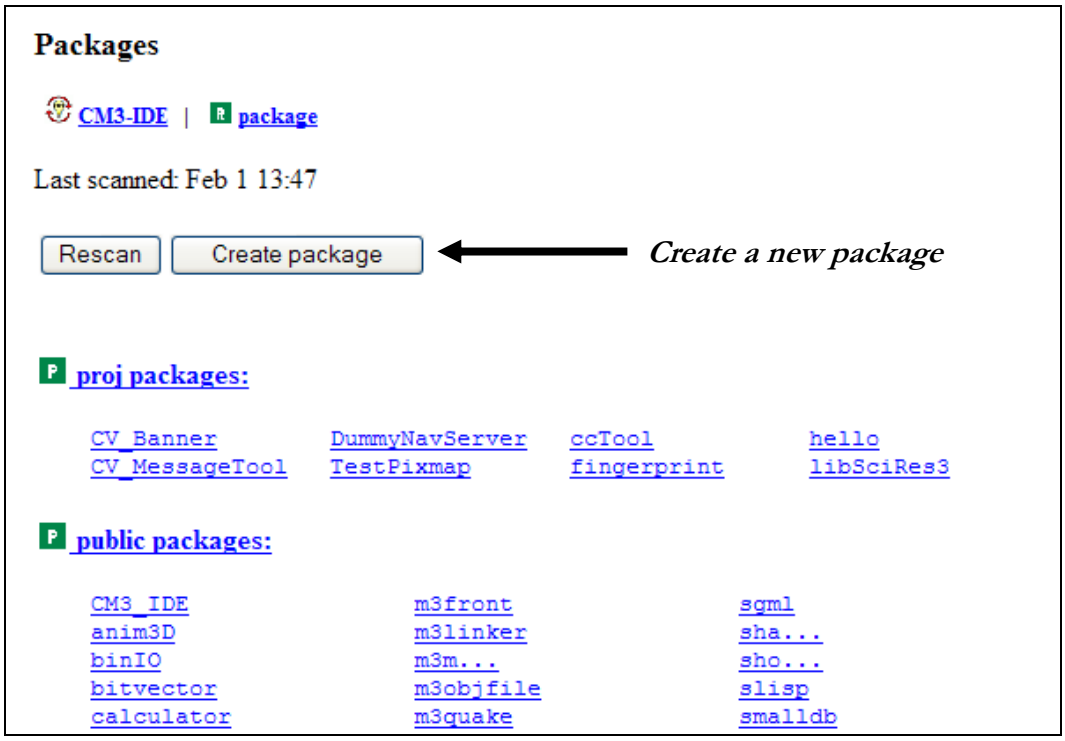


Figure 11. Top of the Packages Page

The Create Package dialog. Before CM3-IDE can create a package, you need to specify some information about the package to be created (Figure 12).

