Newton Book Maker User's Guide

Version 1.1

Printed in the United States of America.

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, APDA will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to APDA.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD “AS IS,” AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

11/95

Contents

About This Book

This book, the *Newton Book Maker User's Guide*, describes how to create digital books and application help for the Newton family of personal digital assistants using Book Maker 1.1.

Audience

This guide is for anyone who wants to create digital books for the Newton family of products. You need not have any programming experience to create a book; however, you should be familiar with basic Macintosh operations and should know how to build a project in Newton Toolkit (NTK). For more information about building packages, see the *Newton Toolkit User's Guide*, which is included with NTK.

Non-programmers can also create content for help screens in Newton applications. However, the incorporation of help in a Newton application requires familiarity with NewtonScript and the Newton object system.

Related Books

This book is one in a set of books included with Newton Toolkit, the Newton development environment. Although this book, *Newton Book Maker User's Guide*, provides all of the information you'll need to create digital books for the Newton family of products, you may wish to refer to these other books in the set:

- *Newton Toolkit User's Guide*. This book introduces the Newton development environment and shows how to develop Newton applications using Newton Toolkit. You should read the section of this book that describes how to build a book package if you are not familiar with this process.

- *Newton Programmer's Guide: System Software*. This set of books is the definitive guide and reference for Newton programming topics other than communications.
- *The NewtonScript Programming Language*. This book describes the NewtonScript programming language. None of this material is required to use Book Maker; however, programmers incorporating help screens in a Newton application need to be familiar with this material. Furthermore, NewtonScript code can be added to a digital book to make the book "come alive."
- *Newton Programmer's Guide: Communications*. This book is the definitive guide and reference for Newton communications programming. This material is not necessary reading for creating digital books.

How to Use This Book

This book contains five chapters; at the very least you need to read Chapters 1 and 2 before attempting to create your own books.

- Chapter 1, "Newton Digital Books," describes the features of Newton digital books. This chapter introduces Newton Book Maker, Newton Book Reader and the system-supplied help browser. It also describes some of the differences between Book Reader packages and application help.
- Chapter 2, "Getting Started With Newton Book Maker," provides a quick introduction to the process of building a Newton digital book.
- Chapter 3, "Using the Book Maker Language," describes the Book Maker commands used to lay out text and graphics on the page.

- Chapter 4, “NewtonScript in Books,” describes the optional use of NewtonScript methods, slots, templates, and frames in digital books. The material in this chapter presumes some NewtonScript programming ability and some familiarity with the Newton object system.
- Chapter 5, “Application Help,” describes how to add a set of help screens created using Book Maker to a Newton application. The material in this chapter presumes some NewtonScript programming ability and some familiarity with Newton application development.
- Appendix A, “The Book Maker Language,” is a description of the different kinds of commands used in Book Maker. This appendix includes a command reference section that describes the individual elements of the Book Maker language.
- Appendix B, “Troubleshooting,” describes several common problems and suggests solutions.
- Appendix C, “Books on Online Help,” lists various books and journal articles which deal with creation of online help.
- Appendix D, “Compatibility,” describes changes to the 1.1. version of Book Maker, and the changes necessary to make a book be displayable by versions Book Reader installed on 1.x Newton devices.

Example Books

The Newton Book Maker product includes a set of Book Maker examples that illustrate most of the topics covered in the *Newton Book Maker User’s Guide*.

Each book example folder contains the following files:

- **NTK Project File**
  This file serves as a repository for all of the files needed to build a project in NTK. This example file has the same name as the

folder storing all the related files for a particular book example; for example, the `Simple` folder contains an NTK project named `Simple`.

- **Book Source File**
  This is a word processor file used as input to Book Maker; the example book source listings shown in this book are found in the book source files. This example file is named by appending the word `story` to the name of the related NTK project file; for example, the book source file for the `Simple` project is named `SimpleStory`.
- **Book Maker Output File**
  Book Maker processes a book source file and produces this file, which is added to the NTK project used to build a book or an application. The name of this example file is created by appending `.f` to the name of the book source file that was used to create this output file; for example, Book Maker produces an output file named `SimpleStory.f` when it processes the `SimpleStory` book source file.
- **Book Reader Package**
  This is the digital book package that appears in the Extras Drawer on the Newton screen. This package is produced by building the NTK project containing the Book Maker output file. The name of the example package is created by appending `.pkg` to the name of the NTK project file used to build it; for example, the `Simple` project builds the `Simple.pkg` book package.

The Newton Developer Technical Support team continually revises the existing samples and creates new sample code. You can find the latest collection of sample code in the Newton developer area on eWorld. You can gain access to the sample code by participating in the Newton developer support program. For information about how to contact Apple regarding the Newton developer support program, see the section “Developer Products and Support,” on page xvii.

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. These terms are also defined in the Glossary.
- `Courier typeface`. Code listings, code snippets, and special identifiers in the text such as predefined system frame names, slot names, function names, method names, symbols, and constants are shown in the `Courier` typeface to distinguish them from regular body text. Items that appear in `Courier` should be typed exactly as shown.
- Geneva typeface. Book Maker source file listings and examples are shown in the Geneva typeface to distinguish them from regular body text. Items that appear in Geneva font should be typed exactly as shown. The Geneva font was chosen because it is supported directly by the Newton digital book reader; use of this font in book source examples facilitates a more exact correspondence between what you see in this book and what appears on the Newton screen.
- *Italic type* is used in book source code and NewtonScript 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.
- Square brackets (`[` and `]`) identify optional parameters in command syntax listings. The brackets are not part of the optional parameter and should not be included in Book Maker commands.

Developer Products and Support

APDA is Apple's worldwide source for a large number of development tools, technical resources, training products, and information for anyone interested in developing applications on Apple platforms. Every four months, customers receive the *APDA Tools Catalog,* featuring all current versions of Apple tools and the most popular third-party development tools. Ordering is easy; there are no membership fees, and application forms are not required for most of our products. APDA offers convenient payment and shipping options, including site licensing.

To order a product or to request a complimentary copy of the *APDA Tools Catalog*:

APDA
Apple Computer, Inc.
P.O. Box 319
Buffalo, NY 14207-0319

Telephone	1-800-282-2732 (United States) 1-800-637-0029 (Canada) 716-871-6555 (International)
Fax	716-871-6511
AppleLink	APDA
America Online	APDA
CompuServe	76666,2405
Internet	APDA@applelink.apple.com

If you provide commercial products and services, call 408-974-4897 for information on the developer support programs available from Apple.

For NewtonScript Programmers

Although you need not have any programming knowledge to create Newton digital books, NewtonScript programmers can use the Newton object system and NewtonScript to provide additional features in books. This section addresses issues that affect NewtonScript programmers only; if you do not plan on using NewtonScript in your digital book, you can skip this section and go on to Chapter 1, “Newton Digital Books.”

Tap Versus Click

Throughout the Newton software system and in this book, the word “click” sometimes appears as part of the name of a method or variable, as in `viewClickScript` or `buttonClickScript`. This may lead you to believe that the text refers to mouse clicks. It does not. Wherever you see the word “click” used this way, it refers to a tap of the pen on the Newton screen (which is somewhat similar to the click of a mouse on a desktop computer).

Frame Code

If you are using the Newton Toolkit (NTK) development environment in conjunction with this book, you may notice that this book displays the code for a frame (such as a view) differently than NTK does.

In NTK, you can see the code for only a single frame slot at a time. In this book, the code for a frame is presented all at once, so you can see all of the slots in the frame, like this:

```
{  viewClass: clView,
   viewBounds: RelBounds( 20, 50, 94, 142 ),
   viewFlags: vNoFlags,
   viewFormat: vfFillWhite+vfFrameBlack+vfPen(1),
   viewJustify: vjCenterH,

   viewSetupDoneScript: func()
      :UpdateDisplay(),

   UpdateDisplay: func()
      SetValue(display, 'text, value);
   };
```

If, while working in NTK, you want to create a frame that you see in the book, follow these steps:

1. On the NTK template palette, find the view class or proto shown in the book. Draw out a view using that template. If the frame shown in the book contains a `_proto` slot, use the corresponding proto from the NTK template palette. If the frame shown in the book contains a `viewClass` slot instead of a `_proto` slot, use the corresponding view class from the NTK template palette.
2. Edit the `viewBounds` slot to match the values shown in the book.
3. Add each of the other slots you see listed in the frame, setting their values to the values shown in the book. Slots that have values are attribute slots, and those that contain functions are method slots.

Undocumented System Software Objects

When browsing in the NTK Inspector window, you may see functions, methods, and data objects that are not documented in this book. Undocumented functions, methods, and data objects are not supported, nor are they guaranteed to work in future Newton devices. Using them may produce undesirable effects on current and future Newton devices.

PREFACE

CHAPTER 1

Newton Digital Books

Newton Book Reader is a system service that displays interactive digital books on the Newton screen. Newton Book Maker allows non-programmers to create source files for Newton digital books using an ordinary word processor application.

The Book Maker application also allows non-programmers to create help content for Newton applications. The installation of help in a Newton application package requires some rudimentary NewtonScript programming ability.

This chapter describes the features of Newton digital books, and introduces Newton Book Maker, Newton Book Reader, and the system-supplied help browser. It also describes some of the differences between Book Reader packages and application help.

Newton Digital Books

Newton digital books can display graphics, multiple-font text, and on-screen controls for content navigation. The user can scroll pages, mark pages with bookmarks, access data directly by page number or subject, mark up pages using electronic ink and perform text searches. The user can also copy and paste text from digital books, as well as print text and graphics from them.

Because Book Reader books are closely coupled with NewtonScript and the Newton object system, NewtonScript programmers can attach slots, scripts, and Newton object system prototypes to book content to further customize the book's behavior.

Newton Book Maker

Newton Book Maker allows non-programmers to create source files for full-featured Newton digital books using an ordinary word processor application and the Book Maker command language.

The needs of those wishing to convert existing paper-based publications into digital books were a prime consideration in the development of Book Maker and its command language. As a result, you can convert an existing work into a full-featured interactive book with very little effort. Only three Book Maker commands are required to create a simple book that provides all of the features mentioned previously, such as bookmarks, electronic ink, and support of the Find service.

The Book Maker language also provides a rich set of features that allow the content provider to exercise greater control over page layout or provide additional services to the user.

Newton Toolkit uses the output file produced by the Book Maker application to create a book package that appears in the Extras Drawer or to create help that can be incorporated into a Newton application, or as a help book package that appears in the Help Folder of the Extras Drawer.

Newton Book Reader

Newton Book Reader is available on all Newton platforms. It is invisible to the user until a digital book package is presented to the system. Book packages can be loaded in RAM or can reside on PCMCIA cards.

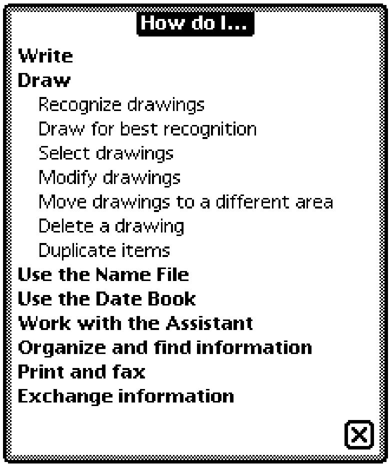
Newton Book Reader provides a user interface for navigation through content that keeps the look and feel of digital books consistent while leaving the design of the content itself up to the book's author. For information about Newton Book Reader's user interface, consult the documentation provided with your Newton device.

Newton Application Help

In addition to creating the source file for a digital book package, Book Maker can be used to create help screens for Newton applications.

When the user taps the "How Do I?" button in the Assist Drawer, the system displays an outline-like overview of help topics. Tapping a topic causes it to display its subtopics or, if there are no further subtopics, to display a help screen. Although you cannot add information to the system-supplied help, you can provide the same kind of help within your Newton application. The system-supplied help overview is shown in Figure 1-1.

Figure 1-1 The system-supplied help overview



How do I...

Write

Draw

Recognize drawings

Draw for best recognition

Select drawings

Modify drawings

Move drawings to a different area

Delete a drawing

Duplicate items

Use the Name File

Use the Date Book

Work with the Assistant

Organize and find information

Print and fax

Exchange information

☒

The source file from which Book Maker creates help is written in much the same manner as the file from which a Book Reader package is created: a word-processor file is tagged with Book Maker commands and processed by Newton Book Maker. The most important difference between the two is the means by which the information is made available to the user. Book packages appear in the Extras Drawer and are displayed by Newton Book Reader when the user taps the package's icon. Application help, on the other hand, is visible to the user only as an outline, and as associated information displayed when the user taps a Help button that the application must provide. The application uses the system-supplied help browser to display help.

The Help Browser

The help browser is intended to present single screens of step-by-step instruction for performing actions in a Newton application. Because of the limited scope of this kind of information, the help browser provides only a subset of the features of Newton Book Reader. This section briefly contrasts application help with book packages.

The screens provided by the help browser are smaller than those used for Book Reader. A Book Reader book package is better suited for the presentation of a full-featured user manual.

When deciding whether to provide information as a Book Reader package or as application help, you'll want to keep in mind these important differences in their respective formats and feature sets.

Further discussion of specific issues involved with designing application help is provided later in this book, in Chapter 5, "Application Help."

Books vs. Applications

Digital books can incorporate protos and templates just as an application can. If you are considering writing an application which uses a large amount of text that spans multiple pages, or if the layout of the text is important, you may wish to consider writing your application as a digital book. This will allow you to use all the tools provided by Book Maker.

CHAPTER 2

Getting Started With Newton Book Maker

This chapter provides a quick introduction to the process of building a Newton digital book. The first section briefly reviews the hardware and software required to use Book Maker. Next, you are shown the composition of the simplest Book Maker input file. After that, you'll use Book Maker and NTK to create a simple Newton digital book.

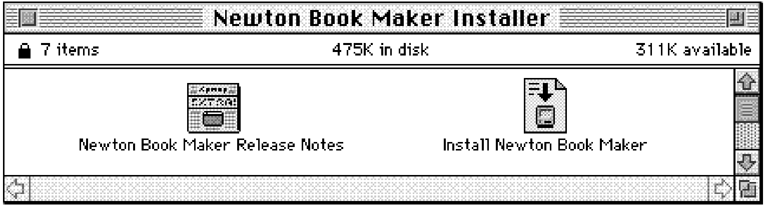
Installing Book Maker

Newton Book Maker is shipped with an installer script, which you run from the distribution disk. Before installing Book Maker, verify that each distribution disk is locked (that is, that the write-protect slider is open) to protect the contents from accidental overwriting.

You will also need to install Newton Toolkit (NTK) in order to process the files created by Book Maker into Newton packages, which can be downloaded onto a Newton device. If you have not already done so, you need to install NTK as described in the *Newton Toolkit User's Guide.*

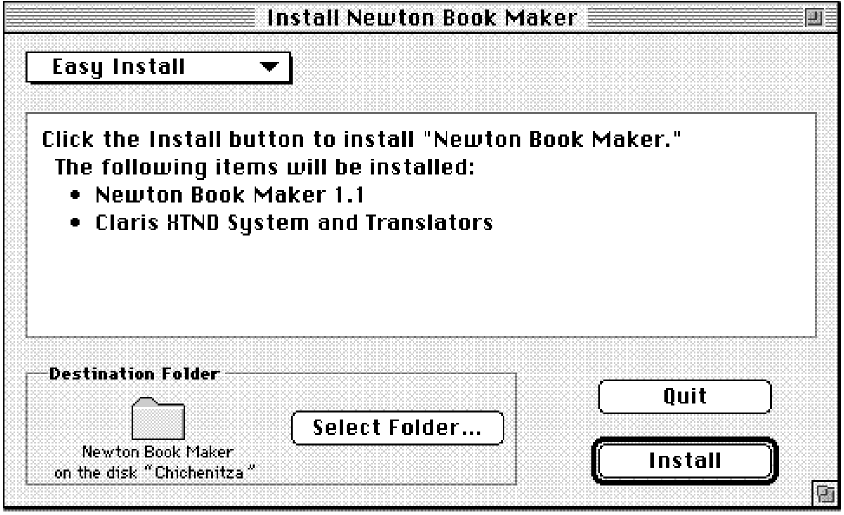
Follow the steps in this section to install Book Maker on the Macintosh.

1. Insert the disk *Newton Book Maker Installer.* The disk opens to show the release notes, and an installer script.


Newton Book Maker Installer
7 items
475K in disk
311K available
Newton Book Maker Release Notes
Install Newton Book Maker

2. Double-click on the release notes and read them to see of there are any updates to the installation procedure. If not, close them and continue on with these directions.

3. Double-click on the *Install Newton Book Maker* icon to begin installation.



Click the Install button to install "Newton Book Maker."
The following items will be installed:
- Newton Book Maker 1.1
- Claris XTND System and Translators

Destination Folder
Newton Book Maker
on the disk "Chichenitza"

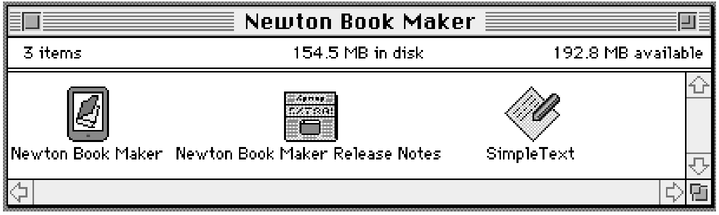
If you are installing Newton Book Maker for the first time, or if you're simply updating an earlier release, leave the Easy Install item selected in the pop-up menu in the top-left corner of the window. You can also choose Custom Install to selectively install parts of the software, or Custom Remove to selectively remove parts of the software.

By default, *Install Newton Book Maker* puts Book Maker in a folder named Newton Book Maker on your startup disk. If you want to change the destination, click Select Folder and specify a new or different folder.

4. Click Install. The installer begins copying and configuring the software, and displays a progress report.

5. When installation is complete you will be prompted to restart your Macintosh; click Restart.

The Newton Book Maker folder contains the following files:



- The Newton Book Maker application.
- The Newton Book Maker Release Notes; read these for any late-breaking information not included in this manual.
- The SimpleText application; provided for reading the release notes. You may delete this file if you wish.

In addition, a Claris folder is installed in your System folder. This folder contains the following:

- Claris Fonts file
- Claris XTND System file
- The Claris Translators folder

Make sure not to move or rename any files in this folder.

Hardware Requirements

Any Macintosh suitable for running Newton Toolkit can be used to run Newton Book Maker; for more information, see the *Newton Toolkit User's Guide*.

System Software Requirements

Newton Book Maker requires Macintosh System software version 7.0.1 or later.

RAM Requirements

Newton Book Maker version 1.0 has a suggested minimum RAM requirement of 3700 KB; the actual requirements vary according to the size of the book source file being processed. Extremely large books may require more RAM to process; smaller ones may require less. Book Maker requires enough RAM to keep the entire book source file in memory while it is being processed.

If you need to conserve memory, you can reduce the amount of RAM allocated to Book Maker by changing the settings in the Get Info box for this application in the Finder. You can also use the `.chain` command to break large book source files into sets of smaller files. For more information about changing the amount of RAM allocated to an application, see your *Macintosh User's Guide*. For more information about the `.chain` command, see its description in the "Miscellaneous Commands" section of Appendix A, "The Book Maker Language."

Claris XTND Translators

Newton Book Maker relies on Claris XTND translator technology to accept files from various word processors as input. When Newton Book Maker is installed correctly, your System folder should contain a Claris folder that includes the Claris XTND translators shown in Table 2-1 as well as the Claris

XTND tool itself. You can add XTND translators for any other word processor format you prefer; Book Maker accepts as input any word processor file for which an XTND translator is installed.

Table 2-1 XTND translators installed by Newton Book Maker installer

Claris	Other
MacWrite® II	TEXT
MacWrite® 5.0	

Fonts

Bitmapped versions of the fonts used in Newton digital books must be installed on the computer that runs Newton Book Maker and NTK.

Currently, the Newton MessagePad supports only the bitmapped versions of font families New York, Geneva, Espy Sans and Espy Sans Bold in sizes 9, 10, 12, 14, and 18 points.

The NTK installer application installs all supported sizes of Espy Sans and Espy Sans Bold for you. You need only install any other fonts actually used in digital books created on this computer.

WARNING

Use of any other fonts or use of TrueType versions of the supported fonts may cause Book Maker to calculate page layouts incorrectly. ▲

Creating Digital Books

This section introduces you to the Book Maker command language and the book-building process.

The process of creating a book is iterative. Because it's hard to guess exactly where pages will fall, you'll typically add Book Maker commands to your document, process it through Newton Book Maker and NTK, then proofread it on a Newton device. After proofreading the book on the Newton screen, you'll probably make changes to your book source file and repeat the process again until you achieve the intended result.

There are three steps in the process of creating a digital book for a Newton device.

1. **Add Book Maker commands to the content file.**

   You can use any Macintosh word processor for which an XTND translator is installed on the computer used to run Newton Book Maker. A content file that includes Book Maker commands is called a book source file.

2. **Use Newton Book Maker to process the book source file.**

   Book Maker produces a file that is used as input to Newton Toolkit.

3. **Use Newton Toolkit to create a Book Reader package or integrate help in a Newton application.**

   Building a book package is discussed later in this chapter, in the section "Building a Book Package With NTK." Integrating help with a Newton application is discussed in Chapter 5, "Application Help."

Creating a Book Source File

A word processor document that includes Book Maker commands is called a **book source file.** The book source file is processed by the Newton Book Maker application, which produces a file used as input to Newton Toolkit. NTK then builds a book package or adds help screens to a Newton application.

All of the information in this chapter applies equally to Book Reader packages or application help.

Before You Start

It's helpful to simulate the dimensions of the screen on the Newton MessagePad by setting your word processor margins to create a page width of 3.33 inches. This gives you an idea of what the book page will look like in the final product. Be aware, though, that Book Maker does not use the margins in the book source file to lay out text, so you should not attempt to create insets or margins in your book text by setting margins of less than 3.33 inches on your word processor. You can also use the full 3.33-inch width without worrying about text running into the edge of the Newton screen—the screen on the Newton device includes a small non-display area between the edge of the display area and the Newton device's plastic case.

The Newton MessagePad supports only the bitmapped versions of New York and Geneva fonts in point sizes 9, 10, 12, 14, and 18. Your book must be written in these fonts to be displayed correctly on the screen of the MessagePad. You can also use point sizes that are multiples of these built-in font sizes.

The MessagePad also has a system font, Espy Sans, which was specially designed to display well on the Newton screen. This font is used in most of the text displayed by the system. This font does not print or fax especially well though; if your book is likely to be printed or faxed often, you want to avoid this font.

Book Maker tries to reproduce the text of your book exactly as it appears in the book source file, observing font, size, and style changes with no additional effort required on your part. As a result, simple books can be created with minimal effort. If you prefer, you can use additional Book Maker commands to specify more complex formatting options. The example shown in the next section demonstrates the minimum command set required to create a book source file.

About Book Maker Commands

Book Maker commands always start with a period (`.`), or dot, which is why they are sometimes referred to as "dot commands."

Book Maker commands always appear at the beginning of a line in the book source file. Each command applies to everything that appears in the book source file until the next command appears. Book Maker commands themselves do not appear on the Newton screen.

The entire Book Maker command language is case insensitive. This book uses mixed capitalization in book source files, as appropriate, to make them easier to read.

Most word proccessors allow for user-defined styles. You may want to define a style called "BM commands" and set it to display in a different color. In this way the Book Maker commands in your book source file will stand out from the rest of the document. See the documentation provided with your word processor for information about how to do this.

Adding Required Commands

All book source files for Book Reader books must include the `.title` and `.isbn` commands, as well as at least one content item. A content item is an item displayed on the screen, such as text or a picture. This section describes how to add these commands and a content item to the book source file.

The Title Command

All book source files must include a `.title` command, which defines the text placed at the top of every page when the book is displayed on the Newton screen. It is possible to suppress the placing of the title at the top of every page with the `NoTitle` flag. Flags are discussed in the section "Flags" beginning on page 3-6; the `NoTitle` flag is documented under "Document Flags" on page A-33.

The `.title` command also defines the book's name in the Extras Drawer, unless the book also has a `.shortTitle` command, which is discussed in "Adding a Short Title" on page 2-17. The title of the book is also used by the system to display the context of a search, in a phrase such as “Searching in The Simple Story...”

