
Adobe® Acrobat®は、PDF（Portable Document Format）ファイルを用いて、
ドキュメントをそのままの体裁で電子的に配信することを可能にしたソフトウェアです。

ご覧のPDFファイルは、用途を配信実験に限定した上で、
開発中のAcrobat Reader 3.0Jを用いて作成したものです。
制作者とアドビシステムズ株式会社両者の事前の承諾なしに、
再配布ならびに、電子掲示板、CD-ROM等への転載、およびデータの改変、
再利用をすることはできません。

Newton® Technology
JOURNAL
Volume II, Number 3 June 1996

## 目　次



Newton®

## Communications Technology

# Newton Internet Enabler

*by Gary Hillerson, Hillysun Enterprises, Inc.*

Newton Internet Enablerは、インターネットへアクセスするアプリケーションの開発を容易にするツールである。Newton Internet Enablerにより、インターネットプロバイダへのリンクを確立し、さまざまなオプションを用いてリンクを構成し、コミュニケーションエンドポイント方式を用いてデータ送受信を行うことができる。さらにNewton Internet Enablerは、ステータス表示とデータ変換を自動化することさえできる。

Newton Internet Enabler を使用すれば、TCPかUDPのいずれかのトランスポートサービスを使用してリンクを確立できる。複数のアプリケーションで１つのリンクを共用できる。Newtonシステムソフトウェアは、すべてのアプリケーションがリンクを解放するまで、リンクをオープンした状態に保つ。Newton Internet Enablerは、現在、2つの低レベル・リンクプロトコル、すなわちSLIPとPPPの使用をサポートしている。

Newton Internet Enablerは、アプリケーション・プログラミング・インタフェース（API）とセットアップ・アプリケーションを提供する。セットアップ・アプリケーションのNewton Internet Setupにより、ユーザは、さまざまなインターネットプロバイダへのリンクの設定を定義できる。例えば、ユーザは、CompuServe用のリンク設定、ローカルのインターネットプロバイダ用のリンク設定、さらに仕事中に電子メールを確認する3つ目の設定をセットアップすることができる。各リンク設定には、ダイヤルする電話番号、使用するリンクレベル・プロトコル、ならびにリンク確立のための初期化シーケンスとログイン・シーケンスが含まれている。

Newton Internet Enablerのアプリケーション・プログラミング・インタフェース（API）は、多様なネットセッション関連タスクを実行するために呼び出す約10個のグローバル関数、リンク設定を制御するために用いる多数のオプション、およびインターネット上でコミュニケーションを実行するためにエンドポイントで使用するコミュニケーションツールから成る。

3ページへ続く

## New Technology

# Windows用のNewton Toolkitのリリース

*by Lee Dorsey, Apple Computer, Inc.*

Newton 2.0プラットフォームのソフトウェア・デベロッパ・コミュニティを拡大する努力の一環として、アップルは、NewtonToolkitのWindows版であるNewton Toolkit for Windows v.1.6（NTK for Windows v.1.6）を今回はじめてリリースとすることになった。本製品はコンパイラとプロファイラを持ち、Newton 2.0を完全にサポートし、Newton Toolkitとしての完全な機能性を備えている。NTK for Windows v.1.6は、Windows 95、Windows NT、およびWin32を含むWindows 3.1をサポートし、完全にWindowsそのものの、ルックアンドフィール（操作性）を与える。

“NTK for Windowsの出荷は、デベロッパ・コミュニティに、最も柔軟で強力なPDAプラットフォーム用ツールを提供するという、非常に大事なステップである。”と、Newton Tools Product Line マネジャーのRick Fleischmanはいっている。さらにFleischmanは、“Newtonプラットフォームが成熟するにつれ、NTK for Windowsを出すことで、Newtonアプリケーションを作りたいという、今までよりずっと多数のデベロッパと協力して行けることになる。”と述べている。NTK for Windowsは、Newtonプラットフォーム用開発ツールを継続的に改善しようとする、アップルの様々な努力の一例にすぎない。アップルは、96年1月に、Mac OS用のNTK v.1.6（Newton Toolkit for Mac OS v.1.6）とMac OSとWindows用のデスクトップ・インテグレーション・ライブラリ（Desktop Integration Libraries for Mac OS and Windows, v. 1.0）を出荷している。NTK v.1.6は、PowerPC用に最適化され、デバッグ・ツールが改善され、Newton 2.0を完全にサポートする。デスクトップ・インテグレーション・ライブラリ（DILs）を使用することで、Mac OSとWindowsベースのアプリケーション供給者は、別個のユーティリティ・プログラムを必要とせずに、Newton PDAのデータとアプリケーションのデータを同期させられる。

