Newton Application Development

Newton C++ Tools User's Guide

Release 1.0

May 16, 1996 Jonathan Simonoff

Contents

About This Book

This book, *Newton C++ Tools User's Guide*, describes programmer software used to prepare C++ code for inclusion in NewtonScript programs. It has the following chapters:

- Chapter 1, "Introduction," gives background information.
- Chapter 2, "Installing the Newton C++ Tools," tells you how to install the Newton C++ Tools.
- Chapter 3, "How to Use the Newton C++ Tools," steps you through using the C++ Tools, and includes a tutorial example.
- Chapter 4, "The Newton C++ Tools MPW Environment," describes the menu commands that the Newton C++ Tools adds to MPW.
- Chapter 5, "MPW Tools," discusses the MPW tools included with the C++ Tools.

Conventions Used in This Book

This book uses the following conventions to present various kinds of information.

Special Fonts

This book uses the following special fonts:

- **Boldface**. Key terms and concepts appear in boldface on first use.
- `Courier typeface`. Code listings, code snippets, and special identifiers in the text such as slot names, function names, method names, symbols, and constants are shown in the

Courier typeface to distinguish them from regular body text. If you are programming, items that appear in Courier should be typed exactly as shown.

- *Italic typeface*. Italic typeface is used in code to indicate replaceable items, such as the names of function parameters, which you must replace with your own names. The names of other books are also shown in italic type, and *rarely,* this style is used for emphasis.

Introduction

The Newton C++ Tools package allows you to write C++ code that can be called from a NewtonScript program. You need to understand how Newton devices are programmed before you can use this product.

In order to use the C++ Tools, you need:

- Newton Toolkit (NTK)
- Apple Macintosh or other Mac OS-based personal computer with a 68030 or later processor, 33Mhz or faster. Any PowerPC Mac OS-based computer will work.
- Mac OS version 7.1 or later.
- At least 12 megabytes of RAM. If you have a PowerPC computer, 20 megabytes are recommended. You need 20 megabytes to use MPW, NTK, and Newtsbug simultaneously.
- A CD-ROM drive
- A hard disk with at least 25MB of available space
- A Newton device running the Newton operating system version 2.0 or later
- Newton Connection Kit for Macintosh v2.0 - Serial cable or Macintosh System Peripheral 8-Cable (M0197LL/B)

You develop C++ code using MPW. You then add the code into an NTK project. The C++ functions can then be called from NewtonScript.

The C++ Tools package is designed so that you do not need to know much about MPW. If you want to know more, MPW documentation is included on the C++ Tools CD. If you want to use all of the capabilities of MPW, you should order a full version of MPW, including documentation, from APDA® (Apple's source for developer tools). You can order MPW, NTK, or any other APDA product through these telephone numbers and addresses:

From the USA	1-800-282-2732
From Canada	1-800-637-0029
From other Countries	(716)871-6555

Fax	(716)871-6511
AppleLink	APDA
Internet	APDA@applelink.apple.com
CompuServe	76666,2405
America Online	APDAorder
Regular Mail	APDA Apple Computer, Inc. P.O. Box 319 Buffalo, NY 14207-0319 USA

Porting C++ Code to the Newton

You can use the C++ Tools to help port programs to the Newton. However, only computational code can generally be ported to the Newton in the form of C++ code. Newton devices have a user interface that is quite different from desktop computers, so you generally cannot port user interface code. Similarly, Newton devices do not have a file system in the same sense desktop computers do, and memory management issues are somewhat different. In order to take advantage of the Newton's built-in software for the user interface, data storage, and memory management, you generally need to write the code for those portions of your application in NewtonScript.

There are also limitations in how C++ code can be written. See the manual *Newton C++ Tool*s for more information on those limitations.

Files and Directories

The C++ Tools and MPW require certain file and directory names and a certain directory structure in order to operate properly. You may want to come back to this section after installing the Newton C++ Tools and going through the tutorial in Chapter 3, "How to Use the Newton C++ Tools."

File and Folder Naming Conventions

The MPW scripts require that files containing C code end in .c, while files containing C++ code end in .cp.

MPW can be confused by certain characters. In general you should limit your folder and file names to letters, digits, spaces, underscores, minus signs, asterisks (*), • (option-8), and π (option-p). Some other characters are also allowable, but certain ones such as ", ', and ∂ definitely are not. See the MPW documentation for information if you want to use unusual characters.

Directories

The C++ Tools files and directory go into your MPW directory in the places indicated by Chapter 2, “Installing the Newton C++ Tools.”

You can place the Newtsbug directory anywhere you want. We recommend, though, that you do not put it in the MPW directory. However, Newtsbug has to be in the MPW command path. The easiest way to make sure it is is to put an alias to Newtsbug in the MPW:NCT_Folder:Tools folder, as specified in the installation instructions. The name of this alias must be Newtsbug.

Your code goes into another directory structure that is contained within a root directory. The root directory that is included with the C++ Tools is the NCT_Projects directory. You normally do your code development in the directory hierarchy contained in the NCT_Projects directory.

