Newton ブックメーカー ユーザーズ・ガイド

for Newton BookMaker Version 1.1

アップルコンピュータ株式会社

翻訳　齋藤匡弘 (Kunihiro Saito)

mailto: brown@ga2.so-net.ne.jp

本書の配賦条件

本書は Apple の著作物 “Newton BookMaker User’s Guide” をベースにして翻訳・変更・加筆された物であり、個人で使用される場合に限りその使用を許可する。但し、Apple は翻訳文の内容について責任を負う物ではない。

本書の著作者

原書の著作者 :

日本語版の著作者 :

訳者のことば

「プログラミング言語 NewtonScript」を公開してから一年が過ぎた。その間 Newton にとっては様々なことが起こったが、それはさておき、まずは本書を一般に公開できた事を素直に喜びたい :-)

1997 年末から、私は Newton ブックメーカーを使って一つの Newton ブックを作成する機会に恵まれた。その時感じたのは Newton ブックの柔軟性・可能性だった。スタイル付きのコンテンツを単に表示できるだけでなく、NewtonScript を組み込めば様々に拡張することができるのだ。

こうして Newton ブックの魅力を発見した私は、もっとたくさんの人に Newton ブックについて知ってもらいたい、そして Newton ブックメーカーが意外に使いやすいものであることを知ってもらいたいと考えた。それが本書の翻訳を開始した動機だった。

原書の内容のうち、すでに現実にそぐわない部分は変更したり、削除した。また、訳者が独自に追加した部分も多少存在する。また、もし本書の内容に間違いを発見したら是非 brown@ga2.so-net.ne.jp までご一報いただきたい。

本書を読んで、本格的な Newton ブックを作ってみたいと思う人が増えることを祈っている。

1998 年 8 月 12 日

齋藤匡弘

目次

空のページ

図表目次

本書について

この *Newton ブックメーカー ユーザーズガイド*では、ブックメーカー 1.1 を使用して、PDA である Newton ファミリー用デジタルブックあるいはアプリケーションヘルプを作成する方法について述べる。

読者対象

このガイドは Newton ファミリー製品用のデジタルブックを作成したい全ての人のためのものである。ブック作成にはプログラミング経験は全く不要である。しかし、基本的な Macintosh の操作方法と、Newton Toolkit(NTK) によるプロジェクトのビルド方法には通じている必要がある。パッケージのビルドについてより詳しくは、NTK に付属の *Newton Toolkit ユーザーズガイド*を参照されたい。

プログラマでない人でも Newton アプリケーション用ヘルプのコンテンツを作成することはできる。Newton アプリケーションにヘルプを組み込むためには、NewtonScript 及び Newton オブジェクトシステムに通じている必要がある。

関連する書籍

本書は Newton 開発環境である Newton Toolkit に含まれる書籍群のうちの一つである。本書 *Newton ブックメーカーユーザーズガイド*は Newton ファミ

リー製品用のデジタルブックを作成するために必要となる情報を全て提供するが、以下の本も参照する必要が生じるだろう：

- *Newton Toolkit ユーザーズガイド*. この本は Newton 開発環境への導入と、Newton Toolkit による Newton アプリケーション開発方法を示す。ブックパッケージのビルドプロセスについてあまり知らない場合は、このガイドの該当セクションを参照すること。
- *Newton プログラマーズガイド：システムソフトウェア*。この本のセットは (通信をのぞく) Newton プログラミングガイドの決定版であり、参考書である。
- *プログラミング言語 NewtonScript*。本書は NewtonScirpt プログラミング言語について述べる。本書のどの部分もブックメーカーの利用のためには必要ではないが、ヘルプスクリーンを Newton アプリケーションに組み込もうとするなら、これについて十分通じている必要がある。さらに、デジタルブックをより良いものにするためには NewtonScirpt コードを追加する必要がある。
- *Newton プログラマーズガイド：コミュニケーション*。本書は Newton 通信プログラミングのためのガイドの決定版であり参考図書である。デジタルブック作成のために本書を読む必要は全くない。

本書の使用方法

本書は 5 つの章からなる。自分でブックを作成する前に、最低限 1 章及び 2 章は読む必要がある。

- 第 1 章　「Newton ブック」では、Newton デジタルブックの機能について説明する。この章では ブックメーカー、ブックリーダー、システム提供のヘルプブラウザについての導入を行う。ここではまた ブックリーダー用パッケージと、アプリケーションのヘルプの違いについても述べる。
- 第 2 章　「はじめの第一歩」では、Newton デジタルブックのビルドプロセスの簡単な導入を行う。
- 第 3 章　「ブックメーカー言語の使用」では、ページ上にテキストやグラフィックをレイアウトするために使うブックメーカーのコマンドについて説明する。

- 第 4 章　「NewtonScript」では、オプショナルな NewtonScript メソッド、スロット、テンプレート、フレームのデジタルブックでの使用に付いて述べる。この章の内容は NewtonScript プログラミングの能力と、Newton オブジェクトシステムに関する知識があることを前提としている。
- 第 5 章　「ヘルプ」では、Newton アプリケーションにブックメーカーを使って作成したヘルプスクリーンを追加するための方法について述べる。この章の内容は NewtonScript プログラミングの能力と、Newton オブジェクトシステムに関する知識が有ることを前提としている。
- 付録 A　「ブックメーカー言語」では、ブックメーカーで使用される様々なコマンドについて説明を行う。この付録に含まれるコマンドリファレンスセクションではブックメーカー言語の個々の要素について述べる。
- 付録 B　「トラブルシュート」では、一般的な問題と、その解決策について述べる。
- 付録 C　「オンラインヘルプに関する書籍」では、オンラインヘルプの作成に関して扱う様々な書籍や雑誌の記事について触れる。
- 付録 D　「互換性」では、ブックメーカー 1.1 に対して行われた変更や、1.x Newton デバイス上のブックリーダーでも表示可能なブックを作成するために必要な変更点について述べる。

サンプルブック

Newton, Inc ホームページからは *Newton ブックメーカーユーザーズガイド* でカバーされるトピックスの多くを例示するサンプル群が取得できる。

それぞれのサンプルブックフォルダは次のファイルから構成される：

- NTK プロジェクトファイル

  このファイルには NTK においてプロジェクトをビルドするのに必要な全てのファイルが記録されている。このサンプルファイルには、サンプルブックに関連する全てのファイルを含むフォルダと同じ名前が付けられている；たとえば、`Simple` フォルダは `Simple` という名前の NTK プロジェクトを含んでいる。

- ブックソースファイル

  これは ブックメーカーの入力となるワープロのファイルである。本書で示されるサンプルブックのソースリストはブックソースファイル内のどこかに入っている。このサンプルファイルは、NTK プロジェクトファイルの名前の後に、`story` という名前が追加されたファイル名を持つ。たとえば、`Simple` プロジェクトのためのブックソースファイルは、`SimpleStory` という名前である。

- ブックメーカー出力ファイル

  ブックメーカーはブックソースファイルを処理し、このファイルを生成する。これは NTK プロジェクトに追加され、ブックまたはアプリケーションをビルドするのに使用される。この出力ファイルの名前は、ブックソースファイル名に `.f` を追加して作成される。たとえば、`SimpleStory` ブックソースファイルが処理されて出力ファイルが作られると、それは `SimpleStory.f` という名前になる。

- ブックリーダーパッケージ

  これは Newton スクリーンの Extras Drawer 上に表示されるデジタルブックパッケージである。このパッケージは NTK プロジェクトのビルドによって生成される。サンプルパッケージの名前は、NTK プロジェクトファイルの名前に `.pkg` を追加して作成される。たとえば、`Simple` プロジェクトをビルドすると、ブックパッケージの名前は `Simple.pkg` となる。

1998 年の 8 月時点では、Newton の開発は終了しており、新たなサンプルコードの提供もしくはデベロッパサポートが Apple Computer からなされることはない。[1]

本書で使用される規約

本書では各種情報を表現するため以下の規約を使用する。

1. 1998/6/26 をもって Newton デベロッパ窓口は閉鎖された事により、この段落は原書と違うものにした。

特殊なフォント

本書は以下の特殊なフォントを使用する：

- **Bold 体**。重要な用語とコンセプトが最初に登場する時はボールド体で記述される。これらの用語は用語集で定義される。(和文では**ゴシック体**を使用。)
- `Courier` 体。プログラムリスト、コードの断片、定義済みシステムフレーム名、スロット名、関数名、メソッド名、シンボル名、定数といった特殊な識別子をテキスト中に入れるときは、クーリエ体で記述し、通常のテキスト本体と区別する。クーリエ体で表示されるアイテムは、その綴り通りにタイプされなければならない。
- Geneva **フォント**。ブックメーカーソースファイルリストは、テキスト本文と区別するため Geneva フォントで記述される。Geneva フォントで登場するアイテムは表示されている通りにタイプされなければならない。Geneva フォントは Newton ブックリーダー で直接サポートされていることから本書で選択した。このフォントをソースファイルで使用することによって、ソースファイル上での見た目と、Newton スクリーン上での見た目をより正確に合わせることができるようになる。
- *イタリック体*がブックソースコード中もしくは NewtonScript コード中で使用される場合、置換可能なアイテムを示す。それはたとえば関数のパラメタであったりする。また、本書以外の書籍の名前もイタリック体で表示される。
- 角カッコ ([]) はコマンド文法リストにおいて、オプションのパラメタを示す。カッコはオプションパラメタの一部分ではないので、ブックメーカーコマンドに含んではならない。

デベロッパ製品とサポート[1]

すでに Apple Computer による Newton 開発は停止されているため。この段落は訳さない。

1. 1998/6/26 時点でデベロッパサポート窓口はクローズされてしまったので、原書のこのセクションの記述は間違ったものとなったため、削除した。

NewtonScript プログラマのために

Newton デジタルブック作成には何らプログラミングの知識は必要ないものの、NewtonScript プログラマならブックに追加の機能を提供するために Newton オブジェクトシステムと NewtonScript を使用できる。このセクションでは NewtonScript プログラマにのみ影響のある問題を述べる。NewtonScript をデジタルブックに使用するつもりでなければ、このセクションはとばして第 1 章「Newton ブック」に進むと良い。

タップとクリック

Newton ソフトウェアシステム及び本書全体で、"click" という単語がメソッドあるいは変数の名前の一部として登場する。たとえば `viewClickScript` とか、`buttonClickScript` とかである。これによりそのテキストがマウスクリックを意味するのだと思ってしまうかもしれないが、それは違う。"click" という言葉がこんな風に登場した場合は、それは Newton スクリーン上でのペンのタップを意味するのだ (それはデスクトップコンピュータにおけるマウスクリックと似たようなものである)。

フレームコード

Newton Toolkit(NTK) 開発環境を本書とともに使用する場合、本書では (ビューのような) フレームのコードを NTK とは異なる方法で表示する。NTK においては、一度に一つのフレームスロットのコードを見られるだけである。本書では、フレームのコードを一度に全て示すので、フレーム内の全てのスロットは以下のように見られる：

```
{   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);
    };
```

NTK を使用している場合、以下の方法で本書に登場するようなフレームを生成できる：

1. NTK テンプレートパレット上で、ブック表示したいビュークラスあるいはプロトを探す。次にそしてそのテンプレートを使うビューを描く。本書に示されるフレームが `_proto` スロットを含んでいる場合、対応するプロトを NTK のテンプレートパレットから使用する。本書で示されるフレームが `_proto` スロットの代わりに `viewClass` スロットを含んでいれば、NTK テンプレートパレットから対応するビュークラスを使用する。

2 `viewBounds` スロットを、本書が示すのと同じに編集する。

3 フレーム内に表れる他のスロットを全て追加して、本書が示しているのと同じ値にする。値を持つスロットはアトリビュートスロットであり、関数を含むものはメソッドスロットである。

文書化されないシステムソフトウェアオブジェクト

NTK のインスペクタウィンドウでブラウジングをしているとき、本書で述べられていない関数、メソッド、データオブジェクトに出くわすことがある。そうした文書化されていない要素はサポートされないかもしくは将来の Newton デバイスにおいて動作が保証されないものである。それらを使用することは現行及び将来の Newton デバイスにおいて予期せぬ結果を生じることがある。

空白ページ

Newton ブック

Newton ブックリーダーは Newton のスクリーンに対話型のデジタルブックを表示するためのシステムサービスである。Newton ブックメーカーを使えば、プログラマでない人でも普通のワープロソフトを使って Newton ブックのソースファイルを作成することができる。

また、ブックメーカーアプリケーションによって、プログラマでない人も Newton アプリケーション用のヘルプを作成することができる。Newton アプリケーションパッケージにヘルプをインストールするには、初歩的な NewtonScript プログラミング能力が必要となる。

この章では Newton ブック の機能について説明するとともに Newton ブックメーカー , Newton ブックリーダー , システム提供のヘルプブラウザについて紹介する。また、ブックリーダーパッケージとアプリケーションヘルプのいくつかの違いについて説明も行う。

Newton ブック

Newton ブック はグラフィック、マルチフォントテキスト、ナビゲーションのためのコントロールを表示できる。ユーザーはページをスクロールでき、ブックマークでページをマークでき、ページ番号あるいはサブジェクトによってデータに直接アクセスでき、電子インクを使ってページにマークをつけたり、テキスト検索を実行したりできる。ユーザーはまた Newton ブック

からテキストをコピー&ペーストできるし、同様にテキストやグラフィックをプリントアウトできる。

ブックリーダーのブックは NewtonScript 及び Newton オブジェクトシステムと密接に結びついているので、NewtonScript プログラマーはスロットやスクリプトや Newton オブジェクトシステムのプロトタイプをブックの内容に追加して、ブックの動作をさらにカスタマイズすることが可能である。

Newton ブックメーカー

Newton ブックメーカーによって、プログラマではない人も、完全な機能を持つ Newton ブック のためのソースファイルを通常のワープロと、ブックメーカーコマンド言語を使って作成することができる。

既存の紙の出版物を Newton ブックに変換するという要望は ブックメーカー開発の第一の検討項目となっていた。その結果既存の出版物を、完全な機能を持つ対話型のブックへ非常に少ない労力で変換することができるようになった。先ほど述べた全ての機能 (ブックマークとか、インクテキストとか、Find サービスのサポートとか) を持つシンプルなブックを作るので有ればたった三つの ブックメーカーコマンドが必要なだけである。

ブックメーカーのコマンド言語はまた、より高度なページレイアウトを試したり、ユーザーに追加のサービスを提供するための豊富な機能の一群を提供する。

Newton Toolkit は ブックメーカーアプリケーションが生成する出力ファイルを用いて、Extras Drawer 内に表示されるブックパッケージを作ったり、Newton アプリケーションに組み込まれるヘルプを作ったり、Extras Drawer の Help フォルダに表示されるヘルプブックパッケージを作成する。

Newton ブックリーダー

Newton ブックリーダーは全ての Newton プラットフォームで利用可能である。これはデジタルブックパッケージがシステムに提供されるまでユーザーに見えることはない。ブックパッケージは内蔵 RAM あるいは PCMCIA カード上にロードする事ができる。

Newton ブックリーダーはブック内容のデザインは著者に任せる一方、デジタルブックのルック＆フィールが一貫するように保ちつつ、ナビゲーション用のユーザーインターフェースを提供する。Newton ブックリーダーのユーザーインターフェースに関しては、自分の Newton デバイスに付属のドキュメントを参照のこと。

Newton アプリケーションヘルプ

デジタルブックパッケージ用のソースファイル作成に加えて、ブックメーカーは Newton アプリケーション用のヘルプスクリーン作成に使用することができる。

ユーザーが Assist Drawer の "How Do I?" ボタンをタップすると、システムはヘルプトピックスのオーバービューをアウトライン形式で表示する。トピックをタップするとそのサブトピックを表示したり、より下位レベルのサブトピックがなければヘルプスクリーンを表示する。システム提供のヘルプオーバービューは図 1–1 に示す。

図 1–1　　システム提供のヘルプオーバービュー

ブックメーカーがヘルプを生成するためのソースファイルは ブックリーダーパッケージ生成用のファイルとほとんど同じ手段で書ける：

ブックメーカーコマンドによってタグ付けされたワープロファイルは Newton ブックメーカーによって処理される。両者のもっとも重要な違いは、情報をユーザーに提供するための手段にある。別の言い方をすると、アプリケーションヘルプはユーザーがアプリケーションのヘルプボタン (それは全てのアプリがサポートすべきものだ) をタップしたときに表示されるアウトラインとして、そしてそこに割り当てられた情報として見えるだけである。アプリケーションはシステム提供のヘルプブラウザを使ってヘルプを表示する。

ヘルプブラウザ

ヘルプブラウザは Newton アプリケーションの操作方法をステップバイステップに一つのスクリーンに表示するためのものである。この種の情報の可視範囲は限定されているので、ヘルプブラウザは Newton ブックリーダーの機能のサブセットしか提供しない。このセクションではアプリケーションヘルプとブックパッケージを手短かに対比する。

ヘルプブラウザが提供するスクリーンは ブックリーダーが使用するものよりも小さい。ブックリーダーのブックパッケージは、完全なユーザーマニュアルを表示するのに適している。

情報をブックリーダーパッケージの形で提供するか、アプリケーションヘルプの形で提供するかを決める際に、それらの表示形態と、機能の集合の重要な違いを頭に留めておく必要がある。

ヘルプについてより詳しくは、第 5 章「ヘルプ」で述べる。

ブックとアプリケーション

デジタルブックはアプリケーション同様プロトタイプやテンプレートを組み込むことができる。もし複数ページにまたがる大量のテキストを使用したり、テキストのレイアウトが重要であるようなアプリケーションを作ろうと考えているので有れば、デジタルブックの形でアプリケーションを作ると良い。そうすることで、ブックメーカーが提供する全てのツールが利用可能になるのである。

Chapter 2

はじめの第一歩

この章では Newton ブックの構築過程を足早に見ていくことにする。最初のセクションではブックメーカーの利用に必要なハード・ソフトの簡単な紹介を行う。次に、シンプルなブックメーカー入力ファイルの構成を見る。その後、ブックメーカーと NTK を使って Newton ブックを作る。

ブックメーカーのインストール[1]

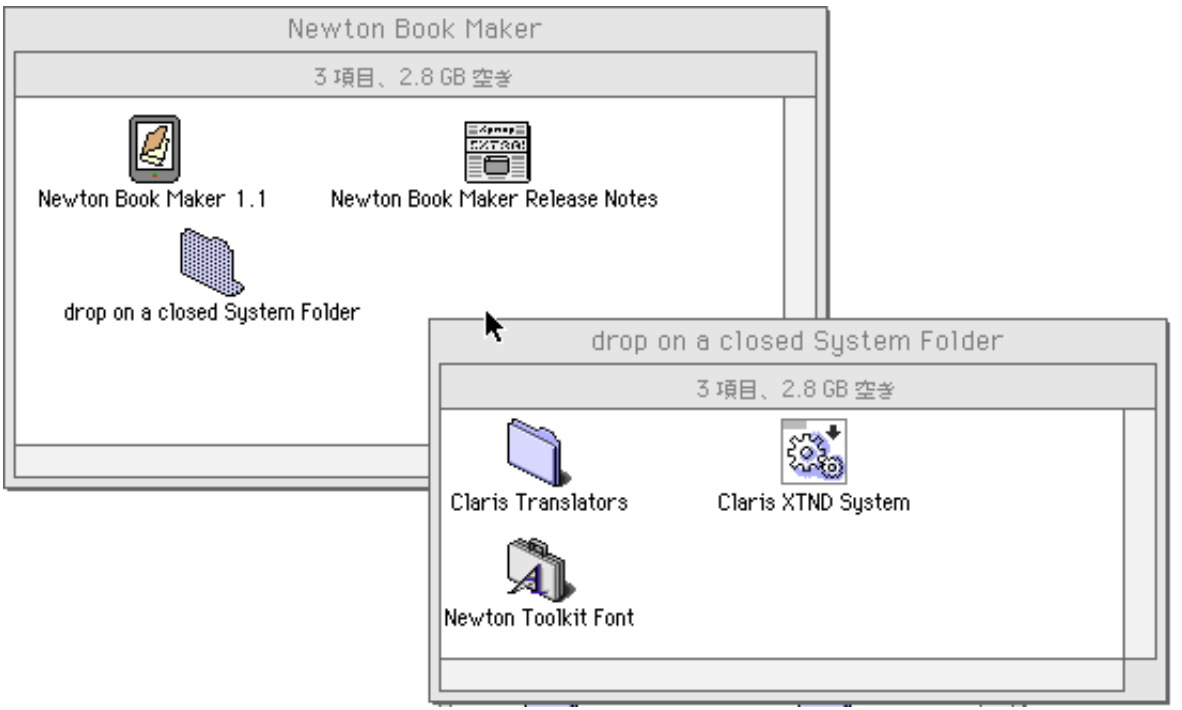
Newton ブックメーカーは Newton, Inc. のホームページ、http://www.newton-inc.com/ から無償ダウンロードできる。現在配布されているものはインストーラを伴わないものである。

また、Newton Toolkit(NTK)[2] もインストールする必要がある。NTK は ブックメーカーが作成したファイルを元に、Newton にダウンロード可能なパッケージを作成するために必要である。NTK のインストール方法は、Newton Toolkit ユーザーズガイドに述べられている。

1. 現在ブックメーカーは http://www.newton-inc.com/ から無償配布されている。そのため、本書に記載のインストール手順は原書と異なっている。
2. NTK も上記サイトから無償ダウンロードできる。

1. ブックメーカーを Macintosh にインストールするには以下の方法で行う。

   ダウンロードしてきたパッケージを開くと、次のようになっている。



   フォルダ "Newton Book Maker" にはパッケージ本体と、リリースノート、そして、システムフォルダにインストールする書類を納めたフォルダ "drop on a closed System Folder" である。このフォルダ内の三つのアイテム (Claris Translators, Claris XTND System, Newton ToolkitFont) を閉じたシステムフォルダにドラッグ＆ドロップすればインストールは完了である。

2 そして Mac を再起動すれば Newton ブックメーカーは機能するはずである。

必要なハードウェア

Newton Toolkit を動かせる Mac なら、Newton ブックメーカーを動かすことができる。より詳しくは、*Newton Toolkit ユーザーズガイド*参照。

必要なシステムソフトウェア

Newton ブックメーカーは MacOS 7.0.1 以降を必要とする。

必要な RAM 容量

Newton ブックメーカーバージョン 1.1 は最低 3700KB の RAM を必要とする。実際に必要な量はブックソースファイルのサイズによって異なる。極端に巨大なブックはさらに多くの RAM を必要とする。小さなものならそれほど必要ではない。ブックメーカーはブックソースファイルを処理している間、そのファイル全体をメモリに格納するに十分な RAM を必要とする。

メモリを節約するには、ファインダの「情報を見る」で、ブックメーカーに割り当てられている RAM 容量を減らしてやればよい。また、`.chain` コマンドを使って大きなソースファイルを小さな複数のファイルに分割しても良い。アプリケーションへの RAM 割り当ての変更について詳しくは、Mac のユーザーズガイドを参照のこと。`.chain` コマンドについては、付録 A「ブックメーカー言語」の「その他のコマンド」セクションで解説されている。

Claris XTND Translators

Newton ブックメーカーは Claris XTND translator 技術に依存している。これは入力として様々な種類のワープロファイルを利用可能にする物だ。Newton ブックメーカーが正常にインストールされると、システムフォルダは Claris フォルダを持つことになり、表 2–1 に見られるような Claris XTND transrator が格納される。必要な他のワープロフォーマット用 XTND トランスレータを追加することもできる。ブックメーカーは、XTND トランスレータがインストールされているあらゆるワープロのファイルを入力として使用できる。

表 2–1　インストールされる XTND トランスレータ

Claris	他
MacWrite® II	TEXT
MacWrite® 5.0	

フォント

Newton ブックメーカーおよび NTK を動かすコンピュータ上には、ブックで使うフォントのビットマップバージョンをインストールしておかないといけない。現在のところ、Newton MessagePad は NewYork, Geneva, Espy Sans, Espy Sans Bold の 9, 10, 12, 14, 18 ポイントのビットマップバージョンのみをサポートしている。

NTK インストーラーは Espy Sans と Espy Sans ボールドの、サポートされている全てのサイズをインストールする。そのコンピュータには、実際にブックで使用する他のフォントもインストールしておかないといけない。

▲ 警告
他の種類のフォント、サポートされているフォントの
TrueType バージョンの使用は不正なページレイアウト計算を
招く。▲

デジタルブックの作成

このセクションでは ブックメーカーコマンド言語と、ブック構築プロセスを紹介する。

ブック作成過程は反復的である。ソース内のどこがページの切れ目になるのかを予測することが難しい。特にブックメーカーコマンドを追加しているときはそうだ。Newton ブックメーカーから NTK を通して初めて Newton 上で校正を行うことができる。校正のあとソースファイルに変更を施すという作業を満足がいくまで何度も繰り返すのだ。

Newton のデジタルブックを作る過程は次の３つである。

1. **コンテントファイルにブックメーカーコマンドを追加する。**

   Newton ブックメーカーを動かそうと思っているコンピュータに XTND トランスレータがインストールされているワープロなら何でも使用できる。ブックメーカーコマンドを含むコンテントファイルはブックソースファイルと呼ばれる。

2 Newton **ブックメーカーでソースファイルを処理する。**

   ブックメーカーは Newton Toolkit の入力となるファイルを生成する。

3 Newton Toolkit **を使い、ブックリーダーパッケージを作ったり、**Newton **アプリケーションにヘルプを追加したりする。**

ブックパッケージの構築については後ほど本章のセクション「NTK でのブックパッケージ構築」で述べる。ヘルプを Newton アプリケーションに統合する方法については、第 5 章「ヘルプ」で述べる。

ブックソースファイルの作成

ブックメーカーのコマンドを含むワープロ文書は**ブックソースファイル**と呼ばれる。ブックソースファイルは Newton ブックメーカーアプリケーションによって処理され、Newton Toolkit 用のファイルを生成する。NTK はその後ブックパッケージを構築したり、アプリケーションにヘルプスクリーンを追加したりする。

この章の全ての情報は ブックリーダーパッケージとアプリケーションヘルプに同様に適用される。

始める前に

ワープロの用紙幅を 3.33 インチに合わせることによって、Newton MessagePad のスクリーンサイズをシミュレートすると便利である。これによって、作品の最終的な画面はどのように見えるのかという見当が付く。しかし、ブックメーカーは、テキストレイアウトの際にソースファイルのレイアウトを使用しないので、ワープロ画面において 3.33 インチ以下のマージンを使うことによってインセットやマージンを作ろうとしてはいけない。また、Newton の画面の端っこからテキストがあふれ出すのではないかなどと心配せずに 3.33 インチの幅を目一杯使えばよい。Newton デバイスは画面の端っことプラスチックケースの間にわずかに隙間を残している。

Newton MessagePad は New York, Geneva の 9, 10, 12, 14, 18 ポイントのビットマップバージョンだけをサポートしている。MessagePad のスクリーン上で正しく表示をしたいのであれば、ソースファイルはこれらのフォントだけを使って書くべきである。また、これら組み込みフォントサイズの倍数ポイント数も使うことができる。

MessagePad はまたシステムフォント Espy Sans も持っており、これは Newton スクリーン上でばっちり表示できるようデザインされている。このフォントはシステムによるテキスト表示のほとんどの部分で使用されている。この

フォントはプリントや fax 向きではないため、プリントあるいは fax を頻繁に行うことを目的としたブックでは、このフォントは使わない方がよい。

ブックメーカーはソースファイルの情報を元に、ブック上での正確な表示が行われるようテキストを再生成する。フォント、サイズ、スタイルの変更に対する監視は特に利用者が意識することなく行われる。結果として単純なブックであればほとんど苦労なく作成することができる。なれてきたところで、より複雑なフォーマットオプションを指定するための ブックメーカーコマンドを追加することができる。次のセクションで示される例では、ブックソースファイルの作成に最低限必要なコマンドを示す。