The book source file can have only one `.title` command.

To add a `.title` command to your book source file, place it at the beginning of the first line in the book source file and place your book’s title on the rest of the line. The following example shows the `.title` command from a book titled *The Simplest Story:*

```
.title The Simplest Story
```

The ISBN Command

All book source files must include an`.isbn` command. Like the `.title` command, there can be only one `.isbn` command in any book source file. This command assigns a unique identifier to the book package. The identifier is used by the Newton Book Reader.

ISBN is an acronym for International Standard Book Number. ISBN numbers are unique numbers used by publishers and others in the book trade to identify books. You need not use an actual ISBN number to identify your book to Newton Book Reader; any unique identifier consisting of 14 or fewer alphanumeric characters will suffice. You can create a unique identifier by using your developer signature as a suffix to a descriptive title; for example, `Simple1:PIEDTS` is a perfectly acceptable value for this identifier.

To add an `.isbn` command to your book source file, place it on the line that follows the title command, as shown in the following example:

```
.title The Simplest Story
.isbn Simple1:PIEDTS
```

For information about acquiring a real ISBN number, see the discussion of the `.isbn` command in Appendix A, “The Book Maker Language.”

The Required Content Item

Aside from the required `.title` and `.isbn` commands, the book source file must include at least one content item, such as text or a picture, to be displayed on the screen. You can use the `.story` command to display text, as in the following example:

```
.story
This is the simplest story ever told:

Once upon a time there was a little girl who lived
happily ever after.
```

Note
The `.story` and `.title` commands process text only. Graphics tagged with these commands are ignored. The `.picture` command, explained later in this section, can be used to add graphics to a digital book. ◆

The Simplest Book Source File

The simplest Book Maker source file contains a `.title` command, an `.isbn` command and at least one content item. It looks like the following example, which is included with Book Maker as the `SimpleStory` file. This file and all of the other example files mentioned in this guide are in the `Book Maker Examples` folder included with Newton Toolkit.

```
.title The Simplest Story
.isbn simple1:PIEDTS
.story
This is the simplest story ever told:


Once upon a time there was a little girl who lived
happily ever after.
```

When this example is displayed on the Newton screen, the fonts and formatting appear just as in the source file: the unusual capitalization of the first letter of the story and the paragraph breaks are preserved, as shown in Figure 2-5 on page 2-17. However, the Book Maker commands (dot commands) do not appear on the Newton screen.

In order to display the book on the Newton screen, you need to use Book Maker and NTK to build a book package from the book source file. The next two sections introduce you to this process.

Processing the Book Source File

The book source file for a Book Reader book or application help is processed using the Newton Book Maker application, which produces a file that is used as input to NTK. This section describes how to use Newton Book Maker to process book source files.

You can use the example book source file `SimpleStory` to complete the steps described in this section. If you use one of the example files provided with Book Maker, it's recommended that you create a new folder in which to experiment; that way you won't accidentally replace one of the original example files.

Take the following steps to process the book source file using Newton Book Maker:

1. **Open Newton Book Maker.**

   As with any other Macintosh application, you can select its icon in the Finder and choose the Open item from the File menu, or simply double-click the application icon to open it.

2. **Choose Open… from the File menu and select the book source file.**

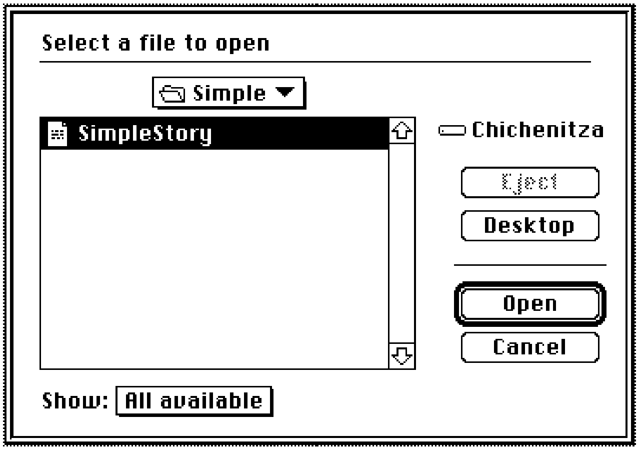
   When you choose the Open… item from the File menu, Book Maker presents a dialog box in which you can specify the book source file to be processed. This dialog box is shown in Figure 2-1.

3. **Open the book source file.**

   Select the book source file's name in the dialog box and click the Open button. Alternatively, you can double-click the file's name to open it. Figure 2-1 depicts the selection of the `SimpleStory` book source file included with Newton Book Maker.

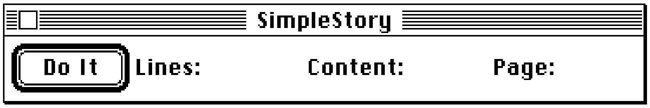
Note
In the Finder, you can simply drag a book source file onto the Book Maker application to open the application and select the book source file. ◆

Figure 2-1 Choosing a book source file in Book Maker


Select a file to open

Book Maker presents a window named for the source file to be processed. Figure 2-2 shows the window Book Maker displays when the `SimpleStory` book source file is opened.

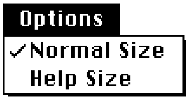
Figure 2-2 The book processing window


SimpleStory

4. **Choose the screen format for which the book is to be processed.**

From the Options menu, choose Normal Size for Book Reader books, or Help Size for application help. A check mark appears in the Options menu next to the size that is selected. Figure 2-3 shows the Options menu as it appears when Normal Size is selected.

Figure 2-3 Specifying the Book Reader destination format



5. **Click the Do It button to process the book.**

   As Book Maker processes the source file, it presents status information regarding the number of lines processed in the source file, the number of content items created, and the number of pages as calculated for the specified destination format.

   If Book Maker cannot process the book source file for any reason, it stops and displays an error message. You must correct the error and reprocess the source file.

   For most errors, Book Maker displays a line number specifying where the error was encountered, and the previous line, in the book processing window. You can copy this information to the clipboard by selecting Copy from the Edit menu (or hitting Command-C on the keyboard).

   The error line number is actually the number of paragraph endings that Book Maker has processed; your word processor may count paragraphs differently than Book Maker.

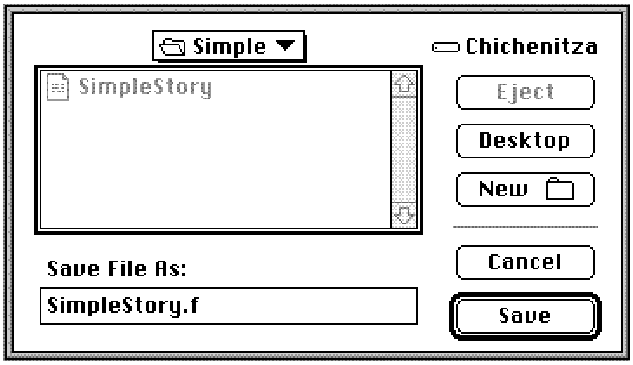
   You can cancel Book Maker's processing of a source file, by typing a period while pressing the Command (or Apple) key.

6. **Save the Book Maker output file.**

   When Book Maker finishes processing the source file, it presents a dialog box in which you can specify the name under which to save the Book Maker output file. Book Maker supplies a default name that is constructed by appending the suffix `.f` to the source file's name; however, you can name the file whatever you prefer by typing in the editable text field and clicking the Save button. Figure 2-4 shows the Save dialog box as it appears after processing the source file `SimpleStory`.

This Book Maker output file is now ready to use as input to NTK.

Figure 2-4 Saving the processed book file



Save File As: SimpleStory.f

IMPORTANT

Do not edit the output file produced by Book Maker. To use one of the example books as a starting point for your own work, modify the book source file and then process it in Book Maker to create a new Book Maker output file. ▲

Book Maker is AppleScript-able

The preceding steps can be automated with AppleScript. If you have AppleScript you can write a short script that has Newton Book Maker open a source file, process it, and save the output. The following script, for example, processes a source file "A Superb Story" in the folder "to do" in the disk "MyDisk," and save the output as "A Superb Story.f" in the disk "My Disk."

```
tell application "Newton Book Maker"
   activate
   open file "MyDisk:to do:A Superb Story"
   (* The following two commands - parse and buildFile -
      are the equivalent of clicking the Do It button.
      They should always be used together. *)
   parse
   buildFile soupFile file "My Disk:A Superb Story.f"
   close document
   quit
end tell
```

A Book Reader Book or Application Help?

The steps you take next depend on whether you are creating a Book Reader book package or application help.

If you processed the book source file with the Normal Size option enabled, you'll use NTK to build a Book Reader package that appears in the Extras drawer. The next section, "Building a Book Package With NTK," describes how to build Book Reader packages.

If you processed the book with the Help Size option enabled, the contents of this file must be integrated with your application, so that it can use the help frames directly. The integration of help in a Newton application is described later in this book, in Chapter 5, "Application Help."

Building a Book Package With NTK

A Book Reader package can be downloaded to the Newton platform using Newton Connection or Newton Toolkit. The book package appears as an icon in the Extras Drawer just like any other application package.

You can use the `SimpleStory.f` file and the NTK project named `Simple` to follow along with the steps in this section if you want to practice building a book package.

Note

These instructions assume that you are familiar with Newton Toolkit. If you need more information, see the *Newton Toolkit User's Guide*. ◆

Take the following steps to build a Newton Book Reader package in NTK:

1. **Open a new project in NTK.**

   If you just want to practice building a book package, you can use the `Simple` project to build from the `SimpleStory.f` file.

2. **Add the Book Maker file to the project using the Add File… command in the Project menu.**

   To remove a file from the project, use the Remove File… command in the Project menu.

3. **Using the Output Settings… command in the Project menu, set Output to Book, and change the book package's name and application symbol from the default values supplied by NTK.**

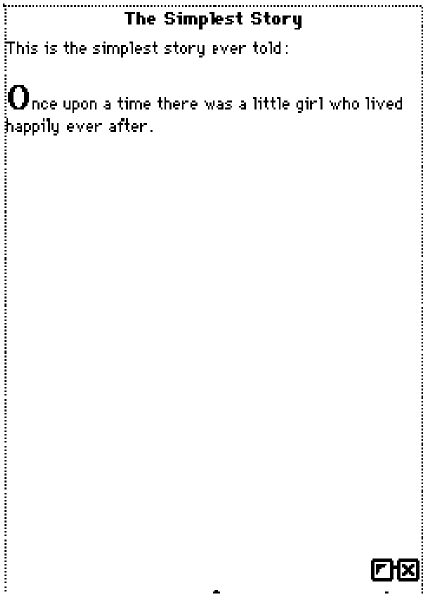
   If you are working with one of the example book project files supplied with Book Maker, this is preset for you.

4. **Build the project just like any other NTK project and download the package that is produced.**

   The book package appears in the Extras Drawer.

When the book package is opened on the Newton MessagePad, it looks like the source file it came from, as illustrated in Figure 2-5.

Figure 2-5 The sample book



The Simplest Story

This is the simplest story ever told:

Once upon a time there was a little girl who lived happily ever after.

Adding a Short Title

Long titles such as `The Simplest Story` do not always fit well in the Extras drawer. You can use the `.shortTitle` command to give your book another title that appears only in the Extras Drawer—the title specified by the `.title` command still appears at the top of the page.

To use the `.shortTitle` command, place it at the beginning of any line in the book source file; the remainder of this line contains the text that is to be the book's Extras Drawer title. In the following example, the short title `Pix` appears in the Extras Drawer and the title that appears at the top of every page in the book is `The Picture Story`:

```
.title The Picture Story
.shortTitle Pix
```

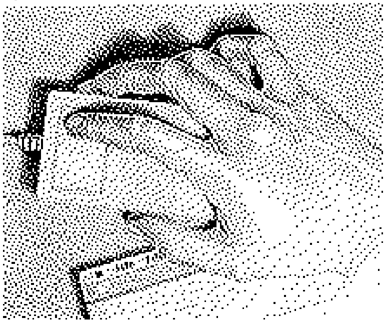
Adding Graphics

Adding graphics to Book Reader books or application help is as easy as adding the `.picture` command to your book source file and pasting a picture on the line that follows it. For example, to add a picture to the previous example, place the `.picture` command at the end of the file and then paste a picture on the line following it. The source file should then look like the following example, taken from the `PictureStory` example file:

```
.title The Picture Story
.isbn simplePicts
.shortTitle Pix
.story
This is the simplest story ever told:

Once upon a time there was a little girl who added
pictures to books with the greatest of ease.

.picture
```



WARNING

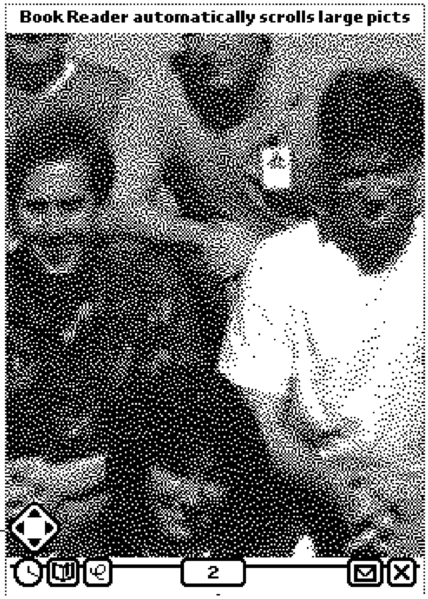
Pictures used in book source files must be in the `'PICT'` format. The Newton MessagePad does not support the use of all `'PICT2'` resources. Book source files using `'PICT2'` resources compile correctly, but the `'PICT2'` graphics do not display on the screen of the Newton device. ▲

An easy way to convert a graphic into `'PICT'` format is to paste it into any version of HyperCard, which automatically converts all graphics to this format. You can then copy the graphic from HyperCard and paste it into your book source file.

Large Pictures Scroll Automatically

The Newton Book Reader automatically displays scroller controls for pictures that are too large for the screen of the Newton device on which the book is displayed. The scroller controls are shown in Figure 2-6. The scroller can be suppressed by using the `NoScroller` flag. Flags are discussed in the section “Flags” beginning on page 3-6, and the `NoScroller` flag is documented under “Content flags” beginning on page A-34.

Figure 2-6 Scroller controls



The `BigPictureStory` file and the `BigPicture.pkg` book package illustrate this feature of Newton Book Reader.

Note

Application help screens do not scroll. ◆

Adding Comments to the Book Source File

As your book source files become increasingly complex, you may want to use comments to document them. A **comment** is a line of text that appears in the source file but does not appear in the compiled book.

Prefix your comments with a dot, and a pound (#) sign so that the Book Maker application ignores them when processing your source file, as in the following example:

```
.# This is a comment. Book Maker ignores it.
```

You'll find many examples of comments in the example book source files supplied with Book Maker.

Where to Go From Here

Now that you have a basic understanding of the book-building process, you can focus on the material in this user's guide that best suits your goals and working style.

- You can go on to Chapter 3, "Using the Book Maker Language," for an overview of the Book Maker command language. This chapter contains information about using layouts and flags to create interesting page designs, and creating browsers and kiosks to allow the user to better navigate through the digital book. All aspects of the Book Maker command language that do not require knowledge of NewtonScript are described in this chapter, and in Appendix A, "The Book Maker Language."
- You can turn to Chapter 4, "NewtonScript in Books," for information on using NewtonScript and Newton object system prototypes in Newton digital books.
- You can consult Chapter 5, "Application Help," for information on using Book Maker and NTK to add help screens to Newton applications.
- You can experiment on your own with Book Maker. All of the commands and flags in the Book Maker language are described in Appendix A, "The Book Maker Language." However, it is strongly recommended that you read Chapter 3, "Using the Book Maker Language," before creating your own books or application help.

CHAPTER 3

Using the Book Maker Language

You've now seen how to create the simplest book source file and build a Newton book package from it. This chapter describes the Book Maker commands used to lay out text and graphics on the page, and to provide navigational tools, browsers, and kiosks, in digital books. This chapter, in conjunction with the previous one, outlines all aspects of the Book Maker command language that do not require knowledge of NewtonScript.

Using Layouts

A Book Maker command beginning with the keyword `.layout` is called a **layout command**, or **layout.** Layouts specify the placement of text and graphics on the page.

The use of layout commands is optional. If no layout commands are defined for a book, Book Maker uses the entire width of the screen to lay out content items. For example, the book *The Simplest Story,* shown in the previous chapter, used this default formatting.

Defining Layout Commands

You can define layout commands as needed while creating the book source file, or you can define all of the layouts used in a book at the beginning of the book source file. Although the approach you take is strictly a matter of personal preference, keeping all of the layouts together at the beginning of the book makes it easier to change every occurrence of a particular layout in the book if you should need to do so. If you eventually build up a library of layouts that you use in multiple books, this approach makes it easier to copy and paste your layouts into a new book source file.

The `.layout` command requires that you specify a name for the format it defines; the format can then be referred to by name. To name a layout command, place its name immediately following the `.layout` keyword. It is important to give each `.layout` a different name, because repeating the names of `.layout` commands causes errors.

The `.layout` command also requires that you specify the widths of the columns it defines. Newton Book Reader divides the page into twelve equal vertical strips; the widths of columns are specified as a number of these strips.

The following example creates a layout named `simple`. This layout defines a single column that is twelve strips wide—the entire width of the page. A named layout such as this one is useful for returning to the default page format after using another layout that formats the page differently.

```
.layout simple 12
```

The next example creates a layout named `threeCol` that divides the page into three equal columns, each four strips wide:

```
.layout threeCol 4 4 4
```

In practice, the size of a usable column is determined by the font size used to display the text and the physical size of the screen on which it is displayed. For most books, you get the best results by dividing a screen the size of the MessagePad into no more than two or three columns.

Applying Layouts

When Book Maker encounters a `.layout` command, it applies the formatting specified by that command to all subsequent content items (such as stories, pictures, and so on) in the book source file until it encounters another layout command.

The simplest way to use layout commands is to place them in your book source file ahead of the content items that they affect. Note that only one layout is allowed per page on the Newton screen; every time you switch layouts the first content item that has the new layout is placed on a new page.

The following example, taken from the `LayoutStory` example book source file, shows the simplest way to apply layouts:

Note

The example book source files included with Book Maker all use the recommended 3.33 inch page width. However, the book source listings shown in this guide use the full width of the page in order to save space. ◆

The first layout, `threeCol`, creates a page that looks like the one shown in Figure 3-1. The second layout, `simple`, returns to the default page format—it defines a page that looks like all of the previous examples in this guide.

```
.# this layout divides the page into three columns
.layout threeCol 4 4 4

.story
.layout threeCol 4 4 4
This paragraph uses the layout defined above. The threeCol layout divides
the page into three equal columns of four grid units each. Text flows from
the top of the leftmost column to the bottom of the rightmost.

Because you can have only one layout command per page, you need to turn
to the next page to see the next layout example in this book.
```

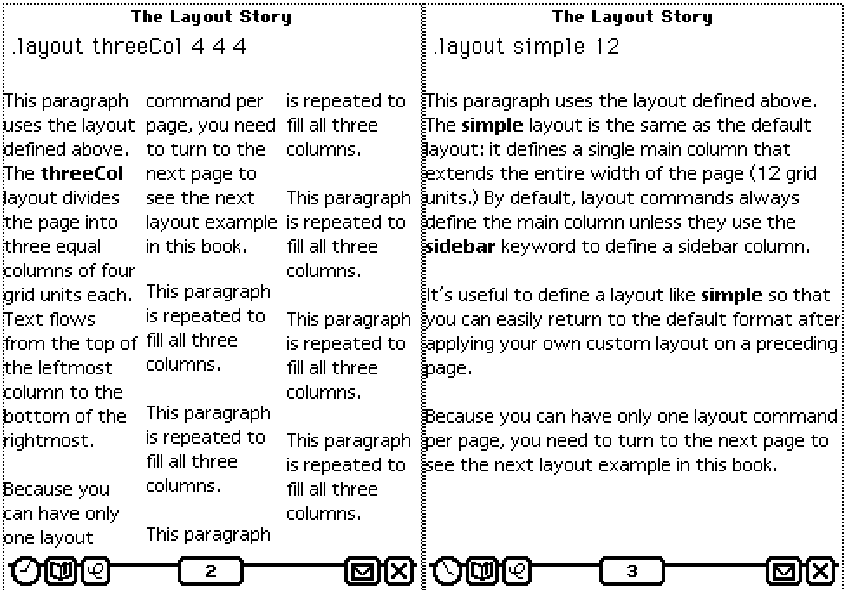
This paragraph is repeated to fill all three columns.

.layout simple 12
.story
This paragraph uses the layout defined above. The **simple** layout is the same as the default layout: it defines a single main column that extends the entire width of the page (12 grid units.) By default, layout commands always define the main column unless they use the **sidebar** keyword to define a sidebar column.

It's useful to define a layout like **simple** so that you can easily return to the default format after applying your own custom layout on a preceding page.

Because you can have only one layout command per page, you need to turn to the next page to see the next layout example in this book.

Figure 3-1 Pages formatted with the `threeCol` and `simple` layouts



The Layout Story

.layout threeCol 4 4 4

This paragraph uses the layout defined above. The **threeCol** layout divides the page into three equal columns of four grid units each. Text flows from the top of the leftmost column to the bottom of the rightmost.

Because you can have only one layout command per page, you need to turn to the next page to see the next layout example in this book.

This paragraph is repeated to fill all three columns.

This paragraph is repeated to fill all three columns.

This paragraph is repeated to fill all three columns.

This paragraph is repeated to fill all three columns.

This paragraph is repeated to fill all three columns.

The Layout Story

.layout simple 12

This paragraph uses the layout defined above. The **simple** layout is the same as the default layout: it defines a single main column that extends the entire width of the page (12 grid units.) By default, layout commands always define the main column unless they use the **sidebar** keyword to define a sidebar column.

It's useful to define a layout like **simple** so that you can easily return to the default format after applying your own custom layout on a preceding page.

Because you can have only one layout command per page, you need to turn to the next page to see the next layout example in this book.

Incidentally, the “layout commands” shown in Figure 3-1 are not real layout commands, but story text used to illustrate the layout in effect for those pages. The lines displaying these commands begin with spaces, not dots, so Book Maker does not consider them commands. The layout commands that actually implement these formats are in the book source files, but they are not displayed on the Newton screen.

The extra space was added to these lines intentionally so that their text would appear in the example book. This technique brings to light a common mistake: if a command in your book source file just won’t work, make sure the line that it’s on begins with a dot, not a space.

Applying Layouts by Name

All of the book source examples shown so far have defined layouts just ahead of the content items that use them. Once a layout has been defined, however, you can use its name to apply it to subsequent content items by using the `layout=` keyword. The following example defines a layout and then applies it by name to a content item:

```
.layout threeCol 4 4 4
.layout simple 12

.story layout=threeCol
This story text uses the threeCol layout.

.story
This story text uses the simple layout because no other layout is specified.
```

Note that content items without `layout=`*layoutName* specified use the default layout, which is the one specified by the most recent `.layout` command. If you define your default layout last in the set of layout definitions placed at the beginning of the file, you’ll need to apply `layout=` commands only to those content items that deviate from the default layout.

Remember that only one layout can apply to any page. Every time you switch layouts the first content item with the new layout is placed on a new page.

Although applying layouts by name is a useful technique, the short examples shown in this book don't require its use. To get a better idea of how named layouts are used, see the `MoreBrowsingStory` and `KioskStory` files provided with Book Maker.

Flags

A **flag** is a keyword that may be appended to a Book Maker command to specify various options. You can use flags singly or can combine them to achieve elaborate effects.

There are a handful of flags used only to modify layouts or entire documents; these flags are called **layout flags** and **document flags,** respectively. The flags that modify the display of content items are called **content flags.** Most of the flags in the Book Maker language are content flags.

This section provides a brief overview of the use of flags; for a complete listing of all of the flags available in the Book Maker command language, see the "Flags" section of Appendix A, "The Book Maker Language."