The C++ Tools build system allows you to build a two-level structure inside a root directory. The first level consists of **_NCT directories** which have names like Calculator_NCT and Maps_NCT. Each _NCT directory contains one or more **projects**. The contents of project directories are described in the next section.

You do not have to organize your NTK projects this way, but the Newton C++ Tools is set up to have your C++ projects set up in this way.

The Contents of the Project Directories

The project directories contain:

- C++ code files
- NTK files that specify Newton applications (layout files and project files)

IMPORTANT

Development of the NTK part of a project needs to take place in the same directory as you use for C++ development, so that the debugger can find the files it needs. ▲

- Items that the Newton C++ Tools uses to build the package. Those items are:
  - FilesInBuild. This file lists the files that are in the build. It is used by the Build Project Makefile command to create a makefile for the project. If you create your own makefile, you do not need this file.
  - *Project*.exp. (The first part of the name is the same as the name of the project directory.) You put in information in this file that identifies the functions you want to call from NewtonScript.
  - Objects. This directory is used for the object files created and used in your build.
  - Makefile. This is the makefile for the project.

When you use the Create New Project command from the Newton C++ Tools menu in MPW, the Newton C++ Tools creates prototypes for FilesInBuild and *project*.exp and also creates the Objects directory. The Build Project Makefile creates the makefile.

Note

The Newton C++ Tools menu in MPW lists all the C++ Tools projects in the current projects directory. It finds those by looking for directories in the current projects directory that end in _NCT and then looking in each sub-directory for a FilesInBuild file or a Makefile. It lists only those directories that contain one or both of those files. ◆

Installing the Newton C++ Tools

How you install the Newton C++ Tools depends on whether or not you already have MPW installed. We recommend MPW version 3.4 or later.

Insert the Newton C++ Tools CD into your CD drive.

IMPORTANT
The Newton C++ Tools is an add-on product to NTK. You must have NTK installed in order to use it. ▲

Installing MPW

To install MPW:

1. Open the Latest MPW folder.
   It contains two folders: MPW and System Additions.
2. Drag the folder MPW to your hard disk.

If you do not have a PowerPC computer you are done with installing MPW. If you have a PowerPC computer, you need to install an extension to use MPW:

1. Open the folder System Additions.
2. Open the folder Required for MPW.
3. Drag the extension in that folder, StdCLibInit, to your system folder.
4. Restart your computer.

Installing the Newton C++ Tools

1. Drag the NCT_Projects directory to your hard disk.

2. Open the MPW Additions folder.
3. Drag the top three items, NCTBuild•UserPrefs, UserStartup•NewtonC++Tools, and NCT_Folder into your MPW folder.
4. Drag the Includes and Libraries folders into the NCT_Projects folder on your hard disk.

Installing Newtsbug

You use Newtsbug to debug C++ programs on the Newton. To install Newtsbug:

1. Drag the Newtsbug folder to your hard disk.
2. On your hard disk, make an alias to the Newtsbug program that is in that folder.
3. Open the NCT_Folder on your hard disk.
4. Drag the alias to Newtsbug into the Tools folder that you find in the NCT_Folder. Change the name of the alias from

   ```
   Newtsbug alias
   ```

   to

   ```
   Newtsbug
   ```

The Newton C++ Tools build system is now installed.

How to Use the Newton C++ Tools

This chapter steps you through using the C++ Tools, MPW, and NTK to:

- Create a small C++ function
- Combine it with a Newton application that calls the function

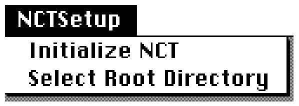
Starting the C++ Tools for the First Time

The C++ Tools MPW start-up files need to know where on your disk you'll be working. It uses that information to build it's menus.

1. Find the MPW folder and open it.
2. Double-click on the MPW icon.

   The start-up script begins executing. After a few moments, you see a dialog box that tells you you need to select a root folder. The C++ Tools needs a root directory. It looks in that directory for your projects.
3. Click OK to dismiss the dialog box.
4. Pull down the NCTSetup menu. It is shown in Figure 3-1.

Figure 3-1 NCTSetup Menu



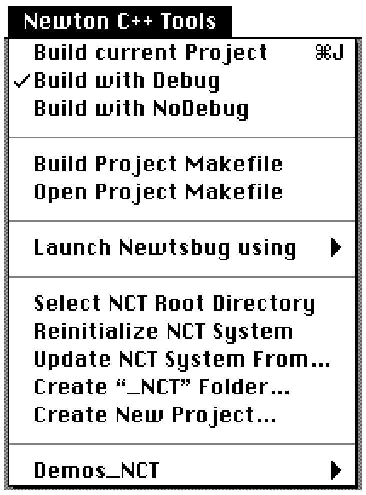
5. Choose the Select Root Directory command.

MPW displays a standard file dialog box.

6. Find the NCT_Projects directory and choose it.

   The C++ Tools start-up files continue processing. They build the Newton C++ Tools menu. When processing is done, the menu should look like the one shown in Figure 3-2.