ブックメーカーコマンドについて

ブックメーカーコマンドは常にピリオド (.)– ドットとも言う – から始まる。そのためコマンドは「ドットコマンド」とも呼ばれる。

ブックメーカーコマンドはつねに行頭から始まる。それぞれのコマンドは、ソースファイル内で次のコマンドが表れるまでの全ての物に対して適用される。ブックメーカーコマンドそれ自身は Newton のスクリーン上には表示されない。

全ての ブックメーカーコマンド言語は、大文字小文字の区別をしない。このブックでは可読性のためブックソースファイルにおいて適切な場所で大文字と小文字を使い分けている。

ほとんどのワープロではユーザー定義のスタイルを使用できる。コマンドだけが別の色で表示されるようなスタイルを定義しておけば、ソースファイルの残りの部分からは、はっきりと目立って表示される。その方法については各自のワープロのマニュアルを参照のこと。

必須コマンドの追加

全てのブックソースファイルは `.title` コマンドと `.isbn` コマンドを含まなければならない。もちろんコンテントアイテムも必要である。コンテントアイテムはスクリーン上に表示されるテキストとか絵といったものである。このセクションではコマンドやコンテントアイテムの追加方法について述べる。

タイトル (title) コマンド

全てのブックソースファイルは `.title` コマンドを一つ含まなければならない。これは Newton のスクリーン上にブックが表示されるとき、その全て

のページのトップに配置されるテキストを定義する。Notitle フラグを使えば各ページのトップにタイトルを置かないようにすることもできる。フラグについてはページ 3–6 からはじまる「フラグ」セクションで述べられ、NoTitle フラグはページ A–55「ドキュメントフラグ」の下に記述される。

.title コマンドはまた (.shortTitle コマンドが指定されていない場合は) Extras Drawer におけるブックの名前も定義する。.shortTitle コマンドについてはページ 2–15「ショートタイトルの追加」参照。ブックのタイトルはまたシステムによる検索内容の表示において使用され、"Searching in The Simple Story..." といった形で表示される。

ブックソースファイルには .title コマンド一つだけを書ける。

.title コマンドをソースファイルに追加するには、ブックソースファイルの先頭にそれを配置し、その行の残りの部分にブックのタイトルを書いて行う。

以下の例では Simplest Story という名のブックから .title コマンドの例を示す :

```
.title The Simplest Story
```

ISBN コマンド

全てのブックソースファイルには .isbn コマンドを一つ含めなければならない。.title コマンドと同じく、一つのブックソースファイルには一つの .isbn コマンドだけが許される。このコマンドはブックパッケージに対して一意な識別子を割り当てる。識別子は Newton ブックリーダーによって使用される。

ISBN は International Standard Book Number の略称である。ISBN は出版社などによって用いられる書籍を識別するための一意番号である。Newton ブックリーダーのために正式の ISBN を使う必要はなく、14 文字以下の一意な識別子を与えればよい。一意な識別子は、自分のデベロッパシグネチャを織り込んで作ることができる。たとえば、Simple1:PIEDTS は有効な識別子である。

.isbn コマンドをソースファイルに追加するには、.title コマンドに続けてそれを配置する。以下の例を参照 :

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

正式な ISBN の取得については、付録 A「ブックメーカー言語」における .isbn の部分を参照のこと。

必須のコンテントアイテム

.title や .isbn コマンドとは別に、ブックソースファイルには最低でも一つ以上のコンテントアイテムがなければならない。画面に表示されるテキストや絵が必要なのだ。以下の例のようにして、.story コマンドを使えばテキストが表示できる：

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

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

注意
.story と .title コマンドはテキストしか処理しない。それらのコマンドにグラフィックを割り当てても無視される。このセクションで後述する .picture コマンドはデジタルブックにグラフィックを追加するために使用される。◆

もっともシンプルなブックソースファイル

もっとも単純なブックメーカーのソースファイルは .title コマンド、.isbn コマンドと一つ以上のコンテントアイテムを含む。これは以下の例のようになる。この例は SimpleStory としてブックメーカーのサンプルに収録されている。本書に含まれるサンプルファイル (このファイルも含む) は、NTK の BookMakerExamples フォルダに格納されている。

```
.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.
```

このサンプルが Newton のスクリーンに表示されるとき、フォントとフォーマットはソースファイルと全く同じになる。物語の最初の一文字目の大文字や段落の改行は維持され、ページ 2–15 の図 2–5 のようになる。ブックメーカーコマンド (ドットコマンド) は Newton スクリーンには表示されないが。

Newton スクリーン上にブックを表示するには、ブックソースファイルからブックパッケージを作るため ブックメーカーと NTK を使う必要がある。次の二つのセクションでこの処理を紹介する。

ブックソースファイルの処理

ブックリーダーブックあるいはアプリケーションヘルプのためのブックソースファイルは Newton ブックメーカーを用いて処理され、NTK 用の入力ファイルとなる。このセクションではブックソースファイルを処理するための Newton ブックメーカーの使い方について述べる。

このセクションに述べられているステップを完了するため、`SimpleStory` ソースファイルを使用できる。ブックメーカーに付いてくるサンプルファイルのどれかを使用するので有れば、練習のために新たなフォルダを作っておくことをお勧めする。そうすることで、うっかりあやまってオリジナルのサンプルファイルを置き換えてしまうようなことがないからだ。

Newton ブックメーカーを使ってブックソースファイルを処理する手順は以下の通り：

1. Newton **ブックメーカーを開く。**

   他の Mac 用アプリケーションと同様、ファインダ内のアイコンを選択してファイルメニューの「開く」を選ぶか、単にアプリケーションファイルをダブルクリックして開くと良い。

2 **ファイルメニューから** Open... **を選択し、ブックソースファイルを選ぶ。**

   ファイルメニューから開くを選択すると、ブックメーカーは処理対象のブックソースファイルを指定するためのダイアログボックスを表示する。

   このダイアログは 図 2–1 に示されている。

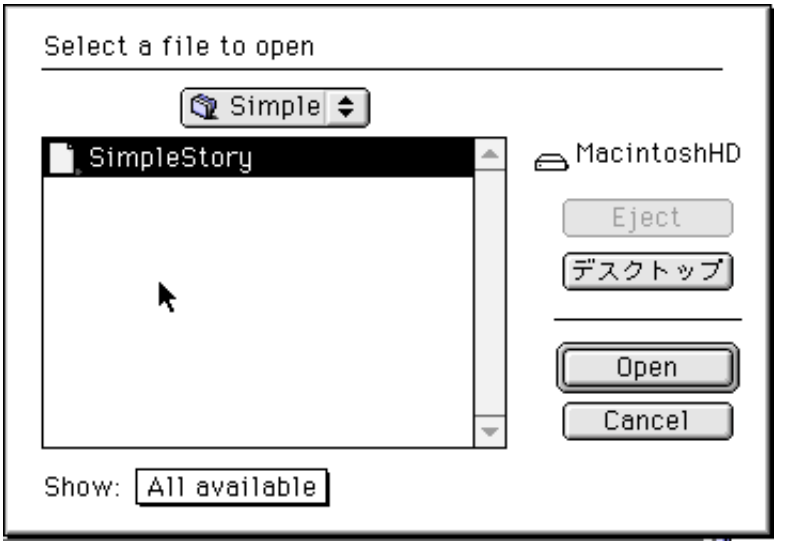
3 **ブックソースファイルを開く。**

ダイアログ中のブックソースファイルを選択し、開くボタンをクリックする。別の方法として、ファイル名をダブルクリックしても良い。図 2–1 には Newton ブックメーカーに含まれる `SimpleStory` ブックソースファイルの選択を図示する。

注意

ファインダ内では、ブックソースファイルを ブックメーカーアプリケーションのアイコンにドラッグするだけで、アプリケーションとソースファイルを開くことができる。◆

図 2–1　　ブックメーカーでのソースファイル選択



ブックメーカーは処理対象のソースファイル名を冠したウィンドウを表示する。図 2–2 には `SimpleStory` ブックソースファイルが開かれたときにブックメーカーが表示するウィンドウを示す。

図 2–2　　ブックの処理ウィンドウ

4 ブックが処理されるスクリーンの形態を選択する。

オプションメニューから、Normal Size を選択すればブックリーダーブックが、Help Size を選択すればアプリケーションヘルプが目的のフォーマットとして指定できる。Options メニュー内で、現在選択されているサイズの横にチェックマークが付く。図 2–3 には Normal Size が選択されているときに表示される Options メニューが示されている。

図 2–3　　ブックリーダー用フォーマットの指定

5 Do It ボタンをクリックしてブックを処理する

ブックメーカーがソースファイルを処理する際、処理中のソースファイル行数、生成されたコンテントアイテム数、指定されたフォーマットにおけるページ数が示される。

ブックメーカーがソースファイルを何らかの理由で処理できなくなると、それは停止してエラーメッセージを表示する。ソースファイルの間違いを直し、再処理する必要がある。

たいていの場合、ブックメーカーはエラーが発生した行数あるいはその一つ前の行数を示す。この情報をクリップボードにコピーしておくこともできる。(Edit メニューの Copy を選択するか、キーボードショートカット CMD-C をたたくかどちらかである)。

行番号は実際には ブックメーカーが処理を停止した段落の番号である。多分、ワープロとブックメーカーの段落の数え方は異なっているだろう。

処理中にコマンドキーを押しながらピリオドを押すと、ブックメーカーの処理を停止できる。

6 ブックメーカー出力ファイルの保存。

ソースファイルの処理が完了すると、ブックメーカーの出力ファイルを保存するための名前を聞くダイアログが表示される。ブックメーカーはデフォルトで、ソースファイル名の後に `.f` をつけた名前を提示してくる。しかし、キーボードを使って好きな名前を指定し、Save ボタンを押

すこともできる。図 2–4 では SimpleStory ソースファイルを処理した後の Save ダイアログボックスを示す。

こうして ブックメーカーの出力ファイルを NTK にかけられるようになった。

図 2–4　　処理されたブックファイルの保存

重要
ブックメーカーによって生成された出力ファイルを編集しないこと。作業の出発点としてサンプルファイルを使い、新たな出力ファイルを作るにはソースファイルの方を変更すること。▲

ブックメーカーは (Apple) スクリプタブルである

前述ステップはアップルスクリプトで自動化できる。アップルスクリプトが使用できるなら、ソースファイルを開き出力ファイルを保存する行程を自動化できる。たとえば以下のスクリプトによって "MyDisk" ディスク内の "todo" フォルダ内の "A Superb Story" を処理し、"A Superb Story.f" として保存することができる。

```
tell application "Newton BookMaker"
    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
```

ブックにするか、アプリケーションヘルプにするか？

次にすべきことは、ブックリーダー用ブックを作るか、アプリケーションヘルプを作るかで変わってくる。

Normal Size オプションを選択してブックソースファイルを処理した場合、NTK を使って Extras Drawer に表示されるブックリーダーパッケージを作ることになる。次のセクション「NTK でのブックパッケージ構築」では、ブックリーダーパッケージの構築方法を説明する。

Help Size オプションを有効にして処理した場合、このファイルのコンテンツはアプリケーションに統合される必要がある。だから、アプリケーションはヘルプフレームを直接使うことができる。Newton アプリケーションにおけるヘルプの統合は第 5 章「ヘルプ」で後ほど述べる。

NTK でのブックパッケージ構築

ブックリーダーパッケージは Newton Connection あるいは Newton Toolkit を使用して Newton プラットフォームにダウンロード可能である。ブックパッケージは他のアプリケーションパッケージと同様、Extras Drawer にアイコンの形で表示される。

`SimpleStory.f` と `Simple` という名前の NTK プロジェクトを使い、以下のステップに沿ってブックパッケージの構築を練習することができる。

注意
これらの方法は、読者が NTK に通じていることを前提にしている。より詳しい情報については、*Newton Toolkit ユーザーズガイド*を参照。◆

以下のステップを使い、NTK でブックリーダーパッケージを作ってみよう：

1. NTK **で新しいプロジェクトを開く**

   ブックパッケージ構築の練習をしたいのなら、`Simple` プロジェクトを使って `SimpleStory.f` ファイルから構築してもよい。

2 **ブックメーカーファイルを** Project **メニューの** Add File... **コマンドを使ってプロジェクトに追加する**

   プロジェクトからファイルを取り除くには、Project メニューの Remove File... を使用する。

3 Project **メニューの** Output Settings... **コマンドを使用し、**Book **への出力をセットし、**NTK **がデフォルトで提供するパッケージの名前、アプリケーションのシンボルを変更する。**

   サンプルブックプロジェクトを使っている場合は、これらの項目はあらかじめ設定されている。

4 **他の** NTK **プロジェクトと同様、プロジェクトを構築し、それをダウンロードする**

   ブックパッケージが Extras Drawer に表示される。

Newton MessagePad 上でブックパッケージを開くと、まるでソースファイルのように見えるだろう。例を図 2–5 に示す。

図 2–5　サンプルブック

ショートタイトルの追加

The Simplest Story のような長いタイトルは Extras Drawer ではちゃんと表示されない。.stortTitle コマンドを使って他のタイトルを付けて、Extras Drawer でのみ表示される名前を指定することができる。その場合でも .title コマンドで指定したタイトルはページのトップに表示される。

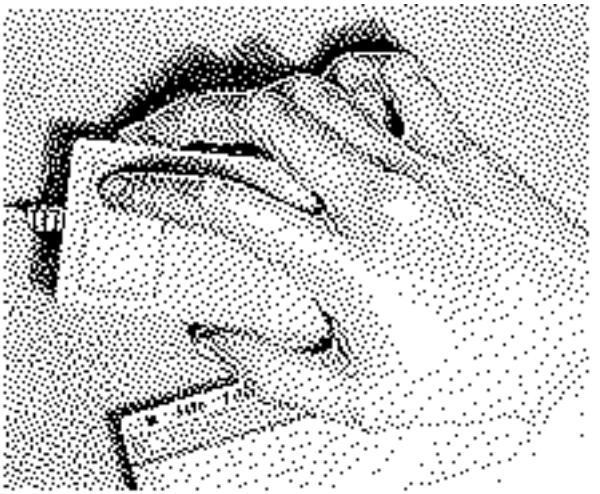
.shortTitle コマンドを使うには、ブックソースファイルの最初の方にそれを追加する。その行の後の部分に、Extras Drawer での表示用のタイトル文字列を指定する。以下の例では、短いタイトルである Pix が Extras Drawer 内で表示され、ブックの各ページのトップに表示されるタイトルは The Picture Story となる。

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

グラフィックの追加

ブックリーダーブックあるいはアプリケーションヘルプにグラフィックを追加するのは簡単で、ブックソースファイルに `.picture` コマンドを追加し、その後に絵をペーストするだけでよい。たとえば、前出のサンプルにおいて、ファイルの最後に `.picture` コマンドを置き、その後に絵を張り付ける。ソースファイルは以下の例のようになる。これは PictureStory サンプルからとった物である：

```
.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
```



▲ 警告
ソースファイル中で使用される絵は `'PICT'` フォーマットでなければならない。Newton MessagePad は `'PICT2'` リソースは全く扱えない。`'PICT2'` リソースを含むブックソースファイルは正常にコンパイルされるが、Newton デバイスの画面上には表示されない。▲

'PICT' リソースにグラフィックを変換する簡単な方法は、HyperCard のどこかの画面にペーストすることである。そうすればあらゆるグラフィックは自動的にこのフォーマットに変換される。その後、そのグラフィックを HyperCard からブックソースファイルにペーストすればよい。

大きな絵の自動スクロール

Newton ブックリーダーは Newton のスクリーンサイズを超える絵を制御するためのスクローラーを自動的に表示する。このスクローラーは図 2–6 に示すような物である。スクローラーは NoScroller フラグを指定して、表示しないようにすることもできる。フラグについてはページ 3–6 からはじまる「フラグ」セクションで解説される。NoScroller フラグはページ A–31 からはじまる「コンテントフラグ」で解説される。

図 2–6　　スクローラー

BigPictureStory ファイルと BigPicture.pkg ブックパッケージは Newton ブックリーダーのこの機能を示す物である。

注意
アプリケーションヘルプのスクリーンはスクロールしない。
◆

ブックソースファイルへのコメント追加

ブックソースファイルが徐々に複雑になると、コメントを追加したくなってくるだろう。コメントはソースファイルには表示されるがブックには表示されないテキスト行である。

コメントの前にドットとシャープ (#) をつけておくと、ブックメーカーはソースファイル中のその行を無視する。以下に例を示す :

.# This is a comment. ブックメーカー ignores it.

ブックメーカーと共に配布されているサンプルブックのソースファイルには、コメントの例がたくさん入っている。

次に行くべき場所

ここまででブック構築プロセスの基本は理解できた。これからは本書の中で、自分の目標や作業スタイルにもっとも適した素材に着目すればよい。

- 第 3 章「ブックメーカー言語の使用」では、ブックメーカーコマンド言語の概要を見ることができる。この章では興味深いページデザインを作るためのレイアウトとフラグの使用方法や、ユーザーをうまく導くためのブラウザやキオスクの作成方法等を提供する。NewtonScript の知識を必要としない ブックメーカーコマンド言語の全ての面について、この章と付録 A「ブックメーカー言語」で述べる。
- 第 4 章「NewtonScript」では、Newton ブックにおいて NewtonScript や Newton オブジェクトシステムプロトタイプの使用方法に関する情報を述べる。
- 第 5 章「ヘルプ」では、Newton アプリケーションにヘルプ画面を追加するための ブックメーカーと NTK の使用方法について述べる。
- 自分自身で　ブックメーカーを使って経験を得ることができる。ブックメーカー言語内の全てのコマンドとフラグは付録 A「ブックメーカー言語」において説明される。しかし、自分でブックやアプリケーションへ

ループを作る前に第 3 章「ブックメーカー言語の使用」を読むことが強く推奨される。

空白ページ

ブックメーカー言語の使用

すでに簡単なブックのソースファイルを作り、Newton Book パッケージをそこから作成する方法について見てきた。本章ではテキストとグラフィックをページ上に配置し、ナビゲーションツール、ブラウザ、キオスクをデジタルブックに提供するための ブックメーカーコマンドについて述べる。本章では、すでに登場のものも一緒に、NewtonScript の知識を必要としない ブックメーカーコマンド言語全体について概説する。

レイアウトの使用

`.layout` キーワードで始まる ブックメーカーコマンドは**レイアウトコマンド**あるいはレイアウトと呼ばれる。**レイアウト**はテキストとグラフィックのページ上での配置を指定する。

レイアウトコマンドの使用はオプショナルである。レイアウトコマンドをブック内で使用しない場合、ブックメーカーはスクリーンの幅全体をコンテントアイテムの配置に使用する。たとえば、前章で登場の *The Simplest Story* ブックは、デフォルトのフォーマットを使用している。

レイアウトコマンドの定義

ブックソースファイル作成中、必要に応じてレイアウトコマンドを定義することができる。あるいは、ブックソースファイルの最初のところで全てのレイアウトを定義しておくこともできる。どのようなアプローチをとろうとそれは全く個人の好みの問題であるが、レイアウトを最初の方にまとめておけ

ば、ブック内のレイアウトを変更したくなったときにやりやすい。複数のブックが使用するレイアウトのライブラリを構築したくなった場合、後者の方法を採っておけば、新たなブックソースファイルにレイアウトをコピー&ペーストするのがより簡単になる。

.layout コマンドには定義するフォーマットの名前を指定する必要がある。そうするとそのフォーマットを名前によって参照できる。レイアウトコマンドに名前を与える場合、.layout キーワードのすぐ後に名前を置く。それぞれの .layout に異なる名前を与えることが重要である。.layout の名前に同じものが出てくると、エラーになるからである。

.layout コマンドはまた、カラムの幅の指定も必要とする。Newton ブックリーダーはページを等幅の垂直な帯に 12 等分する。カラムの幅は、この帯の数だけ指定される。

以下の例では simple という名前のレイアウトを作成する。このレイアウトは一個のカラムを定義し、そのカラムの幅は帯 12 個分、つまりページの幅全体である。

このようにしてネーミングされたレイアウトは、別のページフォーマットを使った後、デフォルトのページフォーマットに戻るとき等に便利である。

```
.layout simple 12
```

次の例は threeCol というレイアウトを作成する。このレイアウトはページを、それぞれが帯 4 個分の幅を持つ、等幅な 3 つのカラムに分割する :

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

実際には、使用可能なカラムの数は、テキスト表示に使用されるフォントサイズ及び、それを表示するスクリーンの物理的サイズにより決定される。たいていのブックでは、MessagePad のスクリーンを 2, 3 カラムより多く分割しない方が良い結果が得られる。

レイアウトの適用

ブックメーカーが .layout コマンドに行き当たると、ブックソースファイル中の後続するコンテントアイテム (ストーリー、絵、等々) に、そのコマンドで指定されたフォーマットを適用し続け、これは、ブックメーカーが別のレイアウトコマンドに行き当たるまで続く。

レイアウトコマンドをそれが影響を与えるコンテントアイテムの前に配置することが、レイアウトコマンドのもっとも簡単な使用方法である。Newton スクリーン上の一つのページに対しては一つのレイアウトだけしか指定できないことに注意。レイアウトを変更した後の最初のコンテントアイテムは、新しいページに配置される。(つまり、レイアウトを変更すると改ページされる)

以下の `LayoutStory` サンプルブックソースファイルからとってきたサンプルでは、レイアウトを適用するもっとも簡単な方法を示す：

注意
ブックメーカーに付属のサンプルブックソースファイルは全て推奨ページ幅 3.33 インチを使用している。しかし、このガイドで示すブックソースファイルリストでは、スペースの節約のため、ページ幅全体を使っている。◆

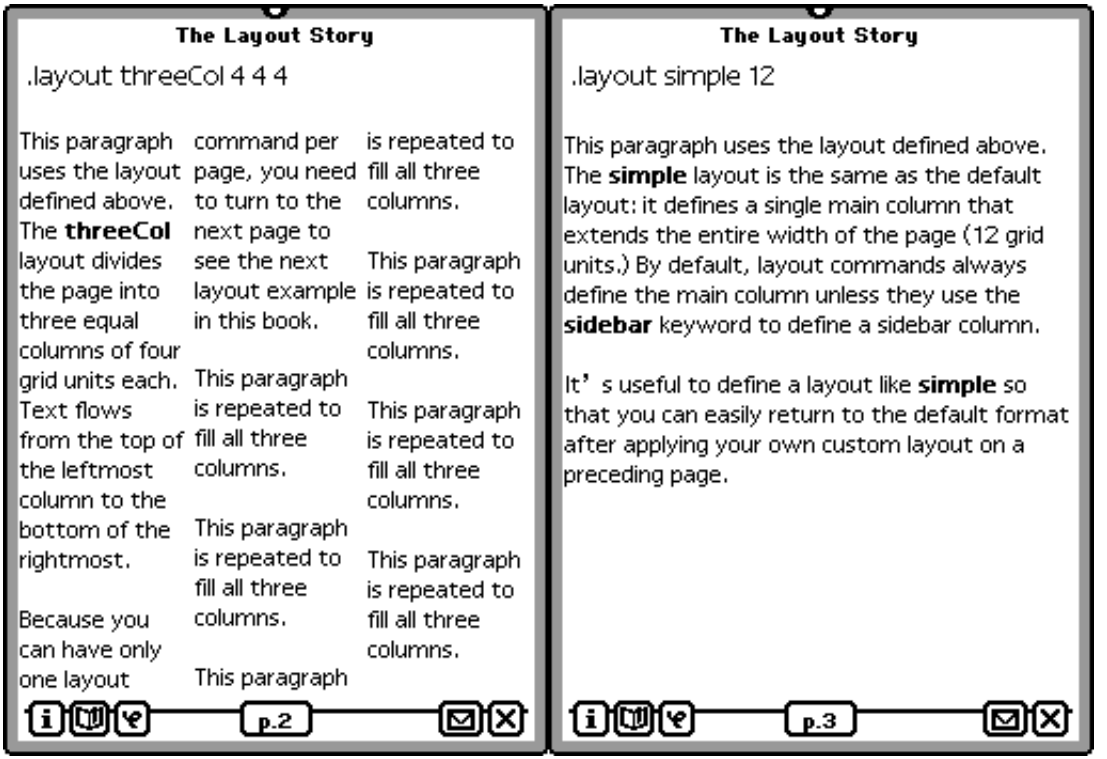
最初のレイアウト `threeCol` は 図3–1 に見られるようなページを作成する。次のレイアウト `simple` によって、(本書のここまで紹介してきたサンプルブック全てに見られるような) デフォルトページフォーマットに戻る。

```
.# 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.

図 3–1 `threeCol` と `simple` **レイアウトでフォーマットされたページ**



ちなみに 図 3–1 に示された「レイアウトコマンド」は実際のレイアウトコマンドではない。しかし、ストーリーテキストがこれらのページに効果を与えるレイアウトを示している。それらのコマンドを表示する行はドットではなく空白で始まっているので、ブックメーカーはそれをコマンドだとは思わないのである。実際にそれらのフォーマットを実装するレイアウトコマンドはブックソースファイル内にあるが、それらは Newton スクリーンに表示されることはない。

そうした行には空白が特別に追加されているので、テキストはサンプルブック表示される。このテクニックは共通のミスを引き起こす : もしブックソースファイル内のコマンドがちゃんと働かない場合、行がドットで始まっているか、スペースで始まっていないかということを確認すると良い。

名前によるレイアウトの適用

ここまで紹介してきたブックソースファイルでは、レイアウトの定義はそれを使用するコンテントアイテムの直前で行っていた。一度レイアウトを定義してしまえば、後続するコンテントアイテムに対して `layout=` キーワードを使えば、レイアウトをその名前によって適用できる。以下の例ではレイアウトを定義し、その後名前によってコンテントアイテムにレイアウトを適用している :

```
.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.
```

`layout=`*layoutName* が指定されていないコンテントアイテムは、デフォルトのレイアウトを使用することに注意。デフォルトのレイアウトは、一番最後に `.layout` コマンドで指定されたレイアウトである。ファイルの先頭におけるレイアウト定義群の一番最後にデフォルトレイアウトを定義すれば、`layout=` コマンドはデフォルトレイアウト以外のものを使うときだけに使用すればよい。

どのページにもただ一つのレイアウトだけを適用できることに注意。レイアウトを変更するたび、新たなレイアウトを持つ最初のコンテントアイテムは新しいページに配置される。

名前によるレイアウトの適用は有用なテクニックだが、本書で紹介される短いサンプルではそれは使わなくてよい。名前付きレイアウトのよりよい使用のアイデアについては、ブックメーカーに付属の `MoreBrowsingStory` と `KioskStory` ファイルを参照のこと。

フラグ

フラグは、ブックメーカーコマンドに追加して様々なオプションを指定するための予約語である。フラグを単独で指定しても良いし、もっと凝った効果を得るため複数組み合わせて使っても良い。

レイアウトあるいは文書全体を変更するためだけに使用される一握りのフラグがある。そうしたフラグはそれぞれ**レイアウトフラグ**及び**ドキュメントフラグ**と呼ばれる。コンテントアイテムの表示を変更するフラグは**コンテントフラグ**と呼ばれる。ブックメーカー言語におけるほとんどのフラグはコンテントフラグである。

このセクションではフラグの使用の概要を手短に示す。ブックメーカーコマンド言語において利用可能なフラグ全ての完全なリストについては付録 A「ブックメーカー言語」の「フラグ」セクションを参照のこと。

フラグの使用

フラグはコマンドの後に追加して使用する。たとえば、`edges` フラグを使ってコンテントアイテムの周囲に境界線を引くことができる。`FlagStory` サンプルブックソースファイルから一部をお目にかける：