Using Flags

To use a flag, append it to the command it will affect. For example, you can use the `edges` flag to draw a box around the edges of a content item, as shown here in an example taken from the `FlagStory` example book source file:

```
.# create narrow columns to heighten the edge flag's effect
.layout doubleCol 6 6

.story edges
On the Newton screen, this story text appears with a line drawn around its
edges, courtesy of the edges flag.
```

The flags defined for a particular content item remain in effect until Book Maker encounters a new content command, such as a `.story` or `.picture` command.

Using Edge Flags

The `edges` flag is one of a set of flags that you can use to ornament content items with lines, boxes, and rounded-corner boxes. You can further modify the content item by adding other edge flags, such as the `rounded` and `edgeWidth` flags, as shown in the following example:

```
.# draw a box with rounded corners around the story text
.story edges edgeWidth=4 round
```

Adding the **rounded** flag causes Book Maker to draw a rounded-corner box around this text. The **edgeWidth** flag is used to specify the width of the line, which for this text is quite heavy at a four-pixel width.

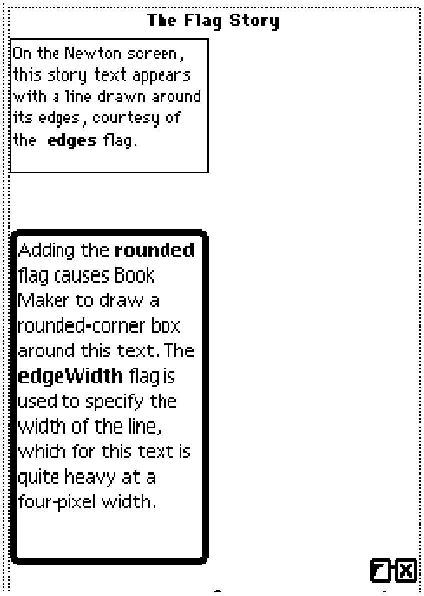
The `rounded` flag specifies a box with rounded corners. You can use the optional `edgeWidth` flag to specify in pixels the width of the line to be drawn by the edge flags. Place the `edgeWidth` flag on the same line as the edge flag it modifies.

edgeCommand `edgewidth=`*number*

When processed through Book Maker and NTK, these examples look like the screen shown in Figure 3-2.

For a complete list of available edge flags, see the “Edge Flags” section of Appendix A, “The Book Maker Language.”

Figure 3-2 Using edge flags



The Flag Story

On the Newton screen, this story text appears with a line drawn around its edges, courtesy of the **edges** flag.

Adding the **rounded** flag causes Book Maker to draw a rounded-corner box around this text. The **edgeWidth** flag is used to specify the width of the line, which for this text is quite heavy at a four-pixel width.

Using the Sidebar Flag

The `sidebar` flag can be used to modify layout commands as well as content commands.

When used in a layout command, the `sidebar` flag defines a column on the right or left edge of the screen. This sidebar column can provide marginal text, hanging indents, and other special effects. In a layout command, the numeric value preceding the `sidebar` flag defines the width of the sidebar column. Similarly, the `main` flag uses the numeric value preceding it to define the width of the main column.

For example, the following command defines a layout named `leftSide` that consists of two columns. The sidebar column is four strips wide (1/3 of the entire width of the page) and begins at the left edge of the screen. The main column is eight columns wide and fills the remaining width of the page:

```
.layout leftSide 4 sidebar 8 main
```

The order in which these identifiers appear determines where the sidebar is placed in relation to the main column. Thus, you can define a layout with a sidebar to the right of the main column as in the following example:

```
.layout rightSide 8 4 sidebar
```

Because the main column is always defined by every layout, the use of the `main` flag in layout commands is optional. It can be included, however, to make clear which column in a layout is the main column.

When the `sidebar` flag appears in a content command, it specifies that the content item is to be placed in the currently defined sidebar column; thus, to place text or graphics in the sidebar, append the `sidebar` keyword to the `.story` or `.picture` command associated with the content item to place in the sidebar column.

Content items without the `sidebar` keyword are placed in the main column by default.

The following example, taken from the `FlagStory` example book source file, uses the `sidebar` flag in layouts and in content commands:

```
.# this layout has a 4-unit sidebar and an 8-unit main
.layout leftSide 4 sidebar 8

.# place this text in the sidebar
.story sidebar
Sidebar text
.# no sidebar keyword, so this text goes in main col
.story
This page uses the layout command defined above. This layout, named
leftSide, defines a sidebar column four grid units wide that appears to the
left of the main column. The remaining eight grid units make up the main
column.

.# place this picture in the sidebar
.picture sidebar
```

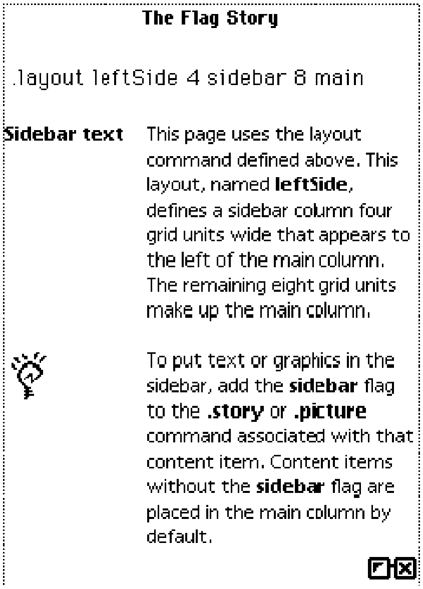
```
.# no sidebar keyword, so this text goes in main col
.story

To put text or graphics in the sidebar, add the sidebar flag to the .story
or .picture command associated with that content item. Content items
without the sidebar flag are placed in the main column by default.
```

The book page produced by this code fragment looks like the one shown in Figure 3-3.

A similar example that defines the sidebar column on the right-hand side of the screen follows this one in the `FlagStory` book source file and the `Flags.pkg` example book package.

Figure 3-3 Using the `sidebar` flag in layouts and content items



The Flag Story

.layout leftSide 4 sidebar 8 main

Sidebar text

This page uses the layout command defined above. This layout, named **leftSide**, defines a sidebar column four grid units wide that appears to the left of the main column. The remaining eight grid units make up the main column.

To put text or graphics in the sidebar, add the **sidebar** flag to the **.story** or **.picture** command associated with that content item. Content items without the **sidebar** flag are placed in the main column by default.

The following example, also taken from the `FlagStory` book source file, defines a layout named `twoCol` that creates a four-unit sidebar and an eight-unit main column:

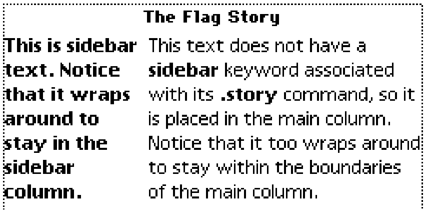
```
.# this layout has a 4-unit sidebar and an 8 unit main
.layout twoCol 4 sidebar 8

.# place this text in the sidebar
.story sidebar
This is sidebar text. Notice that it wraps around to stay in the sidebar
column.

.# no sidebar keyword, so this text goes in main col
.story
This text does not have a sidebar keyword associated with its .story
command, so it is placed in the main column. Notice that it too wraps
around to stay within the boundaries of the main column.
```

Appending the `sideBar` flag to a content command places its associated content item in the sidebar column. Text in either column is automatically wrapped to stay within the confines of the column. The text in the previous example appears on the Newton screen as in Figure 3-4.

Figure 3-4 Text formatted with the `twoCol` layout



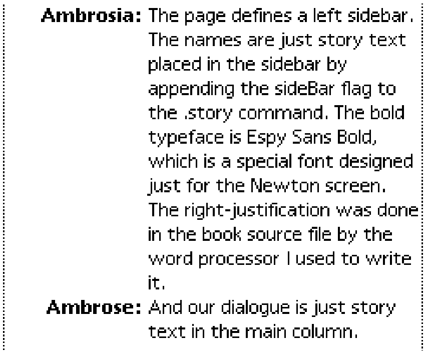
The Flag Story

This is sidebar text. Notice that it wraps around to stay in the sidebar column.

This text does not have a **sidebar** keyword associated with its **.story** command, so it is placed in the main column. Notice that it too wraps around to stay within the boundaries of the main column.

Book Maker uses the font, style, and justification information in the book source file; for example, the screen shown in Figure 3-5 uses right-justified text in a sidebar at the left of the main column.

Figure 3-5 Right-justified text in a sidebar to the left of the main column



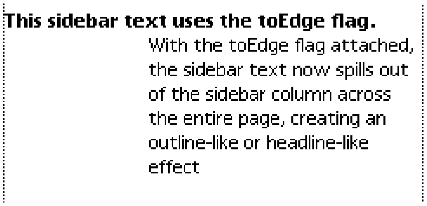
Ambrosia: The page defines a left sidebar. The names are just story text placed in the sidebar by appending the sideBar flag to the .story command. The bold typeface is Espy Sans Bold, which is a special font designed just for the Newton screen. The right-justification was done in the book source file by the word processor I used to write it.

Ambrose: And our dialogue is just story text in the main column.

Using the toEdge Flag

Appending the `toEdge` content flag to sidebar text can produce a "hanging indent" or outline-like effect, as shown in Figure 3-6:

Figure 3-6 Applying the toEdge flag to sidebar text



This sidebar text uses the toEdge flag.

With the toEdge flag attached, the sidebar text now spills out of the sidebar column across the entire page, creating an outline-like or headline-like effect

Although Book Maker does not wrap text around graphics, careful use of the `toEdge` flag can produce a similar effect. The following book source example is taken from the `FlagStory` book source file included with NTK. The named layout `sideby` creates a six-unit sidebar to the right of a six-unit main column, effectively splitting the screen down the middle and allowing you to control placement of items by using the `sidebar` flag on individual content items:

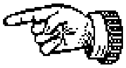
.layout sideby 6 6 sidebar
.story toEdge

More Layout Tricks

.story

As you can see, Dickens supports text next to graphics.

.picture sidebar alignCenter

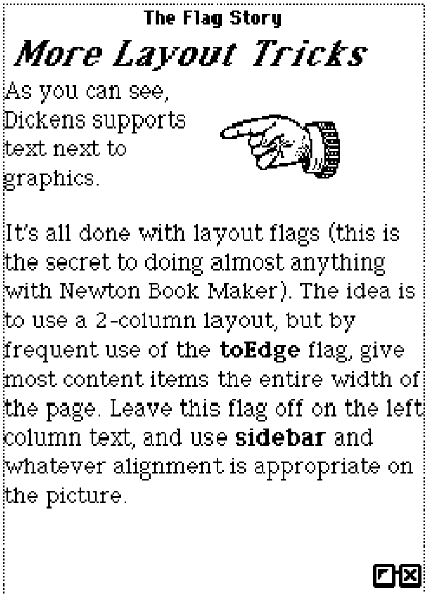


.story toEdge

It’s all done with layout flags (this is the secret to doing almost anything with Newton Book Maker). The idea is to use a 2-column layout, but by frequent use of the **toEdge** flag, give most content items the entire width of the page. Leave this flag off on the left column text, and use **sidebar** and whatever alignment is appropriate on the picture.

When processed with Book Maker and NTK, this book source example produces the screen shown in Figure 3-7.

Figure 3-7 Wrapping text around a graphic



The Flag Story

More Layout Tricks

As you can see, Dickens supports text next to graphics.

It's all done with layout flags (this is the secret to doing almost anything with Newton Book Maker). The idea is to use a 2-column layout, but by frequent use of the **toEdge** flag, give most content items the entire width of the page. Leave this flag off on the left column text, and use **sidebar** and whatever alignment is appropriate on the picture.

Using Sidebar Alignment Flags

You may have noticed the use of the `alignCenter` flag on the picture in the previous example. The `alignCenter` flag is one of several content flags you can use to alter the vertical alignment of items in the sidebar column. These flags—`alignTop`, `alignCenter`, and `alignBottom`— locate the content item in the sidebar relative to the previous content item in the book source file.

For example, attaching the `alignCenter` flag to the picture in the previous example vertically aligns the item in the sidebar column with the center of the story text in the main column.

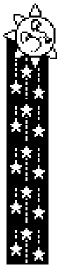
The following example uses the `alignTop` flag to align a picture in the sidebar with the top of the text in the main column.

```
.layout bedtimeStory 3 sidebar 9
.# put this title in the sidebar and spill its
.# contents out of the sidebar column
.story Sidebar toEdge
The Boy Who Flew To The Moon

.# Put this story in the main column
.story
Once upon a time, there was a little boy who dreamed of flying to the moon.
He would fly there in a sleek silver rocket ship with his trusty companion,
Biff The Wonder Dog.

When they arrived, he would eat green cheese all day long, set reduced
gravity sports records and never have to do any homework.

.# Put this pict in the sidebar and align it with the
.# top of the text in the main column
.picture sidebar alignTop
```

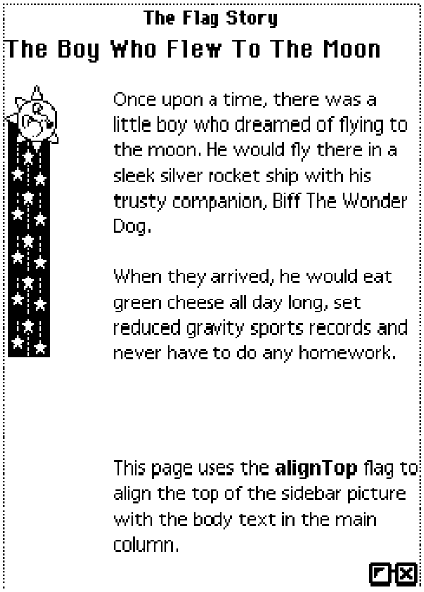


```
.story pageBottom
This page uses the alignTop flag to align the top of the sidebar picture
with the body text in the main column.
```

When processed through Book Maker and NTK, the previous example produces the screen shown in Figure 3-8.

The `alignBottom` flag aligns the sidebar with the bottom of the previously defined content item. The following code example, demonstrating the use of the `alignBottom` flag, is based on the previous example; only the relevant parts of the book source file are shown.

Figure 3-8 Using the alignTop flag



The Flag Story

The Boy Who Flew To The Moon

Once upon a time, there was a little boy who dreamed of flying to the moon. He would fly there in a sleek silver rocket ship with his trusty companion, Biff The Wonder Dog.

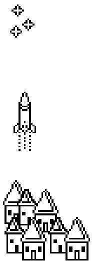
When they arrived, he would eat green cheese all day long, set reduced gravity sports records and never have to do any homework.

This page uses the **alignTop** flag to align the top of the sidebar picture with the body text in the main column.

.story pageBottom
This version of the story uses the **pageBottom** flag to pin this
explanatory paragraph to the bottom of the page. The sidebar picture is
aligned to the bottom of this paragraph by attaching the **alignBottom** flag
to the **.picture** command associated with it.

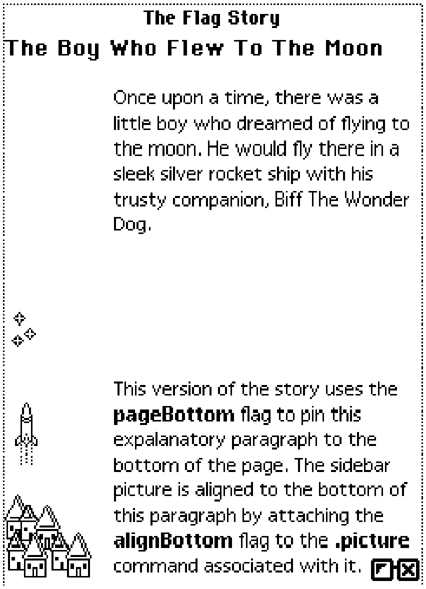
.# Put this pict in the sidebar and align it with the
.# bottom of the text in the main column

```
.picture sidebar alignBottom
```



This code example produces the sidebar graphic and final paragraph on the screen shown in Figure 3-9.

Figure 3-9 Using the alignBottom flag



The Flag Story

The Boy Who Flew To The Moon

Once upon a time, there was a little boy who dreamed of flying to the moon. He would fly there in a sleek silver rocket ship with his trusty companion, Biff The Wonder Dog.

This version of the story uses the **pageBottom** flag to pin this expalanatory paragraph to the bottom of the page. The sidebar picture is aligned to the bottom of this paragraph by attaching the **alignBottom** flag to the **.picture** command associated with it.

The Book Maker language provides a number of other content flags; they are listed in Appendix A, “The Book Maker Language.”

Formatting Recommendations

To obtain the best results from Book Maker, keep the following recommendations in mind when creating a book source file:

- Use tabs, rather than spaces, to align text. (Note, however, that the MessagePad supports left-aligned tabs only.) You'll save space in your book if you set as few tab stops as possible (or set them only in those stories that need them). Each content item can have only one set of tabs, and the tabs must be set in the first paragraph of the content item; that first paragraph need not actually use the tabs, but they must be present for subsequent paragraphs in the story to use them.
- Do not hyphenate text manually (by typing "-" at the end of the line). Although the text may appear correctly on a MessagePad, it will not appear correctly on machines with larger screens and it will prevent the user from using the Find service to search for the hyphenated word.
- Don't interrupt story text with commands; doing so may cause Book Maker to lose style information. Instead, place commands associated with a story at the beginning or the end of the story text.
- If your word processor supports superscripted or subscripted text, you generally need not do anything special to your book source file to obtain appropriate display on the Newton MessagePad; the MessagePad automatically uses a smaller point size to display superscripted and subscripted text. However, use of special text styles can produce unexpected results. The XTND filters used by Book Maker do not support the special styles, such as small caps or all caps, that some word processor programs implement.
- On future Newton platforms, Book Reader may automatically reformat pages to take advantage of a larger screen size. Thus, if a title ends up on the bottom of a page and its associated story on the next page, it's recommended that you use the `KeepWith` flag on the story, rather than `StartsPage` flag on the title (these flags are explained in Appendix A, "The Book Maker Language.") Although both approaches provide the

same appearance on smaller screens, the latter solution may produce inappropriate pagination on future Newton devices.

Creating a Browser Pane

The main Book Reader browser is displayed when the user taps the overview button in the middle of the button bar at the bottom of the MessagePad screen. A browser pane is a floating view which initially displays the top-level heading lines, and either expands to show sub-headings, or to the beginning of that section in the book. You can think of a browser as a table of contents for the book.

Book Maker allows you to define multiple browsers for a book. If you do so, they are listed by name in the main browser, and tapping a browser name takes the reader to that browser. This might be considered analogous to a listing of individual tables of contents for each chapter in a conventional paper-based book.

The items that appear in a browser are text items in the body of the book that have been tagged with the commands `.subject` and `.chapter`. The browser always displays a single line in a single type style for each subject or chapter tagged with either of these commands. The font size and style of items in the browser is predefined by Book Reader; thus, you can use different layouts for text in the body of the book without affecting the appearance of items in the browser.

The `.subject` command tells Book Reader several things about the text on the line following it. First, it identifies this text as the beginning of a new content item and indicates to Book Reader to use this text as the title text for the story or picture immediately following. Second, it indicates the level at which the subject text appears in the browser; this level is specified by the numeric value immediately following the `.subject` command. Book Reader automatically ranks the browser items by subject level to create a hierarchical table of contents.

Following the level argument are the optional flags used to specify the layout of the subject text as it appears in the body of the book. Layouts are applied

to browser items just as they would be to any other content item,
but they do not affect the appearance of the text in the browser. The browser automatically displays level 1 subjects in boldface type; other items are displayed in plain text.

Following these required arguments are optional arguments. The `name=` argument allows you to specify a name for the subject other entities such as kiosks use to find this subject. The `bro=` command specifies a browser in which to place the subject line when you want it to appear in a browser other than the main overview.

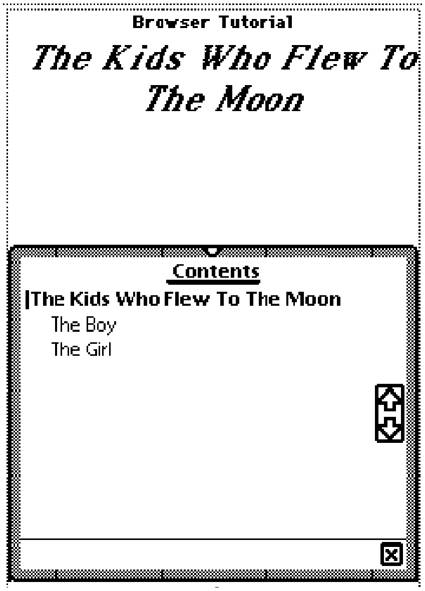
The `.chapter` command is a synonym for the `.subject 1` command. A complete discussion of the parameters to the `.chapter` and `.subject` commands is included in the “Content Commands” section of Appendix A, “The Book Maker Language.”

The example below, from the `BrowserStory` example book source file, creates two browser items. The first `.subject` command places the text *The Kids Who Flew To The Moon* in the center of the page (this command in fact creates the title page for this particular book) and adds it to the top level of the browser pane. The next `.subject` command places the text *The Boy* at the second level in the browser pane at level 2. The Book Reader browser automatically makes this subject line into a subheading of the level 1 browser item.

```
.subject 1 Centered
The Kids Who Flew To The Moon
.layout bedtimeStory 3 sidebar 9
.# put this title in the sidebar and spill its
.# contents out of the sidebar column
.subject 2 Sidebar toEdge
The Boy
```

When the example above is compiled, the browser looks like the one shown in Figure 3-10.

Figure 3-10 Browser pane from `BrowserStory`



Browser Tutorial

The Kids Who Flew To The Moon

Contents

The Kids Who Flew To The Moon

The Boy

The Girl

The BrowserOnly Flag

Adding the `browserOnly` flag to the `.subject` or `.chapter` command causes the text associated with that command to appear only in the browser pane, not in the body of the book. This flag is useful for arranging browser entries under labels that will not appear in the body of the book.

For example, if you wanted to make your title page a little fancier, you might add carriage returns to the text so that it to fills up the page more, as in the following variation on the previous example. Unfortunately, Book Maker uses only the first line of text following the `.subject` command to create the browser entry; thus, the example shown here would place the heading *The Kids Who* in the browser pane at level one.

.subject 1 Centered

The Kids Who

Flew

To The Moon

The following example solves this problem by using the `browserOnly` flag to create a `.subject 1` listing for the title page that is displayed only in the browser. The title page itself is displayed by the `.story` command. Because the `.story` command does not create a browser entry, only one entry for the title page appears in the browser.

.subject 1 browser only
The Kids Who Flew To The Moon

.story Centered

The Kids Who

Flew

To The Moon

See the `MoreBrowsingStory` example book source file for more examples of the creation of browser entries.

Creating a Kiosk

In the real world, a kiosk is a cylindrical structure on which advertisements are posted; in a Book Reader book, a **kiosk** is a navigational page containing "advertisements" for subject matter in the book. Tapping an item in a Book Reader kiosk takes the reader to the subject matter it represents.

Kiosks also provide a place that the user can always return to easily: a button that takes the user to the nearest kiosk always appears in the bookmark dialog box.

In addition to making content easier for the user to navigate, kiosk pages can lend visual interest to the book. As you can see from the example in Figure 3-11, kiosks can make use of graphical items in unusual layouts. The remainder of this section discusses the commands used to create this kiosk page.

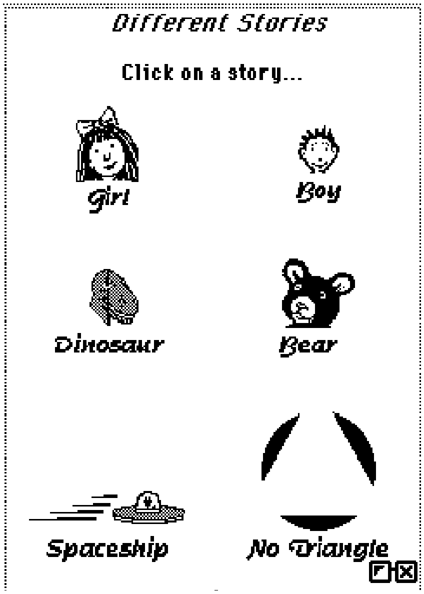
The kiosk page shown here uses a custom layout for placement of the pictures on the page. The layout definition for a kiosk page must include the `kiosk` flag.The layout defined in the following example, named `MyKiosk`, creates a six-unit main column with a six-unit sidebar. The use of the `main` keyword is optional, but is included here for illustrative purposes.

```
.# This is a layout for the kiosk.  Make sure to add the flag “kiosk”
.layout MyKiosk 6 Sidebar 6 Main kiosk
```

The definition of the kiosk itself begins with the `.kiosk` command. The previously defined `myKiosk` layout is applied by name.

```
.kiosk layout=MyKiosk name=Menu
```

Figure 3-11 Kiosk page from the Kiosks example book



The kiosk page itself must be named also. The `name=`*itemName* command attaches the specified name to a content item. The name can then be used by other entities in the book to navigate to the named content item.