Figure 3-2 Newton C++ Tools Menu



Creating a Project

The basic unit of Newton application development is a **project**. A project contains various files used and produced by NTK as well as the files used and produced by the C++ Tools. You do all C++ and NewtonScript development for an application within a single project.

Before you create a project, you need to create a place to put the project. The C++ Tools allows for a three-level hierarchy, which is intended to help you keep multiple projects neatly organized:

- The top level of the hierarchy is the root directory, which was the directory you chose in the preceding section; if you want to create an additional root directory, do it from the Mac OS Finder. You normally just use the NCT_Projects directory, because it contains the includes and Library files that you need.
- The second level of the hierarchy is made up of folders with names that end in _NCT. These folders have special meaning to the C++ Tools, as you will soon see. You use a menu command described in this section to create these folders.
- The lowest level of the hierarchy is made up of project folders. You use a menu command described in this section to create those folders, also.

Look again at the menu in Figure 3-2. The last item in the menu is Sunny_NCT. This is an NCT folder that contains the Sunny application, which is a sample Newton application you can build and use. If you look at the submenu contained in the Sunny_NCT item, you will see that it contains one project, Sunny.

Creating an _NCT Folder and a Project Folder

You now need to create a new _NCT folder to hold the new example application, and then create a project for the application.

1. Choose the Create "_NCT" Folder command from the Newton C++ Tools menu.
   A dialog box displays.
2. In the dialog box, type:

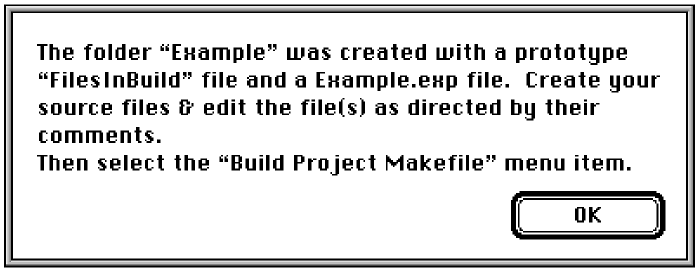
   ```
   Example
   ```

   Notice that you do not type the _NCT ending—the C++ Tools adds that for you.
   You can have any number of _NCT folders.
3. Click in the OK button.
4. Choose the Create New Project command from the Newton C++ Tools menu.
   A dialog box displays.
5. In the dialog box, type:

   ```
   Example
   ```

   This name doesn't have to be the same as the _NCT folder name.
6. Click in the OK button.
7. A dialog box asks you where to put the new project. Choose Example_NCT.
   You've created a new project. Each _NCT folder can contain any number of projects.
   The C++ Tools informs you that you've created a new project by showing the dialog in Figure 3-3. The dialog tells you what the C++ Tools has done and gives general directions for what you need to do to continue.

Figure 3-3 New Project Dialog


The folder "Example" was created with a prototype "FilesInBuild" file and a Example.exp file. Create your source files & edit the file(s) as directed by their comments.
Then select the "Build Project Makefile" menu item.

Every C++ Tools project needs a .exp file. That file lists your module name and function names, and defines the interface used to call your functions from NewtonScript. The FilesInBuild file lists the files that make up the C++ part of your project. You need to have this file in order to use the C++ Tools command that creates the build instructions (makefile) for your project. When you create a new project, the C++ Tools creates prototypes for these files. They contain instructions that tell you how to place your information in the files.

8. Click OK to dismiss the dialog.

Making a Project the Current Project

C++ Tools commands apply to the current project. To make the new project the current project:

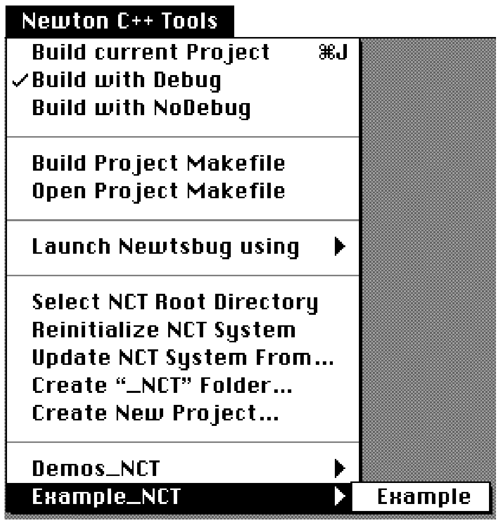
1. Pull down the Newton C++ Tools menu.

   You will see an Example_NCT item at the bottom of the menu.

2. Pull down until you select Example_NCT.

   You'll see a menu item that says Example, as shown in Figure 3-4.

Figure 3-4 Menu Showing New _NCT Folder



3. Choose Example.

   If you pull down the menu again, you'll see that Example_NCT and Example are now underlined, which indicates that Example is the current project. In addition to making this project the C++ Tools current project, choosing a project in this way sets the MPW current directory to be the project directory.