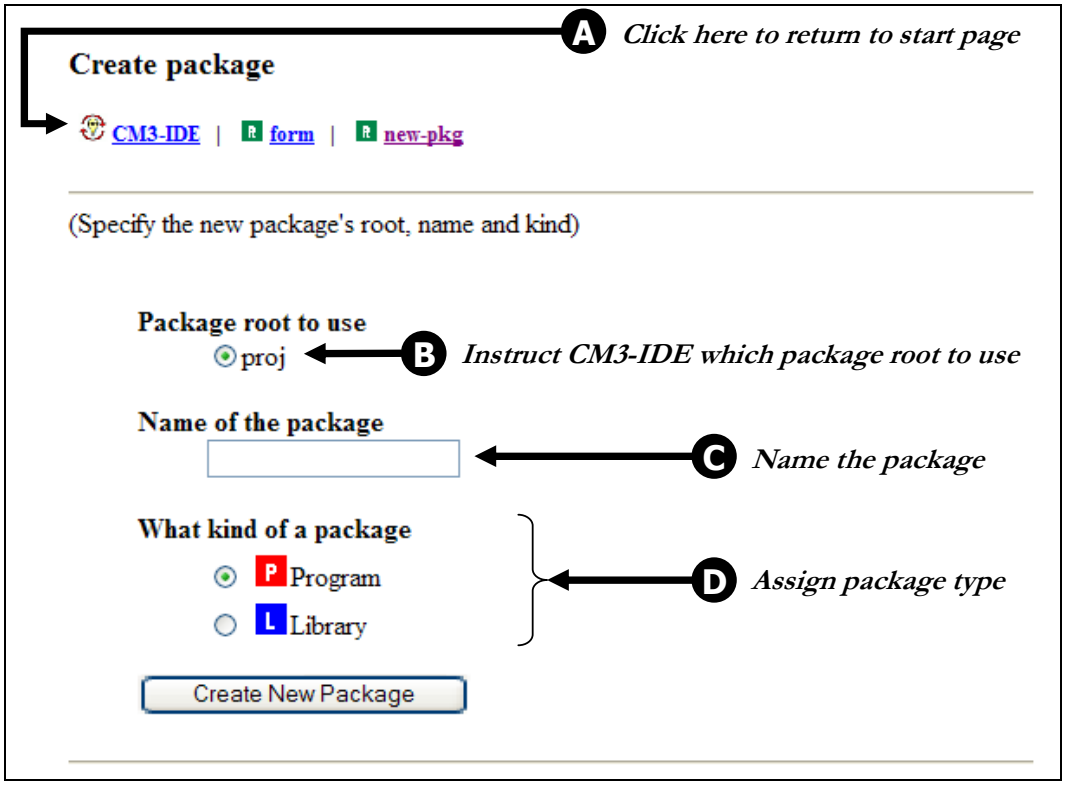


Figure 12. Package Creation Dialog

A To return to CM3-IDE's start page, follow the CM3-IDE link.

B Package roots are used by CM3-IDE to organize your packages. Before CM3-IDE can create any package, it needs to know what root that package will reside in. The package root "`proj`"—where your private packages reside—is a good place for this example package. Choose proj. proj

Step

Step **C** Enter the name of your package here. Under "Name of the Package", enter "`MyPackage`".

D When you create a new package, you're given the option of creating a library and program. You are not locked into your decision here. Under "What Kind of Package," select Program.

Step

Step Click on the "Create new package" button at the bottom of the screen. CM3-IDE will create your package and point your browser to the package summary for the new package.

Package Summary. You have just created the package MyPackage, but it doesn't do anything yet. You will need to write some code.

Step Look for the Modules icon on the part of the page where the program elements are summarized. You'll see that there is only one module listed there, named "MyPackage". Click on it.

Module Summary. You may remember this page from our previous example. A module summary contains the code and relevant information regarding a module in a package. About half-way down the page you should see the module's code:

```
MODULE MyPackage EXPORTS Main;
BEGIN
END MyPackage.
```

Every CM3-IDE program must have a single main module, so when you tell CM3-IDE to create a program, CM3-IDE starts you off with an empty main module. Notice that there is no IMPORT statement here, and there is nothing between the keywords BEGIN and END. You will need to supply these.

Step **Coding a Module.** Near the top of this page, find the row of action buttons. Click on Edit source. CM3-IDE should start your text editor and open the file MyPackage.m3.

Figure 13. The top of the Module Summary Page

Step In your text editor, add the following line between the line containing the word MODULE and the line containing the word BEGIN:

```
VAR name: TEXT; (* a string variable called "name" *)
```

This line uses the keyword VAR to declare a variable called "name", to be a string. The line ends with a semi-colon. Comments in CM3-IDE begin with "(*" and end with "*)"

Step Add the following lines between the line containing "BEGIN" and the line containing "END":

```
IO.Put("Enter the name of your nemesis: ");
name := IO.GetLine();
IO.Put(name & " is a stupidhead.\n");
```

The first line calls the procedure `IO.Put` passing the string "`Enter the name of your nemesis:`".

The second line calls the procedure `IO.GetLine` and puts the string returned from that procedure into the `TEXT` variable you declared above, `name`.

The third line calls `IO.Put` again, passing it the string in the parentheses that follow.

1.3.3 Procedure Calls in CM3-IDE: What is "IO.Put?"

The expression `IO.Put` refers to a procedure `Put` in an interface called `IO`. `IO.Getline` refers to the `Getline` procedure in the interface `IO`. (See Figure 14.)