NTK for Windowsの他に、アップルは、Mac OS用のNewton C++ツール（Newton C++ Tools for Mac OS）をまもなく提供する予定である。このツールを使えば、Newtonデベロッパは、

21ページへ続く

Newton Technology JOURNAL

Volume II, Number 3　1996年6月

発行／Apple Computer, Inc.
マネージング・エディタ／Lee DePalma Dorsey
テクニカル担当、コーディネーティング・エディタ／Gerry Kane
DTSおよびトレーニング担当、コーディネーティング・エディタ／Gabriel Acosta-Lopez
テクニカル・ピア・レビュー・ボード／J. Christopher Bell, Bob Ebert, David Fedor, Ryan Robertson, Jim Schram, Maurice Sharp, Bruce Thompson
寄稿者
Greg Christie, Barney Dewey, Gary Hillerson, Peter Murray, Jeffrey C. Schlimmer, Eileen Tso

制作／Xplain Corporation
発行者／Neil Ticktin
編集助手／John Kawakami
編集助手／Matt Neuburg
アートディレクタ／Judith Chaplin



Editor's Note

# 編集長からのメッセージ

by Lee DePalma Dorsey

## 同じ機会、同じ場所、そして新たなこと

読者の皆さんの多くは、Newton Technology Journalのこの号を、カリフォルニア州サンノゼにおけるアップルのWorld Wide Developers Conference会場の席で、あるいはホテルの部屋でご覧になっていることと思います。なかには、このジャーナルを初めて目にされる方もいらっしゃるでしょう。Newtonデベロッパ・コミュニティの多くの方々が、アップルの毎年恒例のこのデベロッパ・イベントに参加されて、アップルの技術動向に関する理解をさらに深めるとともに、新たなテクノロジーを身につけていただくことを願っております。今回もNewton Systems Groupは、MacintoshとWindowsのデベロッパの方々に、Newtonテクノロジーの現況ならびに私どもの展開するPDAプロダクトカテゴリーとテクノロジーベースに占める、Newtonプラットフォームの可能性をご紹介したいと考えております。参加された皆さんには、Newtonテクノロジーが、その他のアップルテクノロジーといかに調和しているか、また発展するアップル社の事業計画にいかに統合されていくかについても、ご説明させていただきます。経験豊富なNewtonデベロッパの皆さんには、この号で取り上げているNewtonプラットフォームの最新テクノロジーの中のいくつかについて、新しい情報に触れ、さらにデモをご覧いただきます。

皆さんは経験豊富なNewtonプログラマでいらっしゃるかもしれないし、あるいはNewtonプラットフォームに不慣れなデベロッパかもしれません。また単にNewtonテクノロジーを評価しているだけの方もいらっしゃるでしょう。しかし、皆さんがどのような方であろうとも、これほどエキサイティングな時期に、これほどエキサイティングなテクノロジーに触れることはないでしょう。1995年11月のリリース以来ご紹介して参りましたように、Newton 2.0は、ユーザとデベロッパがPDAプラットフォームに対して等しく期待するような機能を実現する上で、非常に大きな前進が達成されたことを意味します。昨今、インターネット、電子メール、通信などをめぐり、めざましい動きが展開されています。Newton PDAは、このタイムリーな発展の波にみごとに対応するものです。NTJの本号では、この発展の最も重要な側面のいくつかについて、その概要をお伝えします。Newton Communicationsのプロダクトマネージャーであるバーニー・デューイは、私どものコミュニケーション戦略を、またゲイリー・ヒラーソンは、待望の最新ツールであるNewton Internet Enablerをそれぞれ解説します。しめくくりとしてEileen Tsoが、これらの戦略やツールが、Newtonプラットフォームにとって、またこれを最新鋭のプラットフォームにして行くソリューションの継続的発展のために必要不可欠である、ということ、そしてそれはなぜか、ということをご説明いたします。ツールの面では、Windows NTKの発表もついに秒読み段階に入りました。Intro to Newton Sessionに参加される方々には、この新製品をはじめてご覧いただけます。昨年話したことが、今ここに実現したのです。昨年と機会も場所も同じですが、新たなことが山ほど起こりました。