```
.# 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.
```

個々のコンテントアイテムに対して定義されたフラグの効果は、ブックメーカーが `.story` や `.picture` と言った新たなコンテントコマンドに出会うまで持続する。

Edge フラグの使用

`edge` フラグはコンテントアイテムを線、箱、角丸の箱で修飾するためのフラグのセットの一つである。以下の例のように、`rounded` や `edgeWidth` といった `edge` フラグを追加することによって、さらにコンテントアイテムを変化させることができる。

.# draw a box with rounded corners around the story text
.story edges edgeWidth=4 round
Adding the **rounded** flag causes BookMaker 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.

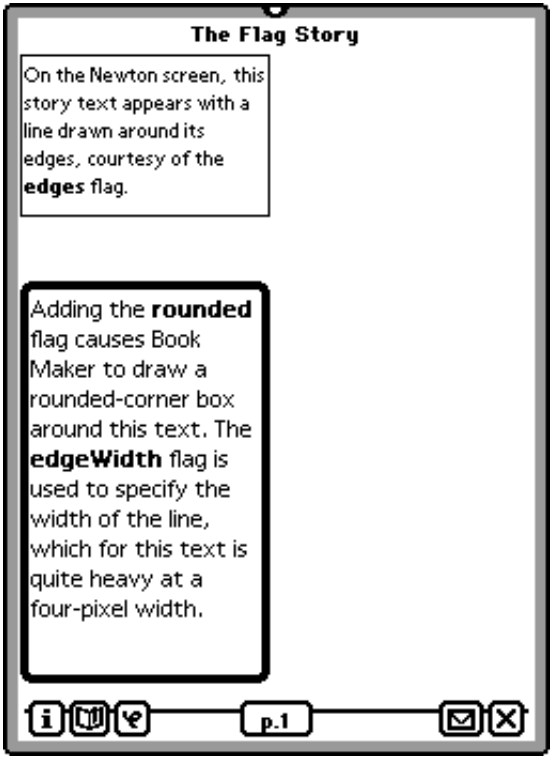
rounded フラグは角丸の箱を指定している。オプションの edgeWidth フラグを指定すれば、edge フラグによって描画される線の幅をピクセル単位に指定することができる。修飾対象となる edge フラグと同じラインに、edgeWidth フラグを配置すること。

edgeCommand edgewidth=*number*

ブックメーカーと NTK を通して処理すれば、これらのサンプルは 図 3–2 にのようになる。

利用可能な edge フラグの完全なリストについては、付録 A「ブックメーカー言語」の「エッジフラグ」セクションを参照。

図 3–2　　edge フラグの使用



SideBar フラグの使用

`sidebar` フラグはコンテントコマンドと同様レイアウトコマンドを修飾するためのものである。

レイアウトコマンド内で使用されるとき、`sidebar` フラグはカラムを左右どちらかの端に定義する。`sidebar` カラムはマージン付きテキスト、懸垂インデント、その他の効果を提供できる。レイアウトコマンドにおいて、`sidebar` フラグに先行する数値がサイドバーカラムの幅を定義する。同様に、`main` フラグに先行する数値はメインカラムの幅を定義する。

たとえば、以降のコマンドは二つのカラムを含む `leftSide` という名のレイアウトを定義している。サイドバーカラムは帯 4 つ分の幅 (ページ幅全体の 1/3) で、左端から始まっている。メインカラムは 8 カラム分の幅で、ページの残りを埋める :

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

サイドバーとメインカラムの配置関係を明瞭にするため、これらの識別子は登場している。だから、メインカラムの右側にサイドバーのあるレイアウトは以下の例のように定義できる：

.layout rightSide 8 4 sidebar

メインカラムは常にどのレイアウトでも定義されるため、レイアウトコマンド内での `main` フラグの使用はオプショナルである。どのカラムがメインカラムなのか明示しておきたい場合には記述して結構である。

`sidebar` フラグがコンテントコマンドに表れた場合、それはコンテントアイテムが現在定義されているサイドバーカラムに配置されることを指定する。だから、テキストやグラフィックをサイドバーに配置するには、サイドバーに配置したいコンテントアイテムに関連する `.story` あるいは `.picture` コマンドに `sidebar` 予約語を追加すればよいのである。

`sidebar` 予約語の指定されていないコンテントアイテムはデフォルトでメインカラムに配置される。

`FlagStory` サンプルブックソースファイルからとられた以下の例では、`sidebar` フラグをレイアウトとコンテントコマンドで使用している：

.# 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

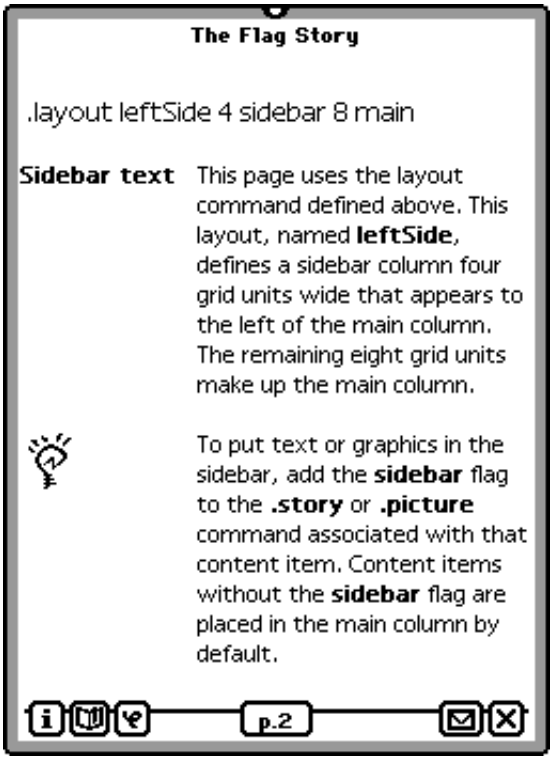
.story

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.

上記コードにより生成されたブックのページは 図 3–3 の様な体裁である。

同様の `sidebar` カラムが右側に定義された例は、`FlagStory` ブックソースファイル及び `Flags.pkg` サンプルブックに示される。

図 3–3　　レイアウトとコンテントアイテムにおける `sidebar` フラグの使用



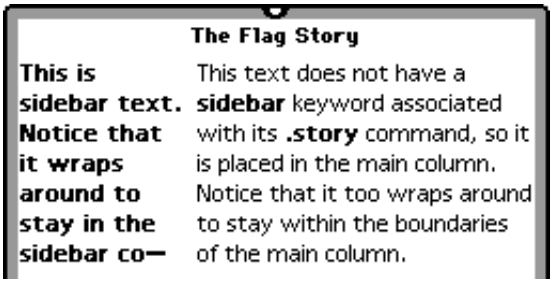
またしても `FlagsStory` からとってきた以下の例では、4 単位のサイドバーと 8 単位のメインカラムを生成する `twoCol` という名のレイアウトを定義する：

```
.# 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.
```

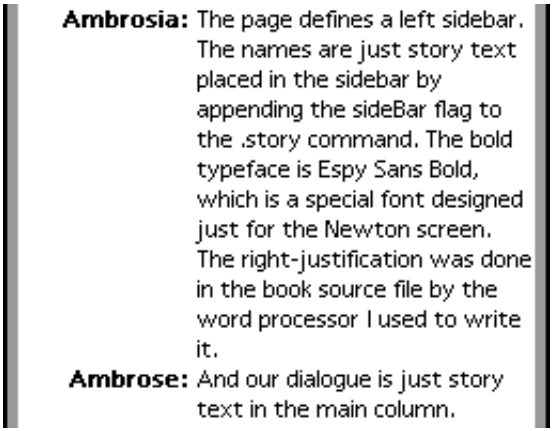
コンテントコマンドに `sideBar` フラグを追加することにより、そこに関連しているコンテントアイテムはサイドバーカラムに配置される。どちらのカラムのテキストもカラムの範囲内で自動的にワードラップされる。先の例におけるテキストは Newton スクリーン上には図 3–4 のように表示される。

図 3–4　　`twoCol` レイアウトでフォーマットされたテキスト



ブックメーカーはブックソースファイル内のフォント、スタイル、行揃え情報を使用する。たとえば、図 3–5 に見られるスクリーンはメインカラムの左側にあるサイドバー内で右寄せされたテキストを使用する。

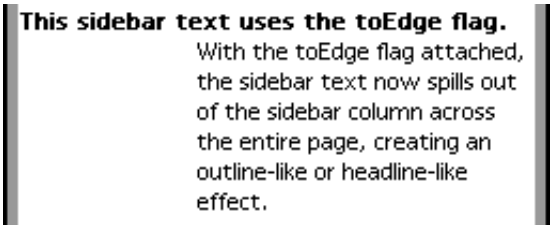
図 3–5　メインカラムの左側のサイドバー内で右寄せされたテキスト



toEdge フラグの使用

サイドバーテキストに対して、`toEdge` コンテントフラグを使用すると、図 3–6 のように、懸垂型やアウトライン的な効果を作り出せる：

図 3–6　サイドバーテキストへの `toEdge` フラグの適用



ブックメーカーはグラフィックへのテキストの回り込みはサポートしないが、`toEdge` フラグをうまく使えば同様の効果が得られる。以下のブックソース例は `FlagStory` ブックソースファイルからとってきたものである。`sideby` という名のレイアウトは 6 単位のサイドバーと 6 単位のメインカラムを作成し、効果的にスクリーンを分割して、`sidebar` フラグを特定のコンテントアイテムに適用することでアイテムの配置をコントロールできるようにしている：

```
.layout sideby 6 6 sidebar
.story toEdge
```

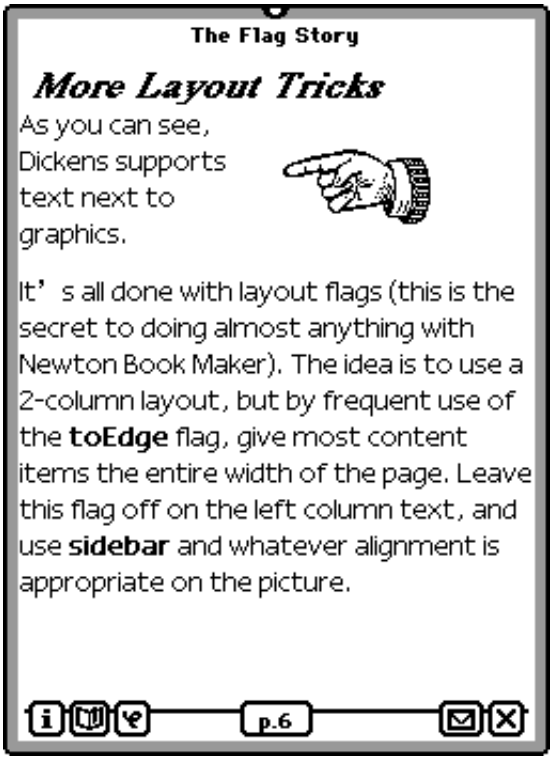
More Layout Tricks

.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.

ブックメーカーと NTK による処理の後、画面表示は 図 3–7 のようになる。

図 3–7　　グラフィックへのテキスト回り込み



sidebar 位置揃えフラグの使用

前出の例で、絵に対する `alignCenter` フラグの使用を見た。`alignCenter` はサイドバーカラムにおける垂直位置揃えを変更するために使用できるいくつかのコンテントフラグの内の一つである。こうしたフラグ、`alignTop, alignCenter, alignBottom` は、ブックソースファイル内の一つ前のコンテントアイテムに対して該当コンテントアイテムをサイドバー内で位置づける。

たとえば、前出の例では `alignCenter` フラグを絵につけることで、メインカラム内のストーリーテキストの垂直方向の中央に、アイテムをサイドバー内で揃えた。

以下の例では `alignTop` フラグを使用してメインカラムのテキストのトップに絵をサイドバー内で揃える。

```
.layout bedtimeStory 3 sidebar 9
```

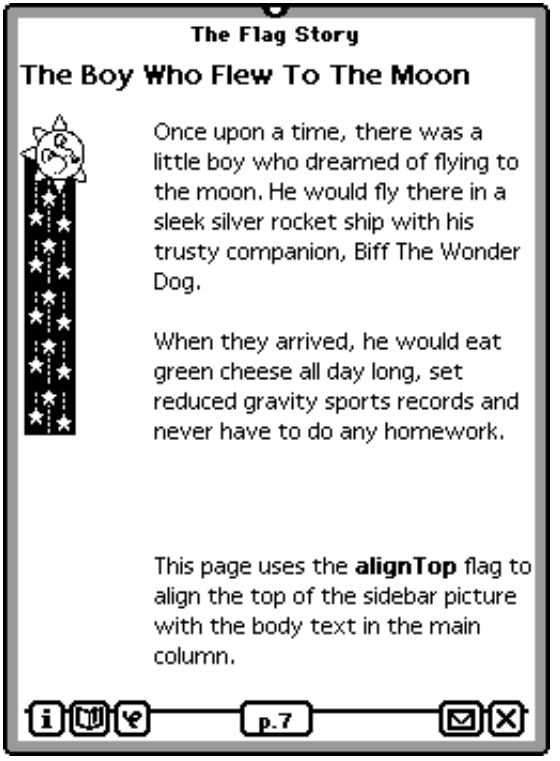
```
.# put this title in the sidebar and spill its
.# contents out of the sidebar column
.story Sidebar toEdge
The Boy Who Flew To The Moon

.# 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.
```

ブックメーカーと NTK での処理後、前出の例は 図 3–8 の様になる。

`alignBottom` フラグは直前のコンテントアイテムの底辺にサイドバーを揃える。以下の例では、`alignBottom` フラグの使用を示しており、これは前出の例に基づいている。ブックソースファイルの関連する部分だけが示されている。

図 3–8　　alignTop フラグの使用



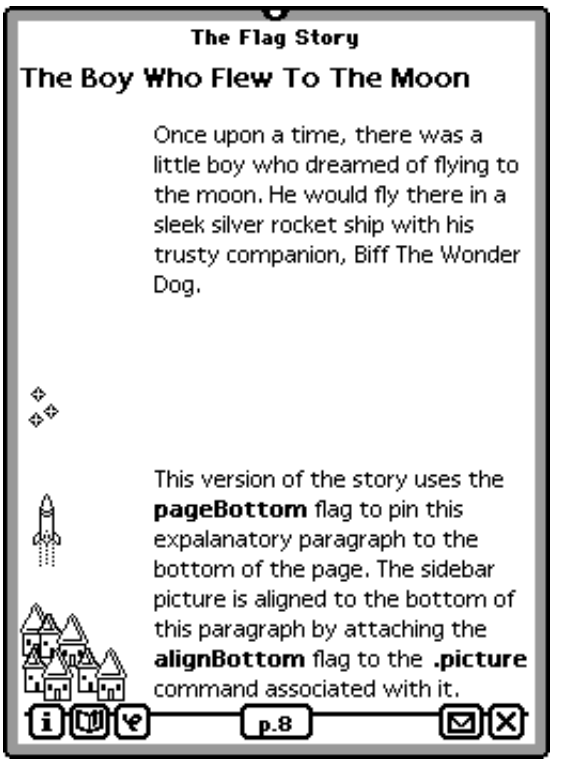
.story pageBottom
This version of the story uses the **pageBottom** flag to pin this explanatory paragraph to the bottom of the page. The sidebar picture is aligned to the bottom of this paragraph by attaching the **alignBottom** flag to the **.picture** command associated with it.

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

```
.picture sidebar alignBottom
```

このサンプルコードはサイドバーの絵と最後の段落を、図 3–9 に示すように生成する。

図 3–9　　alignBottom フラグの使用



ブックメーカー言語は他のコンテントフラグもいくつか用意している。これらは付録 A「ブックメーカー言語」に示される。

フォーマット上の推奨

ブックメーカーから最良の結果を得るには、ブックソースファイル作成時に以下の推奨点を覚えておかれたい：

- テキストの位置を揃えるには、スペースではなくタブを使用する。(ただし、MessagePad は左揃えのタブしかサポートしないことに注意)。できるだけタブストップを少な目にすることで、スペースを節約することができる (あるいは、タブが必要なときだけそれを使用するようにすれば)。それぞれのコンテントアイテムには一組のタブの集合だけを設定することができる。そして、タブはコンテントアイテムの最初の段落で設定されなければならない。実際には最初の段落にタブが必要でなくとも、後続の段落がタブを使うので有れば、タブの設定が必要である。
- 手作業で(行の終わりに "-" をタイプして)ハイフネーションを行わない。MessagePad 上にはテキストが正しく表示されるかもしれないが、別のもっと広いスクリーン上では正しく表示されないだろうし、ハイフネーションされた単語があると、ユーザーが Find サービスを使って検索するときのじゃまになる。
- ストーリーテキストにコマンドを割り込ませない。そういうことをすると、スタイル情報が消滅してしまう。そうする代わりに、ストーリーテキストの最初または最後にストーリーに関連するコマンドを配置すると良い。
- 読者のワードプロセッサが上付き下付き文字をサポートしている場合、ブックのソースファイルに特別なことをする必要はない。MessagePad は上付き下付の文字を表示するため、自動的に小さなポイントサイズの文字を使用する。しかし、特殊なテキストスタイルを使うと、期待しない結果となることもある。ブックメーカーによって使用される XTND フィルタはスモールキャップ等の特殊なスタイルをサポートはしない。
- 将来の Newton プラットフォームでは、ブックリーダーは大型スクリーンサイズを有効に利用するためページを自動的に再フォーマットする可能性がある。だから、本来次のページのストーリーと関連づけられるはずのタイトルが前のページの末尾に来たりする可能性もある。タイトルに `StartPage` フラグを使うのではなく、ストーリーに `KeepWith` フラグを使った方がよい。(これらのフラグについては、付録 A「ブックメーカー言語」で説明されている)。しかし、より小さいスクリーンでは、これら二つのアプローチは同じ結果を生じる。`StartPage` フラグ

の利用は将来の Newton デバイスにおいて不適切なページ送りを生じる可能性がある。

ブラウザペーンの生成

MessagePad のボタンバー上の真ん中にあるオーバービューボタンをタップすると、メインのブラウザが表示される。ブラウザペーンはフローティングビューで、最初に表示されたときはトップレベルのヘッドラインを表示し、そこから下位のサブヘッダや、セクションの最初の部分を展開したりできる。ブラウザは本の目次のような物だ。

ブックメーカーでは一つのブックに複数のブラウザを定義できる。その場合、ブラウザの名前はメインブラウザにおいてリスト表示される。ブラウザの名前をタップすれば、そのブラウザが表示される。これは、紙の本における章ごとの目次とよく似たものである。

ブラウザ内に表れるアイテムは、コマンド `.subject` や `.chapter` によってタグ付けされたブック内のテキストアイテムである。ブラウザはこれら二つのコマンドでタグ付けされたものを一行づつ単一のスタイルで表示する。フォントサイズとスタイルは ブックリーダーによってあらかじめ定義されている。だから、ブック本体のテキストに対して異なるレイアウトを使用しても、ブラウザ内のアイテムに影響することはない。

`.subject` コマンド行内のテキストは、いくつかのことを ブックメーカーに伝える。まず、このテキストは新たなコンテントアイテムの開始であることを意味し、このテキストが直後のストーリーや絵のためのタイトルテキストであることを示す。次に、これはそのサブジェクトテキストがブラウザのどのレベルに表れるかを示す。このレベルは `.subject` コマンドに後続する数値によって指定される。ブックリーダーは階層的な目次を作成するため、このサブジェクトレベルを保ってブラウザアイテムを自動的にランクづけする。

レベル引数に後続するのは、サブジェクトテキスト表示用レイアウトを指定するためのオプションフラグである。レイアウトは、他のコンテントアイテムに対して適用されるのと同様にブラウザアイテムにも適用されるが、ブラウザ内のテキストには全く影響しない。ブラウザはレベル 1 のサブジェクトを自動的にボールド体で表示する。他のアイテムは通常のテキストで表示される。

これらの引数に続くのはオプショナルな引数である。name= 引数によってこのサブジェクトの名前を指定できる。この名前は例えば、キオスクのような他の要素がこのサブジェクトを発見するときに使う。bro= コマンドによって、このサブジェクトをメインブラウザ以外のどのブラウザに表示するかを指定できる。

.chapter コマンドは .subject 1 コマンドと同義である。.chapter と .subject コマンドのパラメタに関する完全な論議は、付録 A「ブックメーカー言語」の「コンテントコマンド」セクションに含まれる。

BrowserStory サンプルブックソースファイルから取り出した以下の例は、二つのブラウザアイテムを作成する。最初の .subject コマンドはテキスト *The Kids Who Flew To The Moon* をページの中央に配置する (このコマンドは実際にはこのブックのタイトルページを作成する)、そしてそれをブラウザペーンのトップレベルに追加する。次の .subject コマンドはテキスト *The Boy* をブラウザペーンの第二レベルに配置する。ブックリーダーのブラウザはこのサブジェクトラインを第一レベルのブラウザアイテムのサブヘディングとする。

```
.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
```

上記の例がコンパイルされると、ブラウザは 図 3–10 の様になる。

図 3–10　　BrowserStory のブラウザペーン

BrowserOnly フラグ

.subject あるいは .chapter コマンドに browserOnly フラグを付加すると、これらのコマンドに関連づけられたテキストはブラウザペーンにのみ表示され、ブック本体には表れない。このフラグはブック本体に表れないブラウザエントリを配置するのに有用である。

たとえば、タイトルページを少しばかり凝った物にしたい場合、テキストに改行を追加することによってページを埋め尽くすようにする。先の例を少しばかり変形したのが下の例である。残念なことにブックメーカーは .subject の次に来る最初の行だけをブラウザエントリ作成に使用する。だから、次の例では *The Kids Who* がブラウザペーンのトップレベルに配置されてしまう。

```
.subject 1 Centered
```

The Kids Who Flew To The Moon

以下の例では、.subject 1 に browserOnly フラグを使うことにより、ブラウザにしか表示されないタイトルページのリストを作成し、この問題を解決している。タイトルページ自信は .story コマンドによって表示される。.story コマンドはブラウザエントリを作らないので、ブラウザにはタイトルページ用のエントリが一つだけしか作られない。

```
.subject 1 browser only
The Kids Who Flew To The Moon
.story Centered
```

The Kids Who Flew To The Moon

さらにブラウザエントリの作成の例を見たければ、MoreBrowsingStory のブックソースファイルを見ると良い。

キオスクの作成

現実の世界では、キオスクは広告を掲げる円柱状の構造物である；Newton ブックにおいては、キオスクはブック内のサブジェクトに対する「広告」を含むナビゲーションページである。ブックのキオスク内のアイテムをタップすることにより、目的の物が表れる。

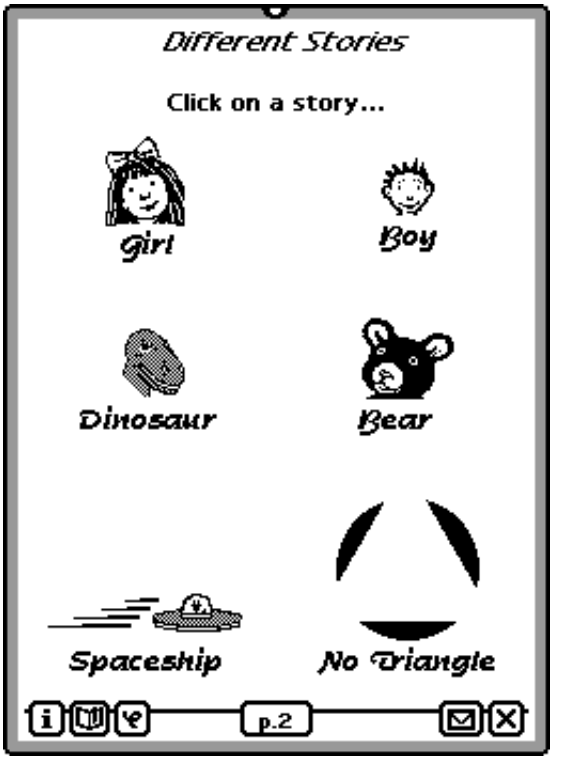
キオスクはまたユーザーが簡単に戻ってこれる場所を提供する。ブックマークダイアログボックスには、最寄りのキオスクへユーザーを導くボタンが常に表示されている。

ユーザーをナビゲートしやすいコンテンツを作るのに加えて、キオスクページはブックに視覚的なおもしろさを与える。図 3–11 に見られるように、キオスクはちょっと変わったレイアウト内にグラフィカルなアイテムを使用できる。

ここで示されるキオスクページはページ上に絵を配置するためのカスタムレイアウトを使用している。キオスクページのレイアウト定義は `kiosk` フラグを含んでないといけない。以下の例で定義されるレイアウト、`MyKiosk` は 6 単位のサイドバーを持つ 6 単位のメインカラムを生成する。`main` キーワードの使用はオプショナルであるが、ここでは使用目的をはっきりさせるために用いている。

.# 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

図 3–11　　`Kiosks` サンプルブックのキオスクページ



キオスクページ自身もまた名前を持っている必要がある。name=*itemName* コマンドはコンテントアイテムに名前を付ける。名前付きのコンテントアイテムにナビゲートを行うため、その名前をブック内の他の要素から使用することができる。

キオスク定義の残りの部分では、絵とかテキストといったコンテントアイテムをキオスクページに配置する。以下の例では *Boy* の絵をキオスクページのメインカラムに配置するために使用される。centered フラグはカラム内でその絵を水平方向にセンタリングする。

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

goto=*destination* コマンドはユーザーがキオスクページ上の少年の絵をタップしたときに、指定したコンテントアイテムが表示されることを指示する。*destination* パラメタは、name=*itemName* コマンドによって名付けられたコンテントアイテムを指定しなければならない。たとえば、上記の例における goto=boy コマンドはキオスクの絵をユーザーがタップしたときに boy という名前の付いたストーリーが表示されることを意味する。

キオスクの定義は .endkiosk コマンドで完了する。.kiosk コマンドと .endkiosk コマンドの間にある全ての物はキオスクを定義する。

図 3–11 に見られるキオスクページの定義全体は、キオスクページ上に他の５つの絵を配置するためのコマンドを含み、それぞれのコマンドは goto=*destination* 指定も行っている。完全なキオスク定義のリストについては、KioskStory ブックソースファイルを参照のこと。

仕上げ

このセクションではブックの体裁にもっと磨きをかけるためのコマンドやテクニックについて解説する。

"About" スリップに、ブック情報を追加する

ブックソースに著者、出版社、発行日、著作権などの情報を付加するためのコマンドを追加することができる。この情報はステータスバー上の info ボタ

ンを通してアクセスできる "About" スリップにおいてエンドユーザーに表示される。この目的のために、ブックメーカーのコマンドが５つ用意されている：.author, .publisher, .date, .expires, .copyright である。これらのドットコマンドを使うには、以下のサンプルのように、コマンドと関連する情報を同一の行に書く：

```
.title Newton BookMaker 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.
```

さらに二つのコマンド .blurb と .key はブックに関する情報を提供するが、現在のところエンドユーザーからは利用不可能である。このセクションで示される全てのコマンドは ページ A–11 からはじまる「コンテントコマンド」において文書化されている。

コンテントアイテム間に空白を追加する

.space コマンドはコンテントアイテム間に適当な量の空白を追加するのに使用される。このコマンドは、様々な位置揃えフラグでは十分制御できないような、グラフィックとテキストの間の位置揃えを行うには便利な物である。ブックソースファイルにおいて空白行を挿入する代わりに .space コマンドを使うことによって、空白行がページのトップにはこないことが保証され、結果としてブックパッケージのサイズもわずかに小さくなるのである。

ブックソースファイルのコンテントアイテムの後に .space コマンドを追加すると、指定された量の空白がコンテントアイテムの間に追加される。指定する空白の単位はタイポグラファーが使うポイントである。1 ポイントは 1/72 インチである。.space コマンドを使って、63 ポイントまでの空白を指定することができる。次の例を見てみる：

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

.space コマンドの使用例は、KioskStory ブックソースファイルに見ることが出来る。

テキストのインデント

コンテントコマンドとそれに関連づけられたストーリーテキストや絵の間に .indent コマンドを置くことができる。これを使うとコンテントアイテムのどちらの側にでもマージンを作ることができる。.indent コマンドは左または右マージンを定義する。0 を指定するとマージンなしとなる。以下の例ではストーリーテキストの左側に 30 ポイント、右側に 20 ポイントのインデントを作る。1 ポイントは 1/72 インチである。

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

ピクチャーヘッダの追加

.pictHeader コマンドを使って、各ページのトップに表示される絵を追加することができる。以下の例は、TouchesStory[1] ブックソースファイルからとった例である。