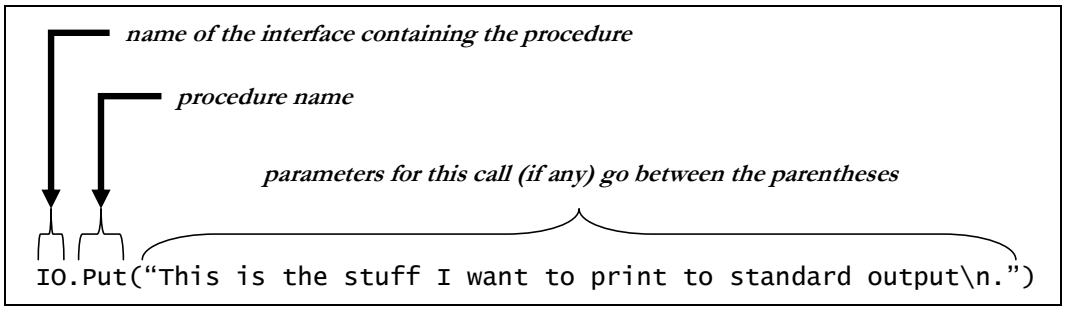


Figure 14. A Procedure Call Crossing Module Boundaries

The identifier to the left of the "." is the name of the interface in which the procedure is declared. The one to the right of the "." is the name of the procedure. Parameters for this call go inside the parentheses that follows the procedure's name.

Interfaces in CM3-IDE are used to bundle relevant procedures, types, and constants in one syntactic unit. For example, the `IO` interface includes all the procedures you may need for simple input/output.

1.3.4 The IO Interface

Some of the procedures in the `IO` interface will be used in the package you are building. Before using a procedure, you may want to make sure you are calling the correct procedure by reviewing the interface where it is defined. The next few steps show how you can explore the `IO` interface from your current package.

Step **Navigating to the IO Interface.** Leave your text editor open and return to your web browser. Your browser should still show the module summary for `MyPackage`.

Step Click on the CM3-IDE icon at the top of the page to return to the start screen.

Step On the start screen, find the Interfaces icon and click on it. to navigate to a list of all available interfaces. Find and click on the word "**IO**." (Depending on your CM3-IDE display settings, you may have to click on an "**I...**" entry first.) This should bring you to the "**IO**" interface (Figure 15).