WWDCのセッションは、このプラットフォームに関心をもつ新しいデベロッパの方々の教育訓練を目的としていますが、経験豊富なデベロッパの皆さんにとっても、興味をそそられる新しいことを経験していただく良い機会です。もし、本号をご覧になっている読者が既にプラットフォームを変更した方であれば、この機会に、お知り合いの方々にNewtonセッションに立ち寄っていただくようお勧めください。そして、MacアプリケーションをモバイルなNewtonクライアントへ拡張し、本当の“モバイル・インターネット”デバイスを初めて市場に送るプロジェクトに参画する可能性について、理解を深めていただいてください。Guy Kawasakiが言うように“百花繚乱を求め”ようではありませんか。皆さん一人一人が、Newtonプラットフォームの伝道者たり得るのです。WWDCは、皆さんのデベロッパ仲間の方々が、その卓越した最新のアイディアをこのプラットフォームで実現する気持ちになっていただける絶好の機会です。機は熟しています。市場にはチャンスがあふれ、ツールもそろっています。ぜひとも仕事仲間の方々に、この最高のPDAプラットフォームについて、アップルがNewton 2.0によって成し遂げた成果について、皆さんの知っている限りのことをお伝えください。

Lee

*1ページからの続き*

# Newton Internet Enabler

図1に、Newton Internet Enablerの構成要素の関係を示す。

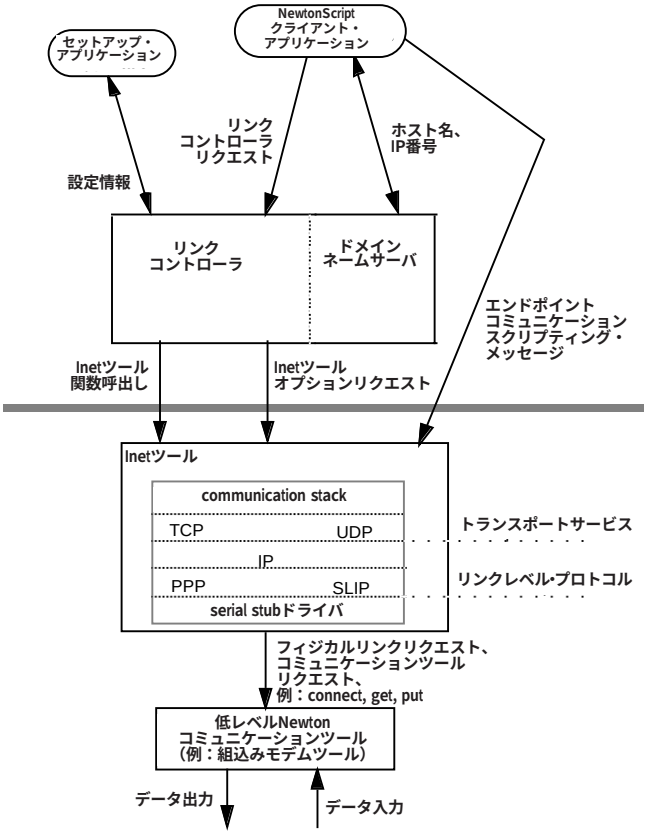


*図1　Newton Internet Enablerの構成要素*

Newton Internet Enablerのアプリケーション・プログラミング・インタフェース（API）には、２種類の関数が用意されている。ドメインネームサービス関数とリンクコントローラ関数である。ドメインネームサービス関数は、インターネットドメインネームとそれに対応するIPアドレスとの間の変換に使用する。リンクコントローラ関数は、インターネットリンクのステータスの確立、解放、および検索に使用する。

ドメインネームサービス関数とリンクコントローラ関数は、Newtonシステムソフトウェアのグローバル関数である。これらの関数は、Newtonコミュニケーションエンドポイントと Inetツールとの間のインタフェースを提供する。Inetツールとは、基本的なコミュニケーションツールであり、リンクの確立、維持、およびリンク経由のコミュニケーションに関する実際の作業を行う。