```
.pictheader
```

このピクチャーヘッダを実際に見たければ、ブックメーカー付属のサンプルパッケージ Touches.pkg を開いてみると良い。

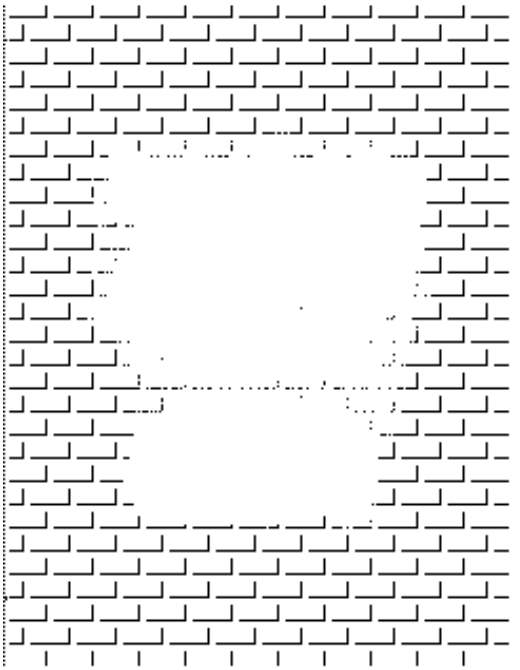
ブックページ内にはみ出るような大きいピクチャーヘッダ

ブックのページ上端に、ヘッダ用に確保された領域は高さ 16 ピクセルである。もしこの高さを超えると、ヘッダーはコンテント用に確保された領域にはみ出す。ページコンテントはピクチャーヘッダが描画された後に描かれるので、ページコンテンツは常にピクチャーヘッダの「上に重なって」に描かれる。結果として、オーバーサイズのピクチャーヘッダを使ってブックページに面白い背景を与えることができる。

1. 原書では TouchesStory となっているが、そのようなサンプルは見あたらなかったので KioskStory からピクチャーヘッダの例を取ってきた。

以下の例は FlagStory ソースファイルからとった物で、このテクニックを示す物である。

.pictHeader

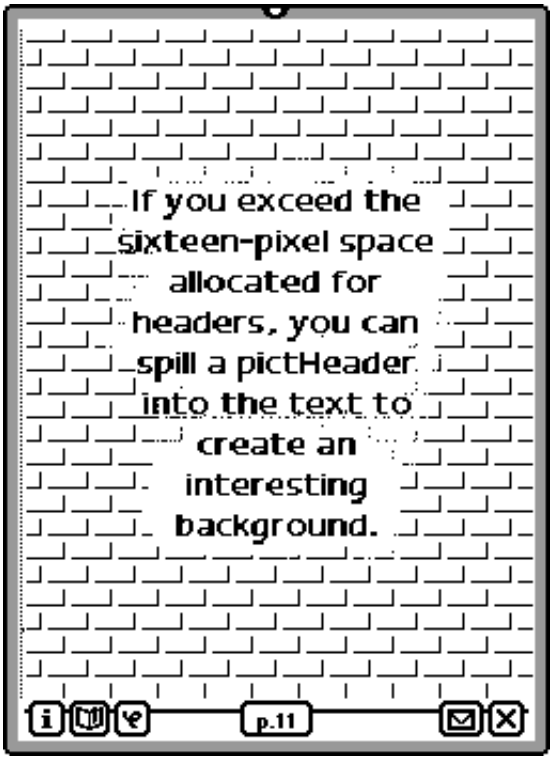


.story centered

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

.pictHeader コマンドはそのコマンドの後にある絵をブック内の全てのページ欄外見出し (ランニングヘッダ) のトップに表示しようとする。サンプルで示した絵は 16 ピクセルより大きいので、それはテキスト本体にあふれ出し、図 3–12 に見られるような効果を作り出す。ストーリーテキストにおける空白行は透明であるため、テキストを垂直方向にセンタリングするための改行は背景を消してしまうようなことはない。

図 3–12　大きなピクチャーヘッダーの使用



外部 PICT ファイルの取り込み

ブックソースファイル内に直接絵をペーストする代わりに、'PICT' ファイル内に絵を格納しておくこともできる。外部のここの 'PICT' ファイルには、ただ一つの絵を入れておくことができる。

ブックソースファイルに外部 'PICT' ファイルを取り込むには、.picture コマンドの後にファイルのフルパス名を記述する。フルパス名は、そのファイルが存在するディスク名の後に、フォルダ名をつづけ、最後に目的のファイルの名前をおいたものである。パス名のそれぞれの要素はコロンで区切り、パス名全体は単一引用符で囲む。以下に例を示す :

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

このコマンドはブックソースファイル内に 'PICT' ファイル 'MyPICTFile' に格納された絵を取り込む。"MyPICTFile" ファイルは "InnerFolder" フォルダに存在する。"InnterFolder" フォルダは

"OuterFolder" フォルダ内に存在し。"OuterFolder" は "MyDisk" 内に存在する。

NewtonScript

ブックメーカーソースファイルに NewtonScript コマンドを追加することにより、Newton ブックの動作をさらにカスタマイズ可能である。このセクションではスクリプト、スロット、ビューテンプレート、フレームを Newton ブックに取り込む方法について述べる。本章で述べる要素は、NewtonScript のプログラミング能力が多少有り、Newton オブジェクトシステムに通じていることを仮定している。

ブックソースファイルでの NewtonScript の使用

`.script` コマンドは NewtonScript コマンドをブックソースファイルに追加するために使用する。これによって後続するもの全てが ブックメーカーのコマンドとしてではなく、NewtonScript として解釈されるようになる。ブックメーカーは、`.endscript` コマンドもしくは他の ブックメーカーコマンドに出くわすことにより、再びブックソースファイルの解釈に復帰する。

`.script` と `.endscript` コマンドの間にある全てのものは「*プログラミング言語 NewtonScript*」において定義されている正しい NewtonScript コードでなければならない。

`.script` コマンドのすぐ後に名前を書くことによって、スクリプトに名前を与えることができる。スクリプトの名前が省略されると、そのスクリプトはデフォルトで `buttonClickScript` となる。ブックリーダーはコンテ

ントアイテムがタップされると常に buttonClickScript メッセージをコンテントアイテムに送信する。

以下の例では単純な viewClickScript メソッドを定義している。

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

コンテントアイテムへのスクリプトの追加

ブックメーカーはファイルを最初から最後まで処理し、.script コマンドの直前にあったコンテントアイテムにそのスクリプトをひっつける。

```
.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 BookMaker commands
// note the NewtonScript comment style
playSound (ROM_FunBeep);
.endscript
.# now we are using BookMaker commands again
.# This command defines a new content item with no
.# script attached
.story name=silent
```

このパラグラフをタップすると音がなる。

この例では、viewClickScript メソッドは beeps と名付けられたストーリーにひっつけられる。ユーザーが beeps ストーリーをタップすれば、システムは ROM_FunBeep サウンドをならす。silent と名付けられたス

トーリーは viewClickScript メソッドがひっついてないので、そこをタップしても音は鳴らない。.script と .endScript の間では ブックメーカーのコマンドではなく NewtonScript を使用しなくてはならないことに注意。

ブックリーダーブック内のコンテントアイテムは実際にはビューであり、アプリケーション内のビューが受信するような全てのメッセージを受信できる。だから、アプリケーションに定義できるあらゆるスクリプトをコンテントアイテム用に定義して良い。たとえば viewClickScript, viewSetupDoneScript 等々をである。

注意
ブックまたはページレベルで viewSetupFormScript を使ってはならない。それは呼び出されないからである。代わりに、viewSetupDoneScript メソッドを用いる。このメソッドはページがイメージングされるとき (プリントされるときは除く)、常に呼び出される。コンテントアイテム用の viewSetupFormScript については特に制限はない。▲

一般に、スクリプト内にページ番号をハードコーディングすべきではない。将来の Newton デバイスは異なるスクリーンサイズを持つ可能性もあり、その際にはページ送りが変わってくる可能性があるからだ。その代わり、名前や属性でコンテントアイテムを検索する NewtonScript メソッドを使うことができる。これらのメソッドは付録 A「ブックメーカー言語」の「NewtonScript メソッド」セクションで説明する。

ブックがイメージングされているのはどのタイプの画面かを調べてからハードコードされたページ番号だけを使用するのであれば、この制限は越えることができるし、検索を行わないのでスクリプトのスピードも向上する。この情報はブックの curRendering スロットに格納されている。curRenderling スロットについては、ページ 4–16「予約済みスロットから取り出せる情報」で、その使用方法を示すコードとともに説明されている。

.script, .endscript コマンドについてより詳しくは、付録 A「ブックメーカー言語」の「コンテントコマンド」セクションを参照のこと。

ページへのスクリプト追加

レイアウトの定義の後に .script コマンドを置いてレイアウトに対してスクリプトをひっつけることができる。そのスクリプトは、そのレイアウトを使っているページにだけ有効である。これらのスクリプトは**ページスクリプト**と呼ばれる。

ページスクリプトは、たとえば特別な背景を描いたり、そのレイアウトでフォーマットされたコンテントアイテムの viewClickScript には提供されていないペンイベントの取り扱いを処理したりするのに使用できる。

ページがスクリプトを持つコンテントアイテムを表示すれば、そのコンテントアイテムに付随するあらゆるスクリプトはページに付随するスクリプトをオーバーライドしてしまう。これにより、コンテントアイテムはページビューの子供で有るかのように考えることができる。だから、ページが viewClickScirpt の定義されたレイアウトを使用しており、ページ上のコンテントアイテムも viewClickScript の定義をされている時に、ユーザーがそのコンテントアイテムをタップすれば、コンテントアイテム側の viewClickScript が呼び出される。ページの viewClickScript メソッドも呼び出したければ、コンテントアイテムの viewClickScript から nil を返せばよい。

ブック全体にスクリプトを追加する

ブックソースファイルの最初のところ (レイアウトまたはコンテントコマンドが登場する前) でスクリプトを定義すると、それはペアレント継承によりブック全体で有効となる。こうしたスクリプトを**ブックスクリプト**と呼ぶ。

コンテントスクリプトがページスクリプトをオーバーライドするのと同様、ページスクリプトはブックスクリプトをオーバーライドする。

NewtonScript コードの共有

共通の動作を持つコンテントアイテム同士でスクリプトを共有することにより、ブックパッケージのスペース効率を若干向上させることができる。スクリプト (あるいは他の NewtonScript コード) は、NTK プロジェクトに含まれるテキストファイルに定義することで、あるいはブックソースの先

頭に定義することで共有できる。それぞれの方法はブックのあらゆる場所からそれが利用可能という点で等しく働く物である。どっちの方法を採るかは、開発の目標によって変わってくるだろう。

ブックソースファイルをできるだけ自己完結にしたいのであれば、ブックの先頭に NewtonScript コードを定義する必要があるだろう。(付録 A 「ブックメーカー言語」のページ A–9 の `.preamble` コマンドの説明を参照)。あるいは、コードを他のブックでも使いたいので有れば、NTK プロジェクトに含まれるテキストファイルに共有コードを定義しておくこともできる。

共有スクリプトの例

`goTo=`*destionation* コマンドはコンテントアイテムに対してナビゲーション機能を提供するが、ハイライト動作は提供しない。以下のサンプルでは、どのコンテントアイテムにもボタン的なハイライト動作を提供するスクリプトを再利用する方法を示す。このスクリプトは、以下の例のように、特にキオスクのエントリに対してハイライト動作を追加するのに有用である：

```
// 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;
```

キオスク内のコンテントアイテムは、この `kioskEntryScript` を `viewClickScript` メソッドから呼び出すことによって、ボタン的動作を身につけることができる、以下の例のようにする：

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

.script コマンドに対する slot パラメタにより、コンテントアイテムをイメージするビューの中にスロットが作られる。このパラメタはこの章で後述する「ビューへのスロット追加」において完全に説明される。

このアプローチによってブックパッケージのサイズを抑えることができる。仮にキオスク内の全てのコンテントアイテムがこの小さな kioskEntryScript メソッドを繰り返し使ったとしても、このスクリプトがバイトコードにコンパイルされるのは NTK プロジェクトがコンパイルされる時に一度だけだからである。

ブック内のプロトとビューテンプレートの使用

.form コマンドによってプロト、ユーザープロト、ビューテンプレートがブックに追加できる。このコマンドによって追加されたそれぞれのビューはコンテントアイテムとは別の物として扱われ、ページ上に完全に機能するコードオブジェクトとして表れる。

.form コマンドへの height, width パラメタはページ上でそのビューが占有するスペースを確保するための物である。.form コマンドによって作られたビューは (通常のコンテントアイテムと同じく)、直前のコンテントアイテムの後に表れる。

以下のサンプルは .form コマンドで myFunkyProto と呼ばれるカスタムビュープロトタイプを取り込んでいる。.form コマンドで追加されたレイアウトとユーザープロトはブックの NTK プロジェクトファイル中に含まれていなければならない。システムプロトタイプは自動的に NTK によって取り込まれる。

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

.form コマンドで行は始まり、ビューの高さと幅の指定が続く。そこからビューが作られるプロト、ユーザープロトあるいはレイアウトは次の行で指定される。protoStaticText のようなシステムプロトはこの行にその名前を書くことで指定できる。ユーザープロトやレイアウトの名前の前には layout_ というキーワードを前置しなければならない。

.form コマンドの後には必ず対応する .endform コマンドがなければならないことに注意。これらのコマンドとそのパラメタについてより詳しくは、付録 A「ブックメーカー言語」の「コンテントコマンド」セクションを参照。

.form コンテントアイテムの検索

.form コンテントアイテムを探索するための方法は三つある。テンプレートまたはプロトにはテキストスロットが追加できるし、FormSearchScript または BookSearchScrit を追加することもできる。

仮に .form コンテントアイテムが text という名のスロットを持っていれば、Find サービスはそれを自動的に検索する。ビューを作成するときにこのスロットを追加することもできるし、以下の例のように .attribute コマンド (ページ 4–8 からはじまる「コンテントアイテムへのスロットの追加」で説明) を使って text スロットを追加することもできる :

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

ブックリーダーは、検索中に .form コンテントアイテムに出会うと FormSearchScript メッセージを .form コンテントアイテムに送信する。このメッセージを処理するには、.form コンテントアイテムに FormSearchScript を追加すればよい。

さらに、ブックに BookSearchScript を提供することによって、システム提供のブック検索エンジンをオーバーライドすることもできる。このメッセージは検索されるべきコンテントアイテムをパラメタとして渡す。そして、特定のコンテントアイテムに対する検索を選択的に処理することができたり、システム提供のサーチエンジンでそのコンテントアイテムを検索したりすることができる。

検索が成功したことを示すため .form コンテントアイテムがハイライトされるとき、システムは FormHiliteScript メッセージをそこに送信する。FormSearchScript, BookSearchScrpt, FormHiliteScript は付録A「ブックメーカー言語」の「ブックリーダーメッセージ」セクションにおいて解説される。

ブック内へのデータ格納とアクセス

このセクションではブックにデータを格納する様々な方法について述べる。グローバルデータはブックパッケージの一部としてブックリーダーが維持するスープに格納できる。あるいはブックの実行時のビューフレーム内のスロットとして格納できる。ローカルデータはコンテントアイテム及び、そのコンテントアイテムをイメージする実行時のビュー内に格納できる。

どこにデータを格納するかを決めるにはいくつかの問題について考えなければならない。

- 利用度 : このデータはブック全体で利用可能でなければならないか？あるいは単一のコンテントアイテムでだけ利用できればいいか？
- メモリマネジメント : このタイプのメモリを使っていいのか、そしてそれはどのくらいの時間使っていていいのか？
- 書き換え可能性 : そのデータは変更されうるか？
- 不変性 : そのデータはブックがオープンされる度に同じ値に初期化されて良いか？

コンテントアイテムへのスロットの追加

コンテントアイテムはフレームとして内部的に表現される。それらのフレームはブックメーカーによって生成され、ブックリーダーがそれをイメージするために必要ないくつかのスロットを含んでいる。.attribute

コマンドを使って、コンテントアイテムのフレームに格納される複数のスロットを作ることができる。実行時には、.attribute コマンドが作ったスロットは現在のコンテントアイテムにのみ有効な、読み込み専用のものとなる。

以下の例では .attributes コマンドの使用例を示す :

```
.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
```

どちらのコンテントアイテムもキャッシュスロットに自分の値を格納していることに注意。

コンテントアイテム内スロットからのデータ取得

ブックリーダーがブックパッケージを開くとき、それは .attribute コマンドを含むコンテントアイテムを参照する、item と呼ばれるスロットを生成する。.attribute コマンドによって生成されたスロットにアクセスするには、スクリプトは NewtonScript のフレームアクセサ (ドット) 演算子を使って item スロットを参照しなければならない。以下の例を参照 :

```
.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
```

上記のサンプルは `Joe’s Pizza` ストーリーが表れたときに "poof" サウンドを再生する。`Joe’s Pizza` に結びつけられた `smoking` スロット (`item` スロット経由でアクセスできる) の値が `true` だからである。

ビューへのスロット追加

コンテントアイテムに結びつけられたスロットを作成するもう一つの方法は、`.scrpt` コマンドに `slot` パラメタを指定することである。この方法でスロットを作成すると、スロットとその内容はコンテントアイテムをイメージするために使用される RAM 内のビューフレーム内に格納される。ブックパッケージ内には格納されない。そのスロットは RAM 内に存在するので、その値は動的に変更できる。もうひとつ覚えておきたいことは、こうやって作った個々のスロットはある程度のヒープスペースを消費するので、本当に必要なときにだけこうしたスロットを作成する必要があるという事である。

また、それぞれのコンテントアイテムは別々のビューによってイメージされることに注意。一つのビュー内のスロットに格納されているデータは、他のビューのスクリプトからは利用不能である。たとえ両方のビューが同じページ上にあったとしてもである。

ラッキーなことに、滅多にビューにスロットを作る必要はないだろう。ビューにスロットを作るのは、それを変更したいときだけにしたほうが良い。ビューの中にスロットを作らないといけないのであれば、ついうっかりパフォーマンスを落とすことがないように気をつけるべきである。さらにページ 4–16「予約済みスロット名」セクションにリストされた予約済み

スロット名の他にも、コンテントアイテムをイメージするビューの中には変更可能ないくつかのスロットが存在することを覚えておいた方がよい。

ビューのスロットはビューそのものに格納されるため、ビュースクリプトはスロットアクセスのために特別なことをする必要がない。以下のサンプルではビュースロットである viewFormat の値を変更し、ビューの縁取りを黒のかわりにグレーにしている：

```
.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.
```

ブック内のグローバルデータ

Newton ブックは、グローバルデータを格納するためのいくつかの構造体を提供する。このセクションではそうした静的あるいは動的なグローバルデータ格納のための構造の使用方法を述べる。

ブックデータ

ブックメーカーがブックソースファイルを処理する際、それは NewtonScript においてブックの構造や内容を定義するファイルを生成する。そのブック定義ファイルを見たいので有れば、ブックメーカーの出力をエディタで開き、そのかなり最初の方のフレームを見てみるとよい。そこにはブック全体の構造が定義されたフレームがある。このフレームは book と呼ばれており、以下のフレームと同じ様な物である：

```
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: []};
```

見てのとおり、book フレームは様々なタイプのデータを含む複数のスロットから構成されている。そこには数値、文字列、配列、フレームなどが含まれている。特に興味深いのは data スロット (太字で表示) であり、これはユーザーのために特別に用意された物である。その中には、自分のデータ、NewtonScript のスロット、フレーム、関数など格納することができる。このデータはブックとともに維持され、ブックが削除されると削除される。

data スロットはブックが開かれている間グローバルに使用可能であるべき低レベルデータの記録場所として用意されている。このスロットには、複数のビューやコンテントアイテムから使用したり、ブックに特に関係のある読み込み専用データを何でも格納できる。たとえば、ブックメーカーはこのスロットに .index コマンドが作り出すインデックスエントリを格納している。

ブックデータの設定と取得

BookData メソッドはブックの data スロット内にあるフレームへの参照を返す。以下のようにすれば、このフレームにデータを出し入れするスクリプトが書ける :

```
.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
```

ブックデータフレームは二つの異なる方法で参照されていることに注意。`book.data` のように明示的に参照されている場合と、ブックリーダーのメソッドである `BookData` の戻り値を使って参照されている場合とがある。これはデータがコンパイル時には格納され、実行時にはアクセスされるためである。

著者データの利用

ブックの状態に関係なく (それが閉じられているときでも) 有効でなければならないデータを格納する事ができる。このデータはスープ内のフレームに格納されている。このフレームはブックデータフレームとは違い、書き込み可能である。だからこれは変更可能なユーザー初期設定を格納するのに自然な場所であるが、状態に関係なく不揮発で (持続的で) なければならない。

最後に開いた８つのブックの著者データが残されている。９つ目のブックを開くと、最初のスープエントリは消滅する。ブックのスープエントリはパッケージが削除された後も残されるので、常に必要なデータだけをここに格納する必要がある。あるいは、９つ目のブックがオープンされた後も書き込み可能なデータが欲しいので有れば、自分自身のスープに書き込むと良い。スープに関しては *プログラマーズガイド* を参照のこと。

AuthorData メソッド

AuthorData メソッドは、Newton ブックリーダーが著者データを格納しているスープベースのフレームへの参照を返す。スクリプトはこの参照を使い、そこにデータを格納したり、そこから取り出したりできる。以下の例を参照 :

```
.# 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
```

ブックへのスロット追加

コンテントアイテムあるいはレイアウトコマンドが登場するより前に、.script コマンドに slot パラメタを使って追加したスロットは、ブックのどのスクリプトからも利用可能になる。ページ 4–10「ビューへのスロッ

ト追加」セクションにも述べられているように、こうしてスロットを追加すると、貴重なヒープを消費することになる。このヒープの使用は特に重要で、なぜならブックが開いている間ずっと存在するからである。ビューが閉じるとその中に有ったスロットがガベージコレクションされるのとは対照的なことである。

しかし、ブックレベルのスロットを作成するというのはビュー同士の通信のためには必要な物である。たとえばレストランに関する情報を格納しているブックを考えてみよう。このブックではユーザーはあるビューでルーマニア料理を指定するかもしれない。他のビューでルーマニア料理店をリストするには、他のビューとコミュニケートしてその情報を取り出さなければならない。このデータは AuthorData フレームに格納するものであるが、これはブックが削除された後でも残ってしまう可能性がある。ブックレベルのスロット追加は、実現性のある選択肢である。

ブックレベルスロットがこの方法で追加されると、このスロットに対する特別なアクセスコードが不要になる。このスロットがどのスクリプトから参照されても、継承によってそれを見つけることができる。たとえば：

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

スクリプト中からこのデータにアクセスするには、単に favoriteTypeOfFood を参照すればよい。

このアプローチの唯一の制限は、そのスロットが使われていようがいまいが生成されると言う事である。これは、このスロットを作り出すコードがブック定義ファイルの先頭に必然的に存在し、それは当然実行されてしまうからである。この制限は、contentArea ビューを直接参照する事によって解消できる。

contentArea ビューはコンテントアイテムをイメージするビューよりももっと高いペアレントチェーン階層に存在する。これはブックがオープンされたときに生成され、ブックが閉じるまで存在する。これはグローバルスロットを格納するには便利な場所である。以下の NewtonScript コード行は contentArea ビューにスロットを作成する：

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

重ねて言うと、このデータにはスロット名だけでアクセスできる。それは継承ルールに沿って発見されるからだ。以下のコードは favoriteTypeOfFood スロット内に格納されている文字列で始まるテキストを持つページへ移動する：

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

予約済みスロット名

Newton ブックリーダーは以下の名前のスロットを OS のために予約している。同名のスロットを作ってはならない。

表 4–1　　予約済みスロット名

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

予約済みスロットから取り出せる情報

printing スロットに TRUE が入っていると、そのページがプリント中であることを示す。同様に、bookmarking スロットにおける TRUE はそのページがブックマークイメージとして参照されていることを示す。

curRendering スロットの値が 0 の場合、現在のページが MessagePad のサイズであることを示す。この情報を使い、以下のコードにより、もっと早いページめくりアルゴリズムを実装可能である：

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

このコードは、ブックが MessagePad サイズのスクリーン上で表現されている場合はすでに決まっている値をページ番号として使用し、さもなくばページ番号を決めるためにコンテントアイテムの探索を行っている。

copyProtection スロット

ブック中のページあるいはコンテントアイテムをコピープロテクトするには、copyPretection という名のスロットを、そのアイテムもしくはページに追加すると良い。ブック全体をコピープロテクトするには、このスロットをブックソースファイルの先頭で、しかもコンテントアイテムがまだ定義されていない場所に定義する。copyProtection スロットには表 4–2 で解説される値を格納できる。

表 4–2　　copyProtection の定数

定数	値	説明
cpNoCopies	1	このビューはコピーできない
cpReadOnlyCopies	2	このビューはコピーできるが、コピーを変更することはできない。
cpOriginalOnlyCopies	4	オリジナルビューはコピー可能だが、そのコピーをさらにコピーすることはできない。copyProtection スロットのコピーの値は 1 (cpNoCopies) となり、さらなるコピーを防ぐようになる。
cpNewtonOnlyCopies	8	ぉのビューは一台の Newton デバイス上でしかコピーできない。コピーは他の Newton デバイスに持っていくことはできない。

以下の例は、どんなコピーも許さない copyProtection スロットを作る :

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

コンテントアイテムのマーク

.mark コマンドを使ってあるコンテントアイテムの属性 (スロット) を、.usemark コマンドによって指定された後続のコンテントアイテムから利用可能にできる。これらのコマンドは現在表示されているコンテントアイ

テムが他のコンテントアイテムの中にあるスロットを、検索をしなくても参照できるようにする。新しいコンテントアイテムは .story, .picture, .subject, .form, .chapter といったコマンドのいずれかによって始まる。

例えば、ガイドブックやカタログの (特定のデータをグラフィカルに図示する) サマリービューで個々のコンテントアイテムをフォローしたいので有れば、そのデータを個々のコンテントアイテムに付随するスロットに格納しておき、そのスロットの値に従いサマリービューをアップデートする。以下のサンプルでは .mark と .usemark コマンドを使い、コンテントアイテム Joe's Pizza Parlor 内にユーザープロト myRestaurantView から利用可能なスロットを作成している :

```
.# 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
```

マークされたコンテントアイテムのスロットの参照

.usermark コマンドによってコンテントアイテムに注釈をつけると、ブックメーカーはそのブック定義ファイルの記述に related という名前のスロットを追加する。related スロットは .mark コマンドで指定されたコンテントアイテムを参照する。.usemark コマンドによって指定されたコンテントアイテムはその related スロットを使い、マークされたアイテムのスロットを参照することができる。

以下のサンプルはマークされたコンテントアイテム内のスロットに related スロットを使用してアクセスする：