Setting Up the FilesInBuild File

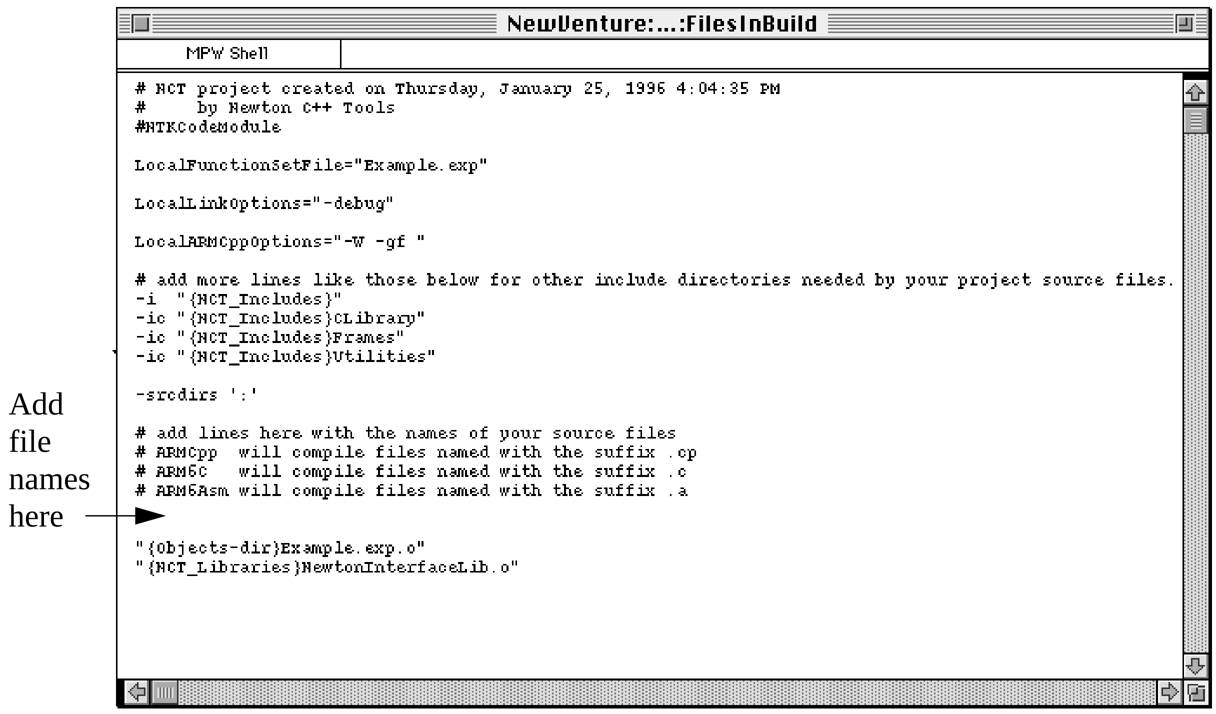
As mentioned before, the FilesInBuild file lists the files that make up the C++ part of your project. You need to have this file in order to use the C++ Tools command that creates the build instructions for your project. To set up FilesInBuild for the example application:

1. Use the File menu's Open command to open FilesInBuild.

   What you see should look like Figure 3-5. Comment lines begin with a pound sign (#). Notice that the file has comments that tell you where to put your information.

2. Type Example.cp in the place where the comments indicate source file names should go, which is indicated by the arrow in Figure 3-5.

Figure 3-5 Default FilesInBuild File



```
# NCT project created on Thursday, January 25, 1996 4:04:35 PM
#     by Newton C++ Tools
#NTKCodeModule

LocalFunctionSetFile="Example.exp"

LocalLinkOptions="-debug"

LocalARMCppOptions="-W -gf "

# add more lines like those below for other include directories needed by your project source files.
-i  "{NCT_Includes}"
-ic "{NCT_Includes}CLibrary"
-ic "{NCT_Includes}Frames"
-ic "{NCT_Includes}Utilities"

-srcdirs ':'

# add lines here with the names of your source files
# ARMCpp  will compile files named with the suffix .cp
# ARM6C   will compile files named with the suffix .c
# ARM6Asm will compile files named with the suffix .a


"{Objects-dir}Example.exp.o"
"{NCT_Libraries}NewtonInterfaceLib.o"
```

3. Close and save the file.

Keeping Code in Sub-Folders

You can have code in sub-folders in your project folder. In that case, you need to give the folder names in the FilesInBuild file. Notice the line in Figure 3-5 that looks like this:

```
-srcdirs ':'
```

You need to list your subfolders in this line. Suppose you have a single subfolder called MyCode. Modify the line so that it looks like this:

```
-srcdirs ': :MyCode:'
```

Notice that the list of folders is surrounded by single quote marks. Suppose you have two subfolders. You separate their names with a space:

```
-srcdirs ': :MyCode1: :MyCode2:'
```

If a folder name contains spaces, you need to surround the name with double-quote marks like this:

```
-srcdirs ': :MyCode1: :MyCode2: ":MyCode Additions:"'
```

Defining a Function

To define the function for this example:

1. Use the New command in the File menu to create a new file.
2. Type this code in the new file's window:

```
#include "objects.h"

extern "C"
Ref SillyString(RefArg)
{
   Ref Unistrg = MakeInt(49);
   return Unistrg;
}
```

Note
You should generally define your functions as `extern "C"`. If you do not do this, the compiler issues a mangled name for the function. You then need to specify the mangled name in the Sample.exp file, which is discussed in the next section. You would also have to use the C++ mangled names when calling the functions from NewtonScript. (C++ uses mangled names to accommodate methods and overloaded functions. A mangled name includes the function name plus its class name and arguments.) You may use mangled names if you wish, but since the mangled names are compiler-dependent and function-signature dependent, you will have to discover and maintain the mangled names yourself; the `DumpAOF` tool is useful for displaying such names. ◆

3. Save the file as Example.cp.

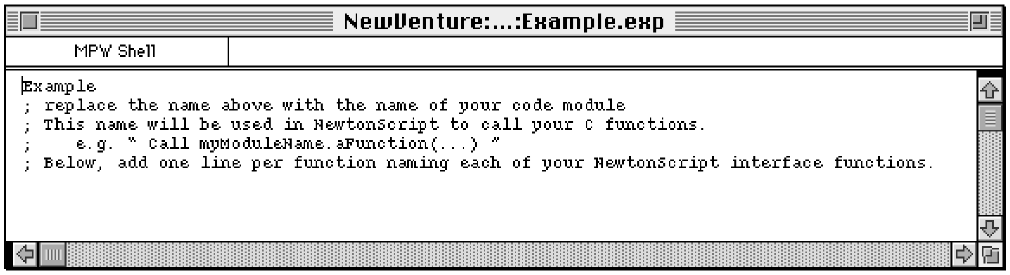
4. Close the file.

Defining the NewtonScript Interface to a Function

You need to tell the build system how the function will be called from NewtonScript. You do that using the .exp file.

1. Use the File menu's Open command to open the Example.exp file. It looks like the file shown in Figure 3-6.

Figure 3-6 Sample.exp File



```
Example
; replace the name above with the name of your code module
; This name will be used in NewtonScript to call your C functions.
;     e.g. " Call myModuleName.aFunction(...) "
; Below, add one line per function naming each of your NewtonScript interface functions.
```

   The first line gives the module name. The C++ Tools build system assumes the module name is the same as the project name. You can change this line if you want to use a different module name. Each .exp file, and therefore each project, can only contain one module.

   In .exp files, comment lines begin with a semicolon (;). Notice that the comments tell you were to add the names of your NewtonScript interface functions.

2. In the place indicated in the file for interface functions, type:

```
SillyString
```

   Press Return. You must always have a return character at the end of every line in this file. If you omit this trailing carriage return, you will get a link error for the last symbol in your module minus its last character.

3. Close and save the file.

Creating a Makefile

The MPW make tool uses makefiles to govern building programs.

1. Select the Newton C++ Tools menu item Build Project Makefile .

   A window named BuildResults opens and the scripts to create an MPW makefile begin executing.

2. Close the BuildResults window.

Note
The file shown in the BuildResults window is marked as "Never save on close." You can save it using the Save command, however, which may be useful if you want to save the position and size of the window. If you do, and there are contents in the window, anything new will be appended to the window's contents. Therefore, in order to save the position of the window, delete the contents of the window before using the Save command. ◆

Building the C++ Source

This step uses the makefile to build the project:

1. From the Newton C++ Tools menu select the menu item Build Current Project .

The Build Results window appears again; messages begin as the Newton C++ Tools build system caches your prior results in the folder YourOldBuildResults: and then a Makeout window opens as the source is compiled and linked .

You now have the file Sample_NCT:Example:Objects:Example.ntkc which has been rebuilt with a new value for the C function called from the NTK application.

IMPORTANT
Development of the NTK part of a project needs to take place in the same directory as you use for C++ development, so that the debugger can find the files it needs. ▲

Importing a C++ Code Module Into an NTK Project

You need to have NTK installed on your hard disk. See NTK documentation of information on using NTK and on developing NewtonScript applications.

1. Select the Finder.
2. Open the NCT_Projects folder.
3. Select the Example.t and Example.π files and drag them into the Example folder.
   These are files that define the NewtonScript part of a small application that you are going to use to call SillyString.
4. Double click Example.π.
   NTK starts and opens the Example project. The project shows only one file.
5. Select the Add file ... menu item from the Project menu.
6. Choose the file Example.t, and click on Add.

This command adds the layout file to the project. (See the NTK documentation for information on layout files.)

7. Select the Add file ... menu item from the Project menu again.
8. Navigate to the Objects folder and select the file Example.ntkc. Click on Add.

   This adds the C++ code module file to the NTK project.
9. Choose the Process Earlier command from the Project menu.

   This command makes sure that the C++ module is processed before the NewtonScript module.
10. Choose the Open command.
11. Select Example.t, and click on Open.