The remainder of the kiosk definition places content items, such as pictures or text, on the kiosk page. The following example uses the `.picture` command to place the *Boy* picture in the main column on the kiosk page. The `centered` flag centers the picture horizontally in the column.

```
.picture layout=MyKiosk Main Centered goto=Boy
```

The `goTo=`*destination* command specifies the content item to be displayed when the user taps the picture of the boy on the kiosk page. The *destination* parameter must specify a content item that has been named with the `name=`*itemName* command. For example, the `goTo=Boy` command in the example above specifies that when the user taps this picture in the kiosk, Book Reader displays the story named `Boy`.

The definition of the kiosk is terminated with the `.endkiosk` command. Everything between the `.kiosk` command and the `.endkiosk` command defines the kiosk.

The entire definition of the kiosk page shown in Figure 3-11 includes commands to place the other five pictures on the kiosk page and specify `goto=` destinations for each; for a complete listing of this kiosk definition, see the `KioskStory` book source file.

Finishing Touches

This section discusses commands and techniques you can use to give your book a more polished appearance.

Adding Book Information for the “About” Slip

You can add commands to your book source file that provide information about the book’s contents, its author, its publisher, and its dates of publication and copyright. This information is presented to the end-user in the “About” slip accessed through the Information button on the status bar.

Five Book Maker commands are provided for this purpose: `.author`, `.publisher`, `.date`, `.expires`, and `.copyright`. To use these dot commands type the relevant information in the same line as the command, as in the following example:

```
.title Newton Book Maker User’s Manual
.isbn dickens:PIEDTS
.date 11/3/93
.author Bob Ebert, PIE DTS
.publisher Apple Computer, Inc.
```

```
.copyright © 1993 Apple Computer, Inc.
.expires 12/31/95
.shorttitle Dickens
.# Rest of book would follow from here.
```

Two additional commands, `.blurb` and `.key`, provides information about the book, but is not currently available to the end user. All the commands mentioned in this section are documented in the section "Content Commands" beginning on page A-11.

Adding Space Between Content Items

The `.space` command can be used to add precise amounts of space between content items; it's handy for aligning text and graphics for which the various alignment flags don't provide enough control. Using the `.space` command instead of inserting blank lines of text in the book source file also ensures that blank lines are not placed at the tops of pages, and results in the creation of slightly smaller book packages.

Placed after a content item in the book source file, the `.space` command inserts a specified amount of blank space below that content item on the Newton screen. The unit of space is specified in typographer's points; one **point** is $^1/_{72}$ inch. You can specify up to 63 points of space using the `.space` command, as in the following example:

```
.story
The following command inserts 63 points of space after this text.
.space 63
```

You can see examples of the use of the `.space` command in the `KioskStory` book source file.

Indenting Text

You can place the `.indent` command between a content command and its associated story text or picture to create margins on either side of the content item. The `.indent` command defines left or right margins; specify a value of `0` for no margin. The following example indents the story text by 30 points on the left and 20 points on the right. (A **point** is a typographer's unit of measure; there are 72 points in an inch.)

```
.story
.indent 30 20
This is my story text.
```

Adding Picture Headers

You can use the `.pictHeader` command to add a picture that appears at the top of every page, as in the following example, taken from the `TouchesStory` book source example file.

```
.pictheader
```

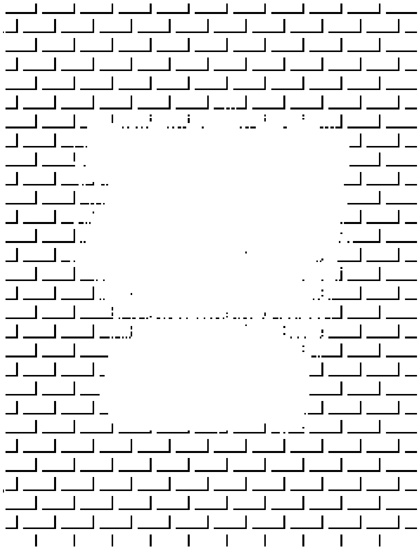
To see what this picture header looks like, open the `Touches.pkg` example book package included with Book Maker.

Oversize Picture Headers Spill Into Book Page

The space reserved for headers at the top of the page in a digital book is 16 pixels high. If you use this entire height or exceed it, the header spills into the area of the page normally reserved for content. Because the page content is drawn after the picture header, the page contents are always "on top of" the picture header. As a result, you can use oversize picture headers to provide interesting backgrounds for your book pages.

The following example, taken from the `FlagStory` book source file, illustrates this technique.

```
.pictHeader
```

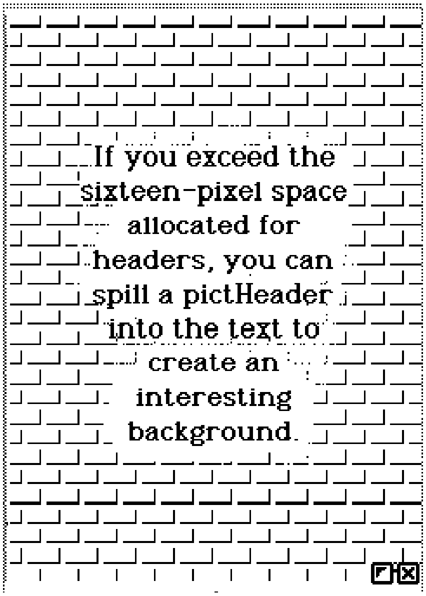


.story centered

If you exceed the sixteen-pixel space allocated for headers, you can spill a pictHeader into the text to create an interesting background.

The `.pictHeader` command causes the picture on the line following it to be used as the running headline that appears at the top of every page in the book. Because the picture in the example is larger than the 16-pixel space allocated for such headlines, it spills into the body text, creating the effect shown in Figure 3-12. Note also that blank space in story text is transparent; the carriage returns used to center this text vertically on the page do not obliterate the background.

Figure 3-12 Using oversize picture headers



If you exceed the sixteen-pixel space allocated for headers, you can spill a pictHeader into the text to create an interesting background.

Including External PICT Files

If you prefer, you can keep pictures in `'PICT'` files rather than paste them directly into the book source file. Only one picture should reside in each external `'PICT'` file.

To include the external `'PICT'` file in your book source, you need to follow the `.picture` command with the full pathname to the file. The full pathname to the file specifies the name of the disk on which the file resides, followed by the names of all folders you need to open to get to the file, in order, and concludes with the name of the file itself. Use a colon to separate each name in the path from the others, and surround the entire pathname with single quotation marks, as in the following example:

```
.picture 'MyDisk:OuterFolder:InnerFolder:MyPICTFile'
```

This command includes the picture stored in the `'PICT'` file “MyPICTFile” in the book source file. The “MyPICTFile” file resides in the “InnerFolder” folder. The “InnerFolder” folder in turn resides in the “OuterFolder” folder. The “OuterFolder” folder is on the disk “MyDisk.”

CHAPTER 4

NewtonScript in Books

You can further customize the behavior of a Newton digital book by adding NewtonScript commands to the Book Maker source file. This section describes how to include scripts, slots, view templates, and frames in a Newton digital book. The material in this chapter presumes some NewtonScript programming ability and some familiarity with the Newton object system.

Using NewtonScript in Book Source Files

The `.script` command is used to add NewtonScript commands to a book source file; it specifies that everything following it in the book source file is interpreted as NewtonScript, rather than as Book Maker command language. Book Maker returns to interpreting the content of the book source file as Book Maker command language upon encountering an `.endscript` command or another Book Maker command.

Everything between the `.script` and `.endscript` commands must be valid NewtonScript code as defined in *The NewtonScript Programming Language*.

You can supply a name for the script by placing the name immediately following the `.script` command. If the script's name is omitted, it becomes a `buttonClickScript` by default. Book Reader sends a `buttonClickScript` message to a content item whenever the user taps the content item.

The following example defines a simple `viewClickScript` method.

```
.# NewtonScript begins on the line following the .script
.# command
.script viewClickScript
// here to .endscript command is NewtonScript,
// not Book Maker commands
// note the NewtonScript comment style
playSound (ROM_FunBeep);
.endscript
.# now we are using Book Maker commands again.
```

Attaching Scripts to Content Items

Book Maker processes the file from top to bottom and attaches the script to the content item defined just before the `.script` command.

```
.story name=beeps
Tap on this paragraph to play the sound.

.# NewtonScript begins on the line following the .script
.# command
.script viewClickScript
// here to .endscript command is NewtonScript,
// not Book Maker commands
// note the NewtonScript comment style
playSound (ROM_FunBeep);
.endscript
```

```
.# now we are using Book Maker commands again
.# This command defines a new content item with no
.# script attached
.story name=silent
Tapping on this paragraph makes no sound.
```

In this example the `viewClickScript` method is attached to the story named `beeps`. When the user taps the `beeps` story, the system plays the `ROM_FunBeep` sound. Because the story named `silent` has no `viewClickScript` method attached to it, no sound plays when the user taps that story. Note that between the `.script` and `.endscript` commands, you must use NewtonScript, rather than Book Maker commands.

The content items in a Book Reader book are actually views that receive all of the same system messages that views in an application receive. Thus, you can define any script for a content item that you might define for a view in an application, such as `viewClickScript`, `viewSetupDoneScript`, and so on.

WARNING

Don't use `viewSetupFormScript` methods at the book or page level—your script will not be called. Instead, use the `viewSetupDoneScript` method. This method is called each time a page is imaged (except for printing). There are no restrictions on `viewSetupFormScript` methods for content items. ▲

In general, you should avoid hard-coded page numbers in scripts, as the book may paginate differently on a future Newton device having a different-sized screen. Instead, you can use the content-related NewtonScript methods to find content items by name or attribute. These methods are described in the "NewtonScript Methods" section of Appendix A, "The Book Maker Language."

This limitation can be overcome, however, and your script's speed increased by not having to perform the search, if you only use hard-coded page numbers after checking which type of screen the book is being imaged on. This information is stored in the book's `curRendering` slot, which is described in “Information Available From Reserved Slots” on page 4-16, along with a code snippet demonstrating how to use it.

For more information regarding the `.script` and `.endscript` commands, see the “Content Commands” section of Appendix A, “The Book Maker Language.”

Attaching Scripts to the Page

You can attach a script to a layout by placing the `.script` command after the definition of the layout. The script is then available on any page that uses that layout. These scripts are referred to as **page scripts.**

Page scripts can be used, for example, to draw a special background or to handle a pen event not provided for in the `viewClickScript` method of a content item formatted with that layout.

If the page displays a content item that has its own script, any scripts attached to the content item override those attached to the page, which makes sense if you think of the content item as the child of the page view. Thus, if the page uses a layout that defines a `viewClickScript` method and a content item on the page also defines a `viewClickScript` method, the content item’s `viewClickScript` method is invoked when the user taps the content item on the screen. If you want the `viewClickScript` method of the page to be invoked as well, return `nil` from the content item’s `viewClickScript` method.

Attaching Scripts to the Entire Book

Scripts defined at the beginning of the book source file (before any content or layout commands) are made available throughout the book by parent inheritance. These scripts are referred to as **book scripts.**

Just as content scripts override page scripts, page scripts override book scripts.

Sharing NewtonScript Code

You can save some space in the book package by sharing scripts among content items that have common behavior. You can share a script (or any other NewtonScript code, for that matter) by defining it in a text file included in your NTK project or in the book's preamble. Either approach works equally well in terms of its ability to make the code available throughout the book; the approach you take is determined by your development goals.

If you want to make your book source file as self-contained as possible, you'll want to define shared NewtonScript code in the book's preamble. (See the discussion of the `.preamble` command, on page page A-9 of Appendix A, "The Book Maker Language," for more information.) On the other hand, if you want to use this code in more than one book, you can define the shared code in a text file and include it in your NTK projects.

Shared Script Example

Although the `goTo=`*destination* command provides navigational capability to content items, it does not provide any highlighting behavior. The following example shows how to reuse a script that provides button-like highlighting behavior for any content item. This script is especially useful for adding highlighting behavior to kiosk entries, as in the following example:

```
// defines button-like behavior for a content item
// place in a text file included in your NTK project
// or in the book's preamble
kioskEntryScript := func(aClick) begin
   if (:TrackHilite(aClick)) then
         begin
            :buttonClickScript();
            :Hilite(NIL);
         end;
```

```
      TRUE;
end;
```

A content item in the kiosk can acquire this button-like behavior by calling the `kioskEntryScript` from its `viewClickScript` method, as shown in the following example:

```
.story centered goto=atlanta
.script slot viewClickScript
kioskEntryScript
.endscript
Atlanta
```

The `slot` parameter to the `.script` command creates a slot in the view that images the content item; this parameter is explained fully in the section "Adding Slots to Views," later in this chapter.

This approach saves space in the book package because even though all of the content items in your kiosks may use this little `kioskEntryScript` method over and over again, the bytecodes for this script are compiled only once, when the NTK project is compiled.

Using Protos and View Templates in Books

The `.form` command allows you to add protos, user protos, and view templates to your book. Each view that you add using this command is considered a separate content item, and appears on the page as a fully functioning code object.

The `height` and `width` parameters to the `.form` command are required to reserve space for the view to occupy on the page. Views created with the `.form` command appear immediately following the previous content item, just as any other content item would.

The following example shows the use of the `.form` command to include a custom view prototype called `myFunkyProto`. Layouts and user protos added with the `.form` command must be included in the book's NTK project file; system prototypes are included automatically by NTK:

```
.form height=32 width=168
.# user protos require the layout_ prefix
layout_myFunkyProto
.endform
```

The `.form` command begins the line, followed by the required values for the view's height and width. The proto, user proto, or layout from which the view is to be created is specified on the next line. System protos, such as `protoStaticText`, are specified by placing their name on this line; the names of user protos and layouts must be prefixed with the `layout_` keyword.

Note that the `.form` command must be followed by a corresponding `.endform` command. For more information regarding these commands and their parameters, see the "Content Commands" section of Appendix A, "The Book Maker Language."

Searching in .form Content Items

There are three methods provided to allow for searching in a `.form` content item. A text slot can be added to the proto or template, or a `FormSearchScript` or `BookSearchScript` can be provided.

If a `.form` content item contains a slot called `text`, the Find service searches it automatically. You can add this slot when you create the view, or use the `.attribute` command (explained in "Adding Slots to Content Items" beginning on page 4-9) to add a `text` slot, as in the following example:

```
.form w=100, h=50
layout_myCoolView
.endform
.attribute text: "sports scores"
```

Book Reader also sends a `.form` content item a `FormSearchScript` message when it encounters a `.form` content item while performing a search. To handle this message, add a `FormSearchScript` to the `.form` content item.

Furthermore, the system-supplied book search engine can be overriden by providing a `BookSearchScript` to the book. This message passes in the content item to be searched as a parameter, allowing the method to selectively handle the search for any particular content item, or let the system-supplied search engine search that content item.

When a `.form` content item is to be highlighted to indicate that the search was successful, the system sends it the `FormHiliteScript` message.

`FormSearchScript`, `BookSearchScript`, and `FormHiliteScript` are discussed in the section "Book Reader Messages"of Appendix A, "The Book Maker Language."

Storing and Accessing Data in Digital Books

This section describes the various ways to store data in your digital books. Global data can be stored as part of the book package, in a soup maintained by Book Reader, or as a slot in the book's run-time view frame. Local data can be stored in a content item, and in the run-time view which images the content item.

There are a number of issues to consider in deciding where to store data,

- Availability; does this data need to be available through out the book, or to a single content item?
- Memory management; is it acceptable to use up this type of memory, and for this length of time?
- Writeability; will this data ever need to change?
- Persistence; is it acceptable to have this data initialized to the same value every time the book is opened?

Adding Slots to Content Items

Content items are represented internally as frames. These frames are generated by Book Maker, and contain a number of slots need by Book Reader to image them. You can use the `.attribute` command to create multiple slots which are stored in a content item's frame. At run time, slots created with the `.attribute` command are read-only and are available to the current content item only.

The following example shows how to use the `.attribute` command:

```
.title Pizza Parlors
.isbn anyUniqueID

.#first content item
.story
Joe’s Pizza
.# This slot is available only to the Joe’s Pizza
.# content item
.attribute cash: true

.#second content item
.story
Luigi’s Pizza
.# These slots available only to the Luigi’s Pizza
.# content item
.attribute cash: nil, credit: nil, beer: true
```

Note that both content items store their own values for the `cash` slot.

Getting Data From Slots in a Content Item

When Book Reader opens the book package, it creates a slot called `item` that references the content item that contains the `.attribute` command. To access slots created using the `.attribute` command your scripts must dereference the item slot using the NewtonScript frame accessor (dot) operator, as in the following example:

```
.title Pizza Parlors
.isbn anyUniqueID

.story
Joe’s Pizza
.# These slots are available only to the Joe’s Pizza
.# content item
.attribute cash: true, smoking: true

.script viewSetupDoneScript
if (item.smoking) then
PlaySound(ROM_Poof);
.endscript
```

The example above plays the "poof" sound when the Joe’s Pizza story appears, because the smoking slot associated with Joe's Pizza (referenced through the item slot) has the value true.

Adding Slots to Views

Another way to create a slot that is associated with a content item is to use the slot parameter to the .script command. When you create a slot in this way, the slot and its contents are stored in RAM, in the view frame used to image the content item, rather than in the book package. Because the slot is stored in RAM, its value may be changed dynamically. Another thing to remember is that each slot you create this way uses up valuable heap space, so you should create these slots only when absolutely necessary.

Note also that each content item is imaged by a separate view. Data stored in a slot in one view will not be available to scipts in any other views, even if both views are imaged on the same page.

Fortunately, you rarely need to create view slots. It is recommended that you create view slots only when the view itself needs to change them. If you must create slots in views, be extremely careful when doing so lest you inadvertently degrade performance. Remember that in addition to the reserved slot names listed in the section "Reserved Slot Names" on page 4-16, there are a number of slots in the views which image content items that which can be altered.

Because view slots are stored in the view itself, view scripts need not do anything special to access them. The following example changes the value of the view slot `viewFormat`, causing the box created by the `edges` command to be drawn in gray instead of black:

```
.story edges
.script slot viewFormat
vfFrameGray+vfFillWhite+vfPen(1)
.endscript
This story uses a script slot to change
the pen pattern. It draws a gray box
around the story.
```

Global Data in Books

Newton digital books provide several structures that you can use to store global data. This section describes how to use these structures to store static and dynamic global data.

Book Data

When Book Maker processes a book source file, it creates a file that defines the structure and content of the digital book in NewtonScript. If you were to view the book definition file, Book Maker's output, with a text editor application, you would see at the very beginning of the file a frame defining the overall structure of the book. This frame, appropriately called `book`, is similar to the following frame:

```
book := {
   version: 1,
   isbn: "0-000-1111-0",
   title: "The Planets",
   author: "Copperfield Team",
   publisher: "Apple Computer, Inc",
   keywords:   "Space Planets Mars Venus Jupiter Pluto
               Neptune Saturn Uranus Mercury Earth",
   publicationDate: 4525593,
   data: {},   // Author's own data
   contents: Array(45, NIL),
   styles: [], hints: Array(45, NIL),
   browsers: [], templates: [], rendering: []};
```

As you can see, the book frame consists of a number of slots containing various kinds of data: numeric data, text strings, arrays, and frames. Of particular interest is the `data` slot—shown in boldface type—which is provided expressly for your use. In it, you can store your own raw data or NewtonScript slots, frames, and functions. This data remains with the book and is removed when the book is removed.

The `data` slot is intended as a repository for static data that must be globally available while the book is running. In this slot you can store any read-only data that is used by more than one view or content item and is specifically related to the book. For example, Book Maker uses this slot to store index entries created with the `.index` command.

Setting and Getting Book Data

The `BookData` method returns a reference to the frame that resides in the book's `data` slot. Your scripts can use this reference to store or retrieve information in this frame, as in the following example:

```
.title Barney's Bad Day
.isbn aRealDinosaur
```

```
.# setting some book data
.preamble
book.data.stuff := {text: “Godzilla gave Barney a stare
that could only mean one thing.”};
book.data.moreStuff := [ ... ];
.endpreamble

.story
“I love yoooou,” Barney said mellifluously. But as far
as Godzilla was concerned, the honeymoon was over.

.# getting some book data
.script viewSetupDoneScript func()
begin
   local stuff := :BookData().stuff;
   local myText := stuff.text;
   // do something with the text here
end
.endscript
```

Note that the Book Data frame is referred to in two different ways in the code above–by explicitly referring to the frame as `book.data` and by using the return value of the Book Reader method `BookData`. This is required since the data is stored at compile time, and is accessed at run time.

Using Author Data

Book Reader allows a developer to store data which must be available across runs, i.e. after the book is closed. This data is stored in a frame in a soup. This frame, unlike the Book Data frame, is writeable. It is thus a natural place to store user preferences which can change, but should be persistent across runs.

Author data is kept for the last eight books opened, opening a ninth book erases the first soup entry. You should store data here only if it needs to be kept across runs, since the book’s soup entry is kept around after the book’s package has been removed. On the other hand, if you want writeable data that will be stored after the ninth book is opened, you will need to write your own soup. For information about soups, see the *Newton Programmer's Guide.*

The AuthorData Method

The `AuthorData` method returns a reference to the soup-based frame in which Newton Book Reader stores author data. Your scripts can use this reference to store or retrieve information in this frame, as in the following example:

```
.# Store the user’s present location in Author Data when
.# the user taps on a city name. Note that each city is
.# its own story, so that they will be instantiated by
.# seperate views
.story
My Current Location is... (tap on a city)
.story
Cupertino
.script viewClickScript
:AuthorData().curCity := "Cupertino"
.endscript
.story
Boston
.script viewClickScript
:AuthorData().curCity := "Boston"
.endscript
.# And so on...

.# Then retrive this data elsewhere to present a
.# map of the user’s current city
.story
tap here for local map
.script viewClickScript
local whichMap := :AuthorData().curCity;
ShowStoryCard( 'mapName, whichMap,
            {left:40 ,right:200 ,top:40, bottom: 260} )
.endscript
```

Adding Slots to the Book

A slot added with the `slot` parameter to the `.script` command before any content or layout commands is available to all scripts in the book. As mentioned in the section “Adding Slots to Views” on page 4-10, adding slots in this way uses valuable heap space. This use of heap space is especially important, since this slot is in existence for as long as the book is opened, whereas a slot added to a view is garbage collected when the view is no longer needed.

However it is sometimes necessary to create a book-level slot when you need to communicate between views. Consider, for example, a book that holds information about restaurants. In this book the user may specify they like Romanian food in one view. To display a list of Romanian restaurants in another view, you will need to communicate this information to the other view. This data could be stored in the Author Data frame, but then it would be kept around after the book has been removed. Adding a book-level slot is thus a viable alternative.

If a book-level slot is added in this fashion, no special code is needed to access this slot. If the slot is referred to in any scripts, it will be found through the inheritance structure. For example:

```
.# This must appear before any content item or layout
.# commands
.script slot favoriteTypeOfFood
"Italian" //initialize to something everyone likes
.endscript
```

Then to access this data in a script simply refer to it as `favoriteTypeOfFood`.

One limitation of this approach is that the slot is created whether it is used or not. This is because the code that creates this slot necessarily resides at the top of the book definition file, and is thus always executed. This limitation can be overcome by referring directly to the `contentArea` view.

The `contentArea` view is higher up the parent hierarchy from the views which image content items. It is created when the book is opened, and exists until the book is closed. It is thus a convenient place to store global slots. The following line of NewtonScript code creates a slot in the `contentArea` view:

```
contentArea.favoriteTypeOfFood:= "Romanian";
```

Again, you can refer to the slot by name to access this data, it will be found through the inheritance scheme. The following code turns to the page whose text begins with the string stored in the `favoriteTypeOfFood` slot:

```
:TurnToPage(:FindPageByContent(favoriteTypeOfFood,0,nil));
```

Reserved Slot Names

Newton Book Reader reserves the following slot names for use by the operating system; do not create slots with these names.