```
.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
```

このサンプルにおいてスタティックテキストビューが生成されると、その `text` スロットの値は `nil` となる。このスタティックテキストビューの `viewSetupDoneScript` メソッドはこのスロットに文字列を格納する。文字列は `related` スロットを使用してマークされたコンテントアイテムの `text` スロットを参照して得られる。

コンテントアイテムへの参照を取得するためのインデックスの利用

`.mark` コマンドは、同じあるいは近いページに `.mark` されたり `.usemark` されたアイテムが登場し、相互に一対一の関係になっていたりすると非常に便利である。しかし、複数のコンテントアイテムをマークすることはできないし、そうしたもの全てに対する参照を持つ別のコンテントアイテムを持つこともできない。最後の一つが記憶されるだけだからである。

このような状況では、`.index` コマンドが必要になってくる。コンテントアイテムが `.index` *Entry* コマンドを持っていると、そこへの参照が配列となり、`book.data` フレームの `alphaIndex` と呼ばれるスロットに格納

される。このコンテントアイテムへの参照はアルファベット順に alphaIndex 配列に格納される。

以下の例では、alphaIndex[0] は二番目 (aadrvark) のストーリーへの参照で、alphaIndex[1] は最初のストーリーへの参照である：

```
.story
.index zebra
Write a story about zebras here.
.story
.index aardvark
And one about aardvarks here.
```

実行時に alphaIndex の要素を参照するには、この配列が格納されている book.data フレームを参照する必要がある。この方法に関してはページ 4–12「ブックデータの設定と取得」を参照。

さらに、subIndex と呼ばれるもう一つの 26 要素の配列が book.data フレームに作成される。subIndex 内のそれぞれの要素はアルファベットの文字に対応している。たとえば、subIndex[0] は "a" に対応し、subIndex[25] は ’z’ に対応する。特定の文字で始まるエントリがない場合、たとえば ’m’ に要素がない場合は ’m’ の値は ’ n’ の値と同じになる。n-z の間に何もエントリがない場合、’m’ は ‘l’ の値と同じになる。

subIndex 配列の生成は .option noSubindex コマンドをブックのどこかに置くことで抑制できる。より詳しくはページ A–29 からはじまる「Option」を参照のこと。さらに、オプションの noSubIndex 引数が .index コマンドに使用されると、subIndex 配列は生成されるが、個々のコンテントアイテムはその中には洗われない。この構文に関してはページ A–28 からはじまる「Index」 を参照。

ソースの末尾 (postamble) においてブックの後処理を行い、インデックスから得られる情報を引き出すことができ、alphaIndex スロットを nil にする事によってパッケージ構築時のメモリを節約できる。以下のコードは book.data フレームに、alphaIndex の最初の要素への参照を含むスロットを追加し、その後メモリを解放している。

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

重要
現在のところ、インデックスは純粋にプログラマ向けに作られているが、将来のブックリーダーのバージョンでは、ナビゲーションの目的からエンドユーザーにもインデックスが見えるようになるだろう。インデックスがエンドユーザーに見える場合、それは特殊なスロットがセットされているときだけ表示される。だから、インデックスをユーザーにみせることに関して不安を抱く必要はない。このことを心にとどめておき、ユーザー可視のインデックスへの移行が簡単に成るようにしておこう。◆

複数のインデックス生成

`.index` コマンドはオプションの引数 `!`*indexName* をとる。これは `book.data` フレームに *indexName*`Index` と呼ばれる分離されたインデックスを作成するのに使用される。たとえば、`.index !mammals rabit` は、`.index !mammals` *Entry* コマンドを持つ全てのコンテントアイテムへの参照を配列にした `mammalsIndex` と呼ばれるインデックスを生成する。これらのコンテントアイテムへの参照は `mammalsIndex` にのみ現れ、`alphaIndex` 内には登場しない。

さらに、別の配列 *indexName*`SubIndex` が生成される。*indexName*`SubIndex` 配列もまた 26 要素の配列で、`alphaIndex` の `subIndex` と同様の性質を持つ。

indexName`SubIndex` 配列の生成を制御する方法はいくつかある。ブックが `.option noSubIndex` コマンドを持っていれば、ブック全体に渡ってサブインデックスは生成されない。`.index @`*indexName Entry* 構文を `.index !`*indexName Entry* 構文に変わって使うと、個々のインデックスは対応するサブインデックスを持たない。さらに、`.index` コマンドにオプションの `noSubIndex` 引数を与えれば、そのエントリはサブインデックスからは排除される。

コンテントアイテムへのページ番号の格納

.option firstPage コマンドはそれぞれの可視なコンテントアイテムに対して firstPage というスロットを追加する。(可視なものとはたとえば NoPage もしくは BrowserOnly フラグを含んでいないコンテントアイテムのことである。) このスロットはそのコンテントアイテムが表示されるページ番号を持っている。将来的には異なるスクリーンサイズが利用可能になり、firstPage スロットはページ番号の配列を含むことに成るだろう。その時には、curRendering スロットがこの配列から正しいページ番号を選択するために必要な情報を提供することになるだろう。

どのコンテントアイテムに対して firstPage スロットを追加するかという選択のコントロールをより行うためには、.option indexPage コマンドが提供されている。これは firstPage スロットを .index コマンドが付加されている可視なコンテントアイテムに追加する。.option indexPage コマンドは .option firstPage コマンドと同様の効果を持つが、コンテントアイテムのいくつかあるいは全てに firstPage スロットが追加されないこともある点が違う。

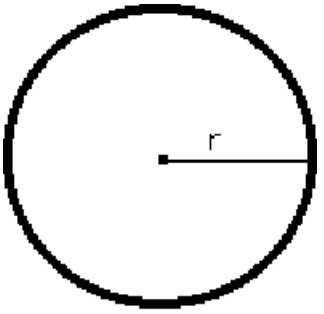
ストーリーカードの生成

ストーリーカードは Newton スクリーン上のどこにでも表れるビューである。そしてそれはそれ自体が提供するクローズボタンによって消滅する。ストーリーカードは少ない情報をポップアップするのに便利である。この方法で、タップされたものに関する追加の情報を提供するようなハイパーテキスト環境を作ることができる。この情報は短い説明、絵、システムプロトタイプやビューテンプレートから生成されるあらゆるビューであって良い。

ストーリーカードはページ A–45 で紹介する :ShowStoryCard メソッドの呼び出しによって生成される。このメソッドには viewBounds フレーム、コンテントアイテムの名前と値が渡される。どのコンテントアイテムもストーリーカードとしてイメージ可能である。しかし、現実的制限として、viewBounds フレームの境界に収まるコンテントアイテムしか使えない。ストーリーカードはスクロールできないからである。また、一度に表示できるストーリーカードの数は最高１つである。

たとえば、以下のコンテントアイテムは円の面積を求める公式を表示する。ユーザーが式をタップするとストーリーカードが円の図解とともに表示される。

```
.story
Circle: A =  r2
.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"
```



π = 3.14159

動的ブラウザの作成

AddBrowser 関数を使えば、ブックリーダーのオーバービューに動的にブラウザを追加できる。これの典型的な使用方法は、カスタム検索結果を表示することである。ユーザーはブラウザを開いておき、その中のアイテムをタップして、好きなところに移動できる。

browser フレームを追加するには以下のようにして AddBrowser を使用する：

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

戻ってきた値は今追加したブラウザを表示するのに使用できるインデックスであり、以下に例を示す：

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

注意
MessagePad 上では、ブラウザを削除する方法はない。◆

リストブラウザフレームの構造は以下の例のようになる：

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

`list` 配列内のエントリは次の形である：

```
{item: content, level: 2}
```

あるいは

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

`item` スロット内のコンテントはブック内のコンテントアイテムである (別の言い方をすれば、`FindContentByValue` メソッドが返してきた値)。`level` スロットは省略可能で、デフォルトは 1 である。`name` スロットはアウトライン中のトピックのテキストである。これが省略されると、`content` のテキストが、ブラウザリストアイテムとして表示される (非テキストのコンテントアイテムに対するエントリは、`name` スロットを必ず持たなければならない)。

ブックへのインテリジェントアシスタントテンプレートの追加

ブックは他のアプリケーションのようなインテリジェントアシスタント (IA) が使用可能である。IA サポートのためには、`.assist` と `.endassist` コマンドの間にタスクテンプレートを書かなければならない。

*Newton プログラマーズガイド*にはタスクテンプレートを書くために必要な情報が全て提供されており、IA の一般的情報も提供されている。だが、アプリケーション用の IA サポートに必要な過程はブックでそれを行うよりも

広範である。`.assist` と `.endassist` コマンドの中では、タスクテンプレートを書く以外の事はする必要がない。

Chapter 5

ヘルプ

ブックメーカーにおいて、Help Size オプションをクリックしてブックソースファイルを処理することにより、アプリケーションヘルプを生成することができる。生じたヘルプブックファイルはアプリケーションと一緒にもしくは単一のパッケージとしてコンパイルされ、ヘルプ表示用のシステム提供ヘルプブラウザを使用する。本章では ブックメーカーによるヘルプブックの作成方法及びアプリケーションからのヘルプブック表示方法を述べる。

本章の内容は本書の他の部分よりも極端にテクニカルなものである。ここでは NewtonScript 言語、Newton オブジェクトシステム、Newton アプリケーションプログラムの生成に関する知識を持っているものと仮定する。こうしたトピックに関してより詳しくは、*プログラミング言語 NewtonScript* や、*Newton プログラマーズガイド*、*Newton Toolkit ユーザーズガイド*を参照のこと。

アプリケーションへのヘルプの追加

ヘルプブックは Help Size オプションをチェックした状態で ブックメーカーがブックソースファイルを処理してできたファイルである。図 5–1 に例を示す。ヘルプブックはアプリケーションのステータスバー上にある info ボタンの help オプションを通じてアクセスすることができる。それは独立パッケージとしてビルドして構築できるし、パッケージの一部としてアプリケーションとともにコンパイルすることもできる。

図 5–1　Newton ブックメーカー の Help Size オプション

ヘルプブックの記述

ヘルプブックの記述工程はブラウザの記述と本質的に同じである。実際、ヘルプブックはブラウザと `.title` と `.isbn` コマンドの集まり以外の何者でもない。ブラウザの詳細に関してはページ 3–19 からはじまる「ブラウザペーンの生成」を参照のこと。

しかしながら、ヘルプブックを書くときには、全てのブラウザには適用されないいくつかの事柄を心にとどめておかなければならない：

- ヘルプアイテムは短くしておくこと。ブラウザはスクロールできないし、将来的にもそうだろう。ヘルプアイテムはブラウザペーンの長さを越えた場合、それは端切りされてしまう。よいヒューマンインターフェースは、ユーザーがヘルプブックにおいて迷うような事がないようなものだ。物事はシンプルにし、単純で、特徴的で、特定のタスクに応じたインストラクションを示すべきである。
- 特定の質問に対してオーバービューアイテム (`.subject` コマンドによって表示された行) で答えること。一つのオーバービュートピックは「トピックを行う方法」について答えるべきである。たとえば、トピックが「ショッピングリストアプリケーションの使用」に沿ったものであれば、「ショッピングリストアプリケーションをどうやって使うか」になるだろうし、このトピックの元では、サブトピックは「ショッピングリストへのアイテムの追加」とか「注文した商品をチェックオフする」等となるだろう。

オンラインヘルプの作成に関して議論した書籍の一覧が付録 C「オンラインヘルプに関する書籍」に示されている。

スタンドアロンのヘルプブック構築

ヘルプブックを単独のパッケージとして構築するのに特別なステップはいらない。通常のブックリーダー用ブックを作成するのと全く同じであり、ブックメーカーの Help Size オプションをチェックするという手順が追加されるだけである。ブックパッケージが Newton デバイスにダウンロードされるとき、Extras Drawer の Help フォルダに自動的にそれは配置される。スタンドアロンのヘルプブックはアプリケーションヘルプをインストールしないとか、読み終わったらヘルプを勝手に消去すると言ったオプションをユーザーに提供する。これによってアプリケーションパッケージのサイズを最小にすることもできる。

Newton ブックメーカーを使用してブックソースファイルを処理し、ブック定義ファイルを得る方法については、ページ 2–9 からはじまる「ブックソースファイルの処理」を参照のこと。ブック定義ファイルを処理してパッケージを得る方法については、ページ 2–13 からはじまる「NTK でのブックパッケージ構築」を参照のこと。

アプリケーションへのヘルプの追加方法

アプリケーションにヘルプを追加するために必要なステップは、ヘルプブックを独立パッケージとして構築するかあるいは、アプリケーションパッケージの一部として構築するかによって異なってくる。

いずれの場合も、アプリケーションは、その `doInfoHelp` スクリプトからヘルプブックを開く `protoInfoButton` (もしくは NewtApp ベースなら `newtInfoButton`) を必要とする。`doInfoHelp` スクリプトは `openHelpBook(`*isbnStr*`)` もしくは `openHelpBookTo(`*isbnStr*`, `*topic*`)` のいずれかを呼び出してヘルプブックを表示する必要がある。`protoInfoButton` と `newtInfoButton` テンプレートについては Newton プログラマーズガイドに記述されている。`OpenHelpBook` と `OpenHelpBookTo` 関数は第 A 章「ブックメーカー言語」で記述されている。

ヘルプブックの存在を確認するための `:WhereIsBook` を、`OpenHelpBook` の前に呼び出す必要もあるだろう。`:WhereIsBook` はグローバル関数でなく、`copperfield`(`copperfield` は全てのブックの親である) のメソッドであるため、`:WhereIsBook` メソッドはこのフレーム

に対して送信しなければならないのである。以下のコードはこれを正しく処理してくれる：

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

`WhereIsBook` は付録 A「ブックメーカー言語」のセクション「NewtonScript メソッド」に記述されている。

これが独立パッケージとしてコンパイルされたヘルプブックを開くためにアプリケーションにしなければならない事の全てである。

アプリケーションのパッケージにヘルプブックを追加する

ヘルプブックはアプリケーションプロジェクトの一部としてコンパイルしてよい。これを行うには NTK の Project メニューから Add File を選択し、ブック定義ファイルを選択する。ヘルプブックがこの方法で構築されると、それはアプリケーションの永続的パートとなり、Extras Drawer の Help フォルダには登場しなくなる。

システムにヘルプブックを登録することで、`OpenHelpBook` がブックを発見できる。ヘルプブックは `RegisterBookRef(`*isbnStr*`, {book:` *book*`})` を呼び出すことでシステムに登録できる。これはアプリケーションベースビューの `viewSetupDoneScript` で行ってよい。

すでに必要ないブックを登録解除するのが良い習慣である。これは `UnRegisterBookRef(`*isbnStr*`)` を呼び出すことで行える。これを行う適切な場所はプロジェクトの `removeScript` である。

`RegisterBookRef` と `UnRegisterBookRef` は付録 A「ブックメーカー言語」の「ブックリーダーメッセージ」に記述されている。`viewSetUpDoneScript` と `removeScript` については、*Newton プログラマーズガイド*を参照のこと。

アプリケーションプロジェクトのヘルプブック内の絵

ブックメーカー出力ファイルはブックソースファイルから ブックメーカーが生成したヘルプブックフレームを含む。ソースファイルが絵を含んでいると、ブックメーカーはそれを `'PICT'` リソースとして出力ファイルのリソースフォークに格納する。ブックがプロジェクト内で、出力オプションを "**Book**" にして処理されると、NTK はブック定義ファイルのリソースを探す必要があることを知っている。しかし、ブックがアプリケーションプ

ロジェクトの一部として処理される際には、リソースが開かれる保証はない。

これを NTK のグローバル関数 OpenResFileX を呼び出すことで行える。OpenResFileX 関数は開く必要のあるリソースファイルの完全なパス名を引数としてとり、開かれたファイルに対する参照のために使用される数値を返す。

以下のコードはディスク myDisk 上のフォルダ myFolder 内の myResouce ファイルを開く。そして、OpenResFileX が返した値を refNum 変数に格納する。このコードはヘルプブックの先頭部分に配置できる：

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

リソースファイルの使用が終わったら、グローバル関数 closeResFileX を呼び出す。この関数の引数としては refNum を渡せばよい。これはブックの最後尾で行える。

ブックメーカー言語

本付録ではブックメーカーコマンド言語で利用可能な全てのコマンドの解説を行う。オプションのコマンドは、(オプション) 等としてマークしている。

コマンドはカテゴリの下にアルファベット順に示す。カテゴリはドキュメントコマンド、コンテントコマンド、ブラウザコマンド、ページレイアウトコマンド、その他のコマンド、フラグ、NewtonScript メソッド、ブックリーダーメッセージ、NewtonScript グローバル関数である。

ブックメーカー言語の概観

このセクションではブックメーカーコマンドのそれぞれのカテゴリについて概観する。だから、様々な効果を生み出すためにそれぞれのコマンドをどのように組み合わせるべきか理解できるだろう。ブックメーカー言語の個々の要素については本付録の後続セクションにおいて解説される。

ブックメーカーコマンドのタイプ

Newton ブックメーカー言語の全てのコマンドは以下の５カテゴリーに大別される：

- ドキュメントコマンドは著者、著作権、アシスト機能等の、ドキュメント全体の特性を定義する。第 2 章「はじめの第一歩」で述べられている `.title`, `.isbn` は典型的なドキュメントコマンドである。
- コンテントコマンドはテキスト、グラフィック、NewtonScript ビューテンプレート、キオスクなどのドキュメントが含んでいる様々な種類の要素について定義する。第 2 章「はじめの第一歩」で述べられている `.sotory`, `.picture` は典型的なコンテントコマンドである。これらのコマンドでタグ付けされたテキストやグラフィックはコンテントアイテムと呼ばれる。
- ブラウザコマンドは、Newton ブックリーダーにおいて、ユーザーナビゲーションに有用なアウトライン的ページの定義を行う。ブラウザコマンドは第 3 章「ブックメーカー言語の使用」の「ブラウザページの生成」セクションにおいて詳細に解説されている。
- ページレイアウトコマンドは、カラム数やその幅、グラフィックの配置などドキュメントの一般的レイアウトの特性を定義する。ページレイアウトコマンドはレイアウトと呼ばれる名前付きフォーマットの作成に使われたり、またページにスクリプトを結びつけるのに使用される。第第 3 章「ブックメーカー言語の使用」のいくつかのセクションでは様々なページレイアウトコマンドの解説に当てられている。
- その他のコマンドは、インデックス、巨大なソースファイルの分割、コメントの組み込みなどを容易にする。
- フラグによってブックメーカーコマンドに様々なオプションを与えることができる。
- NewtonScript メソッドによってブックリーダーブックのコンテントを NewtonScript から操作可能になる。
- ブックリーダーメソッドはブックリーダーがブックに対して、特定の状況で送信するメッセージである。そうしたメッセージを処理するために、メソッドを提供でき、それにより特定の状況に対応するアクションをとることができる。
- NewtonScript グローバル関数はデジタルブックに作用するためにアプリケーションが呼び出せる関数である。

NewtonScript を知らない場合

本付録の最後の３つのセクション、NewtonScript メソッド、ブックリーダーメッセージ、NewtonScript グローバル関数は、NewtonScript プログラマだけが利用できる情報を提供する。

他のセクションは、プログラマでない人も読めるブックメーカーのコマンド言語を解説しているが、いくつかは NewtonScript プログラマのために用意した物もある。ページ A–53 からはじまる「コマンド、関数、メソッドのサマリー」セクションでは、NewtonScript の知識を必要とするコマンドに対して印を付けている。

ブックメーカーコマンドの構文

ブックメーカーコマンドは常にブックソースファイルの行頭に登場する。それぞれのコマンドはブックソースファイル中で別のコマンドが表れるまでの全ての物に適用される。コマンドそれ自身は Newton のスクリーン上には表れない。

以下にブックメーカー言語の構文のサマリーを示す：

- ピリオド (.) で始まる行は、ブックメーカーコマンドとして取り扱われる。
- ブックメーカーコマンドは全て大文字小文字の区別をしない。
- コマンドライン中で # が表れた場所以降はコメントである。
- 空白を含む名前は引用符で囲まなければならない。たとえば `"Futura Heavy"` あるいは `'Futura Heavy'` となる。

注意
本書では、角カッコ ([]) で囲まれるパラメタはオプショナルである。◆

ブックメーカーソースファイルの構造

ドキュメントコマンドはブックソースファイルの最初の部分に配置される。その後にページレイアウトやスタイルが続く。ドキュメントの大半はスクリーン上に表示されるテキストやグラフィックから構成される。このコン

テントはドキュメントの構造を表すブックメーカーコマンドの間に点在している。

実際にはごくわずかのコマンドがブックメーカーに必須のものである。典型的なブックソースファイルにおけるコマンドの大多数は付加的なページレイアウトコントロールやユーザーへの付加的な機構を提供するオプショナルなコマンドである。

典型的なメーカーソースファイルの例については、ブックメーカーに含まれるサンプルファイルを参照のこと。

ドキュメントコマンド

ドキュメントコマンドは、著者、著作権、アシスト機能等ドキュメント全体の特性を定義する。このセクションではブックメーカーコマンド言語におけるドキュメントコマンドそれぞれについて述べる。

Assist

.assist

このコマンドに後続する NewtonScript コードがインテリジェントアシスタントの使用するテンプレートを定義する。(オプショナル)

例 :

```
.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
```

ページ A–7 の .endassist コマンドの記述も参照のこと。

Author

`.author` *author*

ブックの著者を定義する。(オプショナル)

この情報はブックリーダー Ver2.0 の info ボタンからアクセスできる "About" スリップに表示される。

author　　ブックの著者の名前を表す文字列。

例 :

```
.author Charles Dickens
```

Blurb

`.blurb`

blurbText

`blurb` はブックに関する販売用途の情報が含まれるテキストを含む。`.blurb` コマンドから次のドットコマンドまでの間にあるテキストが `blurb` のテキストである。(オプショナル)

この情報は現在 NewtonScript からのみ利用可能である。だが、ブックリーダーの将来のバージョンではエンドユーザーがこの情報にアクセスできるようになる予定である。

blurbText　　ブックの簡単な説明を表す文字列。

例 :

```
.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*

ブックの著作権を定義する。ブックソースファイルにはこのコマンドは一回だけしか書けない。(オプショナル)

この情報はブックリーダー Ver2.0 の info ボタンからアクセスできる "About" スリップに表示される。

copyright　　ブックの著作権情報を表す文字列。

例 :

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

注意 :© をタイプするには、オプションキーを押しながら、小文字の "g" をタイプする。◆

Date

`.date` *date*

ブックの発行日を定義する。ブックソースファイルにはこのコマンドは一回だけ書ける。(オプショナル)

この情報はブックリーダー Ver2.0 の info ボタンからアクセスできる "About" スリップに表示される。

date ブックの発行日を表す文字列。

例 :

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

Endassist

.endassist

`.assist` コマンドによって定義される NewtonScript のブロックを終了させる。`.assist` コマンドには必須。より詳しくはページ A–4 の `.assist` コマンドの説明を参照。

Endpostamble

`.endpostamble`

`.postamble` コマンドによって定義される NewtonScript のブロックを終了させる。`.postamble` コマンドには必須。より詳しくはページ A–9 の `.postamble` コマンドの説明を参照。

Endpreamble

`.endpreamble`

`.preamble` コマンドによって定義される NewtonScript のブロックを終了させる。`.preamble` コマンドには必須。より詳しくはページ A–9 の `.preamble` コマンドの説明を参照。

Expires

`.expires` *date*

ブックに含まれる情報が失効する日付を定義する。ブックソースファイルにはこのコマンドは一回だけ書ける。(オプショナル)

この情報はブックリーダー Ver2.0 の info ボタンからアクセスできる "About" スリップに表示される。

date　　ブックの失効日を表す文字列。

例 :

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

Key

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

読者にブックを説明するためのキーワードリストを定義する。これらのキーワードは検索に利用でき、ブックリーダーはそのブックが Newton デバイスから削除されてもその情報を覚えている。ブックソースファイルにはこのコマンドは一回だけ書ける。(オプショナル)

keyword　　ブックに関連するキーワードとなる文字列。

例 :

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

この情報は現在 NewtonScript からのみ利用可能である。だが、ブックリーダーの将来のバージョンではエンドユーザーがこの情報にアクセスできるようになる予定である。

ISBN

`.isbn` *isbn*

ブックのユニークな識別子を定義し、これは実際の ISBN 番号であって良い。この識別子は 14 文字以下でなければならない。ブックソースファイルにはこのコマンドは一回だけ書ける。(必須)

この情報は現在 Newton ブックリーダーがパッケージを Extras Drawer 内で識別するのに使用される。この情報は現在 NewtonScript からのみ利用可能である。だが、ブックリーダーの将来のバージョンではエンドユーザーがこの情報にアクセスできるようになる予定である。

isbn　　ブックを一意に識別する文字列

例 :

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

ISBN は International Standard Book Number の略である。ISBN は書籍を一意に識別するため、出版社や書籍取り扱い業者が使用している一意な番号である。

頻繁に書籍を出版し、広範に配布する必要が有れば、実際の ISBN 番号を取得して自分の書籍を識別する必要もあるだろう。ISBN 番号は R.R. Bowker compay から通常の手数料で取得できる。コンタクト先は以下：

ISBN Agency
R.R. Bowker
121 Chanlon Road
New Providence, New Jersey 07974
Phone: (908) 665-6770
FAX: (908) 665-2895

Postamble

`.postamble`

`.postamble` と `.endpostamble` の間にあるあらゆる NewtonScript コードはブック定義ファイル内の、ブックメーカーが生成するコードの後に直接書き出される。何らかのクリーンアップ動作が必要な場合、ここは便利な場所である。たとえば、構築時にだけ巨大なオブジェクトが必要な場合、このオブジェクトを文書の末尾で `nil` にセットすることもできる。(オプショナル)

`.endpostamble` と `.preamble` コマンドの説明も参照のこと。

Preamble

`.preamble`

`.preamble` と `.endpreamble` の間にあるあらゆる NewtonScript コードはブック定義ファイル内の、ブックメーカーが生成するコードの前に直接書き出される。(オプショナル)

このコマンドの一つの理法方法は、ブックデータにグローバルで他を格納することである。以下の例：

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

preamble はブックのあらゆる場所から利用可能なメソッドを定義する場所として便利である。これはメモリ消費を最小限に抑えられると言う利点がある。しかし、こうしたメソッドを様々なブックで使用したいのであれば、それを NTK プロジェクトに含まれるテキストファイル内に定義することにより、より便利になるだろうと思う。

`.endpreamble` と `.postamble` コマンドも参照のこと。

Publisher

`.publisher` *publisherInfo*

ブックの出版者を定義する。ブックソースファイルにはこのコマンドは一回だけ書ける。(オプショナル)

この情報はブックリーダー Ver2.0 の info ボタンからアクセスできる "About" スリップに表示される。

publisherInfo　ブックの出版者を表す文字列。

例 :

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

ShortTitle

`.shortTitle` *shortTitle*

ブックの短いタイトルを定義する。これは Extras Drawer で名前として使用される。ブックソースファイルにはこのコマンドは一回だけ書ける。短いタイトルが定義されていない場合、ブックの通常のタイトルが Extras Drawer で使用される。(オプショナル)

shortTitle　　Extras Drawer でアイコンの下に表示される文字列。

例 :

```
.shorttitle Dickens
```

Title

`.title` *titleText*

ブックのタイトルを定義する。ブックソースファイルにはこのコマンドは一回だけ書ける。(必須)

titleText　　ブックのタイトルとなる文字列。短いタイトルが指定されていない場合、この文字列が Extras Drawer で使用される文字列となる。この文字列はまた、ヘッダあるいはピクチャーヘッダによって置換されない限り、ブックの各ページのトップに表示される。`noTitle` フラグを持つレイアウトにおいては、ページトップへのタイトル文字列表示は行われない。

```
.title All About Dickens
```