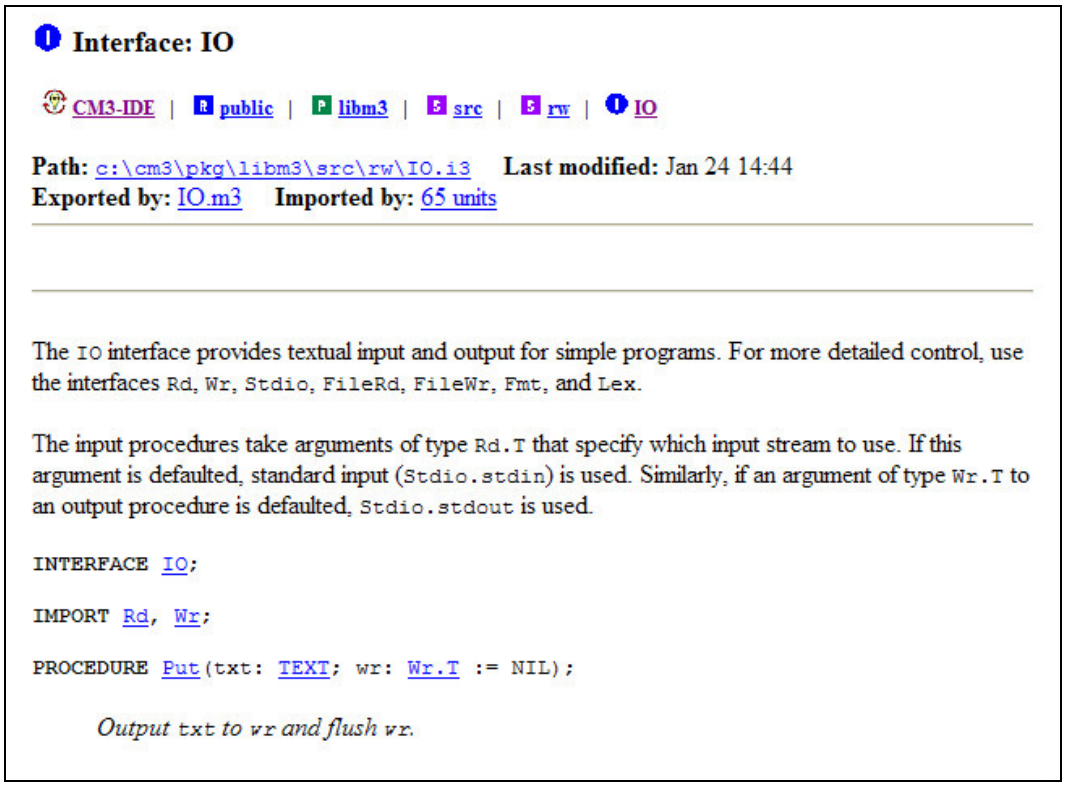


Figure 15. The top of the IO Interface

Look at the top of interface **IO**. Beneath the Quick Access Icons, you'll see information about the path, last modified date, and import and export lists. You can use the links under "Imported by" to find out what units use this interface, or the links under "Exported by" to see where the procedures declared here are defined.

Further down the page is a list of elements declared in the **IO** interface (Figure 16). The procedure **Put** is at the top of the list, and **GetLine** is fifth from the top; these are the procedures used in **MyPackage**. Notice that the procedure names here are highlighted.

```
INTERFACE IO;

IMPORT Rd, Wr;

PROCEDURE Put(txt: TEXT; wr: Wr.T := NIL);

    Output txt to wr and flush wr.

PROCEDURE PutChar(ch: CHAR; wr: Wr.T := NIL);

    Output ch to wr and flush wr.

PROCEDURE PutWideChar(ch: WIDECHAR; wr: Wr.T := NIL);

    Output ch to wr and flush wr.

PROCEDURE PutInt(n: INTEGER; wr: Wr.T := NIL);

    Output Fmt.Int(n) to wr and flush wr.

PROCEDURE PutReal(r: REAL; wr: Wr.T := NIL);

    Output Fmt.Real(r) to wr and flush wr.

PROCEDURE EOF(rd: Rd.T := NIL): BOOLEAN;

    Return TRUE iff rd is at end-of-file.

EXCEPTION Error;

The exception Error is raised whenever a Get procedure encounters syntactically invalid input,
including unexpected end-of-file.

PROCEDURE GetLine(rd: Rd.T := NIL): TEXT RAISES {Error};

    Read a line of text from rd and return it.

A line of text is either zero or more characters terminated by a line break, or one or more characters
terminated by an end-of-file. In the former case, GetLine consumes the line break but does not
include it in the returned value. A line break is either {\tt "\n"} or {\tt "\r\n"}.
```

Figure 16. Some Procedures Defined in the IO Interface

Step Click on the name of the procedure **`GetLine`**. Your browser will display the module summary page of the **`IO`** module, which contains the code for this procedure.

Now, you know what **`IO.GetLine`** and **`IO.Put`** do.

Wrapping up the code. Return to your text editor. Insert the cursor immediately below the line:

```
MODULE MyPackage EXPORTS Main;
```

Step Type:

```
IMPORT IO;
```

The file in your text editor should now read as follows:

```
MODULE MyPackage EXPORTS Main;
IMPORT IO;
VAR name: TEXT; (* string variable called "name" *)
BEGIN
   IO.Put("Enter the name of your nemesis: ");
   name := IO.GetLine();
   IO.Put(name & " is a stupidhead.\n");
END MyPackage.
```

What did you just do? You have imported the `IO` interface in your module, so that you can access the two procedures declared inside it: `IO.Put` and `IO.GetLine`.

Modules and Interfaces: Importing and Exporting

You can control how modules and interfaces interact through `IMPORT` and `EXPORT` statements.

A *module* defines a collection of program elements. These elements could be constants, types, variables, or procedures. The module *exports* an interface to make some of its component elements available to *clients*.

An *interface* is a list of the elements to be made available. Any file that `IMPORT`s an interface is said to be a *client* of the interface. The `MyPackage` module used in this example is a client of the interface `IO`.

You can only access the procedures contained in a module by importing an interface that was exported by that module. To use `IO.Put` and `IO.GetLine`, you needed to import the `IO` interface. You did that by inserting "`IMPORT IO`" right after the module declaration.