    This is the layout file for this project. An Example.t Browser window opens.
12. In the browser window, choose clView: Top.

    The right side of the window shows the slots of clView.
13. Select viewSetupFormScript.

    In the lower part of the browser, you can see the code for viewSetupFormScript.
14. Notice where the number 42 appears. If you built the package now and downloaded it to a Newton, the application would print the number 42.
15. Replace the number 42 with:

    ```
    call Example.SillyString with ()
    ```

    This is the way you call a C++ function from NewtonScript:

    ```
    moduleName.functionName with (arguments)
    ```
16. Close and save the file.
17. From the Project menu, select the Build Package item.

    This command builds the project including the .ntkc module. If you have your NTK preferences set to automatically download after building and you connected a Newton device to the Mac OS-based computer, NTK attempts to download the Test package to your Newton device. Alternatively, if you have loaded the Toolkit App into the Newton device, you can select it from the Extras drawer and download your Test package.
18. Once you have downloaded the package you can run the package. It displays a simple message that contains the number 49.

When you are building interface functions to be called from NewtonScript with the C++ compiler, you should define those functions as `extern "C"`”. If you do not, the C++ compiler will mangle the function names and you must enter the mangled names in the project.exp file and must use the C++ mangled names when calling the functions from NewtonScript. (C++ uses mangled names to accommodate methods and overloaded functions. A mangled name includes the function name plus its class name and arguments.) In addition, when you do not used mangled function names, the Newton C++ Tools build process determines the number of `RefArg` arguments that each interface function requires automatically and passes this information to NTK. If you use mangled function names, the function name will change if the number of arguments changes.

You may use mangled names if you wish, but since the mangled names are compiler-dependent and function-signature dependent, you will have to discover and maintain the mangled names yourself; the `DumpAOF` tool is useful for displaying such names.

Trying Out Project Demos

Demos is a more complex sample program that is included in NCT_Projects. It already has all the code it needs and has a make file. To try it out, you just need to:

- Make it the current project as shown in "Making a Project the Current Project" on page 3-4
- Create a makefile as shown in "Creating a Makefile" on page 3-7
- Build it as shown in "Building the C++ Source" on page 3-8
- Start NTK, add the .ntkc file to the NTK project, build the NTK project, and download the resulting package, as shown in "Importing a C++ Code Module Into an NTK Project" on page 3-8

Debugging

You use the NTK Inspector to debug the NewtonScript portions of your program, but you cannot use it for the C++ portions. Instead, you use Newtsbug, which allows you to debug code at the ARM assembler level. Newtsbug is included with the C++ Tools. Newtsbug and the Inspector are two independent debugging systems, and they do not interconnect.

In order to set up to use Newtsbug, create an alias of the package file that is produced by NTK and place the alias in the Newtsbug directory. The package file ends with the extension .pkg.

See the *Newtsbug User's Guide* for more information.

Note

You need to read this note only if you write your own makefiles or move the package file that NTK produces. Newtsbug needs to have symbol information in order to debug a package. Symbol information is generated by the compiler whether you compile with debugging on or off, and is contained in the .sym file. The package file contains information that defines where the .sym file is located, using a path relative to the .pkg file. The makefiles created by the Newton C++ Tools place .sym files in the Objects folder that is in the project directory. Therefore, although you can move the .pkg file and the .sym file after you build your package, you must keep the relative path from the .pkg file to the .sym file the same if you want to use Newtsbug. In other words, the directory that contains the .pkg file should contain an Objects folder, and the .sym file should be in that folder. ▲

Customizing the Newton C++ Tools

In the MPW folder, there is an MPW shell file called NCTBuild•UserPrefs that contains various things that an expert MPW user might want to change. You can look at that file for information. In general, before making any changes, you should make a copy of the original so you can restore it if necessary.

One thing in the file is information on the SendAE tool, which you can use to send Apple events from MPW to other applications, such as NTK. This allows you to have MPW tell NTK to automatically re-build the NTK project when you re-build the MPW project.

The Newton C++ Tools MPW Environment

You develop and process code for the C++ Tools using MPW. MPW is a programming environment that is similar in many ways to UNIX, in that it uses a command-line shell and has a fairly complex shell programming language. The C++ Tools build system includes scripts written in this shell language that simplify the process of building projects. These scripts are executed using commands in the Newton C++ Tools menu.

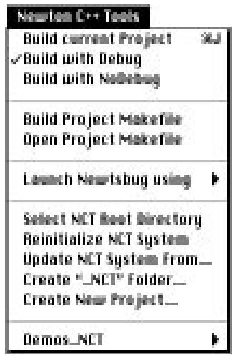
When you start up MPW with the Newton C++ Tools start-up files included in the MPW folder, the start-up files add the Newton C++ Tools menu and also add some commands to the Find menu. These menu commands are described in this chapter.

See the MPW documentation for more information. The MPW documents are included on the Newton C++ Tools release CD.

The Newton C++ Tools Menu

The Newton C++ Tools MPW start-up files add a menu to MPW. Figure 4-1 shows the menu.