コンテントコマンド

コンテントコマンドはブック内の様々なコンテントを定義する。コンテントアイテムとはテキストやグラフィック、ビュー、キオスクなどである。

Attribute

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

コンテントアイテムにスロットを作成することにより、そこに付加情報を定義できるようにする。スロットはビューシステム (と NewtonScript) あるいはインテリジェントアシスタントから利用できるようになる。これはページ 4–8 からはじまる「コンテントアイテムへのスロットの追加」セクションにおいて詳説されている。

name　　このコマンドが生成するスロットの名前。

value　　このコマンドが生成するスロットに格納される値。

たとえば、場所を示すコンテントアイテムには電話番号やアドレスを持つスロットを持っているかもしれない。

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

Chapter

`.chapter` *flags* `[name=`*nameString*`] [goto=`*item*`]`

`[browser=`*browserName*`] [layout=`*layoutName*`]`

subjectText

サブジェクトのレベル 1 を作成する。`.subject 1` コマンドと同じである。

このコマンドへのパラメタは `.subject` へのパラメタと同じである。より詳しくはページ A–21 の `.subject` コマンドを参照。

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

注意
独立の行に書かれなければならない *subjectText* を除いては、`.chapter` コマンド全体はソースファイル中の単一行として記述されなければならない。◆

Endform

`.endForm`

`.form` コマンドで定義される NewtonScript ブロックを終了する。

`.form` コマンドには必須である。`.form` コマンドについては、ページ A–12 を参照。

Endscript

`.endscript`

`.script` コマンドで定義される NewtonScript ブロックを終了する。`.script` コマンドには必須。より詳しくはページ A–18 の `.script` コマンド参照。

Endkiosk

`.endkiosk`

`.kiosk` コマンドの定義を終了させる。`.kiosk` コマンドには必須。より詳しくはページ A–15 の `.kiosk` コマンド参照。

Form

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

ビューのページに指定された量のスペースを確保し、オプショナルに指定されたシステムプロトタイプ、ビューテンプレート、レイアウト、ユーザーテンプレートからビューを生成する。.form コマンドには対応する .endform コマンドが必要である。.form と .enform の間にある全ての物がこのビューを定義する。(オプショナル)

注意

別の行に表れなければならないレイアウト、プロトタイプを除けば、.form コマンド全体はブックソースファイル中の単一行として記述されなければならない。◆

height 必須。ビューをイメージするために必要な空間の高さをピクセル単位で指定する。

width 必須。ビューをイメージするために必要な空間の幅をピクセル単位で指定する。

name= オプショナル。このフォームをブック内の他の構成要素から参照可能な *nameString* を指定する。たとえば、.form コマンドが name=FredForm パラメタを定義していれば、他のコンテントアイテムは goTo=*FredForm* をそのパラメタとして指定することにより、このフォームを表示することができる。

flags ビューに結びつけられたオプショナルなコンテントフラグ。

browser= オプショナル。このコマンドが生成するビューが配置されるブラウザの名前を指定する。たとえば、.form コマンドが browser=Cities パラメタを定義していれば、このビューは Cities という名前のブラウザに配置される。

protoName オプショナル。ビューを作成する元になるシステムプロトタイプ。

layout_*nameOfLayout*

オプショナル。ビューを作成する元になるユーザープロトタイプ、ビューテンプレート、レイアウトあるいはユーザーテンプレート。これはその名前の前に layout_ を前置することで指定する。たとえば、mySpecialUserProto というテンプレートを指定するには、layout_mySpecialUserProto のようになる。

以下に、.form コマンドを使ってブックに protoTextButton ビューを取り込む方法を示す：

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

ビューを定義する NewtonScript コマンドは .form コマンドの後に続けて良い。このテクニックを使って、以下の例のように、ビューが生成されるときにそのスロットの値を初期化することができる：

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

システムプロトあるいはある種のビューテンプレートを使うかわりに、以下の例のようにしてビュー全体をプログラム的に生成することができる：

```
.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*

現在のコンテントアイテムの左マージンを *left* ポイントだけインデントし、右マージンを *right* ポイントだけインデントする。(1 ポイントは 1/72 インチ)

left コンテントアイテムの左マージンのインデント量を表すポイント数。

right コンテントアイテムの右マージンのインデント量を表すポイント数。

このコマンドを対象となるコンテントコマンドの後で、しかしそこに結びつけられたコンテント (テキストや絵) の前に置く。以下の例ではストーリーテキストを左 30 ポイント、右 20 ポイントインデントする :

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

Kiosk

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

名前付きキオスクページを定義する。これはブックコンテントをナビゲートするためのオプショナルなページである。キオスクページに関してより詳しくは、第 3 章「ブックメーカー言語の使用」の「キオスクの作成」セクションを参照。

`.kiosk` コマンドと `.endkiosk` の間では、`goto` フラグを使うあらゆるコンテントコマンドが、キオスクページのボタンとなる。`goto` フラグを持たないコンテントコマンドは全て単なるラベルまたは絵となる。全てのコンテントアイテムがキオスクの名前によって参照されうることに注意。

`name=` 必須。ブック内の他の構成要素がこのキオスクを参照するために使用できる *nameString* を指定する。たとえば、`.kiosk` コマンドが `name=Cats` パラメタを持っていれば、他のコンテントアイテムはそのパラメタとして `goTo=Cats` を指定することでキオスクを表示できる。

`layout=` オプショナル。キオスクページに適用するレイアウトの *layoutName* を指定する。名前でレイアウトを適用する場合、`layout=` パラメタはコマンドラインの最後のパラメタでなければならない。レイアウトとスタイルのオプションについては、この付録の「ページレイアウトコマンド」セクションを参照のこと。

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

注意

キオスクページに使用するためのレイアウトを作っておいた方がよい。このレイアウトには kiosk フラグを含んで置くこと。これにより、ブックリーダーは、ユーザーを最寄りのキオスクに導くようなボタンを自動的に提供してくれる。◆

Mark

`.mark`

現在のコンテントアイテムへの参照を保存する。`.usemark` コマンドを持つこれによって後続アイテムが、マークされたアイテムのスロットを参照できる。

他の `.mark` コマンドが `.usemark` 出現前に登場した場合は、最初のコンテントアイテムへの参照は失われる。つまり、`.usemark` コマンドはもっとも直前に有る `.mark` コマンドと対になる。`.usemark` コマンドについてはこのセクションの後の方で解説される。

Picture

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

このコマンドの後に続く 'PICT' をコンテントアイテムにすることを指定する。ブックソースファイルに 'PICT' をペーストする代わりに、外部の 'PICT' ファイルへのフルパス名を指定しても良い。

注意

`.picture` コマンド全体はブックソースファイルの単一行として入力されなければならない。十分大きく読みやすいフォントで印字する目的で、本書では二行に分けて書いている。◆

Newton ブックメーカーがページをレイアウトするとき、絵を格納するのに可能な限り大きな領域を与えようとするので、ユーザーが絵をスクロールするよう強制されることはない。たまに絵が関連する主題やストーリーと分かれて次のページに持ち越されたりするのはそのためである。この問題を `KeepWith` フラグを使うことによって、前のコンテントと絵を一緒に保ったり、絵のサイズをより小さくするため `width`, `height` フラグを使ったりして解消できる。`height`, `width` フラグは絵をスケーリングしないことに注意。これは絵の表示可能領域の境界を指定するだけの事である。Newton ブックリーダーは、指定された境界より絵のサイズが大きい場合はユーザーがスクロールを制御できるディスプレイコントロールを自動的に表示する。

flags このコンテントに関連づけられるオプションのコンテントフラグ。

`name=` オプショナル。この絵をブック内の他の構成要素から参照可能な *nameString* を指定する。たとえば、`.picture` コマンドが `name=Fred` パラメタを定義していれば、他のコンテントアイテムは `goTo=Fred` をそのパラメタとして指定することにより、この絵を表示することができる。

`goTo=` ユーザーがこのサブジェクトをブラウザ内でタップしたときに表示されるコンテントアイテムを指定する。たとえば、`.subject` コマンドが `goTo=Cats` パラメタを指定していて、ユーザーがブラウザ内で *subjextText* をタップすると `Cats` コンテントアイテムが表示される。

`browser=` オプショナル。このコマンドが生成するビューが配置されるブラウザの名前を指定する。たとえば、`.story` コマンドが `browser=Cities` パラメタを定義していれば、このビューは `Cities` という名前のブラウザに配置される。

`height` オプショナル。絵をイメージするために必要な空間の高さをピクセル単位で指定する。絵が指定されたサイズより大きい場合は、スクローラーコントロールが自動で表示される。

`width` オプショナル。絵をイメージするために必要な空間の幅をピクセル単位で指定する。絵が指定されたサイズより大きい場合は、スクローラーコントロールが自動で表示される。

`layout=` オプショナル。このコンテントアイテムに *layoutName* レイアウトが適用される。名前でレイアウトを指定するとき、

layout= パラメタはこのコマンドラインの最後のパラメタでなければならない。レイアウトとスタイルオプションに関しては、この付録の「ページレイアウトコマンド」セクションを参照のこと。

例 :

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

Script

`.script [slot]` *event*

現在のコンテントアイテムに対して、NewtonScript メソッドを割り当てる。スクリプトを含むファイルへのオプショナルなパス名を指定しても良い。`.script` と `.endsrcipt` コマンドの間にある全ての物はスクリプト本体を定義する NewtonScript コードでなければならない。

ドキュメントの最初 (コンテントやレイアウトコマンドの出現前) に定義されたスクリプトはブック全体で利用可能であり、ブックスクリプトと呼ばれる。レイアウトコマンドの後でレイアウトに割り当てられたスクリプトは、ページスクリプトと呼ばれる。ページスクリプトはそのレイアウトを使用する任意のページで利用可能である。

event `slot` キーワードが指定されていない場合、このパラメタはスクリプトを起動するためのイベントを指定する。(これがもっとも普通の構文である)。このパラメタのデフォルト値は `buttonClickScript` であるが、他に `viewDrawScript`, `viewClickSclipt`, `viewSetupScript` も指定して良い。

`slot` オプショナル。このキーワードがあれば、*event* パラメタで指定される名前が、コンテントアイテムの中に定義されるスロット名となる。

以下の例では現在のコンテントアイテムに、そのコンテントがタップされると `ROM_click` サウンドを再生するスクリプトを割り当てる :

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

以下の例では、現在のコンテントアイテムをイメージするビューに値 "blue" を格納するスロット color を割り当てる :

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

注意 : スクリプトスロットは慎重に注意深く使用すること。スクリプトスロットはそれが割り当てられたコンテントアイテムをイメージするビューの中に作成される。だから、それぞれのスクリプトスロットは NewtonScript のヒープをそれなりに消費する。◆

しかし関数がパラメタをとる場合、slot 引数は必須である。たとえば、ブックリーダーが FromSearchScript をコンテントアイテムに送る場合、searchStr, stringLen パラメタを一緒に渡す。FormSearchScrpt は以下の例のように slot 引数を使って定義されなければならない :

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

Space

`.space` *n*

現在のコンテントアイテムの後に *n* ポイントの空白を残す。(1 ポイントは 1/72 インチ)。コンテントアイテムがページの下端にあれば、次のページの先頭に空白が作られることはない。

n 現在のコンテントアイテムの後に残すべき空白の量。この値は 0–63 ポイントの値を取る。

ブックソースファイルに複数の空行を置く代わりに .space コマンドを使うことによって、ブックパッケージはわずかに小さくなり、空白がページの先頭に配置されないことが保証され、コンテントアイテム間に配置する空白の量をより正確に制御できるようになる。

Story

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

現在定義されているサブジェクト (があれば) その中に新たなコンテントアイテムを定義する。このコンテントアイテムはデフォルトで現在の subject よりも位置レベル深くなる。

注意

.story コマンドとは独立に記述されるストーリーテキストは別として、.story コマンド全体はブックソースファイル中で同一行に記述されなければならない。十分大きく読みやすいフォントで印字する目的で、本書では二行に分けて書いている。◆

flags このコンテントに関連づけられるオプションのコンテントフラグ。

name= オプショナル。このストーリーをブック内の他の構成要素から参照可能な *nameString* を指定する。たとえば、.story コマンドが name=myStory パラメタを定義していれば、他のコンテントアイテムは goTo=myStory をそのパラメタとして指定することにより、このストーリーを表示することができる。

goTo= ユーザーがこのサブジェクトをブラウザ内でタップしたときに表示されるコンテントアイテムを指定する。たとえば、.subject コマンドが goTo=Cats パラメタを指定していて、ユーザーがブラウザ内で *subjextText* をタップすると Cats コンテントアイテムが表示される。

browser= オプショナル。このコマンドが生成するビューが配置されるブラウザの名前 *brsNm* を指定する。たとえば、.story コマ

ンドが browser=Cities パラメタを定義していれば、このストーリーテキストは Cities という名前のブラウザに配置される。

height　オプショナル。絵をイメージするために必要な空間の高さをピクセル単位で指定する。絵が指定されたサイズより大きい場合は、スクローラーコントロールが自動で表示される。

width　オプショナル。絵をイメージするために必要な空間の幅をピクセル単位で指定する。絵が指定されたサイズより大きい場合は、スクローラーコントロールが自動で表示される。

layout=　オプショナル。このコンテントアイテムに *layoutName* レイアウトが適用される。名前でレイアウトを指定するとき、layout= パラメタはこのコマンドラインの最後のパラメタでなければならない。レイアウトとスタイルオプションに関しては、この付録の「ページレイアウトコマンド」セクションを参照のこと。

例 :

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

Subject

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

subjectText

ブックの目次に指定されたレベルで表示され、ブック本体にも表示される新たなコンテントアイテムを定義する。レベルが指定されなければ、レベル 1 のサブジェクトが生成される。このコマンドと次のコンテントアイテム定義コマンドの間にあるテキストあるいはグラフィックはサブジェクトコンテントアイテムとなる。サブジェクトテキストがブック本体に現れないようにするには、browserOnly フラグを追加する。

注意

.subject コマンドとは独立に記述される *subjectText* は別として、.subject コマンド全体はブックソースファイル中で同一行に記述されなければならない。◆

level ブラウザでこのサブジェクトが登場するレベル。1 を指定すれば、サブジェクトは `.chapter` コマンドと同義であり、2 はヘッドラインであり、3 はサブヘッドで ...

flags このコンテントに関連づけられるオプションのコンテントフラグ。

`name=` オプショナル。このストーリーをブック内の他の構成要素から参照可能な *aStr* を指定する。たとえば、`.subject` コマンドが `name=Cats` パラメタを定義していれば、他のコンテントアイテムは `goTo=Cats` をそのパラメタとして指定することにより、このコンテントアイテムを表示することができる。

`goTo=` ユーザーがこのサブジェクトをブラウザ内でタップしたときに表示されるコンテントアイテムを指定する。たとえば、`.subject` コマンドが `goTo=Cats` パラメタを指定していて、ユーザーがブラウザ内で *subjextText* をタップすると `Cats` コンテントアイテムが表示される。

`browser=` オプショナル。このコマンドが生成するビューが配置されるブラウザの名前 *brsNm* を指定する。たとえば、`.subject` コマンドが `browser=Cities` パラメタを定義していれば、このサブジェクトは `Cities` という名前のブラウザに配置される。

`layout=` オプショナル。このコンテントアイテムに *layoutName* レイアウトが適用される。名前でレイアウトを指定するとき、`layout=` パラメタはこのコマンドラインの最後のパラメタでなければならない。レイアウトとスタイルオプションに関しては、この付録の「ページレイアウトコマンド」セクションを参照のこと。

例 :

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

Usemark

`.usemark`

現在のコンテントアイテムに related という名のスロットを追加する。このスロットの値はもっとも近い .mark コマンドを持つ先行するコンテントアイテムへの参照である。これによって .usemark されたコンテントアイテムは .mark されたコンテントアイテムを検索することなくその中のスロットにアクセスできる。

例 :

```
.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
```

ブラウザコマンド

ブラウザコマンドは Newton ブックリーダーの目次に表示される新しいブラウザを定義する。

Browser

.browser *name type*

目次ブラウザ内に、指定された名前の新しいブラウザペーンを作成する。

name ブラウザメニューに表れるテキストを指定する。このブラウザペーンの名前である。このテキストはまた browser=*name* フラグを使ったコンテントアイテムにおいても使用される。

type ブラウザの定義方法を指定する。list あるいは form を値として指定できる。list を指定すれば、ブックメーカーは browser= フラグを使用するコンテントアイテムからブラウザアイテムのリストを構築する。form を指定すると、ブラウ

ザペーンビューの定義を .browser コマンドの後に続けなければならない。この定義ではシステム提供のブラウザペーンより大きな viewBounds を指定しない方が良い。

例 :

```
.browser Hotels list
```

ページレイアウトコマンド

ページレイアウトコマンドはページのための一般的なフォーマット特性を定義する。たとえばカラム数やグラフィックの配置などである。ページレイアウトコマンドはまた Newton デジタルブック内のテキストやグラフィックにスクリプトを関連づけるのにも使用される。

Layout

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

指定されたカラム数とカラム幅で名前付きのページレイアウトを作成する。ワープロなどで一般に使用されるユーザー定義スタイルのような物だと思えばわかりやすいだろう。Newotn スクリーン側の一ページにつき一つだけレイアウトを指定できる。

name 必須。ブック中の他の構成要素がこの文字列を使って名前でレイアウトを適用できる。たとえば、.layout コマンドが *myLayout* で名前を定義し、他のコンテントアイテムは layout=myLayout というパラメタを用いて myLayout を適用することができる。それぞれの .layout に異なる名前を適用する必要がある。.layout コマンドに同じ名前を繰り返すとエラーになる。

width 必須。このレイアウトで定義される段語との幅を指定する一つ以上の数字。パラメタとして指定された数字の合計は 12 までででなければならない。12 はブックメーカーが Newton の画面を縦に分割する数である。この分割幅は全て均等である。この分割された帯の実際の幅はブックを表示するデバイスのスクリーンサイズに依存する。レイアウトにおけるカラム幅は、こうした分割帯の数で指定される。だから、レイアウトの比

率は、使用されるデバイスのスクリーンサイズがどうなっていようと維持される。

flags このレイアウトを使うそれぞれのコンテントアイテムに適用されるオプショナルなコンテントフラグ。たとえば、レイアウトは `edges` フラグを含むことができ、このレイアウトを使う個々のコンテントアイテムの周囲には箱が描かれる。

layoutFlags `sidebar` とか `main` フラグを適用するためのオプションのレイアウトフラグ。これは指定された個々のカラム幅に対して、それがメインカラムになるのか、サイドバーになるのかといったことを指定するために使う。

典型的なレイアウトコマンドは次のような物である：

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

このコマンドは `twoColumn` という名のページレイアウトを定義する。`twoColumn` レイアウトは二つの等価なメインカラムを定義し、それぞれの幅は 6 単位である。`edges` フラグはその周りに箱を描く。別の例を示す：

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

このコマンドは `annotated` というレイアウトを定義し、これは二つのカラムを持つ。左は 4 単位の幅で、つまりページの 1/3 幅である。こちらはサイドバーである。右のカラムは 8 単位で、つまりページ幅の 2/3 である。こちらがメインカラム。

`.script` コマンドが `.layout` コマンドの後に続くとそのスクリプトはレイアウトに割り当てられる。この方法を使うと、スクリプトで背景に何か描いたり、コンテントアイテムが処理しないようなクリックイベントを処理したりできる。

コンテントコマンドが使用するレイアウトを指定しない場合、もっとも最後に定義されたレイアウトが適用される。

レイアウトを名前で適用する場合、`layout=` パラメタはコマンドラインの最後のパラメタでなければならない。

Header

`.header`

現在のレイアウトコマンドに、センタリングされたテキストヘッダを割り当てる。このテキストヘッダはそのレイアウトを使うそれぞれのページのトップに表示される。ヘッダのフォントとスタイルはレイアウトの他の部分とは別に維持される。

Pictheader

`.pictHeader`

現在のレイアウトコマンドに、ピクチャーヘッダを追加する。このヘッダはそのレイアウトを使うそれぞれのページのトップに表示される。16 ピクセル以上の高さを持つヘッダはコンテントビューの中にあふれ出してくる。ページコンテントはピクチャーヘッダの描画後に描画されるので、巨大なピクチャーヘッダで風変わりな背景を作り出せる。

Running

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

直前のレイアウトコマンドに絵 , ストーリー、フォームのいずれかのタイプのコンテントを割り当て、自動的にそのコンテントを毎ページに配置する。コンテントはページのトップ (で、かつヘッダの下) にデフォルトで配置される。

type ランニングヘッダとして使用されるコンテントアイテムのタイプ。このパラメタは `picture`, `story`, `form` のどれか一つを指定しなければならない。コンテントアイテムは、対応するコンテントコマンド (`.story`, `.picture`, `.form`) で定義されたコンテントコマンドと同じ扱いを受ける。

`pageBottom` このオプショナルフラグによって、コンテントはページの末尾に配置される。

flags ランニングヘッダコンテントアイテムに適用されるオプショナルなコンテントフラグ。

例 :

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

ランニングヘッダを作成するためにこのコマンドを使うことによって、同じデータをブックソースファイルのページごとにペーストするよりもブックパッケージは小さくなり、ブックにおけるランニングタイトルの作成により柔軟性を出すことができる。しかし、固定的な情報のために各ページに大きなスペースをとってしまわないかどうか確認しておく必要はある。

その他のコマンド

これらのコマンドを使って、インデックスを作ったり、巨大なファイルを複数の小さなファイルのセットに分割したり、コメントをソースファイル中に取り込んだりすることができる。

Chain

`.chain "`*fullPathToFile*`"`

ブックソースファイルの最後に記述して、このソースファイルの後に別のソースファイルが続いていることを示す。このコマンドを使って、巨大なブックソースファイルを小さな複数のファイルに分割することができる。

ブックの `.title` と `.isbn` コマンドはチェーンの最初のファイルにだけ表れなければならない。チェーンの全てのファイルは同じフォーマットでなければならない。たとえば、MacWrite ファイルに TeachText ファイルを続けることはできない。`.chain` コマンドで指定されたそれぞれのファイルをブックメーカーが開くとき、前のファイルは閉じられる。

fullPathToFile チェーン中の次のファイルに対するフルパス名。パス名は引用符で囲んでおかなければならない。

例 :

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

このコマンドは巨大なブックソースファイルを、より扱いやすい断片に分割するのに有用である。複数のメンバーで同時に編集を行うときには便利であろう。これはまた、ブックメーカーがソースファイル全体を取り扱えないほど極端に大きいブックファイルを分割するのにも使用できる。(ブックメーカーはブックソースファイル全体を RAM に読み込んで処理しよう

とする。このコマンドを使えば、巨大なブックを処理する際に必要な RAM 容量も減らすことができる。)

Comment Symbol

`.#` *commentText*

はこの後に続くテキストが、スクリーン上には表示されないコメントラインであることを表す。

commentText　コメントとなるテキスト。

例 :

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

Index

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

アルファベット順のインデックスを作成し、そこに指定したエントリを追加する。現在、インデックスデータは NewtonScript からのみ利用可能であるが、将来の Book Maker ではこのデータをエンドユーザーにも提供する予定である。

entry　インデックス *indexName* に追加する名前。ブックが複数のインデックスを持っている場合、特定のインデックスと指定するため名前が使用される。

それぞれのインデックスエントリは `.index` コマンドが使用されたコンテントアイテムへの参照を格納している。ブックメーカーがファイルを処理する時、それはコンテントアイテムのアルファベット順に並んだ配列を出力し、それを `alphaIndex` というブックデータのスロットに格納する。また、別の `subIndex` という名のスロットに第二の配列も出力する。これはアルファベット各文字についての `alphaindex` へのインデックス配列である。`.option noSubIndex` コマンドを使えば、サブインデックスの生成を抑制できる。

!*indexName*　引数は単一のブックに複数のインデックスを生成する。*indexName* パラメタの値は特定のインデックスを識別する NewtonScript 変数を作る。インデックスは *indexName*`Index` という名でブックデータフレームに格納される。

*indexName*SubIndex もまた作成される。@*indexName* 構文を使用すれば、サブインデックスの生成を抑制できる。特定の *indexName* は、.index !*indexName* と .index @*indexName* 形式のどちらでも一つのブックの中に登場できる。この場合、最初に *indexName* が登場すれば、*indexName*SubIndex が生成されることを指定する。@*indexName* シンタックスの方が最初に使われると、*indexName*SubIndex 配列は生成されない。

注意

*indexName*Index と *indexName*SubIndex は NewtonScript のシンボルとなる。だから、*indexName* には英数字もしくはアンダースコア以外の文字を含んではいけない。他の文字は、それが縦棒 (|) に囲まれていても使うことができないので、'|weird%symbol!| は NewtonScript のシンボルとしては正しいが、'|weird%symbol!|Index は正しくない。◆

Option

.option *optionName*

現在のブックに指定したオプションを有効にする。.option コマンドはそれぞれのアプリケーションオプションに関して一回だけ使われるべきである。

optionName 有効にするオプションを指定する。このパラメタは以下の値だけを持つ :

firstPage 可視なそれぞれのコンテントアイテムに対して firstPage という名のスロットを追加する。firstPage スロットはそのアイテムが登場するページ番号を含む。

indexedPage

.index コマンドが生成したインデックスにも登録されている可視なコンテントアイテムそれぞれに対して firstPage という名のスロットを追加する。firstPage スロットはそのアイテムが登場するページ番号を含む。

noSubIndex .index コマンドによって作成されるインデックスのためのサブインデックス配列を作らない。

例 :