Back to the Package. Now that you have reviewed the `IO` interface, it is time to build your program.

Step Save your file, `MyPackage.m3`, and quit your text editor.

Step In your web browser, find "Build Package: MyPackage" in your browser's history (usually listed under the "Go" menu.) Or just hit the "Back" button of your browser enough times to get back to the `MyPackage` summary.

In the next few steps, we will quickly review the makefile for this project. Then we will build and run the program.

1.3.5 CM3-IDE Makefiles (m3makefile)

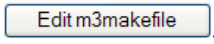
A CM3-IDE makefile is named m3makefile.

A CM3-IDE makefile is named `m3makefile`. A makefile is a text file containing instructions that tell CM3-IDE's builder how to build a program or a library. An instruction is followed by one or more arguments in parentheses, similar to a procedure call in a programming language. Indeed, to build your package, CM3-IDE interprets your `m3makefile` as a little program.

Each instruction may specify a library, interface, or module to be included as part of the build. Comments in makefiles start with `%` and continue to the end of line.

For simple programs you can omit the makefile, and the builder will automatically find your modules and interfaces and their dependencies. However, creating makefiles for CM3-IDE packages is a good idea in general, especially since they are easy to create.

The button on the far right of the row of buttons near the top of the "a package summary" page is labeled "Edit m3makefile":

Step Click on 

. CM3-IDE will start your text editor, and open the file "`m3makefile`". Here is what you should see in your text editor:

```
% Makefile for MyPackage
import("libm3")
implementation("MyPackage")
program("MyPackage")
```

When you create a new package, CM3-IDE automatically creates a basic makefile for you. As your package grows and becomes more complex, it is up to you to make sure your makefile is up-to-date, though doing so is straightforward.

Let's take a look at the makefile for this package, line by line.

The first line:

```
% Makefile for My Package
```

is a comment. The rest of the line after `%` is ignored by CM3-IDE.

The second line:

```
import("libm3")
```

is a *makefile import command*. The `import` command tells the compiler that the program uses routines in the standard library, `libm3`. That's the library that contains the `IO` interface and module.

Libraries

In CM3-IDE, a *library* is a package whose code may be reused as part of another library or an executable program. To use functionality of a library, you must import it in your makefile.

To learn more about libraries see Chapter 3, **Building And Sharing Packages** on page 47. To see a list of available libraries in CM3-IDE, click the L Libraries icon on the start screen.

Most makefiles include one or two import commands. If you use routines from other libraries, you must include other import commands that tell the compiler which libraries to include.

The third line:

```
implementation("MyPackage")
```

marks the program `MyPackage.m3` as a module implementation for your package. In your makefile, there must be one `implementation` command for each "`.m3`" file in your program. In this case, there was only one such file: `MyPackage.m3`.

Finally, the last line:

```
program("stupidhead")
```

tells the compiler to name the resulting executable file "`stupidhead`". On Windows, executables have an "`exe`" extension.

Step Quit your text editor, and, if you've modified your makefile, make sure you don't save the changes. The makefile is fine as it is.

Step Click the Build button in the package summary for `MyPackage`. CM3-IDE will build your program, and point you to the Build Results page for `MyPackage` which will show any compiler error messages (in this case, you should not have gotten any) and warnings (which you may ignore for the moment.)

Your program is now ready to run. This program, however, is a bit more interactive than the one in this chapter's first tutorial. You will need to run this one from a

command-line prompt. Once CM3-IDE has created an executable, you can run it directly from the operating system. This is what we'll do with this program.

Step Click the MyPackage icon in the Quick Access Icons area, or on the Back button of your browser to return to the package summary for `MyPackage`. You should see a link to the `stupidhead` program next to the Programs category. If you don't, click the Reload button of your web browser.

Step Click on the name of your new program to go to its program summary. At the top of that page, immediately beneath the Quick Access icons, you can read the location of the new program in your file system, right after "Path:".