Figure 4-1 The Newton C++ Tools Menu



The menu items in the last division of the menu vary; that division shows all of the _NCT folders in the root directory. The _NCT folder that contains the current project is underlined in the submenu. You change the current project by selecting a new one from the submenu.

The rest of this section describes the menu commands. You may want to see Chapter 3, "How to Use the Newton C++ Tools," for information on using these commands.

Build Current Project

When you choose this command, MPW uses the makefile to build the project. (You must already have a makefile, of course. You can create one using the Build Project Makefile command.)

Build with Debug

This menu item sets the makefile so that when you build the project again it is built with the compiler's debug switch on. This switch is used by the compiler's preprocessor. You can have sections of your code that begin with:

```
#ifdef forDebug
```

These sections should end with:

```
#endif
```

When you build with debug, these sections are included in the compilation.

This is the initial mode.

Note

Whenever you use `DebugPrint` and `printf`, those functions should be enclosed within `#ifdef forDebug` and `#endif`. If you do not do that, compilation will fail when you build with NoDebug. ◆

Build with NoDebug

This menu item sets the makefile so that when you build the project it is built with the compilers debug switch off, which excludes the parts marked with `#ifdef forDebug` and `#endif`.

Symbol information that is used by the debugger, Newtsbug, is still included in the object file.

The object file is placed in the folder ObjectsNoDebug, instead of the Objects folder. (ObjectsNoDebug is created, if necessary.) After you build with NoDebug, you need to use the NTK Add File command to add the NoDebug version of the .ntkc file to the project.

Build Project Makefile

This menu item creates a makefile for the current project. In general, you do not need to modify the resulting makefile.

Open Project Makefile

This menu item opens a window showing the current project's makefile.

Launch Newtsbug Using...

This menu item has a submenu that lets you choose the ROM image to use and then runs Newtsbug, the debugger that you use with the Newton C++ Tools. See the *Newtsbug User's Guide* for information on using Newtsbug.

Select NCT Root Directory

This menu item sets the root directory. MPW rebuilds the last menu items to reflect the projects that are in that directory.

Reinitialize NCT System

This menu item reruns the MPW start-up file that sets up the Newton C++ Tools system.

Update NCT System From...

This menu item lets you update your C++ Tools build system from an update CD. It displays a standard file dialog box. You navigate to the Newton C++ Tools CD and locate the folder named "MPW Additions." When you click on Open, the installed version of C++ Tools is updated.

Create "_NCT" Folder...

You can create a new "_NCT" folder with this menu item. This is what is called a "projects directory." Enter the name of the projects directory you wish to create (without the suffix _NCT) and a folder of that name is added to your NCT_Projects directory.

_NCT folders are intended to be used to organize your projects. You put groups of related projects in a single _NCT folder.

Create New Project...

Use this to create a new project folder within the current projects directory.

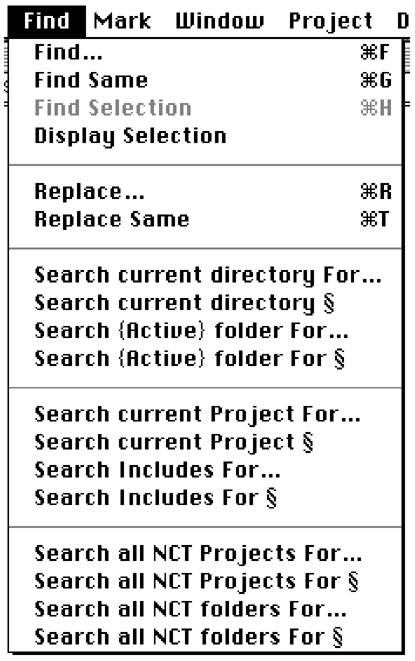
The project name must not contain any spaces, because the name will be used as a C++ module name.

The Newton C++ Tools build system creates the folder, unless one already exists with that name. Within that folder, it creates prototypes for the files and folders that you need. Specifically, this command creates a prototype FilesInBuild file, an Objects folder, and a prototype .exp file. See "The Contents of the Project Directories" on page 1-3 for information on these items.

Additional Commands in the Find Menu

Figure 4-2 shows the Find menu. The commands in the top two divisions are standard MPW commands; see the MPW documentation for information on those commands.

Figure 4-2 The Find Menu


Find
Mark
Window
Project
D
Find... ⌘F
Find Same ⌘G
Find Selection ⌘H
Display Selection
Replace... ⌘R
Replace Same ⌘T
Search current directory For...
Search current directory §
Search {Active} folder For...
Search {Active} folder For §
Search current Project For...
Search current Project §
Search Includes For...
Search Includes For §
Search all NCT Projects For...
Search all NCT Projects For §
Search all NCT folders For...
Search all NCT folders For §

The additional commands are not specific to the C++ Tools, but are powerful and helpful search commands.

▲ **WARNING**
You should not enter search strings that contain MPW escape characters such as ∂ (option-d). This character can cause problems with searching; for example, ∂n causes the search commands to go into an infinite loop. ▲

Search Current Directory For...