Table 4-1 Reserved slot names

bookmarking
bookRef
browser
contentArea
cuPage
curRendering
data
destPage
edgeWidth
flags
kioskDest
layout
look
printing
related
scripts
type

Information Available From Reserved Slots

A value of `TRUE` in the `printing` slot indicates that the page is being printed. Similarly, A value of `TRUE` in the `bookmarking` slot indicates that the page is being rendered as a bookmark image.

A value of 0 in the curRendering slot indicates that the current page is MessagePad size. You can use this information to implement a faster page-turning algorithm using the following code:

```
if (curRendering = 0) then
   :TurnToPage(destPage);
else
   :TurnToPage(:FindPageByContent(dest, 0, NIL));
```

This code uses a preestablished value as a page number if the book is being rendered on a MessagePad-sized screen; otherwise it searches for the content item to establish the page number.

The copyProtection Slot

To copy protect a content item or page in a book, add a script slot named copyProtection to that item or page. To copy protect the entire book, define this slot at the beginning of the book source file, before any content items are defined. The copyProtection slot accepts the values described in Table 4-2.

Table 4-2 copyProtection constants

Constant	Value	Description
cpNoCopies	1	The view cannot be copied.

Table 4-2 copyProtection constants

Constant	Value	Description
`cpReadOnlyCopies`	2	The view can be copied, but the copy cannot be modified.
`cpOriginalOnlyCopies`	4	The original view can be copied, but copies of it cannot be copied. When a copy is made, the copy's `copyProtection` slot is changed to 1 (`cpNoCopies`) to prevent further copying.
`cpNewtonOnlyCopies`	8	The view can be copied, but on one Newton device only. Copies cannot be exported to a different Newton device.

The following example creates a `copyProtection` slot that allows no copies to be made:

```
.script slot copyProtection
cpNoCopies
.endscript
```

Marking Content Items

You can use the `.mark` command to make a content item's attributes (slots) available for use by a subsequent content item specified by the `.usemark` command. These commands allow the currently displayed content item to refer to another content item's slots without having to conduct a search for them. A new content item begins with any of the commands `.story`, `.picture`, `.subject`, `.form`, or `.chapter`.

For example, if you wanted to follow each content item in a guidebook or catalog with a summary view that graphically depicts certain data, you could store the data in slots associated with the content item and update the display in the summary view according to the values of the slots. The following example uses the `.mark` and `.usemark` commands to make slots in the content item `Joe’s Pizza Parlor` available to the user prototype `myRestaurantView`:

```
.# The name of the restaurant is centered on the page
.story layout=centered
Joe’s Pizza Parlor
.# associate these slots with the content item
.attribute cash: true, credit: true, beer: nil
.# make this content item’s slots available
.# to the usemarked item
.mark

.# Apply a different layout to the description
.# of the restaurant
.story layout=flushleft
This place has great pizza. Please eat here and leave a
big tip for the waitstaff.

.# create the view defined in myRestaurantProto
.form height=32 width=168
layout_myRestaurantView
.endform

.# The usemark command specifies that this view is to
.# use the slots in the content item having the .mark
.# command. The slots in the first story are referenced
.# even though another story appears between the marked
.# story and the usemarked view.
.# Notice also that the .usemark command applies to the
.# current view definition, regardless of its placement
.# before, within or after the view definition.
.usemark
```

Dereferencing Slots in a Marked Content Item

When you annotate a content item with the `.usemark` command, Book Maker adds a slot named `related` to its description in the book definition frame. The `related` slot references the content item designated by the `.mark` command. The content item designated by the `.usemark` command can then use its `related` slot to reference slots in the marked item.

The following example uses the `related` slot to access the slots in the marked content item:

```
.story
Annie-politan Pizza Parlor
102743 Fourth Street
Annapolis, MD.

.# This command defines a frame in the slot myFrame
.attribute myFrame: {cash: true, credit: true, pizza:
true, beer: true, text: "Best crab pizza in Annapolis!"}

.# make these slots available to the usemarked item
.mark

.# set the text slot in the viewSetupDoneScript
.form height=60 width=170
protoStaticText
.endform

.# The current content item is the view created
.# from myRestaurantView so this is
.# its viewSetupDoneScript method

.script viewSetupDoneScript
text := related.myFrame.text;
.endscript
.usemark
```

When the static text view in this example is created, its text slot has the value `nil`. The static text view's `viewSetupDoneScript` method stores a text string in this slot. The string is obtained by using the `related` slot to reference the `text` slot in the marked content item.

Using an Index to Obtain References to Content Items

The `.mark` command is especially useful when the `.mark`ed and `.usemark`ed items appear on the same or on nearby pages, and there is a one-to-one correspondence between them. However, you cannot mark a number of content items, and have a separate content item contain a reference to all of these, since only the last one will be remembered.

In this situation, it makes more sense to use the `.index` command. When a content item contains the `.index` *entry* command, a reference to it is stored in an array called `alphaIndex` as a slot in the `book.data` frame. The reference to this content item will be stored alphabetically, by the *entry* argument, in the `alphaIndex` array.

In the following example, `alphaIndex[0]` is a reference to the second–the aardvark–story, and `alphaIndex[1]` is a reference to the first story:

```
.story
.index zebra
Write a story about zebras here.

.story
.index aardvark
And one about aardvarks here.
```

To dereference the elements of `alphaIndex` at run time you need a reference to the `book.data` frame in which this array is stored. See "Setting and Getting Book Data" on page 4-12, for information on how to do this.

In addition, another 26-element array is created in the `book.data` frame, called `subIndex`. Each element in `subIndex` corresponds to a letter of the alphabet; i.e. subIndex[0] is “a,” and subIndex[25] is “z.” If no *entry* begins with a particular letter, say “m,” then m’s value will be the same as the n’s. If no *entry* begin with n-z, then m will have l’s value.

The creation of the subIndex array can be suppressed by including the .option noSubIndex command anywhere in the book; see “Option” beginning on page A-31 for more information. Furthermore, if the optional noSubIndex argument is used in the .index command, the subIndex array is generated, but this particular content item is not represented in the array. See “Index” on page A-30 for a description of the syntax.

You can post-process a book, in the postamble, to pull out the information desired from the index, and then set the alphaIndex slot to nil so that it doesn’t take up space in the built package. The following code adds a slot to the book.data frame containing a reference to the first element of alphaIndex, and then frees up that space:

```
.postamble
book.data.myFavoriteContent := book.data.alphaIndex[0];
book.data.alphaIndex := nil;
.endpostamble
```

Note

Currently, indices are provided purely for the developer’s use, but future versions of Book Reader may make indices visible to the end-user as a navigational aid. When indices do become visible to the end-user, they will only be displayed if a special slot has been set. Therefore, while you do not have to worry about users seeing your indices, you may want to keep this in mind, to make the transition to user-visible indices easier. ▲

Creating Multiple Indices

The .index command takes an optional !*indexName* argument, which is used to create a separate index in the book.data frame, called *indexName*Index. For example, .index !mammals rabbit creates an index called `mammalsIndex` containing an array of references to all the content items with the `.index !mammals` *entry* command. References to these content items appear only in `mammalsIndex`; they are not repeated in `alphaIndex`.

In addition, a separate array called *indexName*`SubIndex` will be created. The *indexName*`SubIndex` array is also a 26-element array, and has the same properties as `alphaIndex`'s `subIndex`.

There are a number of ways to control whether an *indexName*`SubIndex` array is created. If the book contains the `.option noSubIndex` command, then no subindices will be created in the entire book. If the `.index` @*indexName entry* syntax is used instead of the `.index !`*indexName entry* syntax, then that particular index will not have a matching subindex. In addition, particular entries can be excluded from its index's subindex by using the optional `noSubIndex` argument to the `.index` command. See "Index" on page A-30 for a description of the syntax.

Storing Page Numbers in a Content Item

The `.option firstPage` command adds a slot called `firstPage` to each visible content item; i.e., each content item that does not contain a `NoPage` or `BrowserOnly` flag. This slot contains the page number that the content item appears on. In the future, when different-sized screens are available, the `firstPage` slot will contain an array of page numbers. At that time the `curRendering` slot will provide the information needed to select the proper page number from this array.

For more selective control over which content items have a `firstPage` slot added to them, the `.option indexedPage` command is provided. This adds a `firstPage` slot to those visible content items that have an `.index` command attached. The `.option indexedPage` command has the same effect as the `.option firstPage` command, except that some, and not all, of the content items have a `firstPage` slot added to them.

Creating Story Cards

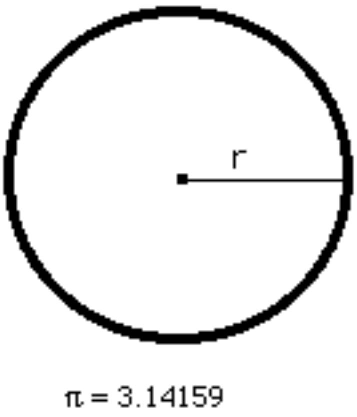
Story cards are views that can appear anywhere on the Newton screen, and are dismissed via a close button they provide. A story card is a convenient way to pop up a small amount of information. In this way you can create a hypertext environment where tapping part of a page provides additional information about what was tapped. This could be a short description, a picture, or any view created from a system prototype or a view template.

A story card is created by calling the `:ShowStoryCard` method, described on page A-49. This method is passed a `viewBounds` frame, and the name and value of a slot in a content item. Any content item can be imaged as a story card. However, as a practical limitation, use only content items that fit within the `viewBounds` frame specified, since story cards are not scrollable. Also, only one story card can be displayed at any time.

For example, the following content item displays the formula for the area of a circle. When the user taps this formula a story card is displayed with a diagram of a circle.

```
.story
Circle: A = π r^2
.script //it's a ButtonClickScript by default
:showStoryCard ('card,"circle",
        {left:40 ,right:200 ,top:60, bottom: 240});
.endscript

.picture NoPage
.#NoPage flag so that it will not be imaged normally
.attribute card : "circle"
```



Creating Dynamic Browsers

You can use the `AddBrowser` function to add your own browsers to the Book Reader overview dynamically. A typical use might be to display the results of a custom search—the user could keep the browser open and tap items in the browser to navigate between hits.

To add a browser frame use `AddBrowser` as in the following example:

```
index := :AddBrowser(browser);
```

The value returned is an index that can be used to show your browser, as in the next example.

```
:OpenBrowser(index);
```

Note
On the MessagePad, there's no way to remove a browser. ◆

The structure of a list browser frame looks like the following example:

```
      browser := {
         name: "Cheap Restaurants", // Name of browser
         list: […]// Topics
      };
```

Entries in the list array are in the form:

```
      {item: content, level: 2}
or
      {item: content, level: 2, name: "Important Stuff"}
```

The `content` in the `item` slot is a content item within the book (in other words, a structure returned by the `FindContentByValue` method). The `level` slot can be omitted; it defaults to 1. A `name` slot is the topic's text in the outline. If this is omitted, the text of `content` is displayed as the list item in the browser (entries for nontext content items must have a `name` slot).

Adding Intelligent Assistant Templates to Books

Digital books can use the Intelligent Assistant (IA) just as an application can. To provide IA support for your book you need to write a task template, and surround it with the `.assist` and `.endassist` commands.

The *Newton Programmer's Guide* provides all the information necessary to write the task template, and information about the IA in general. Note though that the process involved in providing IA support for an application is more extensive than that for a book. You do not need to take any steps other than the writing of the task template within the scope of the `.assist` and `.endassist` commands.

Application Help

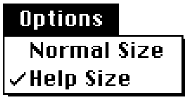
Book Maker allows you to create application help by processing the book source file with the Help Size option checked. The resulting help book file is compiled either with your application or as an independent package, which uses the system-supplied help browser to display help. This chapter describes how create **help books** with Book Maker, and how to display your help book from your application.

The material in this chapter is of a significantly more technical nature than the rest of this book; it assumes intimate knowledge of the NewtonScript language, the Newton object system and the creation of Newton application programs. For further information on these topics, see the *NewtonScript Reference*, *Newton Programmer's Guide*, and *Newton Toolkit User's Guide*.

Adding Help to Your Application

A help book is the file that Book Maker produces when it processes a book source file with the Help Size option checked, as shown in Figure 5-1. A help book is accessed through the Help option of the Information button on the status bar of an application. It can be built and downloaded as an independent package, or compiled with an application as part of a single package.

Figure 5-1 The Help Size option in Newton Book Maker



Writing Help Books

The process of writing a help book is essentially the same as that for writing a browser. In fact, a help book is nothing more than a browser plus the required `.title` and `.isbn` commands. For a detailed discussion of browsers see the section "Creating a Browser Pane" beginning on page 3-19.

There are, however, a few things to keep in mind when writing help books that do not apply to all browsers

- Keep help items short, since the help browser doesn't scroll and probably never will. If your help item exceeds the length of the browser pane, it will be truncated.Good human interface dictates that it should be impossible for the user to get lost in the help book. Keep things simple; provide simple, specific, task-oriented instruction.
- Answer a specific question in your overview item, the line displayed by the `.subject` command. An overview, *Topic*, should answer "How Do I *Topic*?" For example, *Topic* could be something along the lines of "Use the Shopping List Application," and the question would then be "How Do I Use The Shopping List Application?" Under this topic could be subtopics such as "Adding Items To The Shopping List" and "Checking Off Purchased Items."

A list of books that discuss the creation of online help manuals is provided in Appendix C, "Books on Online Help."

Building Stand-Alone Help Books

No special steps need to be taken to build a help book as an independent package. The process is identical to building a normal Book Reader book, with the addittional step of setting the Help Size option in Book Maker. When the book package is downloaded onto the Newton device, it is automatically placed in the Help folder of the Extras Drawer. Stand-alone help books give users the option of not installing application help, or removing it after they have read it. This also helps to minimize the size of the application package.

For instructions on using Newton Book Maker to process book source files and get book definition files, see the section "Processing the Book Source File" beginning on page 2-11. For information on processing a book definition file to get a package see "Building a Book Package With NTK" beginning on page 2-15.

How to Add Help to Your Application

The steps required to add help to your application differ depending on whether the help book is being built as its own package, or as part of the application's package.

In either case, your application needs to have a `protoInfoButton` (or `newtInfoButton` for NewtApps) that opens the help book in its `doInfoHelp` script. The `doInfoHelp` script should call either `openHelpBook(`*isbnStr*`)`, or `openHelpBookTo(`*isbnStr*, *topic*`)` to display your help book. The `protoInfoButton` and `newtInfoButton` templates are described in the *Newton Programmer's Guide*. The `OpenHelpBook` and `OpenHelpBookTo` functions are described in Appendix A, "The Book Maker Language."

You may want to call `:WhereIsBook` before calling `OpenHelpBook` to test for the existence of the help book. Since this is not a global function, but a method of `copperfield`, the parent of all books on the Newton device, you need to send the `:WhereIsBook` message to this frame. The following code does this properly:

```
if getRoot().copperfield:WhereIsBook( myISBNstring )
   then OpenHelpBook( myISBNString );
```

`WhereIsBook` is documented in the section “NewtonScript Methods” of Appendix A, “The Book Maker Language.”

This is all that needs to be done for an application to open a help book that was compiled as an independent package.

Adding a Help Book to Your Application’s Package

A help book may be compiled as part of an application project. To do this choose Add File from NTK’s Project menu, and select the book definition file. When a help book is built in this manner, it becomes a permanent part of the application, and does not appear in the Help folder in the Extras Drawer.

You also need to register the help book with the system, so that it can be found for the call to `OpenHelpBook`. A help book can be registered with the system by calling `RegisterBookRef(`*isbnStr*`, {book:` *book*`})`. This can be done in the application base view’s `viewSetUpDoneScript`.

It is also good manners to unregister your book when it is no longer needed. This is done with a call to `UnRegisterBookRef (`*isbnStr*`)`. A good place for this function call is in the project’s `removeScript`.

`RegisterBookRef` and `UnRegisterBookRef` are documented in the section “Book Reader Messages” of Appendix A, “The Book Maker Language.” For information about the `viewSetUpDoneScript` and the `removeScript` see the *Newton Programmers Guide*.

Pictures in Help Books in an Application Project

The Book Maker output file contains the help book frame that Book Maker generates from your book source file. If the source file contains pictures, Book Maker stores them as `'PICT'` resources in the resource fork of the output file. When a book is processed in a project with the Output option of “Book,” NTK knows to look in the book definition file for resources. But when a book is processed as part of an application project, you must ensure that the resources are opened.

You can do this with a call to the NTK global function `OpenResFileX`. The `OpenResFileX` function accepts as its argument the full pathname to the resource file to be opened, and returns a number to be used for subsequent references to the open file.

The code example below opens the file `myResource`, which resides in the folder `myFolder` on the disk `myDisk`, and stores the value returned by `OpenResFileX` in the `refNum` variable. This code can be inserted in the help book's preamble:

```
refNum := OpenResFileX("My Disk:My Folder:myResource");
```

When you're finished using the resource file, close it by calling the global function `CloseResFileX`, passing as its argument the variable `refNum`. This can be done in the help book's postamble.

The Book Maker Language

This appendix describes all of the commands available in the Book Maker command language. Optional commands are marked with the designation "(optional)."

The commands are organized alphabetically under the categories Document Commands, Content Commands, Browser Commands, Page Layout Commands, Miscellaneous Commands, Flags, NewtonScript Methods, Book Reader Messages, and NewtonScript Global Functions.

Overview of the Book Maker Language

This section provides an overview of the different categories of Book Maker commands, so that you'll understand how you can combine the different kinds of commands to produce various effects. Individual elements of the Book Maker language are described in subsequent sections of this Appendix.

Types of Book Maker Commands

All commands in the Newton Book Maker language fall into one of the following five categories:

- **Document** commands define the overall characteristics of the document, such as author, copyright information, assist functions, and so on. The `.title` and `.isbn` commands described in Chapter 2, "Getting Started With Newton Book Maker," are typical document commands.
- **Content** commands define the various kinds of elements that the document contains, such as text, graphics, NewtonScript view templates, kiosks, and so on. The `.story` and `.picture` commands described in Chapter 2, "Getting Started With Newton Book Maker," are typical

content commands. Text and graphics tagged with these commands are called **content items.**

- **Browser** commands define outline-like panes useful for navigating content in the Newton Book Reader. Browser commands are explained in detail in the "Creating a Browser Pane" section of Chapter 3, "Using the Book Maker Language."
- **Page Layout** commands define general layout characteristics for the document, such as the number and size of columns or the placement of graphics. Page layout commands can be used to create named formats called **layouts**, and are also used to attach scripts to pages. Several sections in Chapter 3, "Using the Book Maker Language," are devoted to explaining the various page layout commands.
- **Miscellaneous** commands make the Newton Book Maker language easier to use by allowing you to create indexes, break large book source files into sets of smaller ones, and embed in your source file comments that are not displayed on the Newton screen.
- **Flags** allow you to enable various options to Book Maker commands.
- **NewtonScript Methods** allow you to manipulate the contents of Book Reader books from NewtonScript.
- **Book Reader Messages** are messages Book Reader sends to a book when certain conditions arise. You can provide methods to handle these messages, so as to take action appropriate to these conditions.
- **NewtonScript Global Functions** are functions that an application can call that affect digital books.

If You Do Not Know NewtonScript

The last three sections in this appendix—NewtonScript Methods, Book Reader Messages, and NewtonScript Global Functions—provide information that can only be used by a NewtonScript programmer.

The other sections describe the Book Maker command language and should be read by non-programmers, although some of the commands listed are provided for NewtonScript programmers. The section "Summary of

Commands, Functions, and Methods" beginning on page A-59 flags commands that require knowledge of NewtonScript to use.

Syntax of Book Maker Commands

Book Maker commands always appear at the beginning of a line in the book source file. Each command applies to everything that appears in the book source file until the next command appears. The commands themselves do not appear on the Newton screen.

The following points summarize the syntax of the Book Maker language:

- Any line beginning with a period (.) is treated as a Newton Book Maker command.
- The entire Book Maker command language is case insensitive.
- Comments can be added to a command line by preceding the comment with a pound (#) sign.
- Names that have spaces in them must be enclosed in quotation marks; for example, `"Futura Heavy"` or `'Futura Heavy'`.

Note
In this book, parameters shown in square brackets (`[]`) are optional. ◆

Structure of a Book Maker Source File

Document commands are generally placed at the beginning of the book source file, followed by any definitions of page layouts or styles. The bulk of the document is comprised of the text and graphics to be displayed on the screen. This content has interspersed in it the Book Maker commands that indicate the document's structure.

Very few commands are actually required by Book Maker; the majority of commands in a typical book source file are optional commands used to exert additional control over page layout or provide additional functionality to the user.

For an example of the structure of a typical Book Maker source file, see the example book source files included with Book Maker.

Document Commands

Document commands define the overall characteristics of the document, such as its author, copyright information, assist functions, and so on. This section describes each of the document commands in the Book Maker command language.

Assist

```
.assist
```

Specifies that NewtonScript code following this command defines templates used by the Intelligent Assistant. (optional)

For example:

```
.assist
{isa: {},
isbn: book.isbn,
primary_act: {isa: {value: "Reserve action"},
            lexicon: [["reserve", "hold",
                           "reservation"]]},
signature: [{isa: {value: "action"},
            lexicon: [["reserve", "hold",
                        "reservation"]]},
            {isa: {"reserve where"},
            lexicon: [["hotel", "room", "motel",
                        "bed"]]],
preconditions: ['action, 'place],
postparse: begin
   app := GetRoot().Copperfield;
   app:OpenBook(isbn);     // Open the book
   app:TurnToContent('name, "hotels");
   end,
score: NIL};
.endassist
```

See also the description of the `.endassist` command on page A-7.

Author

`.author` *author*

Defines the book's authors. (optional)

This information is displayed in the "About" slip accessed through the Information button in Book Reader version 2.0.

author A string that contains the names of the book's authors.

For example:

```
.author Charles Dickens
```

Blurb

`.blurb`
blurbText

The blurb contains text about the book that might be used for sales oriented information. The text of the blurb begins on the line following the `.blurb` command and continues until the next dot command. (optional)

This information is currently available only from NewtonScript; however, future versions of Book Reader may use it or provide end users with access to it.

blurbText A string that contains a short description of the book.

For example:

```
.blurb
This is a great book that tells all about the life and
times of that great author Charles Dickens. Please buy
me!
```

Copyright

`.copyright` *copyright*

Defines the book's copyright notice. Only one such command may exist in the book source file. (optional)

This information is displayed in the "About" slip, accessed through the Information button in Book Reader version 2.0.

copyright	A string that contains the book's copyright information.

For example:

```
.copyright © 1993 Apple Computer, Inc.
```

Note
To obtain the © character, type a lowercase "g"
while pressing the Option key. ◆

Date

`.date` *date*

Defines the publication date of the book. Only one such command may exist in the book source file. (optional)

This information is displayed in the "About" slip, accessed through the Information button in Book Reader version 2.0.

date	A string that contains the book's date of publication

For example:

```
.date July 7, 1992
```

Endassist

```
.endassist
```

Terminates the definition of a NewtonScript block defined by the `.assist` command. Required for `.assist` commands; see the discussion of the `.assist` command on page A-4 for more information.

Endpostamble

```
.endpostamble
```

Terminates the definition of a NewtonScript block defined by the `.postamble` command. Required for `.postamble` commands; see the discussion of the `.postamble` command on page A-9 for more information.

Endpreamble

```
.endpreamble
```

Terminates the definition of a NewtonScript block defined by the `.preamble` command. Required for `.preamble` commands; see the discussion of the `.postamble` command on page A-9 for more information.

Expires

`.expires` *date*

Defines the date on which the information contained in the book is no longer valid. Only one such command may exist in the book source file. (optional)

This information is displayed in the "About" slip accessed through the Information button in Book Reader version 2.0.

date A string that contains the book's expiration date.