```
.option noSubIndex
```

注意

`.option indexedPage` は `.option firstPage` のサブセットであることに注意。前者は、後者がスロットを追加するコンテントアイテム群のサブセットに対してスロットを追加する。同一のブックで `.option firstPage` と `.option indexedPage` コマンドを使うのは余分なことである。◆

将来的に、異なるサイズのスクリーンが使用可能になったら、`firstPage` スロットはページ番号の配列を含むことになる。その時には `curRendering` スロットを使えば、この配列から正しいページ番号を選択することができるようになるだろう。

フラグ

ブックメーカー言語はドキュメント、コンテント、レイアウトコマンドを修飾するキーワードを提供する。これらのキーワードはフラグと呼ばれ、単一もしくは組み合わせて様々な効果を得るのに使用される。このセクションでは Book Maker コマンド言語で利用可能なフラグについて述べる。

ドキュメントフラグ

ドキュメントフラグは個々のコンテントアイテムもしくはレイアウトに影響するというよりもブック全体に効果を及ぼす。表 A–1 には Newton BookMaker コマンド言語のバージョン 1.0 において有効なドキュメントフラグについて述べる。

表 A–1　ドキュメントフラグ一覧

フラグ	説明
`NoReLayout`	スクリーンサイズが変更されても再フォーマットを行わない。

レイアウトフラグ

レイアウトフラグは `.layout` コマンドを修飾する。だから、そのレイアウトを使うあらゆるコンテントアイテムもまたこのフラグによって影響を受

ける。表 A–2 は Newton Book Maker コマンド言語のバージョン 1.1 において有効なレイアウトフラグを示す。

表 A–2　レイアウトフラグ一覧

フラグ	説明
`Kiosk`	このレイアウトコマンドフラグはキオスクページレイアウトを定義する。
`Main`	メインカラムに表示される。(デフォルト)
`NoTitle`	ランニングタイトル用のスペースを確保しない。
`Sidebar*`	サイドバーカラムに表示する。

注意
特定のレイアウトフラグはまたコンテントアイテムを修飾するのにも使用される。詳しくは表 A–3 参照。◆

コンテントフラグ

ブックメーカー言語のほとんどのフラグはコンテントアイテム定義コマンドを修飾するために使用される。コンテントアイテムとは `.story, .picture, .subject` 等である。だから、これらはコンテントフラグと呼ばれる。

ある種のコンテントフラグはレイアウトコマンドの修飾にも使用される。そうしたフラグはアスタリスクでマークされている (アスタリスクはフラグの一部ではない)。レイアウトはそれを使用するページ全体に適用されるので、これらのフラグはレイアウトの修飾に使用されると、ページ全体に影響を及ぼす。

表 A–3　コンテントフラグ一覧

フラグ	説明
`Main`	メインカラムに表示する (デフォルト)。
`Sidebar*`	サイドバーカラムに表示する。
`ToEdge`	現在のレイアウトをオーバーライドし、コンテントがそのカラムの端を越えてページの端まで流れるようにする。

`Centered`	カラム内でセンタリング。
`NeverBreak`	テキスト内にページ区切りを入れない。
`StartsPage`	常に新しいページのトップに配置する。
`PageMiddle`	常に新しいページの中央に配置する。
`PageBottom`	常に新しいページの下端に配置する。
`AlignTop`	サイドバーを直前のコンテントアイテムと上端揃えする。
`AlignBottom`	サイドバーを直前のコンテントアイテムと下端揃えする。
`AlignCenter`	コンテントアイテムがサイドバーに有れば、直前のコンテントアイテムと中央揃えする。さもなくばページの中央で揃える。
`BrowserOnly`	コンテントがブラウザペーンにのみ登場するようにする。ブックのページには表示されない。
`BrowserAutoClose`	選択が行われれば、ブラウザが自動的に閉じられるようにする。システム提供のオーバービューのような動作を提供する。
`Overlay`	直前のコンテントアイテムと、オーバーレイする。
`KeepWith`	直前のコンテントアイテムと同一ページになるようにする。
`NoExtend`	コンテントブロックのページ下端までの自動拡張を抑制する。キオスクにおいてページ全体は占めるが、ページ全体を塗りつぶしたくないと言うような、`viewClickScript` メソッドの、不要なハイライト動作をさけるには有用である。
`NoPage`	このコンテントを自動的にはページに表示しないが、後で表示できるよう情報は保持しておく。コマンドによってのみ表示されるようなストーリーカードでは有用である。

NoScroller	カラムより大きいグラフィックに対する、自動的なスクローラ表示を抑制する。
NoSearch	このコンテントテキストを検索対象にしない。テキストサーチの高速化に使用される。

エッジフラグ

エッジフラグはコンテントアイテムの周縁に線を描くのに使用する特別なコンテントフラグである。エッジフラグは表 A–4 に記述されている。ほとんどのエッジフラグはコンテントアイテムと同様レイアウトも変更するのに使用される。

表 A–4　エッジフラグ一覧

フラグ	説明
TopEdge	コンテントアイテムの上のエッジに線を描く。
LeftEdge	コンテントアイテムの左のエッジに線を描く。
BottomEdge	コンテントアイテムの下のエッジに線を描く。
RightEdge	コンテントアイテムの右のエッジに線を描く。
Edges	コンテントアイテムの周囲に箱を描く。
Round	コンテントアイテムの周囲に角丸の箱を描く。
Reverse	黒塗りの上に白抜きのコンテントアイテムを表示する。
EdgeWidth	エッジフラグで描かれる線の幅を指定する。

エッジフラグで描画される線の幅をピクセル単位で指定するには edgeWidth フラグを使用できる。edgeWidth フラグは次の例のように、エッジフラグと同じ行に記述する：

```
edgeCommand edgewidth=number
```

NewtonScript メソッド

以下の NewtonScript メソッドが Newton ブックメーカーによって提供され、実行時に利用可能である。

AddBookmark

`:AddBookmark (`*pageNumber*`)`

現在のブックの指定されたページにブックマークを追加する。一つのブックには６つしかブックマークをつけれない。

pageNumber　ブックマークするページ番号。

AddBookRouting

`:AddBookRouting(`*routingArray*`)`

指定した通りにルーティングメニューを追加したり削除したりする。このブックが追加した全てのアイテムを削除するには、*routingArray* パラメタに `nil` を渡す。

routingArray　追加すべきルーティングフレームの配列。ルーティングフレームは以下のスロットを持っていなければならない。

`title`　必須。この文字列がルーティングメニューに表れる。

`icon`　オプション。この関数がルーティングメニューに追加する文字列の横に表示されるアイコン。

`routeScript`

メニューアイテムのルーティングスクリプトとなるページスクリプトもしくはブックスクリプトを指定する。

ルーティングに関してより詳しくは*プログラマーズガイド*を参照。

AddBrowser

`:AddBrowser(`*browser*`)`

ブックに指定されたブラウザを追加する。このメソッドはブラウザへの参照を返す。これはブラウザを表示したり削除したりする :OpenBrowser と :CloseBrowser メソッドへのパラメタとして使用される。

browser ブラウザ定義を含むフレーム。このフレームは以下の二つのスロットを持っていなければならない :

name ブラウザペーンのトップに表示される名前を表す文字列。

list ブラウザの一行ごとに、何を表示してタップされたときにどこにジャンプするかという情報を持ったフレームの配列。

list 配列内のフレームは以下のスロットを持つ必要がある :

level オプショナル。このエントリがブラウザの何レベル目に表示されるかを指定する。これは .subject コマンドに対するオプショナルな level 引数と同様の効果を持つ。level 引数についてはページ A–22 参照。このスロットがない場合は、行はレベル 1 に表示される。

item 必須。ブラウザ行がタップされたときに表示されるコンテントアイテムへの参照。

name このアイテムがテキストコンテントアイテムへの参照である場合はオプショナル。ブラウザ行に表示されるテキストを指定する文字列。*item* が参照するコンテントアイテムがテキストコンテントアイテムで有れば、このスロットは省略でき、そのテキストがブラウザ中に表示される。

第 4 章「ブックでの NewtonScript」のページ 4–24 からはじまる「動的ブラウザの作成」セクションと、この付録中のページ A–37「CloseBrowser」、ページ A–43 「OpenBrowser」の記述も参照のこと。

AuthorData

```
:AuthorData()
```

ブックスクリプトの作者がデータを格納できるスープベースのフレームへの参照を返す。このデータは Newton デバイスのスープ上に存在することになる。このスープエントリはブックが削除された後も維持される。

重要

ブック内部に存在するデータをこのフレームにおいては成らない。ブックはいつ削除されるかわからないので、ブックからコピーされたデータはデータフレーム全体を不正な物にしてしまう可能性がある。▲

BookData

```
:BookData()
```

ブックスクリプトの作者がブック特有のデータを格納できるデータフレームへの参照を返す。たとえば、`.index` コマンドは `alphaIndex` と `subIndex` 配列を個々に格納している。このデータはブックとともにコンパイルされ、ブックパッケージの一部となる。ここに保持されているあらゆるデータは実行時には読み込み専用の物となる。

Bookmarks

```
:Bookmarks ()
```

現在のブックのブックマークの配列を返す。

[*number*, …]

BookTitle

```
:BookTitle ()
```

現在のブックのタイトルを含むスロットを返す。

ChangeScrolledOrigin

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

ブックリーダーの組み込みスクローラーによって表示されるピクチャーのスクロール位置を設定する。`.picture` によって作られた全てのビューで、スクリーンよりも大きい物は `scroller` スロットを持ち、このスロットに `ChangeScrolledOrigin` メッセージを送信する。

dX 水平方向へのスクロールピクセル数。正の値は可視のエリアを右にオフセットする。負の値は左にオフセット。

dY 垂直方向へのスクロールピクセル数。正の値は下方向へ、負の値だと上方向へ可視領域をオフセットする。

CloseBrowser

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

指定されたブラウザが開いていればそれを閉じる。

browserRefNum 閉じたいブラウザへの参照。`AddBrowser` 関数が返してくるのと同じ形式。

CountPages

`:CountPages ()`

現在のブックのページ数を返す。

CurrentKiosk

`:CurrentKiosk ()`

現在表示されているページからページ番号が若いページに向かって探索したときに見つかった最初のキオスクページへの参照を返す。キオスクが見つかればこのメソッドはフレームを返し、さもなくば `nil` を返す。このメソッドが返してくるフレームは次のスロットを持つ：

`page` このキオスクが表示されるページ番号。

`name` キオスク定義において、`name=` パラメタでキオスクに割り当てられた名前。

CurrentPage

`:CurrentPage ()`

現在のブックで現在表示されているページを返す。

Find

`:Find (`*string*`, results, `*scope*`, `*statusContext*`)`

一つまたはそれ以上のブックで指定された文字列を検索する。

string `Find` がブックコンテントを検索する際のユーザー指定文字列。

`results` システムによって `Find` に渡されるスロットの配列。`Find` メソッドはスロット `found` をこの配列に追加する。`hits` 配列

はそれぞれの見つかったアイテムが登場するページ、それが表れるブロック、アイテムの長さ、ブロック中でのオフセット文字数を含む。グローバル `find` が進行すると `results` 配列は他のアプリケーションのサーチメソッドが作り出したスロットを含む。

scope この引数の値は常にシンボル `'local` か `'global` のどちらかとなる。`'local` はそのブックリーダーが現在のブックを検索することを指定する。`'global` はブックリーダーサーチが全てのブックを検索することを指定する。

statusContext ブックリーダーが `SetStatus` メッセージを送信するフレーム。`SetStatus` 関数は、検索が進行中にその進捗度をユーザーに報告する。このメソッドは `results` に配列を返す。`results` は指定された文字列が見つかったブックごとに一つのエレメントを持つ。この配列のそれぞれのエレメントは以下のサンプルに示すスロットから成るフレームである：

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

`Find` メソッドについてより詳しくは、*プログラマーズガイド*の「追加のシステムサービス」の章を参照のこと。

FindContentBySlot

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

現在開いているブックから *aSymbol* パラメタで指定される名前のスロットを持つコンテントアイテムの配列を返す。

aSymbol 検索するスロットの名前。

aDepth 検索の深さを指定する。*aDepth* パラメタの値が `nil` の場合、このメソッドは検索条件にマッチする全てのコンテントアイテムを返す。*aDepth* パラメタが `true` なら、このメソッドは検索条件に合致した最初のコンテントアイテムだけを返す。

Assist スクリプトは、ブックが開いていないときにインテリジェントアシスタントが使用するブックフレームを提供するために、*aDepth* パラメタを使用できる。ブックフレームに関してより詳しくは第 4 章「NewtonScript」の「ブックデータ」セクション参照。

FindContentByValue

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

現在開いているブックから *aSymbol* で指定される名前を持ち、*aValue* で指定される値を持つコンテントアイテムの配列を返す。

aSymbol 検索するスロットの名前。

aValue 指定された名前のスロット内でマッチすべき値。

aDepth 検索の深さを指定する。*aDepth* パラメタの値が nil なら、このメソッドは検索条件に合致する全てのコンテントアイテムを返す。*aDepth* パラメタが ture なら、このメソッドは検索条件に合致する最初のコンテントアイテムだけを返す。

Assist スクリプトは、ブックが開いていないときにインテリジェントアシスタントが使用するブックフレームを提供するために、*aDepth* パラメタを使用できる。ブックフレームに関してより詳しくは第 4 章「NewtonScript」の「ブックデータ」セクション参照。

FindContentsByPage

```
:FindContentsByPage(aPage)
```

このメソッドは指定されたページ上にあるすべてのコンテントアイテムの配列を返す。

aPage この関数はこのページからコンテント返す。

FindPageByContent

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

anOffset 位置に *aContent* の文字列が発見されたページを返す。

aContent このメソッドが探索を行うコンテントアイテム。

anOffset *aContent* の中で、何文字目から検索を始めるかという数。

二番目のパラメタ *anOffset* は重要である。なぜなら、コンテントアイテムはしばしば複数のページに分割されるからである。このパラメタを 0 にすると *aContent* が最初に配置されるページ番号を返す。*aContent* が１０段落の長さで、6 段落目がどこにあるのかを知りたい場合、最初の５段落に含まれる文字数を *anOffset* の値として渡せばよい。

FindPageBySubject

`:FindPageBySubject(`*subjectNum*`)`

指定したサブジェクトが登場するページ番号を返す。

subjectNum 位置づけるサブジェクトの番号。サブジェクトはブック定義ファイルの最初から連続的に番号付けられる。ブック定義ファイル内の最初のサブジェクトの番号は１である。

FindPageByValue

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

現在開いているブックから、指定されたスロットに指定された値を持つコンテントアイテムを含むページ番号の配列を返す。

aSymbol 検索するスロットの名前。

aValue 指定された名前のスロットの値とマッチさせる値。

aDepth 検索の深さを指定する。*aDepth* パラメタの値が `nil` なら、このメソッドは検索条件に合致する全てのコンテントアイテムを返す。*aDepth* パラメタが `ture` なら、このメソッドは検索上限に合致する最初のコンテントアイテムだけを返す。

`Assist` スクリプトは、ブックが開いていないときにインテリジェントアシスタントが使用するブックフレームを提供するために、*aDepth* パラメタを使用できる。ブックフレームに関してより詳しくは第 4 章「NewtonScript」の「ブックデータ」セクション参照。

GetLibraryEntry

`:GetLibraryEntry(`*isbn*`)`

指定されたブックのライブラリエントリフレームを返す。このメソッドが返すフレームは以下に示すスロットを含む :

`bookPresent` この値はブックが現在オンラインなら非 0 である。

`title` ブックタイトル。

`isbn` ブックの isbn 文字列。

`curPage` ブックで開いているページ番号。

`prevPage` `curPage` の前のページ。

`inkMarks` インクマーク配列の配列。`rendering` ごとに一つ。

`marks` ブックマーク配列の配列。`rendering` ごとに一つ。

`data` `author` データフレーム。

`curRendering`

ブックの現在の `rendering` を示す整数。値が 0 なら、ブックはメッセージパッドサイズのスクリーン上でレンダリングされていることを意味する。

`flags` ブックレベルのフラグ。

`packageID` ブックパッケージの ID。

GetOfflineBooks

`:GetOfflineBooks()`

Newton 上に記憶されているが、現在利用不可能な全てのブックの配列を返す。

GetOnlineBooks

`:GetOnlineBooks()`

Newton 上に現在存在する全てのブックの配列を返す。

HiliteBlock

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

aContent の *anOffset* 位置から *aLen* 文字だけコンテントアイテムに含まれるアイテム (通常はテキスト) をハイライトする。

aContent ハイライトするテキストを含むコンテントアイテム。

anOffset ハイライトするテキストに先行する文字列の文字数。*anOffset* の値はコンテントアイテムの最初から数える。それは、現在のページがコンテントの一部しか表示していなくてもである。

aLen ハイライトする文字の数。*anOffset* の位置にある最初の文字から数えての数である。

HideStoryCard

`:HideStoryCard ()`

現在表示されているストーリーカードを隠す。

InsertForm

`:InsertForm (`*aForm*`)`

現在のページに指定されたビューを追加する。そして、挿入されたビューを返す。

aForm ページに追加するシステムプロト、ビューテンプレート、レイアウトあるいはユーザーテンプレート。

指定されたビューは `InsertForm` メソッドによって暗黙の内に可視にされたりはしない。以下の例のように自分自身で可視にする必要がある：

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

OpenBook

`:OpenBook (`*isbnStr*`)`

指定されたブックを開く。

isbnStr 開きたいブックの ISBN 文字列。

OpenBookTo

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

指定されたブックのサブジェクトを開く。

isbnStr 開きたいブックの ISBN 文字列。

subjectNum ジャンプ先のサブジェクトの番号。サブジェクトはブック定義ファイルの最初から連続番号が振られる。ブック定義ファイルの最初のサブジェクトの番号は 1 である。

OpenBrowser

```
:OpenBrowser (browserRefNum)
```

指定されたブラウザを開く。

browserRefNum `AddBrowser` メソッドによって作成されたブラウザへの参照。

PageSize

```
:PageSize (nil)
```

現在表示されているページの高さと幅を返す。このメソッドは次のフレームを返す :

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

PageThumbnail

```
:PageThumbnail (aPage)
```

指定されたページの 1/8 サイズのビットマップビューを生成する。以下のフレームを返す :

```
{bits: <page÷8>, bounds: {}}
```

aPage サムネールビューとしてイメージするページの番号。

PreviousPage

```
:PreviousPage ()
```

現在のブックが表示しているページの前のページを返す。

RegisterBookRef

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

このメソッドはシステムにヘルプブックを登録する。ブックが独立パッケージとしてダウンロードされた場合はこのメソッドは呼ぶ必要がない。

isbnStr 開きたいブックの ISBN 文字列。

bookRef ヘルプブックのブックフレームへの参照。

ページ A–46 の「UnRegisterBookRef」も参照のこと。

RemoveBrowser

```
:RemoveBrowser(browserRefNum)
```

指定されたブラウザを削除する。

browserRefNum 削除したいブラウザへの参照。AddBrowser 関数が返してくるのと同じもの。

ScrollPage

```
:ScrollPage (aDelta)
```

現在のページから *aDelta* ページだけスクロールする。

aDelta スクロールするページ数。*aDelta* を 0 にすると、現在のページが再描画される。

ScrollReceiver

```
:ScrollReceiver(aView);
```

ユーザーが組み込みの矢印をタップしたときに生成されるスクロールイベントを、ページ組み込みの独自のビューで使うために横取りすることを可能にする。ビューは viewScrollUpScript と viewScrollDownScript を、送信されてくるメッセージを処理するために持たなければならない。

aView スクロールメッセージが送信されるビュー。nil を渡すとブックのスクロールメッセージは利用不能となり、組み込み矢印の制御はブックリーダーに返される。

SetStatusButtons

```
:SetStatusButtons({left: [btn1, btn2], right:[btn]})
```

ステータスバーに指定されたボタンを追加する。

`left` ページ番号ボタンの左に表れるテンプレートの配列。空の配列を渡すと、そちらには何もボタンは追加されず、前回の `SetStatusButtons` 呼び出しで生成された既存のボタンは全て削除される。

`right` ページ番号ボタンの右に表れるテンプレートの配列。空の配列を渡すと、そちらには何もボタンは追加されず、前回の `SetStatusButtons` 呼び出しで生成された既存のボタンは全て削除される。

ShowStoryCard

`:ShowStoryCard (`*aSymbol*`, `*aValue*`, `*aBounds*`)`

スロット *aSymbol* を持ち、*aValue* を格納するコンテントアイテムを、指定された *aBounds* に基づくサイズと位置を持つウィンドウ内に表示する。コンテントアイテムはどんなタイプのコンテントであってもよく、それはストーリー、ピクチャーも含む。一度に一つのストーリーカードだけが表示できる。

aSymbol 検索するスロットの名前。

aValue 指定された名前を持つスロットの値と合致する値。

aBounds ストーリーカードのサイズと、それを表示したい位置を指定する `viewBounds` フレーム。

TurnToContent

`:TurnToContent (`*aSymbol*`, `*aValue*`)`

aSymbol スロットを持ち、*aValue* を持つコンテントアイテムを含む最初のページを表示する。

aSymbol 検索したいスロットの名前。

aValue 指定された名前を持つスロットの値と合致する値。

TurnToPage

`:TurnToPage (`*aPage*`)`

指定されたページを表示する。

aPage　　表示対象となるページの番号。

TurnToSubject

`:TurnToSubject(`*subjectNum*`)`

指定されたサブジェクトが存在するページに移動する。

subjectNum　　ジャンプ先のサブジェクトの番号。サブジェクトはブック定義ファイルの最初から連続番号が振られる。ブック定義ファイルの最初のサブジェクトの番号は 1 である。

UnRegisterBookRef

`:UnRegisterBookRef(`*isbnStr*`)`

このメソッドはシステムからヘルプブックを登録解除する。`RegisterBookRef` を呼び出したので有れば、アプリケーションの `removeScrpt` から `UnRegisterBookRef` を呼び出す。

isbnStr　　ヘルプブックの ISBN 文字列。

ページ A–43 の `RegisterBookRef` 参照。

WhereIsBook

`:WhereIsBook(`*isbn*`)`

指定されたブックのライブラリエントリとブックフレームを含むフレームを返す。

isbn　　ブックの ISBN 文字列。

このメソッドが返すフレームは以下のスロットを持つ :

`library`　　指定されたブックのためのライブラリエントリフレーム。ライブラリエントリフレームの完全な解説については、ページ A–40 の `GetLibraryEntry` メソッドの説明を参照。

`bookSoup`　　指定されたブックのためのブックフレーム。ブックフレームはブックを構成するデベロッパ提供のコードと全てのデータを含む。

ブックリーダーメッセージ

このセクションではシステムがデジタルブックに送信するメッセージの一覧を示す。これらはブックの適切なビューにメッセージと同名のメソッドを提供することによって処理できる。これらのメッセージによってブックは実行時に発生した特定の状況に応答するアクションがとれる。

Book で始まるメッセージはブックに送信される。メッセージを処理するスクリプトはブックレベルで、レイアウトやコンテントアイテムの定義が行われるより前に書かれなければならない。Book から始まらないメッセージはコンテントアイテムに対して送信される。

メッセージがパラメタを持っている場合、ページ A–18「Script」で述べるように、メソッドは .script コマンドに slot パラメタを使用して実装されなければならない。

BookInstallScript

`BookInstallScript(`*bookFrame*`)`

ブックパッケージがインストールされたときにブックに送信される。

bookFrame book スロットを含むフレーム。book スロットはブック全体の最上位のフレームである。

BookOKClose

```
BookOKClose();
```

ユーザーがブックのクローズボックスをタップしたときにブックに送信される。BookOkClose メソッドはブックが閉じて良いなら true を返し、そうでない場合はブックが閉じないようにするため nil を返す。

BookRemoveScript

`BookRemoveScript(`*removeFrame*`)`

ブックパッケージが削除されたときにブックに送信される。

removeFrame ISBN スロットだけを持つフレーム。ISBN は削除されるブックの .isbn の値を含む。

BookHideScript

`BookHideScript();`

ブックが閉じたときに実行されるオプショナルなメソッド。

BookPrint

`BookPrint();`

ブックが提供するオプショナルなメソッド。これはプリント時に有用なあらゆるデータを返す。システムはこのメソッドが返したデータをブックの `printData` スロットに格納する。

ブック内のページの実際のプリントはたくさんのページ変更あるいはブックが閉じた後に起こりうるので、`BookPrint` メソッドはプリントスリップが開いたときのデータをそのまま格納するように成っている。このデータはページが実際にプリントされるときに、読者のスクリプトからアクセスされる。このメソッドは特にスクロール位置や状態をリストアするのに使用されるので、正しい物がプリントされる。

例：

```
.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　検索対象の文字列。

stringLen　*searchStr* の長さ。

theContent　検索対象のコンテントへの参照。

bookData　book.data から。

bookFrame　ブックフレーム全体。

このメソッドは bookSearchScript メッセージを受信したブック上でのカスタム検索を実行する。このメソッドはブックに割り当てられるものである。つまり、以下のコードは全てのレイアウトもしくは全レイアウト、全コンテントアイテムより前に記述されなければならない：

```
.script bookSearchScript
func ( searchStr, stringLen, theContent, bookData,
        bookFrame)
begin
// Perform a search of theContent
end
.endscript
```

ブックリーダーは NoSearch フラグの付いていない全てのコンテントアイテムに対して bookSearchScript を送信する。

BookSearchScript がシステム提供のブックリーダーサーチエンジンを、このコンテントのサーチに必要とするなら、nil を返せばよい。スクリプトがサーチを処理して何にもヒットしなかった場合、true を返す。スクリプトがサーチを処理し、なにかマッチした物が有れば、このメソッドは以下に述べるようなスロットを含む結果フレームを返さなければならない：

len　見つかった文字数。ブックが見つかった物をハイライトする機能を提供する場合、カスタムハイライト関数での利用に便利な値を len スロットに含めることができる。

char　見つかった文字列位置のオフセット数。ブックが見つかった物をハイライトする機能を提供する場合、カスタムハイライト関数での利用に便利な値を char スロットに含めることができる。

title　Find オーバービューに表示するテキスト。

BookShowScript

```
BookShowScript();
```

ブックがオープンしたときにそこに送信される。

FormHiliteScript

`FormHiliteScript(`*anOffset*`, `*aLen*`)`

見つかったアイテムをハイライトするのに必要なアプリケーション指定のアクションを実行する。このメソッドはコンテントアイテムに割り当てられる。

anOffset　見つかった文字列の最初の文字の位置。

aLen　ハイライトされる文字数。`FormSearchScript` メソッドからの戻り値あるいは、`.form` コンテントアイテムの `text` スロット内で見つかった物。

FormSearchScript

`FormSearchScript(`*searchStr*`, `*stringLen*`)`

searchStr　検索対象文字列。

stringLen　*searchStr* 文字列の長さ。

`formSearchScript` メッセージを受信した `.form` コンテントアイテム上でのカスタム検索を実行する。これはコンテントアイテムに割り当てられるべきである。このメソッドは何も見つからないときは `nil` を返す。最低一つでもターゲット文字列が見つかれば、このメソッドは以下のスロットを含む結果フレームを返さなければならない：

`len`　見つかった文字数。ブックが見つかった物をハイライトする機能を提供する場合、`len` スロットの値にカスタムハイライト関数での利用に便利なものを入れることができる。

`char`　見つかった文字列位置のオフセット数。ブックが見つかった物をハイライトする機能を提供する場合、カスタムハイライト関数での利用に便利な値を `char` スロットに含めることができる。

`title`　`Find` オーバービューに表示するテキスト。

MungeContentScript

`MungeContentScript (`*contentRef*`)`

ブックリーダーがコンテントアイテム内のデータへのアクセスを必要とする時にいつでもコンテントアイテムに送信する。つまり、コンテントアイテムがイメージ・検索・プリント・ブックマークされるときである。このメッセージを処理するメソッドは、コンテントアイテムを返さなければならない。このメソッドはたとえば、以下の例のように実行中にデータを解凍する時などに使用される：

```
.script slot MungeContentScript
func (aContent)
begin
    local newContent := Clone(aContent);
    newContent.data := :DecompressData(newContent.data);
    newContent;
end
```

ThumbNailScript

`ThumbNailScript();`

ブックマークするときにコンテントアイテムに送る。これはカスタマイズされたブックマークへのビューテンプレートを返す必要がある。

例：

```
.script thumbnailScript
local ocean:= {_proto: protostatictext,
               text: "A big aqua one",
               viewFont: ROM_fontsystem14bold};
return ocean;
.endscript
```

NewtonScript のグローバル関数

いくつかの NewtonScript グローバル関数が、ブックリーダーへのアプリケーションインターフェースを提供する。これらの関数はアプリケーションからの呼び出しのためのものであり、デジタルブックから呼び出すための物ではない。

BookAvailable

`BookAvailable ({book:` *bookFrame*`}, 0)`

デジタルブックを利用可能にする。ブックのインストールに問題が有れば `nil` を返し、インストール成功なら `ture` を返す。

bookFrame ブックのもっとも外側のフレーム。このフレームはデジタルブックを定義する全ての情報を含む。これはブックメーカーによってそのブック定義ファイルに生成されたものである。

BookRemoved

`BookRemoved ({isbn:` *isbnStr*`})`

指定されたブックを削除する。この関数は、何も削除されるブックがなかったときには 0 を返す。

isbnStr 開きたいブックの ISBN 文字列。

OpenHelpBook

`OpenHelpBook (`*isbnStr*`)`

指定されたヘルプブックを開く。

isbnStr 開きたいブックの ISBN 文字列。

OpenHelpBookTo

`OpenHelpBookTo (`*isbnStr*`, `*topic*`)`

特定のトピックに対する指定されたヘルプブックを開く。これはヘルプブックを開き、そのトピックをブラウザでタップしたのと同じ効果である。

isbnStr 開きたいブックの ISBN 文字列。

topic ヘルプブラウザで表示したいトピックを指定する文字列。これはヘルプブック中の `.subject` ラインのどれかとマッチしなければならない。

OpenHelpTo

`OpenHelpTo (`*topic*`)`

指定されたトピックに対するシステム提供のヘルプブラウザを開く。

topic NewtonOS2.0 では以下の文字列が指定可能 : `"write", "draw", "notepad", "names", "dates", "todo", "extras", "iobox", "prefs", "exchange", "route", "find", "assist"`