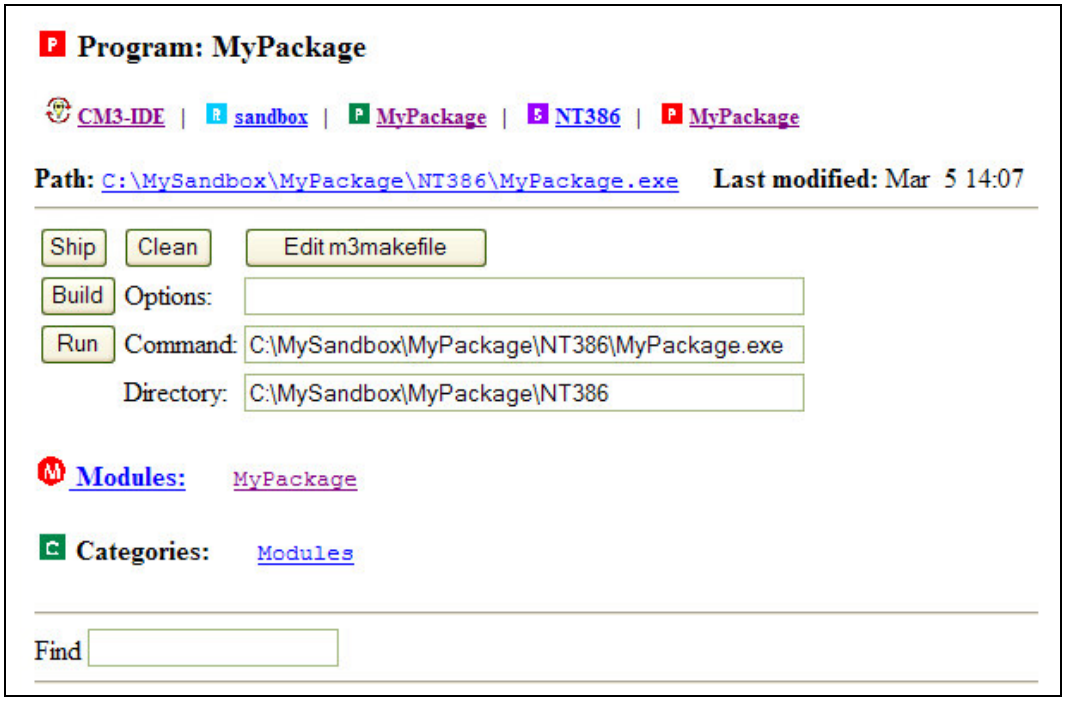


Figure 17. The top of a Program Summary page

Step At the command-line, change your working directory to the one containing the executable. Type "`MyPackage`" at the shell prompt to run the program. Here is what you should see in your system window:

```
Enter the name of your nemesis:
```

Step Do what it says; type the name of your nemesis here. If you enter "`My boss`" the program will write:

```
My boss is a stupidhead.
```

at the shell window and exit. What an intelligent and well-conceived program!

1.4 Summary

In CM3-IDE, projects are divided into *packages*. A project can consist of one more packages. A CM3-IDE package comprises one or more modules and interfaces, along with a makefile that tells the compiler how to put everything together. Unlike their ancestors, CM3-IDE makefiles don't need dependency definitions.

Modules and *interfaces* are the building blocks of a CM3-IDE package. A module is a named collection of declarations, including constants, types, variables, and procedures. An interface can be thought of as a window into a module's functionality. To use another module, you *import* an interface that was *exported* by that module.

CM3-IDE is case-sensitive.

All keywords in CM3-IDE are capitalized.

Both interfaces and modules may use *import statements*. By looking at the import statements for a module, you can easily discover its dependencies on other interfaces.

The basic form of a module and an interface is:

```
MODULE module-name;          INTERFACE interface-name;
IMPORT intf-1, intf-2,…;     IMPORT intf-1, intf-2,…;
Declarations;
BEGIN                        Declarations;
   Statements;
END module-name.             END interface-name.
```

Statements terminate with a semicolon ("`;`"). Comments begin with "`(*`" and end with "`*)`".

CM3-IDE makefiles, usually named `m3makefile`, define the steps for building a package. Here is a simple makefile:

```
% Makefile for a simple package
import("libm3")
implementation("module_name")
program("program_name") or library("lib_name")
```

Comments in makefiles start with "`%`" and continue to the end of the line. The call "`program`" at the end of a makefile marks that this package should be built as an executable program; the call "`library`" means this package should be built as a reusable library.

This page left blank intentionally.