For example:

```
.expires July 7, 1993
```

Key

`.key` *keyword0* `[,` *keyword1*`,` *… keywordN-1*`,` *keywordN*`]`

Defines a list of keywords that describe the book to readers. These keywords can be searched on and are remembered by Book Reader even when the book (card) has been removed from the Newton device. Only one such command may exist in the book source file. (optional)

keyword	A string that is a keyword associated with this book.

For example:

```
.key biography author dickens England writer novelist
```

This information is currently available only from NewtonScript; however, future versions of Book Reader may use it or provide end users with access to it.

ISBN

`.isbn` *isbn*

Defines a unique identifier for the book which may be an actual ISBN number. This identifier must be 14 or fewer characters. Only one such command may exist in the book source file. (required)

This information is currently used by Newton Book Reader to identify book packages in the Extras Drawer. This information is available only from NewtonScript; however future versions of Book Reader may provide end users with access to it.

isbn	A string that contains an identifier unique to this book

For example:

```
.isbn 0-316-08275-9
```

ISBN is an acronym for International Standard Book Number. ISBN numbers are unique numbers used by publishers and others in the book trade to identify books.

If you eventually publish your book and distribute it to a large audience, you may wish to obtain a real ISBN number to identify your book. ISBN numbers may be obtained from the R.R. Bowker company for a nominal fee. You can contact them at the following address and phone number:

ISBN Agency
R.R. Bowker
121 Chanlon Road
New Providence, New Jersey 07974

Phone: (908) 665-6770
FAX: (908) 665-2895

Postamble

```
.postamble
```

Any NewtonScript code between the `.postamble` command and the matching `.endpostamble` command is output directly into the book definition file, following all the code created by Book Maker. This is a handy place to do any clean-up that might be needed. For example, if a large object is needed at build time, but not at run time, this object can be set to `nil` in the postamble. (optional)

See also the descriptions of the `.endpostamble` and `.preamble` commands.

Preamble

```
.preamble
```

Any NewtonScript code between the `.preamble` command and the matching `.endpreamble` command is output directly into the book definition file preceding all definitions of content items. (optional)

One use for this command is to store global data in Book Data, as in the following code:

```
.preamble
   book.data.stuff := { ... };
.endpreamble
```

The preamble is a convenient place to define methods that will be used in various parts of the book, which has the advantage of minimizing memory consumption. However, if you plan to use these methods in various books, you might find it more convenient to define the methods in a text file that can be included in your NTK project.

See also the descriptions of the `.endpreamble` and `.postamble` commands in this section.

Publisher

`.publisher` *publisherInfo*

Defines the book's publisher. Only one such command may exist in the book source file. (optional)

This information is displayed in the "About" slip, accessed through the Information button in Book Reader version 2.0.

publisherInfo The string describing the book's publisher.

For example:

```
.publisher Apple Computer, Inc.
```

ShortTitle

`.shortTitle` *shortTitle*

Defines the book's short title, which is used as its name in the Extras drawer. Only one such command may exist in the book source file. If no short title is defined, the book's full title is used in the Extras Drawer. (optional)

shortTitle	The string that appears under the book's icon in the Extras Drawer.

For example:

```
.shorttitle Dickens
```

Title

`.title` *titleText*

Defines the book's title. Only one such command may exist in the book source file. (required)

titleText	The string that is the book's title. This string is used as the book's title in the Extras Drawer if no short title is supplied. This string also appears at the top of every page in the book unless it is replaced by a header or picture header. The title string at the top of book pages can be suppressed in layouts that include the `noTitle` flag.

```
.title All About Dickens
```

Content Commands

Content commands define the various content items that the book contains. Content items are text, graphics, views, kiosks, and so on.

Attribute

`.attribute` *name1*`:=`*value1* `[ ,`*name2*`:=`*value2*`, ...,`*nameN*`:=`*valueN* `]`

Allows additional information to be defined for a content item by creating a slot associated with the content item. The slot can be used by the view system (and thus, NewtonScript) or by the intelligent assistant, via a special `item` slot. This is explained in more detail in the section "Adding Slots to Content Items" beginning on page 4-9.

name The name of the slot this command creates.

value The value to store in the slot created by this command.

For example, a content item describing a place might have associated with it slots containing its telephone number and address.

```
.attribute phone: "602-555-1212"
.attribute type: 4, color: "blue"
```

Chapter

`.chapter` *flags* `[name=`*nameString*`] [goto=`*item*`] [browser=`*browserName*`] [layout=`*layoutName*`]`
subjectText

Creates a subject on level 1; a synonym for a `.subject 1` command. The parameters to this command are the same as those used for the `.subject` command; for more information, see the discussion of the `.subject` command on page A-23.

```
.chapter StartsPage name=c1 layout=Simple
Chapter One:The Beginning
```

Note

Except for the subject text, which must appear on its own line, the entire `.chapter` command must appear on a single line in your book source file. ◆

Endform

`.endForm`

Terminates a NewtonScript block defined by the `.form` command. Required for `.form` commands; see the discussion of the `.form` command on page A-13 for more information.

Endscript

`.endscript`

Terminates a NewtonScript block defined by the `.script` command. Required for scripts. For more information, see the discussion of the `.script` command on page A-19.

Endkiosk

`.endkiosk`

Terminates the definition of a kiosk defined by the `.kiosk` command. Required for kiosks. For more information, see the discussion of the `.kiosk` command on page A-16.

Form

`.form  height=`*h* `width=`*w* `[name=`*nameString*`] [browser=`*browserName*`]` *flags* `[`*protoName*`|layout_`*nameOfLayout*`]`

Reserves the specified amount of space for a view on the page and, optionally, creates the view from a specified system prototype, view template, layout, or user template. The `.form` command must be followed by a corresponding `.endform` command; everything between `.form` and `.endform` defines the view created by this command. (optional)

Note

Except for the layout or prototype, which must appear on its own line, the entire `.form` command must appear on a single line in your book source file. ◆

`height` Required. Specifies in pixels the height of the blank space reserved for imaging the view.

`width` Required. Specifies in pixels the width of the blank space reserved for imaging the view.

`name=` Optional. Specifies a *nameString* that other entities in the book can use to reference this form. For example, if the `.form` command defines a `name=FredForm` parameter, other content items can display the form by specifying `goTo=FredForm` as one of their parameters.

flags Optional content flags associated with this view.

`browser=` Optional. Specifies a named browser in which the view this command creates is placed. For example, if the `.form` command defines a `browser=Cities` parameter, this view is placed in the browser named `Cities`.

protoName Optional. The name of the system prototype from which to create the view.

`layout_`*nameOfLayout*

Optional. The user prototype, view template, layout, or user template from which to create the view. It is specified by prefixing its name with the `layout_` keyword. For example, to specify the template `mySpecialUserProto`, the syntax would be `layout_mySpecialUserProto`.

You can include a `protoTextButton` view in your book by using the `.form` command as in the following example:

```
.form height=35 width=60
protoTextButton
.endform
```

NewtonScript commands that define the view may follow the `.form` command. You can use this technique to initialize slots in the view when it is created, as in the following example:

```
.form height=30 width =100
{_proto: layout_nameOfLayout ,
slotName: value}
.endform
```

Instead of using a system proto or some kind of view template, you can create the whole view procedurally, as in the following example:

```
.form name=rect height=100 width=50
   {
   viewClass: clView,
   viewFlags:vVisible,
   viewDrawScript:func()
      begin
         DrawRect(3, 3, 13, 13);
      end}
.endform
```

Indent

`.indent` *left right*

Indents the left margin of the current content item by *left* points and the right margin of the current content item by *right* points. (A point is a typographer's unit of measure; there are 72 points in an inch.)

left	The amount of space, in points, by which to indent the left margin of the content item.
right	The amount of space, in points, by which to indent the right margin of the content item.

Place this command after the content command it affects, but before the content (text or picture) associated with that command. The following example indents the story text by 30 points on the left and 20 points on the right:

```
.story
.indent 30 20
This is my story text.
```

Kiosk

`.kiosk name=`*nameString* `[layout=`*layoutName*`]`

Defines a named kiosk page, which is an optional page used for navigating book content. For a more detailed description of a kiosk page, see the section "Creating a Kiosk," in Chapter 3, "Using the Book Maker Language."

Between the `.kiosk` command and its corresponding `.endkiosk` command, any content commands that use the `goto` flag become buttons in the kiosk page. Any content commands without the `goto` flag are simply labels or pictures. Note that all content items can be referred to by the name of the kiosk.

`name=`	Required. Specifies a *nameString* that other entities in the book can use to reference this kiosk. For example, if the `.kiosk` command defines a `name=Cats`

	parameter, other content items can display the kiosk by specifying `goTo=Cats` as one of their parameters.
`layout=`	Optional. Specifies that the *layoutName* layout applies to this kiosk page. When applying layouts by name, the `layout=` parameter must be the last parameter on the command line. For more information on layout and styles options, see the section "Page Layout Commands," later in this Appendix.

```
.kiosk name=aKiosk layout=myKiosk
.story
Tap on a subject…
.story goto=subj1
Car Rental Companies
.story goto=subj2
Airlines
.endkiosk
```

Note

It is recommended that you create a layout to use with kiosk pages. In this layout be sure to include the `Kiosk` flag. This allows Book Reader to automatically provide a button that returns the user to the nearest kiosk. ◆

Mark

```
.mark
```

Saves a reference to the current content item. This allows a subsequent content item that has a `.usemark` command to refer to the marked item's slots.

If another `.mark` command is encountered before a `.usemark` command, then the reference to the first marked content item is lost. That is, a `.usemark` command is matched with the nearest preceding `.mark` command. See also the description of the `.usemark` command later in this section.

Picture

```
.picture ["path"] [flags] [name=nameString] [goto=item] [browser=brsNm]
[height=h] [width=w] [layout=layoutName]
```

Specifies that the `'PICT'` following this command be used as a content item. A full pathname to an external `'PICT'` file may also be specified instead of pasting the `'PICT'` into the book source file.

Note

The entire `.picture` command must appear on a single line in your book source file. In order to print the command in a font large enough to read comfortably, this book continues the command on a second line. ◆

When Newton Book Maker lays out pages, it tries to give the maximum amount of space possible to pictures so users are not forced to scroll the picture. Often this means that pictures are placed on the next page instead of staying with a related subject or story. You can avoid this problem by using the `KeepWith` flag to keep the picture with the previous content or by using the `width` and `height` flags to specify a smaller size for the picture. Note that the height and width flags do not scale the picture; they specify the boundaries of the displayable portion of the picture. Newton Book Reader automatically displays controls with which the user can scroll the picture if its size exceeds the specified boundaries.

flags Optional content flags associated with this content item

`name=` Optional. Specifies a *nameString* that other entities in the book can use to reference this picture; for example, if the `.picture` command defines a `name=Fred` parameter, other content items can display the picture by specifying `goTo=Fred` as one of their parameters.

`goTo=` Specifies the content item to be displayed when the user taps this subject in the browser. For example, if the `.subject` command defined a `goTo=Cats` parameter, the named content item `Cats` is displayed when the user taps the *subjectText* string in the browser.

`browser=` Optional. Specifies a named browser *brsNm* in which the content item this command creates is placed; for example, if the `.story` command defined a `browser=Cities` parameter, the story text would be placed in the browser named `Cities`.

`height` Optional. Specifies in pixels the height of the space reserved for imaging the picture. If the picture is larger than the specified size, scroller controls are displayed automatically.

`width` Optional. Specifies in pixels the width of the space reserved for imaging the picture. If the picture is larger than the specified size, scroller controls are displayed automatically.

`layout=` Optional. Specifies that the *layoutName* layout is to be applied to this content item. When applying layouts by name, the `layout=` parameter must be the last parameter on the command line. For more information on layout and styles options, see the section “Page Layout Commands,” later in this Appendix.

For example:

```
.picture "myDisk:proj:portrait" Centered
```

Script

`.script [slot]` *event*

Attaches a NewtonScript method to the current content item. An optional path to a file containing the script may be specified. All lines between the `.script` command and its corresponding `.endscript` command must be NewtonScript code defining the body of the script.

Scripts defined at the beginning of the document (before any content or layout commands) are available throughout the book and are referred to as book scripts. Scripts defined after a layout command are attached to the layout and are referred to as page scripts; page scripts are available on any page that uses that layout.

event In the absence of the `slot` keyword, this parameter specifies the event that invokes the script. (This is the most common syntax for using this command.)

The default value of this parameter is `buttonClickScript`, but other examples might include `viewDrawScript`, `viewClickScript`, or `viewSetupDoneScript`.

`slot` Optional. In the presence of this keyword, the *event* parameter defines the name of a slot to be associated with the currently defined content item.

The following example attaches a script to the current content item that plays the sound `ROM_Click` when the user taps the content item:

```
.#  by default, it’s a buttonClickScript
.script
   playSound(ROM_click);
   inherited:buttonClickScript
.endscript
```

The following example attaches a slot named `color` that stores the `"blue`" value to the view that images the current content item:

```
.script slot color
   "blue"
.endscript
```

Note

Use script slots sparingly and with caution. Script slots are created in the view that images the content item with which they are associated; hence, each script slot uses a certain amount of space on the NewtonScript heap. ◆

The `slot` argument is required, however, if the function needs to take parameters. For example, Book Reader sends a `FormSearchScript` to a content item, passing in *searchStr* and *stringLen* parameters. The `FormSearchScript` must be written using the `slot` argument as in this example to use these parameters:

```
.script slot FormSearchScript
func (searchStr, stringLen)
begin
// code which highlights would go here
end
.endscript
```

Space

`.space` *n*

Leaves *n* points of space after the current content item. (A point is a typographer's measure; there are 72 points per inch.) If the content item is at the bottom of a page, no space is left at the top of the next page.

n	The amount of space to leave after the current content item. The value of *n* may range from 0-63 points.

Using the `.space` command instead of placing blank lines in the book source file results in the creation of slightly smaller book packages, ensures that blank lines are not placed at the tops of pages, and provides finer control over the amount of white space that is placed between content items.

Story

`.story [`*flags*`] [name=`*nameString*`] [goto=`*item*`] [browser=`*brsNm*`] [layout=`*layoutName*`]`

Defines a new content item within the currently defined subject (if any). The content item is by default one level deeper than the current subject.

Note
Except for the story text, which must appear after the `.story` command, the entire `.story` command must appear on a single line in your book source file. In order to print the command in a font large enough to read comfortably, this book continues the command on a second line. ◆

flags Optional content flags associated with this content item

`name=` Optional. Specifies a *nameString* that other entities in the book can use to reference this story text. For example, if the `.story` command defines a `name=myStory` parameter, other content items can display the story text by specifying `goTo=myStory` as one of their parameters.

`goTo=` Specifies the content item to display when the user taps this subject in the browser. For example, if the `.subject` command defined a `goTo=Cats` parameter, the named content item `Cats` is displayed when the user taps the *subjectText* string in the browser.

`browser=` Optional. Specifies a named browser *brsNm* in which the content item this command creates is placed; for example, if the `.story` command defined a `browser=Cities` parameter, the story text would be placed in the browser named `Cities`.

`layout=` Optional. Specifies that the *layoutName* layout is to be applied to this content item. When applying layouts by name, the `layout=` parameter must be the last parameter on the command line. For more information on layout and styles options, see the section “Page Layout Commands,” later in this Appendix.

For example:

```
.story MainColumn layout=TwoColumn
```

Subject

`.subject` [*level*] [*flags*] [*name*=*aStr*] [`goto=`*item*][`browser=`*brsNm*] [`layout=`*layoutName*] *subjectText*

Defines a new content item that appears in the book's table of contents browser at the specified level and appears in the body of the book as well. If no level is specified, the subject is created at level 1. The text or graphic between this command and the next command defining a content item in the book source file is the subject content item. To suppress the appearance of subject text in the body of the book, add the `browserOnly` flag.

Note

Except for the *subjectText* text, which must appear after the `.subject` command, the entire `.subject` command must appear on a single line in the book source file. ◆

level The level at which this subject appears in the browser. A value of 1 specifies that the subject text is a chapter, a value of 2 specifies a headline, 3 specifies a subhead, and so on.

flags Optional content flags associated with this content item.

`name=` Optional. Specifies a name string *aStr* that other entities in the book can use to reference this content item. For example, if the `.subject` command defines a `name=Cats` parameter, other content items can display this content item by specifying `goTo=Cats` as one of their parameters.

`goTo=` Specifies the content item to display when the user taps this subject in the browser. For example, if the `.subject` command defined a `goTo=Cats` parameter, the named content item `Cats` is displayed when the user taps the *subjectText* string in the browser.

`browser=` Optional. Specifies a named browser *brsNm* in which to place the content item this command creates. For example, if the `.subject` command defined a `browser=Cities` parameter, the subject would be placed in the browser named `Cities`.

`layout=` Optional. Specifies that the *layoutName* layout is to be applied to this content item. When applying a layout by name, the `layout=` parameter must be the last parameter on the command line. For more information on layout and styles options, see the section “Page Layout Commands,” later in this Appendix.

For example:

```
.subject 2 Reverse StartsPage layout=TwoColumn
This is the subject text
```

Usemark

`.usemark`

Adds a slot named `related` to the current content item. The value of this slot is a reference to the nearest preceding content item that has a `.mark` command. This allows the content item with the `.usemark` access to slots in the content item with the `.mark`, without having to conduct a search for it.

For example:

```
.story
.attribute boring : true
.mark
Something long and boring here.

.form height=60 width=170
   protoStaticText
.endform
.usemark
```

```
.script viewSetupDoneScript
      text := “Isn’t this” && if related.boring then
“boring.” else “exciting.”
.endscript
```

Browser Commands

Browser commands define a new browser to be displayed in the Newton Book Reader table of contents.

Browser

`.browser` *name type*

Creates a new browser pane having the specified name in the table of contents browser.

name — Specifies the text that appears in the browser menu; the name of this browser pane. This text is also used by content items that use the `browser=`*name* flag.

type — Specifies how the browser is defined; may have either of the values `list` or `form`. If the type is `list`, Book Maker builds the list of browser items from the content items that use the `browser=` flag. If the type is `form`, the definition of the browser pane view must follow the `.browser` command. This definition may not specify `viewBounds` to be larger than the size of the system-supplied browser pane.

For example:

```
.browser Hotels list
```

Page Layout Commands

Page layout commands define general formatting characteristics for the page, such as the number of columns and placement of graphics. Page layout commands are also used to attach scripts to text and graphics in a Newton digital book.

Layout

`.layout` *name width* `[`*flags*`]` `[`*layoutFlags*`]`

Creates a named page layout with the specified number and widths of columns. You may find it helpful to think of `.layout` commands as resembling the user-defined styles or formats commonly used by many word processors. Only one layout is allowed per page on the Newton screen.

name Required. Other entities in the book can use this string to apply this layout by name. For example, if the `.layout` command defines the name `myLayout` parameter, other content items can apply `myLayout` by including the parameter `layout=myLayout`. It is important to give each `.layout` a different name; repeating the names of `.layout` commands causes errors.

width Required. One or more numbers specifying the width of each column defined in this layout. The numbers specified as the value of this parameter must add up to 12, which is the number of vertical strips into which Book Maker divides the Newton screen. All of the strips are equal in width; the actual width of an individual strip depends on the screen size of the device that displays the book. The column widths in a layout are specified as a number of these strips; thus, the proportions of the layout are preserved regardless of the actual size of the screen on the display device.

flags Optional content flags that are applied to each content item using this layout. For example, the layout can include the `edges` flag to specify that each content item using this layout has a box drawn around its edges.

layoutFlags Optional layout flags to be applied, such as the `sidebar` and `main` flags used to designate particular column widths in the layout as being those of the sidebar and main columns respectively.

A typical layout command looks like the following example:

```
.layout twoColumn 6 6 edges
```

This command defines a page layout named `twoColumn`. The `twoColumn` layout defines two equal main columns that are each six strips wide. The `edges` flag draws a box around them. Here's another example:

```
.layout annotated 4 Sidebar 8
```

This command defines a page layout named "annotated" that has two columns. The left column is four strips wide, or one-third of the page, and is a sidebar. The right column is eight strips wide, or two-thirds the width of the page, and is the main column.

If a `.script` command follows a `.layout` command, the script is attached to the layout. In this manner a script can handle the drawing of a special background or click event not handled by the content item.

If a content command does not specify the layout to be used, the most recently defined one is applied.

When applying layouts by name, the `layout=` parameter must appear last on the command line.

Header

`.header`

Assigns to the current layout command a centered text header that is displayed at the top of every page that uses the layout. The font and style of the header are maintained separately from the rest of the layout.

Pictheader

`.pictHeader`

Adds to the current layout command a picture header that is displayed at the top of every page that uses the layout. Headers taller than 16 pixels spill into the content view. Because page content is drawn after the picture header, oversize picture headers can provide interesting page backgrounds.

Running

`.running` *type* `[pageBottom] [`*flags*`]`

Assigns to the previous layout command a content of type `picture`, `story` or `form` and automatically places that content on every page. The content appears at the top of the page (below the header if applicable) by default.

type — The kind of content item used as the running head; this parameter must have one of the values `picture`, `story`, or `form`. The content item is used the same way as an item defined by the corresponding content command (`.story`, `.picture`, `.form`).

`pageBottom` — This optional flag causes the content to appear at the bottom of the page.

flags — Optional content flags applied to the running header content item.

For example:

```
.layout full 12
.running form height=100 width=200
layout_CoolHead
```

Use of this command to create running headers results in a smaller book package than pasting the same data in the book source file and allows great flexibility for creating running titles in your book. However, it is recommended that you exercise discretion with regard to taking up large amounts of space on every page for static information.

Miscellaneous Commands

You can use these commands to create indicies, break large book source files into sets of smaller ones, and embed comments that are not displayed on the Newton screen in your source file.

Chain

`.chain "`*fullPathToFile*`"`

Placed at the end of a book source file to indicate that the remainder of the book source file is contained in another file. Use this command to split a large book source file into several smaller ones.

The book's `.title` and `.isbn` commands must appear only in the first file in the chain. All files in the chain must be of the same format; for example, you cannot chain a MacWrite file to a TeachText file. As Book Maker opens each file specified by the `.chain` command, it closes the previous file in the chain.

fullPathToFile The full pathname to the next file in the chain. You must enclose the pathname in quotation marks.

For example:

```
.chain "disk:folder:BookSourcePart2"
```

This command is useful for breaking a large book source file into more manageable pieces, perhaps to allow several people to work on it simultaneously. It can also be used to break apart an extremely large book file that Book Maker may be unable to process in its entirety. (Book Maker reads the entire book source file into RAM to process it. You can also use this command to reduce the RAM requirements necessary to process a large book.)

Comment Symbol

`.#` *commentText*

The pound sign (#) indicates that the text following it is a comment line that does not appear on the screen.

commentText The text that is the comment.

For example:

```
.# This is a comment. Book Maker ignores it.
```

Index

`.index [{!|@}`*indexName*`] [noSubIndex]` *entry*

Creates an alphabetical index and adds the specified *entry* to it. Currently, index data is available only from NewtonScript; however, future versions of Book Maker may use this data or make it available to the end user.

entry The word to add to the index.*indexName*The name used to identify a particular index when the book has multiple indicies.

Each index entry stores a reference to the content in which the `.index` command was used. When Book Maker processes the file, it outputs an array of content items in alphabetical order and stores it with the slot name `alphaIndex` in book data. It also outputs a second array in a slot named `subIndex,` which creates an array index into `alphaIndex` for each letter of the alphabet. You can use the `.option noSubIndex` command to suppress generation of subindices.

The `!`*indexName* argument creates multiple indicies for a single book. The value of the *indexName* parameter creates a NewtonScript variable that identifies a particular index. The index is stored with the slot name *indexName*`Index` in the book data frame. An *indexName*`SubIndex` is also created. You can use the `@`*indexName* syntax to suppress generation of subindices. A particular *indexName* can appear in a book in both the `.index !`*indexName* and `.index @`*indexName* formats. In this case the first time the *indexName* appears determines whether an *indexName*`SubIndex` is generated. If the `@`*indexName* syntax is the first use of *indexName* encountered, then the *indexName*`SubIndex` array is not generated.

Note

The *indexName*`Index` and *indexName*`SubIndex` form NewtonScript symbols; therefore, *indexName* may not contain any characters other than alphanumeric characters and underscores. Other characters cannot be used even if enclosed in vertical bars (`|`), since even though `'|weird%symbol!|` is a valid NewtonScript symbol, `'|weird%symbol!|Index` is not.