ShowManual

`ShowManual ()`

システム提供のヘルプブラウザを開く。

コマンド、関数、メソッドのサマリー

このセクションでは全てのブックメーカーコマンド、NewtonScript 関数、メソッドのリストを示す。

ブックメーカーコマンド

以下に示されるコマンドのいくつかは NewtonScript プログラミングとともに使用される。そのようなものは *NSrequired* とマークされている。

ドキュメントコマンド

`.assist` *NSrequired*

`.author`

`.blurb`

`.copyright`

`.date`

`.endassist` *NSrequired*

`.endpostamble` *NSrequired*

`.endpreamble` *NSrequired*

`.expires`

`.key`

.isbn

.postamble *NSrequired*

.preamble *NSrequired*

.publisher

.shortTitle

.title

コンテントコマンド

.attribute *NSrequired*

.chapter

.endform *NSrequired*

.endscript *NSrequired*

.endkiosk

.form *NSrequired*

.indent

.kiosk

.mark *NSrequired*

.picture

.script *NSrequired*

.space

.story

.subject

.usemark *NSrequired*

ブラウザコマンド

.browser

ページレイアウトコマンド

.layout

.header

.pictheader

.running

その他のコマンド

.chain

.# *(comment symbol)*

.index *NSrequired*

.option *NSrequired*

フラグ

ドキュメントフラグ

NoReLayout

レイアウトフラグ

Kiosk

Main

NoTitle

Sidebar

コンテントフラグ

Main

Sidebar

ToEdge

Centered

NeverBreak

StartsPage

PageMiddle

PageBottom

AlignTop

AlignBottom

AlignCentered

BrowserOnly

BroswserAutoClose

Overlay

KeepWith

NoExtend

NoPage

NoScroller

NoSearch

エッジフラグ

TopEdge

LeftEdge

BottomEdge

RightEdge

Edges

Round

Reverse

EdgeWidth

NewtonScript メソッド

: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*)

ブックリーダーメッセージ

ブックメッセージ

BookInstallScript

BookOKClose

BookRemoveScript

BookHideScript

BookPrint

BookSearchScript

BookShowScript

コンテントアイテムメッセージ

FormHiliteScript

FormSearchScript

```
MungeContentScript
ThumbNailScript
```

NewtonScript グローバル関数

```
BookAvailable ({book: bookFrame}, 0)
BookRemoved ({isbn: isbnStr})
OpenHelpBook (isbnStr)
OpenHelpBookTo (isbnStr, topic)
OpenHelpTo (topic)
ShowManual ()
```

Appendix B

トラブルシュート

本章では、Newton ブック開発時に予想されうるトラブルとその対策について述べる。

フォントに関するトラブル

サポートされるフォント

Newton ブックメーカーでサポートされているフォントは以下の通りである。

フォント名	ポイント数
New York	9, 10, 12, 14, 18
Geneva	9, 10, 12, 14, 18
Espy	9, 10, 12, 14, 18

表 B–1　Newton ブックメーカーでサポートされるフォント

これらのフォントは、いずれもビットマップバージョンを使う必要がある。TrueType バージョンとビットマップバージョンではそのスペーシングに若干の違いがあるため、パソコン画面上での結果と Newton スクリーン上での結果に差が生じる事がある。

また、2 バイト文字は全くサポートされておらず、一体全体なんのために Unicode を採用したのかわからない状態である。漢字コードは漢字コードとしてではなく、アスキー文字二文字として認識される。よって、日本語を使った場合は自分でレイアウトその他をなんとか調整しなければならない。

非サポートフォントを使うことによる問題

現行の Newton デバイスは、ビットマップフォントをスケーリングすることによって、画面に文字を表示している。一方、MacOS 及び Windows ベースのマシンでは TrueType により画面表示を行っている。その結果、同一サイズであってもビットマップフォントと TrueType フォントではスケーリングに差が生じうる。

ブックソースファイルが NewtonBookMaker によってコンパイルされるとき、ソースファイル中で使われているフォントが (ビットマップバージョンではなく)TryeType バージョンの物だったりすると、コンパイルは成功するが、画面表示に狂いが生じる可能性がある。

この問題を避けるには、Newton システムフォントの適切なビットマップバージョンをインストールしたマシンでブックソースを再処理するしかない。

レイアウトの問題

ブックソースファイルのコンテントアイテムに非サポートフォントを適用すると、段落のテキストがちゃんと表示されなかったり、スクロールしても続きがみれない等ということが起こる。特に、Newton のスクリーン下部で段落の最後が ... となっている場合は、次のページをめくってもその消えた部分が表示されることはない。

この問題を解決するには、B–1 ページの表 B–1「Newton ブックメーカーでサポートされるフォント」に示されるフォントだけをブックソースファイルのコンテントアイテムに使い、再コンパイルするほかない。

Espy Sans とそのボールド体

Espy Sans と、**Espy Sans Bold** とは全く別のフォントである。よって、Espy Sans のボールド体を使いたい場合は、Espy Sans Bold を明示的に適用しなければならない。単純に Espy Sans にボールドスタイルを適用しただけだと、フォントのスペーシングの違いから、レイアウトの問題が発生する。

Espy フォントの置換

Newton は、サポートされないフォントで書かれたテキストを、システムフォント (Espy) で置き換えて表示する。レイアウトの問題が発生する。

この問題を回避するには、ソースファイル中で使っているフォントを全てサポートされている物にし、適切なビットマップフォントがインストールされたマシン上でソースファイルを再処理しなければならない。

LaserWriter フォント置換問題その１

LaserWriter は Geneva, New York フォントを持たないので、それぞれ Times, Helvetica に置換される。当然これらのフォントは Newton 側と LaserWriter 側で異なったサイズとなるため、プリント時にテキストがページの下端で切り取られると言う問題が生じる。

ブックソース中の Geneva を Helvetica に、New York を Times に置き換えて Book Maker と NTK でソースファイルを再処理することによってこの問題を回避できる。

LaserWriter フォント置換問題その 2

LaserWriter は Espy Sans フォントも持っていないので、Helvetica で置き換えられてしまう。そのため、このフォントを含むページをプリントすると、期待しない結果を生むことになる。ブック内で Espy Sans フォントを使う場合は、このことを考慮しなければならない。

ブックメーカーに関するトラブル

スタイルの消失

ストーリーテキストブロック内にドットコマンドが表れると、ブックメーカーは現在のスタイル情報を失う。たとえば、ストーリーテキストブロック内にコマンド行が表れると、そこまで記憶されていたボールド体、イタリック体などの書式はクリアされる。ストーリーと関係のあるコマンドはストーリーテキストブロックの最初あるいは最後においておけば、矛盾のない結果が得られる。

不正なエラーメッセージ

ある種のワープロで高速保存オプションを使ってブックソースファイルを保存すると、Book Maker は考えられないようなエラーを出すことがある。(この手の問題を起こす既知のワープロとしては MS Word がある)。

これを回避するには、高速保存オプションを使わずに「別名で保存」コマンドを使ってブックソースファイルを保存するとよい。その後は、普通の保存機能を使っても、高速保存オプションは無効のままである。

XTND と巨大なファイル

Book Maker がワープロファイルの変換に使っている Claris XTND translarots は 1MB 以上のファイルを変換しようとするとエラーを返すことがある。

これを回避するには、巨大なファイルを二分割し、`.chain` コマンドを使って複数のファイルを一つのブックとしてコンパイルすればよい。`.chain` コマンドについてよりくわしくは、付録 A「ブックメーカー言語」の「その他のコマンド」セクションを参照。

スクリプトの内 .header のかわりに、.running ストーリーを使う

.header コンテント内ではスクリプトは使えない。代わりに .running コンテントアイテムを使う。デフォルトのヘッダを消すには、NoTitle フラグを .layout ラインに使用すればよい。例 :

```
.layout Default 12 NoHeader
.running story Centered
Tap here to do something cool.
.script viewClickScript
...  // do something cool
.endScript
```

Find 結果の表示方法の制御

ユーザーが Find ボタンで検索を実行し、一つ以上の結果が見つかった場合、find オーバービューはアイテムのテキストを少しばかり表示する。ブックリーダーはソースドキュメントからその表示用テキストを生成する。これは検索された単語の前後数単語を一緒に選択するのである。(空白、タブ、改行も単語として扱われる)。二つのコンテントの間 (例 : 別々の .story アイテムなど) に境界が有る場合、find オーバービュー上に隣接する領域のテキストは表示されない。

近くにあるテキストをオーバービューの一部として表示したくない場合、余分にスペース、タブ、改行を追加するか、テキストそれぞれを個別の .story コンテントアイテムにしてしまうかすればよい。

ゴミの書き出し

NBM がソースファイルの処理を行っている間に、NBM をフォアグラウンドにしたりバックグラウンドにしたりすると、できあがった定義ファイルに妙なゴミが混じってエラーの元になることがある。

NBM がソースファイルを処理している間は、できるだけそっとして置いた方がよい。

勝手な行の分断[1]

NBM は文字列サイズが数百[2] バイトを越えると、勝手に文字列を分割してしまうことがある。これをやられると特に日本語を含むブックソースの場合、あとでコンパイルエラーやブックの予期せぬ不正な表示に出くわすことになるので要注意。

NTK に関するトラブル

このセクションではブックパッケージの構築やダウンロード時に遭遇する共通の問題について述べる。

古いパッケージを削除できない

NTK プロジェクトの `Settings...` ダイアログボックスにおいて "Delete Old Package On Download" オプションをチェックしていれば、NTK はダウンロードしようとしているファイルの旧バージョンが Newton 上にすでに有れば、それを新しい物で置き換える。これにより、開発とともに、Newton デバイス上にすでに存在するブックあるいはアプリケーションを新バージョンで置き換えることができる。

NTK はパッケージを置換するのにそのパッケージシグネチャを使用する。もしブックに使用するパッケージシグネチャを変更しても、他の識別子、たとえば `ISBN` やアプリケーションシンボルを変更していなければ、NTK にプログラムエラーによりパッケージはダウンロードできなかったと通知されることになる。これは初心者がサンプルファイルを使ってテストしているとき、何度かのダウンロードの後パッケージのシグネチャを変更してダウンロードしようとしたときなどによく発生する問題である。

この問題を避けるには、以下のどちらの方法を使っても良い：

- 古いパッケージのアイコンを選択して、ルーティングメニューから Delete を選んで削除する

1. 日本語関連
2. 正確なところはわからないが、256 バイトくらいかもしれない。

- 問題のある識別子を、実際に Newton 上に存在するものが削除されるように変更する

最新ブックをロードしたのに、タイトルが更新されない

Newton のブックリーダーアプリケーションは `title` とか現在のページのようないくつかの情報をキャッシュしている。タイトルが変更されたことをエンジンに通知するには、アップデートしたブックをロードする前に `.isbn` を変更する必要がある。

ブックをアップデートするので有れば、`.isbn` コマンドでダッシュを使い、バージョンや日付を表示しておくと良い。例 :

```
.title My Info Book of 7/94
.isbn  IB-7-94:PIEDTS
```

最後に表示されたあるいはブックマークされたページの削除

ブックの開発中、表示あるいはブックマークされた最後のページが削除されると、エラー `-48205` が発生する。ブックメーカーは表示された最後のページと全てのブックマークされたページへの参照を残そうとするからである。

これを回避するには、`ISBN` を変更し、ブックメーカーと NTK で再処理することである。この問題は開発過程でのみ発生し、エンドユーザーはこの問題に遭遇することはない。

特殊な引用符はコンパイルできない

NewtonScript は 7bit アスキーに制限されている。この範囲を超える文字を追加しようとすると NTK から "illegal character" と怒られることになる。たとえば、この問題はブックソースファイルを書くときに使うワープロの機能「スマート引用符」が原因となりうる。スマート引用符は ASCII 文字集合の一部ではないのである。普通の引用符だけを NewtonScript ステートメントで使用しているので有れば、そうした変なメッセージはなくなる。例 :

```
.script
    Print(“Whoops”) // won’t compile
    Print("Whoops") // will compile
.endscript
```

グローバル関数へのワーニング

NTK がプラットフォームファイル中で見つけられない関数があると、インスペクタ内にグローバル関数に関するワーニング (エラーではない) が生じる。ブックメーカーは使用しているプラットフォームファイルにない関数を呼び出すこともある。これはブックの構築に特に影響はしない。

NTK の `Project` メニューから `Project Settings...` ダイアログを呼び出し、Check global function calls option をチェックオフしてもよい。

オンラインヘルプに関する書籍

この付録ではオンラインヘルプ生成についての議論を収録した本や雑誌の記事をリストしている。

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 D

互換性

この付録では 1.x Newton マシン用にデジタルブックを書く際に発生する互換性の問題に付いて述べる。本書で説明されるいくつかの機能は OS2.0 マシンにインストールされたバージョンの Newton ブックリーダー (NBR) でのみ利用可能なものである。1.x マシン上でも表示可能なデジタルブックのための適切な変更点はこの付録に記録されている。

ブックメーカーバージョン 1.1 での追加点についても触れている。

この付録は以下の三部構成となる：

- Newton ブックメーカーバージョン 1.1 において拡張された機能の説明。
- NBR 1.x におけるバグの内、NBR 2.0 で解消されたものについての説明。こうした問題のほとんどは適切な回避策が示されている。
- NBR 2.0 において追加されたブックリーダーの拡張された機能の説明。

ブックメーカー の機能拡張

ブックメーカー 1.1 では、より効果的にサブインデックスの生成をコントロールするため、`.index` コマンドにパラメタが追加された。

サブインデックス生成のコントロール

`.index` コマンドは *@indexName* パラメタを受け付けるようになった。ブックメーカー 1.0 は *!indexName* パラメタのみを受け付けていた。より詳しくはページ A–28 の `.index` の説明を参照。

ブックリーダーにおけるバグの解消

このセクションで説明される NBR1.x のバグはバージョン 2.0 で解消された。バグとともに、適切な回避策が示される。

ブックマーク及びプリント

NBR1.x において、ブックマークの生成時、あるいはページのプリント時、何らかのブック関数 (`BookData, AuthorData` 等々) を動かそうとするページ上のスクリプトは実行失敗していた。以下の関数は NBR 1.x でプリント・ブックマーク時には利用不可能である：

```
AddToContentArea
AuthorData
BookData
CountPages
CurrentBook
CurrentKiosk
CurrentPage
FindContentBySlot
FindContentByValue
HiliteBlock
InsertForm
WhereIsBook
```

NBR1.x では、プリント・ブックマーク時に呼び出される可能性のある `.script` 内でこれらの関数のどれかを呼ぶ場合は、条件付きメッセージ送信 (`:?`) を使用するか、適切な例外ハンドラを提供する必要がある。

ISBN 文字列の長さ

MessagePad 100 では ISBN 文字列は最大 15 文字までである。

About スリップに表示されない情報

NBR 1.x では `.author, .data, .publisher, .copyright` コマンドで提供される情報が表示されなかった。

ページスクリプトがブックスクリプトをオーバーライドする

NBR1.x では、ブックレベルのスクリプトはページレベル (レイアウト) スクリプトをオーバーライドする。もしブックに同名のスクリプトがあると、ページレベルスクリプトは呼び出されないかもしくは、ブック及びページの両方がデフォルトの `buttonClickScript` を提供していた。

マルチページコンテンツにおけるエッジフラグ

NBR1.x では、`topEdge` と `bottomEdge` フラグは、複数ページにまたがるコンテントアイテムの前ページに一行を書いていた。複数ページにまたがるコンテントアイテムに対してこのフラグを使ってはならない。

ストーリーカードにおけるメモリマネジメント

`HideStoryCard` 関数が呼ばれたときあるいはユーザーが複数のストーリーカードをスイッチした時、NBR 1.x ではストーリーカードをクローズするのではなく隠していた。このメモリーリークへの回避策としては、`ShowStoryCard` の前に以下のコードを追加すればよい :

```
if (storyCard <> NIL) then
begin
    storyCard:Close();
    storyCard := NIL;
end
```

ストーリーカードが常に左寄せになる

NBR 1.x におけるストーリーカードは常に左寄せテキストとなっていた。

.form コンテントアイテムが veiwJustify を反映しない

`.form` コマンドによって追加されたコンテントアイテムは NBR 1.x では `viewJustify` スロットの値を反映しなかった。

InsertForm 関数は View を返さない

NBR1.x において `InsertForm` 関数は挿入されたビューを返さなかった。挿入されたビューへの参照を得るには、`GetView` 関数を呼び出す。

.Form コンテントアイテムの検索

NBR1.x において、システム提供の Find サービスは `.form` コンテントアイテムを検索しなかった。NBR 2.0 は `.form` アイテムを検索する二つの方法を提供している : テキストスロットを追加するか、`FormSearchScript` メソッドを使用するか。いずれの機構も NBR 1.x では利用不可能である。

回避策としては、検索されるテキストとともに `story` アイテムを用意し、それを `.form` アイテムで隠すようにする。`.story` アイテムを名前とその場所とともにページに追加すると、検索機能がそれを見つけだすが、ストーリーは現在のフレームの表面下に存在するので、ストーリーはユーザーのビューから隠される。

以下に NewtonScritp での実装方法を示す :

```
.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
```

ストーリー段落のビューはカラム全体の幅一杯に拡張されるので、フレームの右端を十分な幅にしておく必要がある。また、検索されるテキストはハイライトされない (が、正しいページをを得ることができる)。

ヘルプブック

Newton システムソフトウェアの 1.x において、スタンドアロンのヘルプブックを構築するのは不可能であった。ヘルプブックはアプリケーションパッケージの一部としてコンパイルされねばならなかった。さらに、このシステムはヘルプを呼び出すための `protoInfoButton`(もしくは `newtInfoButton`) を提供しない。よって、アプリケーション内でヘルプを表示するのはプログラマの責任であった。

"Show my Help" ボタンのような立ち入ったインターフェースを提供する必要がある。このセクションではユーザーが "Show my Help" ボタンをタップしたときにヘルプブックを表示するために最低限必要となるコードを説明する。

1. **ブック定義ファイルから以下の二行を取り除くかコメントアウトする。これらはスタンドアロンヘルプブックをサポートするためにブックメーカーによって生成されるものである** :

   ```
   output.book := book;
   output.help := TRUE;
   ```

   これらの二行はブック定義ファイルの一番上の方で、ブックフレームの定義の後に有る。

2 **以下の行をブックの** `postamble` **に置いて、コンパイル時定数を生成する** :

```
DefConst('kMyHelpBook, book)
```

3 **ヘルプブックフレームを記憶するためのスロットをアプリケーションベースビューに生成する。それを** theBookFrame **とする。**

4 **アプリケーションのベースビューの** setupDoneScript **において、以下の代入を行う。(これは** Newton **システム** 1.x **では利用不可能な** RegisterBookRef **呼び出しの代わりである**):

```
theBookFrame := BuildContext({_proto:
GetRoot().TinyTim._proto, bookRef: kMyHelpBook})
```

5 Show My Help" **ボタンのボタンクリックスクリプトで、以下のコードを実行する** :

```
GetRoot().TinyTim:Close(); // in case system help is open
theBookFrame:OpenManual(kMyHelpBook);
```

6 **ヘルプブックを閉じるには、以下のようにする** :

```
if theBookFrame then
begin
    theBookFrame:Close(); //so card can be removed
    theBookFrame := nil //so help book can be GC'd
end
```

この最後のステップは非常に重要である。これを実行しないと、ユーザーは一度アプリケーションが開いた後にシステムのヘルプを使用できなくなってしまう。

ブックリーダーへの拡張

NBR 2.0 は NBR1.x にはないいくつかの機能を提供する。このセクションではそうした追加点について述べる。1.x の Newton デバイスでも自分のブックを動かしたい場合はこのセクションで述べられるメソッドを使用してはならない。

利用不可能なメソッド

以下のメソッドは NBR2.0 から導入された。以下の関数のうちいずれかを呼び出しているブックを NBR 1.x 上で動かそうとすると例外が派生する。

NBR 1.x 上でも動かしたいブックでは、以下の関数を呼び出してはならない。あるいは適切な例外ハンドラを用意すべきである。

```
AddBookRouting
BookSearchScript
ChangeScrolledOrigin
CloseBrowser
FindContentsByPage
FindPageBySubject
GetLibraryEntry
GetOnlineBooks
GetOfflineBooks
OpenBook
OpenBookTo
PageSize
RegisterBookRef
RemoveBrowser
ScrollReceiver
SetStatusButtons
TurnToSubject
UnRegisterBookRef
```

送信されないブックリーダーメッセージ

ページ A–47 からはじまる「ブックリーダーメッセージ」のセクションでリストされた、ブックリーダーによって送信されるメッセージの全てはNBR2.0 でのみ送信される。それらのメッセージに応答するために書いたあらゆるメソッドは呼び出される事はない。

利用不可能なフラグ

`browserAutoClose` フラグは NBR 1.x では利用不能である。ブラウザに `outlineClickScript` を追加することによって似たような動作をさせることができる。

用語集

ブック定義ファイル
ブックメーカーにより生成される出力ファイル。ブックパッケージあるいは Newton アプリケーションヘルプを生成するための Newton Toolkit (NTK) への入力として使用される。

ブックメーカー
ブックソースファイルを処理してブック定義ファイルを生成するためのアプリケーション。

ブックリーダー
Newton のスクリーン上に対話型のデジタルブックを表示するシステムサービス。

ブックスクリプト
ブックソースファイルにおいて、何らかのコンテントアイテム定義が登場する前に `.script` コマンドによって生成されるスクリプト。こうして生成されたスクリプトはブックパッケージ全体に有効となるので、ブックスクリプトと呼ばれる。

ブックソースファイル
ブックメーカーコマンドによりタグ付けられたコンテントアイテムを含むワードプロセッサで作られたファイル。

コメント
行頭がコメントシンボル (`.#`) であるブックソースファイルの行。ソースファイル上のコメントは、コンパイルされたブックには反映されない。

コンテントコマンド
テキストやグラフィックなどの、Newton のスクリーン上に表示されるコンテントアイテムを定義するブックメーカーのコマンド。

コンテントフラグ
個々のコンテントアイテムを修飾するフラグ。ブックメーカー言語のほとんどのフラグはコンテントフラグである。

フラグ
ブックメーカーコマンドに追加して何らかの機能を有効にするキーワード。

ドキュメントコマンド
ブックソースファイル全体に効果を及ぼすブックメーカーのコマンド。

ドキュメントフラグ
ブックソースファイル全体に効果を与えるフラグで、たとえば `noReLayout` フラグ等がある。

グローバル
ブック全体からアクセス可能な変数あるいは関数。

ヘルプブック
Help Size オプションがチェックされたときにブックメーカーがブックソースファイルから生成するファイル。

キオスク
`.kiosk` コマンドによって作られたナビゲーション用ページ。キオスクページ上のアイテム (たとえば何かの絵) をタップすると、ユーザーはそのアイテムが表す目的のアイテムに直接移動できる。

レイアウト
ページ上でテキストやグラフィックの配置を指定するためのブックメーカーのコマンド。

レイアウトコマンド
レイアウトを定義するブックメーカーのコマンド。

レイアウトフラグ
レイアウトコマンドを修飾するためのフラグ。フラグはそのレイアウトを使用する全てのページに影響を与える。

ページスクリプト
レイアウトに張り付けられたスクリプト。そのレイアウトを使用する全てのページでそのスクリプトは利用可能であるため、ページスクリプトとして参照される。

ポイント 活字の単位。1 インチは 72 ポイントである。

プロジェクトファイル NTK 用のファイルで、プログラム構築時に必要なファイルのリスト及び構築方法の指定を含む。

索引

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

あ

い

お

か

き

く

こ

さ

す

た

ち

て

と

は

ひ

ふ

へ

ほ

ま

め

よ

れ

わ

原書の作成方法・スタッフ (原書からそのまま引用)

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.

WRITERS	Adrian Yacub, John Perry
PROJECT LEADER	Christopher Bey
ILLUSTRATOR	John Perry
EDITOR	Linda Ackerman
PRODUCTION EDITOR	Rex Wolf
PROJECT MANAGER	Gerry Kane

Special thanks to Gabriel Acosta, David Dunham, Bob Ebert, and Scott Shwarts.

本書に登場の登録商標について

責任の制限

訳者からの謝辞

本書の作成に当たっては以下の方々に大変お世話になった。みなさんに感謝したい。文体がえらそうだが、ここだけ「ですます調」にすると妙なので、そのあたりはどうかご勘弁いただきたい。

theNewtonShop 店長、大杉信雄さん。彼は本書の公開のために、様々な形で訳者を支援してくださった。本当に感謝している。

大宮さん、池上さんをはじめとするアップルコンピュータ株式会社の方々。本書の公開にあたり、彼らの多大なご支援をいただいた。

「おいしい Book」チームの、ののがきあつこさん、西ゴンこと西田博昭さん。彼らは本書のベータ版から数多くの間違いを発見して下さった。愛すべき彼らのホームページは http://www.k-inet.com/oishii/ である。実は私もそのメンバーの一人である。

武藤琴美さんは、「プログラミング言語 NewtonScript」のための特製バインダーを制作され、このことは私にとって大きな励みとなった。

私の友人であり、k-inet 運営者の河口 ojin 忠志さん。彼は本文書ダウンロードのミラーサイトとなっている k-inet のサーバスペースを私に提供してくれた。

そして私の妻と二人の息子はいつも私を応援してくれた。

訳者について

齋藤匡弘 (Saito, Kunihiro)、男、1964 年 1 月大阪生まれ。現在愛媛県に在住。職業は会社員で UNIX プログラマ。数年前から Newton プログラミングに凝る一方、その英文ドキュメントの翻訳も行っている。最近の目標はプログラマーズガイドとリファレンスを完訳する事と、「Inside Of the おいしい Book」を書くことである。ホームページは http://www.k-inet.com/brown/, メールアドレスは brown@ga2.so-net.ne.jp である。

本書の作成方法

本書の原書の英文は、AppleDockViewer からテキストに落とされた。そうして得られた原稿は全て、田川 洋一 氏開発のフリーウェア YooEdit1.6.3 上で和訳された。

和訳原稿は Adobe FrameMaker5.5J 上で編集され、その際以下のフォントが使用された :Times, Helvetica, Geneva, EspySans, EspySans Bold, NewYork, Courier, Zapf Dingbats, リュウミンライト L-KL, 中ゴシック BBB。編集に際しては NTK を使って MessagePad2.1K から取り込んだハードコピー、サンプルプログラムから取り出したリソースデータ、Machintosh 画面のハードコピーを Adobe PhotoShop4.0J で編集の上使用した。

掲載されているプログラムテキストは原書から取り出したままであり、一切チェックは行っていないので注意されたい。

上記環境で得られた本書の原稿は Adobe Acrobat Distiller3.0.2J を経て PDF 形式に変換され、Adobe Acrobat Exchange 3.0aJ により内容を一部変更の上、本文書を得ている。本文書の内容は一旦 EPSON LP-9200PS によって出力された後校正を受け、再度 Acrobat 書類を作成するという手順を繰り返した。

上記の作業は全て Apple PowerMacintosh 8500/180 上で行われた。

バージョン履歴

1998/7/12	Ver. 0.9b	校正用原稿
1998/8/1	Ver. 1.0b	ベータ版
1998/8/10	Ver. 1.1b	ベータ版その 2
1998/8/11	Ver. 1.11b	限りなく本物に近いベータ版
1998/8/12	Ver. 1.2b	最終確認バージョン
1998/8/17	Ver. 1.2	一般公開バージョン

Newton **ブックメーカー**
ユーザーズ・ガイド

アップルコンピュータ株式会社

翻訳 齋藤匡弘 (Kunihiro Saito)

mailto: brown@ga2.so-net.ne.jp