This menu command puts up a dialog box that asks for text to search for. The search is limited to the current directory. It does not search sub-directories.

Search Current Directory §

The symbol § refers to the current selection. This menu command searches the current directory for other occurrences of the current selection. It does not search sub-directories.

Search {Active} Folder For...

This menu command puts up a dialog box that asks for text to search for. The search is limited to the currently active folder, which is the folder containing the file that is shown in the active window. (Technically, this is the folder whose name is contained in the MPW variable {Active}. See the MPW documentation for more information on MPW variables.) This command does not search sub-directories.

Search {Active} Folder For §

This menu command searches the active directory for other occurrences of the current selection. It does not search sub-directories.

Search Current Project For...

This menu command puts up a dialog box that asks for text to search for. It searches the current project’s directory, including sub-directories.

Search Current Project §

This menu command searches for other occurrences of the current selection in the current project’s directory, including sub-directories.

Search Includes For...

This menu command puts up a dialog box that asks for text to search for. It searches the Include directory, including sub-directories.

Search Includes For §

This menu command searches for other occurrences of the current selection in the Include directory, including sub-directories.

Search All NCT Projects For...

This menu command puts up a dialog box that asks for text to search for. It searches all project directories in the current _NCT directory, including sub-directories.

Search All NCT Projects For §

This menu command searches for other occurrences of the current selection in all project directories in the current _NCT directory, including sub-directories.

Search All NCT Folders For...

This menu command puts up a dialog box that asks for text to search for. It searches all directories in the current root directory, including sub-directories.

Search All NCT Folders For §

This menu command searches for other occurrences of the current selection in all project directories in the current root, including sub-directories.

MPW Tools

The Newton C++ Tools contains a number of MPW tools. You generally do not have to know that much about these tools; the Newton C++ Tools menu commands build a make file that calls these tools for you. See the Tools folder for the complete set of tools. The most important are:

- AIFtoNTK, which converts the link file into a form that can be incorporated in a Newton program. See “AIFtoNTK” below for information.
- ARM6asm assembler. See the ARM documentation for information.
- ARM6c C compiler. See the ARM documentation for information.
- ARMCpp C++ compiler. See the ARM documentation for information.
- ARMLink linker. . See the ARM documentation for information.
- DumpAOF, which t emits code in ARM Object Format. Sophisticated programmers may find this useful for debugging code. See the ARM documentation for information.
- DumpAIF, which dumps the output format from the linker, and is analogous to the MPW tool DumpCode. See “DumpAIF” on page 5-3 for information.

If you want to see the available options (and short descriptions) for most of the tools, you can execute:

toolname `-help`

on the MPW worksheet. For some tools, you can also simply give an invalid option to see the help information.

AIFtoNTK

This tool is invoked for you by the Newton C++ Tools scripts, and you should not need to understand its options yourself. This section is informational only.

`AIFtoNTK [-p] -via` *exportsFile* `-o` *outputFile* *inputFile*

This tool converts an ARM Image Format (AIF) file into an NTK “streamed frame” file.

-p	This is an optional parameter that requests the display of progress messages to Dev:Stderr.
-via *exportsFile*	Specifies the path of an “exports” list file. When you use the C++ Tools menu commands, they build a prototype exports file; you fill in the file with the information you need. The exports file is a normal text file that contains lines of text. The first line must contain the module name. Subsequent lines each contain a single function name. These are the names of functions that you want to be able to call from NewtonScript. Each function name must be the first non-blank text on the line and there must be one function name per line. You can also have comment lines that begin with a semicolon (;); those lines are ignored. You use the module name in NewtonScript call statements when invoking a function in the module. For example: `call modulename.function1 with (....)`
-o *outputFile*	Specifies the path of the output file to be created. If the file already exists, its contents will be destroyed. The normal convention is to use the same name as the input file with the suffix “.ntkc” added .
inputFile	Specifies the path of the input file to be converted into a .ntkc file for use by Newton Toolkit. You can only give one input file. You must give an input file; AIFtoNTK ignores standard input.

None of the parameter pathnames can contain aliases; the tool will not resolve the aliases.

The AIFtoNTK tool converts the data fork of the AIF file created by the ARMLink tool into a streamed frame format file which is accepted as input by the Newton ToolKit (NTK). The binary image of the read-only code and the table of relocations or “fix ups” are converted into binary objects within the streamed frame.

The tool checks for signatures in the file data fork to verify that the input file is an ARM Image Format file.

AIFtoNTK can return the following status codes.

0	no errors; output file created.
1	usage, syntax or missing or invalid parameter error.
2	execution error; usually invalid data encountered in input file
3	internal or fetal error; usually system or memory overflow

Here is an example of the use of this tool:

```
AIFtoNTK -via mymodule.exp -o mymodule.ntkc mymodule
```

DumpAIF

This is a tool for advanced users who want to see the linker output code.

`DumpAIF [-s]` *filename*

The command displays the contents of the AIF file *filename*. The -s option adds a symbol table.