Option

`.option` *optionName*

Enables the specified option for the current book. The `.option` command should only be used once for each applicable option.

optionName Specifies the option to enable. This parameter can have the following values only:

`firstPage` Adds a slot named `firstPage` to each visible content item. The `firstPage` slot contains the number of the page on which the content item appears.

`indexedPage`
Adds a slot named `firstPage` to each visible content item that is also in an index created by the `.index` command. The `firstPage` slot contains the number

of the page on which the content item appears.

`noSubIndex`

Don’t create a subindex array for indicies created by the `.index` command.

For example:

```
.option noSubIndex
```

Note that `.option indexedPage` is a subset of `.option firstPage`. The former adds slots to a subset of the content items that the latter adds slots to. It is thus redundant to use both the `.option firstPage` and the `.option indexedPage` commands in the same book.

In the future, when different-sized screens are available, the `firstPage` slot will contain an array of page numbers. At that time the `curRendering` slot will provide the information needed to select the proper page number from this array.

Flags

The Book Maker language provides keywords that can be used to modify document, content, and layout commands. These keywords, called flags, can be used singly or can be combined to produce various effects. This section describes the flags available in the Book Maker command language.

Document Flags

A document flag affects the entire book, rather than individual content items or layouts. Table A-1 describes the document flags available in version 1.0 of the Newton Book Maker command language.

Table A-1 Document flags

Flag	Description
`NoReLayout`	Don't reformat if screen size changes

Layout Flags

A layout flag modifies the `.layout` command to which it is attached; hence, any content item using that layout is affected by the flag as well. Table A-2 describes the layout flags available in version 1.1 of the Newton Book Maker command language.

Table A-2 Layout flags

Flag	Description
`Kiosk`	This `.layout` command flag defines a kiosk page layout
`Main`*	Display in a main column (default).
`NoTitle`	Do not reserve space for a running title.
`Sidebar`*	Display in the sidebar column.

* Can also be used to modify content commands.

Note

Certain layout flags may be also be used to modify content items; see Table A-3 for details. ◆

Content flags

Most of the flags in the Book Maker language are used to modify the commands that define content items, such as `.story`.`picture`, `.subject`, and so on; thus, they are referred to as **content flags.**

Some content flags may also be used to modify layout commands; these flags are marked with an asterik (the asterik is not part of the flag). Because a layout applies to the entire page on which it is used, these flags affect the entire page when they are used to modify a layout.

Table A-3 Content flags

Flag	Description
`Main`*	Display in a main column (default).
`Sidebar`*	Display in the sidebar column.
`ToEdge`	Override current layout to spill content out of its column to the edge of the page.
`Centered`	Center within column.
`NeverBreak`	Avoid splitting text between pages.
`StartsPage`	Always place at the top of a new page.
`PageMiddle`	Always place in the middle of a new page.
`PageBottom`	Always place at the bottom of the page.
`AlignTop`	Align sidebar with top of previous content item.
`AlignBottom`	Align sidebar with bottom of previous content item.
`AlignCenter`	If content item is in a sidebar, align with center of previous content item, or else align with center of page.
`BrowserOnly`	Content appears only in browser pane; not drawn on the book page.
`BrowserAutoClose`	Close browser automatically whenever a selection is made, causing it to behave like the system-supplied overview.

Table A-3 Content flags (continued)

Flag	Description
`Overlay`	Overlay previous content item without wrapping.
`KeepWith`	Keep with the previous content item.
`NoExtend`	Suppress automatic extension of content block to the bottom of the page. Useful to avoid undesirable highlighting behavior in `viewClickScript` methods for kiosks that occupy an entire page but do not fill it with content items.
`NoPage`	Do not automatically display this content item on the page but keep its information available for display later. Useful for items like story cards, which are displayed only on command.
`NoScroller`	Suppress automatic display of scroller controls for graphics that are larger than the column.
`NoSearch`	Do not search this content text. Used to improve speed of text searches.

* Can also be used to modify layout commands

Edge Flags

Edge flags are specialized content flags that are used to draw lines around the edges of content items. Edge flags are described in Table A-4. Note that most edge flags can be used to modify layouts as well as content items.

Table A-4 Edge flags

Flag	Description
`TopEdge`	Draw line above content item.
`LeftEdge`	Draw line at left edge of content item.
`BottomEdge`	Draw line at bottom edge of content item.
`RightEdge`	Draw line at right edge of content item.

Table A-4 Edge flags

Flag	Description
`Edges`	Draw box around content item.
`Round`	Draw box with rounded corners around content item.
`Reverse`	Draw content item in white on black.
`EdgeWidth`	Specify width of line drawn by edge flags.

You can use the `edgeWidth` flag to specify in pixels the width of the line to be used with the edge flags. Place the `edgeWidth` flag on the same line as the edge flag, as in the following example:

edgeCommand `edgewidth=`*number*

NewtonScript Methods

The following NewtonScript methods, provided by Newton Book Reader, are available at run time.

AddBookmark

`:AddBookmark (`*pageNumber*`)`

Adds a bookmark at the specified page number in the current book. Only six bookmarks can be stored per book.

pageNumber The number of the page to bookmark.

AddBookRouting

`:AddBookRouting(`*routingArray*`)`

Adds or removes routing menu items as specified. To remove all items added by this book, pass `nil` as the value of the *routingArray* parameter.

routingArray An array of routing frames to be added. The routing frame must contain the following slots.

`title`	Required. The string that appears in the routing menu.
`icon`	Optional. An icon that appears next to the string that this function adds to the routing menu.
`routeScript`	A symbol specifying the page script or book script that is this menu item's routing script.

For more information on routing see the *Newton Programmer's Guide*.

AddBrowser

`:AddBrowser(`*browser*`)`

Adds the specified browser to the book. This method returns a reference to the browser, which can be used as parameters to the `:OpenBrowser` and `:CloseBrowser` methods, which display and remove the browser.

browser A frame containing the definition of the browser. This frame must contain the following two slots:

`name`	A string denoting the name to display at the top of the browser pane.
`list`	An array of frames, one per browser line, containing information about how to display the line and where to go when it is tapped.

The frames in the `list` array should contain the following slots:

`level` Optional. Specifies the level at which to display this entry in the browser. This has the same effect as the optional *level* argument to the `.subject` command, which is described on page A-23. If this slot is not included, it is displayed at level 1.

`item` Required. A reference to the content item to display when this browser line is tapped.

`name` Optional if `item` is a reference to a text content item. A string specifying the text to display in the browser line. If the content item that is referenced by `item` is a text content item, this slot can be omitted, and that text is displayed in the browser.

See also the section “Creating Dynamic Browsers” beginning on page 4-25 of Chapter 4, “NewtonScript in Books,” and the descriptions of “CloseBrowser” on page A-40, and “OpenBrowser” on page A-47 of this Appendix.

AuthorData

```
:AuthorData()
```

Returns the reference of a soup-based frame in which book script authors may store data that is to remain in the a soup on the Newton device. This soup entry is maintained even after the book has been removed.

IMPORTANT

Do not put any data in this framethat is resident in the book itself. Since books might be removed at any time, any copied data might render the entire data frame invalid. ▲

BookData

```
:BookData()
```

Returns the reference of the data frame in which book script authors can place book-specific data. For example, the `.index` command also puts the `alphaIndex` and `subIndex` arrays here. This data is compiled with the book and becomes part of the book's package itself. Any data kept here is considered read-only at run time.

Bookmarks

```
:Bookmarks ()
```

Returns an array of bookmarks for the current book.

```
[number, …]
```

BookTitle

```
:BookTitle ()
```

Returns the slot containing the title of the current book.

ChangeScrolledOrigin

`scroller:ChangeScrolledOrigin(`*dX*`, `*dY*`)`

Sets the scroll position of a picture displayed by the built-in scroller in Book Reader. All views created by a `.picture` that is larger than the screen have a `scroller` slot, send the `ChangeScrolledOrigin` message to this slot.

dX The number of pixels to scroll horizontally. Positive values offset the visible area to the right; negative ones, to the left.

dY The number of pixels to scroll vertically. Positive values offset the visible area downward; negative ones, upward.

CloseBrowser

`:CloseBrowser(`*browserRefNum*`);`

Closes the specified browser, if open.

browserRefNum	A reference to the browser to close, as returned by the `AddBrowser` function.

CountPages

```
:CountPages ()
```

Returns the number of pages in the current book.

CurrentKiosk

```
:CurrentKiosk ()
```

Returns a reference to the first kiosk page found when searching backwards in the current book from the currently displayed page. If a kiosk is found, this method returns a frame; otherwise, it returns `nil`. The frame returned by this method has the following slots:

`page`	The number of the page on which the kiosk is displayed.
`name`	The name assigned to the kiosk by its `name=` parameter in the kiosk definition.

CurrentPage

```
:CurrentPage ()
```

Returns the currently displayed page in the current book.

Find

`:Find (`*`string`*`, `*`results`*`, `*`scope`*`, `*`statusContext`*`)`
Searches for the specified string in one or more books.

string — The user-specified string for which `Find` is to search the book content.

results — An array of slots passed to the `Find` method by the system; the `Find` method appends a slot, `found`, to this array. The `hits` array contains the `page` on which each foulnd item appears, the `block` in which it appears, the length (`len`) of the item, and its `char` offset in the block. If a global find is in progress, the `results` array may contain slots created by other applications' search methods.

scope — The value of this argument is always one of the symbols `'local` or `'global`. The symbol `'local` specifies that Book Reader search in the current book; the symbol `'global` specifies that Book Reader search in all books.

statusContext — A frame to which Book Reader sends the `SetStatus` message. The `SetStatus` function reports progress to the user while the search is in progress.

This method returns an array in `results` that has an element for each book in which the specified string was found. Each element of this array consists of a frame with the slots shown in the following example:

```
{title: "book_title",
items: [{title: "string_found",
         isbn: book_isbn,
         found: {len: stringLength, item: theContent,
         char:starting_char},
         {.}}]
}
```

For more discussion of the `Find` method, see the "Additional System Services" chapter of the *Newton Programmer's Guide*.

FindContentBySlot

`:FindContentBySlot (`*aSymbol*`, `*aDepth*`)`

Returns an array of content items from the currently open book with a slot with the name specified by the *aSymbol* parameter.

aSymbol The name of the slot to find.

aDepth Specifies the depth of the search. If the value of the *aDepth* parameter is `nil`, this method returns all content items that meet the search criteria; if the value of the *aDepth* parameter is `true`, this method returns only the first content item that meet the search criteria.

Assist scripts can use the *aDepth* parameter to supply a book frame for use by the Intelligent Assistant when the book is not open. For more information about book frames, see the "Book Data" section of Chapter 4, "NewtonScript in Books."

FindContentByValue

`:FindContentByValue (`*aSymbol*`, `*aValue*`, `*aDepth*`)`

Returns an array of content items from the currently open book with a slot with the name specified by the *aSymbol* parameter and with the value specified by the *aValue* parameter.

aSymbol The name of the slot to find.

aValue The value to match in slots with the specified name.

aDepth Specifies the depth of the search. If the value of the *aDepth* parameter is `nil`, this method returns all content items that meet the search criteria; if the value of the *aDepth* parameter is `true`, this method returns only the first content item that meet the search criteria.

Assist scripts can use the *aDepth* parameter to supply a book frame for use by the Intelligent Assistant when the book is not open. For more information about book frames, see the "Book Data" section of Chapter 4, "NewtonScript in Books."

FindContentsByPage

`:FindContentsByPage(`*aPage*`)`

This method returns an array of all of the content items on a page.

aPage	The page from which this method returns content.

FindPageByContent

`:FindPageByContent (`*aContent*`, `*anOffset*`, nil)`

Returns the page on which the characters of *aContent* at *anOffset* can be found.

aContent	The content item for which this method searches.
anOffset	The number of characters into *aContent* where the part of *aContent* that you are interested in is.

The second parameter, *anOffset,* is important, since content items are frequently split over multiple pages. A value of `0` for this parameter returns the first page on which *aContent* is laid out. If *aContent* was ten paragraphs long, and you wanted to know which page the sixth paragraph was laid out on, you would need to pass in a value for *anOffset* equal to the number of characters in the first five paragraphs.

FindPageBySubject

`:FindPageBySubject(`*subjectNum*`)`

Returns the page number on which the specified subject appears.

subjectNum	The number of the subject to locate. Subjects are numbered consecutively from the beginning of the book definition file. The first subject in the book definition file is numbered `1`.

FindPageByValue

`:FindPageByValue (`*aSymbol*`,` *aValue*`,` *aDepth*`)`

Returns an array of pages from the currently open book that contain content items that have a slot with the specified symbol and value.

aSymbol	The name of the slot to find.
aValue	The value to match in slots having the specified name.
aDepth	Specifies the depth of the search. If the value of the *aDepth* parameter is `nil`, this method returns all content items that meet the search criteria; if the value of the *aDepth* parameter is `true`, this method returns only the first content item that meet the search criteria

Assist scripts can use the `aDepth` parameter to supply a book frame for use by the Intelligent Assistant when the book is not open. For more information about book frames, see the "Book Data" section of Chapter 4, "NewtonScript in Books."

GetLibraryEntry

`:GetLibraryEntry(`*isbn*`)`

Returns the specified book’s library entry frame. The frame returned by this method contains the slots described below:

`bookPresent`	This value is non-0 if the book is currently on-line.
`title`	Title of the book.
`isbn`	The book’s isbn string.
`curPage`	The page the book was (is) on.
`prevPage`	The page it was on before `curPage`.
`inkMarks`	An array of ink marks arrays, one per rendering.
`marks`	An array of book marks array, one per rendering.
`data`	The author data frame.
`curRendering`	Integer indicating the book’s current rendering. A value of 0 means that the book was (is) being rendered on a MessagePad-sized screen.
`flags`	Book level flags.
`packageID`	The book package’s ID.

GetOfflineBooks

`:GetOfflineBooks()`

Returns an array of all books remembered by the Newton but not currently available.

GetOnlineBooks

`:GetOnlineBooks()`

Returns an array of all books currently on the Newton.

HiliteBlock

`:HiliteBlock (`*aContent*`, `*anOffset*`, `*aLen*`)`

Highlights the item (usually text) contained in the content item *aContent* at the offset *anOffset* with a length of *aLen*.

aContent	The content item containing the items to highlight.
anOffset	The number of characters preceding the first character of the text to highlight. The value of *anOffset* is measured from the beginning of the content item, even if the currently displayed page shows only part of it.
aLen	The number of characters to highlight, beginning from the first character after the *anOffset* value.

HideStoryCard

`:HideStoryCard ()`

Hides the currently displayed story card.

InsertForm

`:InsertForm (`*aForm*`)`

Adds the specified view to the current page, and returns the view inserted.

aForm	The system prototype, view template, layout or user template to be added to the page.

The specified view is not implicitly made visible by the `InsertForm` method; you may need to make it visible yourself. For example:

```
:InsertForm(aForm)
local theView;
theView := GetView(aForm);
theView:Show();
```

OpenBook

`:OpenBook (`*isbnStr*`)`

Opens the specified book.

isbnStr The ISBN string of the book to open.

OpenBookTo

`:OpenBookTo (`*isbnStr, subjectNum*`)`

Opens the specified book to the specified subject.

isbnStr The ISBN string of the book to open.

subjectNum The number of the subject to locate. Subjects are numbered consecutively from the beginning of the book definition file. The first subject in the book definition file is numbered `1`.

OpenBrowser

`:OpenBrowser (`*browserRefNum*`)`

Displays the specified browser.

browserRefNum A reference to a browser created by the `AddBrowser` method.

PageSize

`:PageSize (nil)`

Returns the height and width of the currently displayed page. This method returns the frame:

`{height: `*heightPixels*`,width:`*widthPixels*`}`

PageThumbnail

`:PageThumbnail (`*aPage*`)`

Creates a bitmap view of the specified page at one-eighth size. Returns the frame:

`{bits: <page÷8>, bounds: {}}`

aPage The number of the page to image as a thumbnail view.

PreviousPage

`:PreviousPage ()`

Returns the previous page displayed by the current book.

RegisterBookRef

`:RegisterBookRef(`*isbnStr*`, {book: `*bookRef*`})`

This method registers a help book with the system. You need to call this method if your help book was not downloaded as an independent package.

isbnStr The ISBN string of the help book.

bookRef A reference to the help book's `book` frame.

See also "UnRegisterBookRef" on page A-51.

RemoveBrowser

`:RemoveBrowser(`*browserRefNum*`)`

Removes the specified browser.

browserRefNum A reference to the browser to remove, as returned by the `AddBrowser` function.

ScrollPage

`:ScrollPage (`*aDelta*`)`

Scrolls the specified number of pages from the current page.

aDelta The number of pages to scroll. Setting the value of the *aDelta* parameter to zero causes the current page to be redrawn.

ScrollReceiver

`:ScrollReceiver(`*aView*`);`

Enables books to intercept scroll events generated when the user taps the built-in arrows for use in their own on-page views. The view needs to have a `viewScrollUpScript` and a `viewScrollDownScript` to handle the messages that will be sent to it.

aView The view to which scrolling messages are sent. Pass `nil` to disable scrolling messages in the book, and give back control of the built-in arrows to Book Reader.

SetStatusButtons

```
:SetStatusButtons({left: [btn1, btn2], right:[btn]})
```

Adds the specified buttons to the status bar.

left An array of templates to appear to the left of the page number button. Passing an empty array specifies that no buttons are to be added to that side, and removes any existing buttons that were created by a previous call to `SetStatusButtons`.

right An array of templates to appear to the right of the page number button. Passing an empty array specifies that no buttons are to be added to that side, and removes any existing buttons that were created by a previous call to `SetStatusButtons`.

ShowStoryCard

`:ShowStoryCard (`*aSymbol*`, `*aValue*`, `*aBounds*`)`

Displays the content item that has the slot `aSymbol`, storing the value `aValue`, in a window at the size and location specified by `aBounds`. The content item can be any type of content, including stories and pictures. Only one story card can be displayed at a time.

aSymbol The name of the slot to find.

aValue The value to match in slots with the specified name.

aBounds A `viewBounds` frame specifying the size of the story card and the location in which to display it.

TurnToContent

`:TurnToContent (`*aSymbol*`, `*aValue*`)`

Displays the first page containing a content item that has a slot with the symbol *aSymbol*, and the value *aValue*.

aSymbol The name of the slot to find.

aValue The value to match in slots with the specified name.

TurnToPage

`:TurnToPage (`*aPage*`)`

Displays the specified page.

aPage The number of the page to display.

TurnToSubject

`:TurnToSubject(`*subjectNum*`)`

Turns to the page on which the specified subject is located.

subjectNum The number of the subject to locate. Subjects are numbered consecutively from the beginning of the book definition file. The first subject in the book definition file is numbered `1`.

UnRegisterBookRef

`:UnRegisterBookRef(`*isbnStr*`)`

This method unregisters a help book from the system. If you have called `RegisterBookRef`, call `UnRegisterBookRef` from your application's `removeScript`.

isbnStr The ISBN string of the help book.

See also "RegisterBookRef" on page A-48.

WhereIsBook

`:WhereIsBook(`*isbn*`)`

Returns a frame containing the specified book's library entry and book frame.

isbn The book's ISBN string.

The frame returned by this method contains the slots described below:

`library` The library entry frame for the specified book. For a complete description of the library entry frame, see the description of the `GetLibraryEntry` method on page A-45.

`bookSoup` The book frame for the specified book. The book frame contains all of the data and developer-supplied code that comprises the book.

Book Reader Messages

This section provides a list of messages the system sends to a digital book. These can be handled by providing a method named after the message in the appropriate view in the book. These messages allow the book to take action in response to certain conditions that arise at run time.

The messages that begin with “`Book`” are sent to the book. The scripts that handle these messages must be written at the book level; i.e., before the definition of any layouts or content items. Messages that do not begin with “`Book`” are sent to content items.

Note that if the message includes parameters the methods must be implemented using the `slot` parameter to the `.script` command, as described in the description of “Script” beginning on page A-19 of this appendix.

BookInstallScript

`BookInstallScript(`*bookFrame*`)`

Sent to the book when the book package is installed.

bookFrame	A frame containing the `book` slot. The `book` slot is the topmost frame of an entire book.

BookOKClose

```
BookOKClose();
```

Sent to the book when the user taps the book’s close box. Your `BookOkClose` method must return `true` to allow the book to close; otherwise, it must return `nil` to prevent the book from closing.

BookRemoveScript

`BookRemoveScript(`*removeFrame*`)`

Sent to the book when the book package is removed.

removeFrame	A frame containing only the ISBN slot. The ISBN slot contains the `.isbn` value for the book being removed.

BookHideScript

```
BookHideScript();
```

An optional method to execute when the book closes.

BookPrint

```
BookPrint();
```

An optional method supplied by the book; it returns any data that is useful at print time. The system stores the data returned by this method in the book's `printData` slot.

Because the actual printing of pages in a book may occur after many page changes or when book is closed, the `BookPrint` method, allows the you to store away data at the time the print slip is opened. This data can then be accessed by your scripts when the page is actually printed. This method is typically used to restore scroll position or state so that the right thing is printed.

For example:

```
.script bookprint
{scrollPos: firstCity, selection: curCity};
.endscript
.# This data is could then be used by, for example, a
.# viewDrawScript, to image the proper information .#
when this page is printed, as shown below:

.script viewDrawScript
local myScrollPos := printData.scrollPos;
local mySelection := printData.selection;
//code to do actual scrolling to be supplied.
.endscript
```

BookSearchScript

`BookSearchScript(`*searchStr*`,`*stringLen*`,`*theContent*`,`*bookData*`,` *bookFrame*`)`

searchStr The string to find

stringLen The length of the *searchStr* string.

theContent Reference to the content to search.

bookData From `book.data`.

bookFrame The overall book frame.

This method performs a customized search on the book that receives the `bookSearchScript` message. This method should be attached to the book; that is, the following code should go before all definitions of layouts or content items:

```
.script bookSearchScript
func ( searchStr, stringLen, theContent, bookData,
      bookFrame)
begin
// Perform a search of theContent
end
.endscript
```

The Book Reader sends a `bookSearchScript` message for each content item that is not flagged `NoSearch`.

If your `BookSearchScript` wants the system-supplied Book Reader search engine to search this content, it should return `nil`. If the script handled the search but there was no hit, it should return `true`. If the script handled the search and there was a match, this method must return a results frame containing the slots described below:

`len` The number of characters found. If the book provides its own hit-highlighting function, the `len` slot can contain any value that is useful to the custom hit-highlighting function.

`char` The character offset to the beginning of the string that was found. If the book provides its own hit-highlighting function, the `len` slot can contain any value that is useful to the custom hit-highlighting function.

`title` The text to display in the `Find` overview.

BookShowScript

`BookShowScript();`

Sent to the book when it opens.

FormHiliteScript

`FormHiliteScript(`*anOffset*`,`*aLen*`)`

Performs any application-specific actions necessary to highlight the found item. This method should be attached to a content item.

anOffset The first character of the found string.

aLen The number of characters to be highlighted, as returned by the `FormSearchScript` method or as found in the `.form` content item's text slot.

FormSearchScript

`FormSearchScript(`*searchStr*`, `*stringLen*`)`

searchStr The string to be found

stringLen The length of the *searchStr* string.

Performs a customized search on the `.form` content item that receives the `formSearchScript` message. It should be attached to a content item. This method must return `nil` when no match is found. When at least one occurrence of the target string is found, this method must return a results frame containing the slots described below:

`len` The number of characters found. If the book provides its own hit-highlighting function, the `len` slot can contain any value that is useful to the custom hit-highlighting function.

`char` The character offset to the beginning of the string that was found. If the book provides its own hit-highlighting function, the `len` slot can contain any value that is useful to the custom hit-highlighting function.

`title` The text to display in the `Find` overview.

MungeContentScript

`MungeContentScript (`*contentRef*`)`

Sent to a content item whenever Book Reader needs to access the data in that content item; that is, when the content item is imaged, printed, searched, or bookmarked. The method that handles this message must return a content item. This method can be used, for example, to decompress data on the fly, as in the following code:

```
.script slot MungeContentScript
func (aContent)
begin
    local newContent := Clone(aContent);
    newContent.data := :DecompressData(newContent.data);
    newContent;
end
```

ThumbNailScript

Sent to a content item at bookmarking time. It should return a view template that provides a customized bookmark. For example:

```
.script thumbnailScript
local ocean:= {_proto: protostatictext,
             text: "A big aqua one",
             viewFont: ROM_fontsystem14bold};
return ocean;
.endscript
```

NewtonScript Global Functions

A number of NewtonScript global functions exist to provide an application with an interface to Book Reader. These functions are intended to be called by an application, and not a digital book.

BookAvailable

`BookAvailable ({book:` *bookFrame*`}, 0)`

Makes a digital book available. It returns `nil` if there was a problem installing the book, and `true` if installation was successful.

bookFrame The outermost frame of a book. This frame contains all the information that defines a digital book. It is created by Book Maker in the book definition file.

BookRemoved

`BookRemoved ({isbn:` *isbnStr*`})`

Removes the specified book. This function returns 0 if there were no problems removing the book.

isbnStr The book’s ISBN string.

OpenHelpBook

`OpenHelpBook (`*isbnStr*`)`

Opens the specified help book.

isbnStr The ISBN string of the book to open.

OpenHelpBookTo

`OpenHelpBookTo (`*isbnStr*`, `*topic*`)`

Opens the specified help book to a certain topic. This has the same effect as opening the help book, and having that topic tapped in the browser.

isbnStr The ISBN string of the book to open.

topic A string specifying which topic to display in the help browser. It must match one of the `.subject` lines in the help book.

OpenHelpTo

`OpenHelpTo (`*topic*`)`

Opens the system-supplied help browser to the specified topic.

topic In the 2.0 version of the Newton system software, this can be any of the following strings: `"write"`, `"draw"`, `"notepad"`, `"names"`, `"dates"`, `"todo"`, `"extras"`, `"iobox"`, `"prefs"`, `"exchange"`, `"route"`, `"find"`, or `"assist"`.

ShowManual

`ShowManual ()`

Opens the system-supplied help browser.

Summary of Commands, Functions, and Methods

This section contains a list of all the Book Maker commands, NewtonScript functions, and methods described in this chapter.

Book Maker Commands

Some of the commands listed below are used only in conjunction with NewtonScript programming; these are flagged as **NSrequired**.

Document Commands

`.assist` **NSrequired**
`.author`
`.blurb`
`.copyright`
`.date`
`.endassist` **NSrequired**
`.endpostamble` **NSrequired**
`.endpreamble` **NSrequired**
`.expires`
`.key`
`.isbn`
`.postamble` **NSrequired**
`.preamble` **NSrequired**
`.publisher`
`.shortTitle`
`.title`

Content Commands

`.attribute` **NSrequired**
`.chapter`
`.endform` **NSrequired**
`.endscript` **NSrequired**
`.endkiosk`
`.form` **NSrequired**
`.indent`
`.kiosk`
`.mark` **NSrequired**
`.picture`
`.script` **NSrequired**
`.space`
`.story`
`.subject`
`.usemark` **NSrequired**

Browser Commands

`.browser`

Page Layout Commands

`.layout`
`.header`
`.pictheader`
`.running`

Miscellaneous Commands

`.chain`
`.#` (comment symbol)
`.index` **NSrequired**
`.option` **NSrequired**

Flags

Document Flags

`NoReLayout`

Layout Flags

`Kiosk`
`Main`
`NoTitle`
`Sidebar`

Content Flags

`Main`
`Sidebar`
`ToEdge`
`Centered`
`NeverBreak`

```
StartsPage
PageMiddle
PageBottom
AlignTop
AlignBottom
AlignCentered
BrowserOnly
BroswserAutoClose
Overlay
KeepWith
NoExtend
NoPage
NoScroller
NoSearch
```

Edge Flags

```
TopEdge
LeftEdge
BottomEdge
RightEdge
Edges
Round
Reverse
EdgeWidth
```

NewtonScript Methods

:AddBookmark (*pageNumber*)
:AddBookRouting (*routingArray*)
:AddBrowser(*browser*)
:AuthorData ()
:BookData ()
:Bookmarks ()
:BookTitle ()
scroller:ChangeScrolledOrigin(*dX*, *dY*)

```
:CloseBrowser(browserRefNum)
:CountPages ()
:CurrentKiosk ()
:CurrentPage ()
:Find (string, results, scope, statusContext)
:FindContentBySlot (aSymbol, aDepth)
:FindContentByValue (aSymbol, aValue, aDepth)
:FindContentsByPage(aPage)
:FindPageByContent (aContent, anOffset, nil)
:FindPageBySubject(subjectNum)
:FindPageByValue (aSymbol, aValue, aDepth)
:GetLibraryEntry(isbn)
:GetOfflineBooks()
:GetOnlineBooks()
:HiliteBlock (aContent, anOffset, aLen)
:HideStoryCard ()
:InsertForm (aForm)
:OpenBook (isbnStr)
:OpenBookTo (isbnStr, subjectNum)
:OpenBrowser (browserRefNum)
:PageSize (nil)
:PageThumbnail (aPage)
:PreviousPage ()
:RegisterBookRef(isbnStr, {book: bookRef})
:RemoveBrowser(browserRefNum)
:ScrollPage (aDelta)
:ScrollReceiver(aView);
:SetStatusButtons ({left: [btn1, btn2], right:[btn]})
:ShowStoryCard (aSymbol, aValue, aBounds)
:TurnToContent (aSymbol, aValue)
:TurnToPage (aPage)
:TurnToSubject(subjectNum)
:UnRegisterBookRef(isbnStr)
:WhereIsBook(isbn)
```

Book Reader Messages

Book Messages

```
BookInstallScript
BookOKClose
BookRemoveScript
BookHideScript
BookPrint
BookSearchScript
BookShowScript
```

Content Item Messages

```
FormHiliteScript
FormSearchScript
MungeContentScript
ThumbNailScript
```

NewtonScript Global Functions

`BookAvailable ({book:` *bookFrame*`}, 0)`
`BookRemoved ({isbn:` *isbnStr*`})`
`OpenHelpBook (`*isbnStr*`)`
`OpenHelpBookTo (`*isbnStr*`,` *topic*`)`
`OpenHelpTo (`*topic*`)`
`ShowManual ()`

APPENDIX B

Troubleshooting

This Appendix presents solutions to common problems encountered when using Book Maker and Book Reader.

Font-Related Problems

This section lists a number of font-related problems and workarounds for them.

Truncated Paragraphs or Improper Layout

You may experience text-layout problems if you have compiled your book source file on a Macintosh computer that does not have the appropriate bitmapped fonts installed. Book source files compiled with TrueType versions of the Newton system fonts (New York, Geneva, or Espy), rather than the bitmapped versions of these fonts in the appropriate sizes, compile successfully but may not display correctly on the Newton device.

Because Newton currently does not use TrueType fonts, the bitmapped system font is scaled when Newton displays the book. As a result, you may experience problems with incorrect text layout because the scaled bitmap font image measures slightly differently than the TrueType image of the same font.

To work around this problem, you must reprocess the book source file on a system with the appropriate bitmapped versions of the Newton system fonts.

Text Layout Problems

Text that does not display correctly, or paragraphs that truncate when they are scrolled, result from using unsupported fonts or font sizes for content items in your book source file. Typically, the remainder of a paragraph that is truncated with an ellipsis (…) at the bottom of the Newton screen does not appear at the top of the next page.

To solve this problem, use only the supported fonts —New York, Geneva, or Espy in sizes 9, 10, 12, 14, and 18—for content items in the book source file. After reprocessing the book source file in Book Maker and NTK, the text should display correctly.

Improper Espy Sans Bold Style

You must use the Espy Sans Bold font in the book source file to obtain a boldface Espy Sans typeface in the digital book created from it. Applying the bold style to the ordinary Espy Sans font is not equivalent to using the Espy Sans Bold font and may cause page layout problems in the book package.

Espy Font Substituted

Newton substitutes the system font (Espy) when displaying book text written in an unsupported font. This may result in the improper truncation of paragraphs that straddle page boundaries.

To work around this problem, you must change all content items in the book source file to supported fonts, and reprocess the book source file on a system that has the appropriate bitmapped versions of the Newton system fonts.

Printing on a LaserWriter

There are two font-substitution problems related to LaserWriters:

LaserWriters do not have the Geneva and New York fonts, so they use the Times and Helvetica fonts instead. The LaserWriter and the Newton device use different-sized versions of these fonts. Since the LaserWriter’s version of these fonts is slightly bigger, some text may be truncated at the bottom of the page.

Substituting the Helvetica font for the Geneva, and Times for New York in the book definition file, and processing again through Book Maker and NTK should solve this problem.

Also, LaserWriters do not have the Espy Sans font. They substitute this font with Helvetica, which can sometimes cause unexpected results when you print a page from a book that uses this font. If you use the Espy Sans font in your book, you should be aware of this limitation.

Book Maker Problems

This section lists problems that are commonly encountered in writing book definition files and processing them with Book Maker.

Lost Styles

Placing dot commands within story text may cause Book Maker to lose style information; for example, story text that is interrupted by a command may no longer be bolded, italicized, and so on. Placing the commands associated with a story either at the beginning or end of its text ensures consistent results.

Incorrect Error Messages

If Book Maker notifies you of errors that don't appear to exist in your source file, you may be experiencing a known problem with the format used by the Fast Save option available on some word processors (Microsoft Word is known to cause this problem).

To work around the problem, turn off the Fast Save option and use the Save As… command to save your book source file. After you have re-saved the source using the Save As… command, you can use the Save command to save future changes to the file as long as the Fast Save option remains disabled.

XTND and Large Files

The Claris XTND translators used by Book Maker to convert various word processor files may return an error when trying to translate files larger than 1 MB.

To work around this problem, break the large file into two or more smaller files and use the `.chain` command to compile the files as a single Book Maker book. For more information, see the description of the `.chain` command later in this chapter, in the "Miscellaneous Commands" section of Appendix A, "The Book Maker Language."

No Scripts in Page .headers, Use .running story Instead

You cannot use scripts in `.header` content. Use a `.running` content item instead. To turn off the default header, use the `NoTitle` flag in the `.layout` line. For example:

```
.layout Default 12 NoHeader
.running story Centered
Tap here to do something cool.
.script viewClickScript
...  // do something cool
.endScript
```

Controlling How Find Results Are Displayed

When the user preforms a search with the Find button and more than one item is found, the find overview displays some text to show the item "in context." Book Reader generates the "in context" text from your source document. It chooses a few words before and after the word that was found as the context, (spaces, tabs, and carriage returns are each treated as a word). If there is a borderline between two content areas (e.g., separate `.story` items), text from the adjoining area is not displayed in the find overview.

To avoid displaying nearby text as part of the overview, you can add additional separation in the form of extra spaces, tabs, or carriage returns, or you could make each piece of text a single `.story` content item.

NTK Problems

This section lists common problems encountered in the process of building and downloading book packages.

Can’t Delete Old Package

If you have the Delete Old Package On Download option checked in the Settings… dialog box for your NTK project, NTK replaces the version of the package on the Newton with the version being downloaded. This allows you to replace the book or application already on the Newton device with new versions as you develop them.

NTK uses the package signature to find the package to replace; if you have changed the package signature for your book package but not changed its other identifiers, such as the ISBN string or application symbol, you may be notified by NTK that it couldn’t download the package because of a program error. This commonly happens when new users are experimenting with the example files and wind up changing the package signature after downloading one version of the package to the Newton device.

You can take either of the following steps to remedy the situation:

- Remove the old package by selecting its icon, and choosing Delete from the Routing menu.
- Change the offending identifier to cause a package that is actually present on the Newton device to be removed.

Title Not Updated on Loading Updated Book

The Book Reader application on Newton caches certain parts of the book, such as the title and the current page. To get the engine to notice the title has changed, you have to change the `.isbn` of the book before you load the updated one.

If you are likely to update your book, you should use a dash in the `.isbn` command to indicate version or date. For example:

```
.title My Info Book of 7/94
.isbn  IB-7-94:PIEDTS
```

Removed Last Page Displayed or Bookmarked Page

If in the course of development of a book, the last page displayed or a bookmarked page is removed, you will get an error number -48205. This is because Book Reader keeps a reference to the last page displayed and to all the bookmarked pages.

To work around this problem change the ISBN number of the book, and reprocess through Book Maker and NTK. This problem only occurs during the development process, an end user should never have this problem.

Curly Quotes Don’t Compile

NewtonScript is restricted to 7-bit ASCII. If you add characters that are outside this range you will get an “illegal character” error message in NTK. For example, the problem can be caused by the “smart quotes” feature in the word processor you’re using to write your book source file. Curly—or smart—quotes are not part of the ASCII character set. If you use only straight quotation marks in NewtonScript statements, those irritating messages go away. For example:

```
.script
Print(“Whoops”) // won't compile
Print("Whoops") // will compile
.endscript
```

Global Functions Warnings

A warning (not error!) about global functions is produced in the Inspector if NTK cannot find a function in the platform file. Book Maker creates function calls that might not be in the platform file you are using. This does not affect the book being built in any way.

You may want to uncheck the Check global function calls option in the Project Settings... dialog box, accessed through the Project menu in NTK.

Books on Online Help

This appendix lists books and journal articles that discuss the creation of online help.

Aaronson, A., and J. M. Carroll. "Intelligent Help in a One-Shot Dialog: A Protocol Study." In *CHI + GI'87 Conference Proceedings: Human Factors in Computing Systems and Graphics Interface*, edited by J. M. Carroll and P. P. Tanner, 163-168. New York: ACM, 1987.

Brockmann, R. John. "The Documentation Problem." Part I of *Writing Better Computer User Documentation: From Paper to Hypertext, Version 2.0*. New York: Wiley, 1990.

Cohill, A., and R. Williges. "Retrieval of HELP Information for Novice Users of Interactive Computer Systems." *Human Factors* 27, no. 3 (1985): 335-343.

Conklin, J. "Hypertext: An Introduction and Survey." *IEEE Computer* (September 1987): 17-41.

Duffy, T., B. Mehlenbacher, and J. Palmer. "The Evaluation of Online Help Systems: A Conceptual Model." In *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Reality*, edited by E. Barrett, 362-387. Cambridge, MA: MIT Press, 1989.

Horton, William K. *Designing and Writing Online Documentation: Help Files to Hypertext*. New York: Wiley, 1990.

Kearsley, G. *Online Help Systems: Design and Implementation*. Norwood, NJ: Ablex, 1988.

Queipo, L. "User Expectations of Online Information." *IEEE Transactions on Professional Communications* 29, no. 4 (1986): 11-15.

Rubens, P., and R. Krull. “Application of Research on Document Design to Online Displays.” *Technical Communications* 32, no. 4 (1985): 29-34.

Schriver, K. A., J. R. Hayes, and M. D. Langston. “The Design of Information for Computer Users: A Review of the Literature on Hardcopy and Online Documentation.” In *Designing Computer Documentation: A Review of the Relevant Literature*, edited by K. A. Schriver. Communications Design Center Technical Report No. 31, Pittsburgh, PA: Carnegie Mellon University, 1986.

Walker, J. “Issues and Strategies for Online Documentation.” *IEEE Transactions on Professional Communication* 30 (1987): 235-248.

APPENDIX C

Books on Online Help

Compatibility

This appendix describes compatibility issues involved in writing digital books for 1.x Newton machines. Some of the features described in this book are only available in the version of Newton Book Reader (NBR) installed in 2.0 machines. Appropriate changes are noted in this appendix for digital books intended for display on 1.x machines.

Enhancements to the 1.1 version of Book Maker are also discussed.

This appendix is divided into three parts:

- **Book Maker Enhancements** describes enhancements to the 1.1. version of Newton Book Maker.
- **Bugs Fixed in Book Reader** describes problems in NBR 1.x which have been fixed in NBR 2.0. Most of these problems have an appropriate work around listed.
- **Book Reader Enhancements** describes additions to NBR in the 2.0 version.

Book Maker Enhancements

Book Maker 1.1 has introduced a parameter to the .index command to give you better control of the generation of subindices.

Controlling Generation of Subindices

The `.index` command can now accept an `@`*indexName* parameter. Book Maker 1.0 accepted only `!`*indexName* parameters. See the description of the `.index` command on page A-30 for more information.

Bugs Fixed in Book Reader

The NBR 1.x bugs described in this section have been fixed in the 2.0 version. Appropriate workarounds are described with the bugs.

Bookmarking and Printing Context

In NBR 1.x, when a bookmark is being created or a page is being printed, any scripts on the page fail when they try to run any book functions (`BookData`, `AuthorData`, etc.). The following functions are not available at print/bookmark time in NBR 1.x:

```
AddToContentArea
AuthorData
BookData
CountPages
CurrentBook
CurrentKiosk
CurrentPage
FindContentBySlot
FindContentByValue
HiliteBlock
InsertForm
WhereIsBook
```

In NBR 1.x if one of these functions is called in a `.script` that might be called at print/bookmark time, use a conditional message-send (`:?`), or provide the appropriate exception handler.

Length of ISBN Strings

On the MessagePad 100 ISBN strings can only be up to 15 characters long.

Information in “About” Slip Not Displayed

NBR 1.x does not display the information provided by the `.author`, `.date`, `.publisher`, and `.copyright` commands.

Page Scripts Override Book Scripts

Book-level scripts override page-level (layout) scripts in NBR 1.x. No page-level script is called if the book has one by the same name, or both the page and the book provide the default `buttonClickScript`.

Edge Flags With Multi-Page Contents

In NBR 1.x, the `topEdge` and `bottomEdge` flags cause a line to be drawn on every page of a content item that spans multiple pages. Do not use these flags with content items that span multiple pages.

Memory Management for Story Cards

When the `HideStoryCard` function is called or the user switches between multiple story cards, NBR 1.x hides the story cards rather than closing them. To work around this memory leak add the following code before any calls to `ShowStoryCard`:

```
if (storyCard <> NIL) then
begin
   storyCard:Close();
   storyCard := NIL;
end
```

Story Cards Always Left-Justified

Story Cards in NBR 1.x always left-justify text.

.form Content Items Do Not Respect viewJustify

A content item added by the .form command in NBR 1.x does not respect the value of its viewJustify slot.

InsertForm Function Does Not Return the View

The NBR 1.x version of the InsertForm function does not return the view inserted. Call the GetView function to obtain a reference to the view that was inserted.

Searching in .Form Content Items

In version 1.x, the system-supplied Find service did not search in .form content items. NBR 2.0 provides two ways to search in .form items: adding a text slot, and the FormSearchScript method. Neither of these mechanism is available in NBR 1.x.

A workaround is to have a .story item with the text to be searched, but covered with your .form item. Add the .story item to the page with the name of the location, so that a search can find it, but have the story live beneath the current frame, so the story is hidden from a user's view.

Here's how the NewtonScript would look:

```
.story
Joe's Place
.attribute myForm: layout_myForm
.script viewSetupDoneScript
   local theform, theview;
   theForm := {_proto: item.myForm, viewBounds:
viewBounds};
   theView := GetView(:InsertForm(theform));
   theView:show();
   SetValue(self, 'text, "");
.endscript
```

Note that you might have to tweak the right bounds of the frame since the story paragraph view extends all the way across its column. Also, the text that is to be found will not be highlighted (but you will get to the right page).

Help Books

In versions 1.x of the Newton system software, it is impossible to build stand-alone help books. Help books must be compiled as part of an applications package. Furthermore, this system does not supply the `protoInfoButton` (or `newtInfoButton`) to summon help from, so it is up to you to create a way to display help in your application.

You should supply a non-intrusive interface such as a "Show My Help" button. This section describes the minimum code necessary to have your help book displayed when the user taps the "Show My Help" button.

1. Comment out (or cut out) the following two lines from your book definition file, which are created by Book Maker to support stand-alone help books:

```
output.book := book;
output.help := TRUE;
```

These two lines are close to the top of your book definition file, after the definition of the `book` frame.

2. Create a compile-time constant by placing the following line in the book's postamble:

```
DefConst('kMyHelpBook, book)
```

3. Create an `evaluate` slot in you application's base view to hold the help book frame, call it `theBookFrame`.
4. In your application's base view `SetupDoneScript`, make the following assignment (this is in place of a call to `RegisterBookRef`, which is unavailable in the 1.x versions of the Newton system):

```
theBookFrame := BuildContext({_proto:
GetRoot().TinyTim._proto, bookRef: kMyHelpBook})
```

5. In your “Show My Help” button’s `buttonClickScript`, execute the following code:

```
GetRoot().TinyTim:Close(); // in case system help is open
theBookFrame:OpenManual(kMyHelpBook);
```

6. To close the help book, execute:

```
if theBookFrame then
begin
   theBookFrame:Close(); //so card can be removed
   theBookFrame := nil //so help book can be GC'd
end
```

This last step is extremely important; if this is not executed, the user will not be able to use the system help after your application has opened once.

Enhancements to Newton Book Reader

NRB 2.0 provides a number of features not present in NBR 1.x. This section lists these additions. Do not include the features mentioned in this section if your book might run on a 1.x Newton device.

Unavailable Methods

The following methods are new to NBR 2.0. If your book calls any of these functions, and the book is being run NBR 1.x, an exception will be thrown by the operating system. If the book might be run on NBR 1.x, either do not call these functions, or else provide appropriate exception handlers.

```
AddBookRouting
BookSearchScript
ChangeScrolledOrigin
CloseBrowser
FindContentsByPage
FindPageBySubject
GetLibraryEntry
GetOnlineBooks
GetOfflineBooks
OpenBook
OpenBookTo
PageSize
RegisterBookRef
RemoveBrowser
ScrollReceiver
SetStatusButtons
TurnToSubject
UnRegisterBookRef
```

Unsent Book Reader Messages

All of the messages sent by Book Reader listed in the section “Book Reader Messages” beginning on page A-51 are sent only by NBR 2.0. Any methods you have written to respond to these messages will not be called.

Unavailable Flag

The browserAutoClose flag is unavailable in NBR 1.x; you can mimic this behavior by adding an outlineClickScript method to the browser.

Glossary

book definition file
An output file produced by Book Maker. It is used as input to Newton Toolkit (NTK) to build a book package or create Newton application help.

Book Maker
An application that processes a book source file to produce a book definition file.

Book Reader
A system service that displays interactive digital books on the Newton screen.

book script
A script created with the `.script` command before the definition of any content items in the book source file. Because such a script is available throughout the entire book package, it is called a book script.

book source file
A word processor file containing content items tagged with Book Maker commands.

comment
A line of text in the book source file that is preceded by the comment symbol (`.#`), which is a dot and a pound sign. A comment appears in the source file but does not appear in the compiled book.

content command
A Book Maker command that defines a content item, such as text or graphics, to be displayed on the Newton screen.

content flag
A flag that modifies an individual content item; most flags in the Book Maker language are content flags.

flag
A keyword that is added to a Book Maker command to enable a feature.

document command
A Book Maker command that affects the entire book source file.

document flag
A flag that affects an entire book source file, such as the `noReLayout` flag.

global
A variable or function that is accesible throughout the book.

help book
The file that Book Maker produces when it processes a book source file with the Help Size option checked.

kiosk
A navigational page created by the `.kiosk` command. Tapping an item on the kiosk page, such as a picture, takes the user directly to the subject matter it represents.

layout
A Book Maker command that specifies the placement of text and graphics on the page.

layout command
A Book Maker command that defines a layout.

layout flag
A flag used to modify a layout command; the flag affects any page using that layout.

page script
A script attached to a layout. Because the script is available to any page using that layout, it is referred to as a page script.

point
A typographer’s unit of measure; there are 72 points in an inch.

project file
An NTK file that contains a list of files to be included in a build and the build specifications.

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers. Proof pages were created on an Apple LaserWriter Pro 630 printer. Final page negatives were output directly from the text and graphics files. PostScript™, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

WRITER
John Perry, Adrian Yacub

PROJECT LEADER
Christopher Bey

ILLUSTRATOR
Peggy Kunz

EDITOR
Linda Ackerman

PRODUCTION EDITOR
Rex Wolf

PROJECT MANAGER
Gerry Kane

Special thanks to Gabriel Acosta, David Dunham, Bob Ebert, and Scott Shwarts.