Inetコミュニケーションツールは、TCP/IPあるいはそれ以下のレベルのプロトコルに関して、構成可能なスタックを提供する。 Inetツールは標準的なNewtonコミュニケーションツールであるため、組込みモデムツールや組込みシリアルツールなど、他の組込みコミュニケーションツールが提供するエンドポイントサービスをすべて提供する。Inetツールは、他のコミュニケーションツール同様、コミュニケーションオプションを用いて設定を制御できる。

Inetツールは、さまざまな低レベルコミュニケーションサービスを用いて、フィジカルリンクを確立することができる。各コミュニケーションサービスは、組込みモデムツールなどのNewtonコミュニケーションツールが提供する。Inetツールは、PPPやSLIPなどのNewtonシステムソフトウェアにより提供されるさまざまなリンクレベル・プロトコルを実行できる。

アプリケーションは、いくつかのエンドポイントを同じNewton Internet Enablerリンクで使用することができる。しかし、各エンドポイントはかなりの量のメモリを要する。アクティブになれるエンドポイントの総数は（すべてのアプリケーションに関して）、ユーザのハードウェア構成とデバイス上で現在起動しているソフトウェアとの組み合わせにより制約される。

この記事では以降、Newton Internet Enablerのアプリケーション・プログラミング・インタフェースを解説する。1995年11月発行のNewton Technology Journal（Volume 1のNumber 5）には、エンドポイントとコミュニケーションツールを含むNewtonコミュニケーションテクノロジーの概要が述べられている。

## Newton Internet Enabler とコールバック関数

Newton Internet Enabler関数の多くは、コールバック関数を必要とする。コールバック関数とは、リクエストしたオペレーションの実行中や実行後に、Inetツールが呼び出す関数である。コールバック関数はステータスとエラー情報を受け取る。

例えば、`InetCancelLink`関数は、そのオペレーションを終了した後に、デベロッパが設定したコールバック関数を呼び出す。`InetCancelLink`のコールバック関数は、エラーが発生したかどうかを決定し、取消しを希望したリンクのカレント・ステータスを決定する。

コールバック関数を１度以上呼び出すようなオペレーションもある。例えば、`InetGrabLink`関数は、オペレーション実行中に、設定したコールバック関数を頻繁に呼び出す。`InetGrabLink`コールバック関数は、呼出しごとにカレント・ステータスを報告するため、接続の進行を監視するのに利用できる。

関数がコールバック関数の指定を必要とするときは、コンテキストフレームを指定し、そのフレーム内に定義し、コールバック関数として用いたい関数のシンボルを指定する。例えば、`InetGrabLink`関数は３つのパラメータをとる。次に一例を示す。

```
InetGrabLink(linkID, clientContext, clientCallback);
```

`InetGrabLink`を呼び出すときは、`clientContext`の値としてフレーム（あるいはアプリケーションフレーム）を指定し、`clientCallback`の値として、そのフレーム内に定義されている関数を指定しなければならない。

`InetGrabLink`呼出しに対応するコールバック関数の設定方法の一例を次に示す。

```
myApp.GrabLinkCallback := func(linkID, stat, err)
  begin
  if err=nil and stat.linkStatus <> 'connected then
    ;     // display status
  if err then
    ;     //handle the error
  // link established, so resolve the address
  end;
```

アプリケーションで`InetGrabLink`関数を呼び出すときは、コールバック関数の名前（シンボル）を渡す。例えば、次のようにする。

```
myApp.TestGrab := func()
  begin
  myStatusView := InetStatusDisplay(nil, nil, nil);
  InetGrabLink(nil, self, 'GrabLinkCallback);
  ...
  end;
```

この関数はまず最初に、ステータスビューの作成と表示を行うInetStatusDisplay関数を呼び出す。InetGrabLinkの呼出しには、デフォルトリンクIDを使用し、clientContextパラメータの値として自己（アプリケーションフレーム）を、clientCallbackパラメータの値として 'GrabLinkCallback（コールバック関数のシンボル）を指定する。GrabLinkCallback関数は、システムがリンクの接続を試みる間、ステータスが 'connectedになるか、あるいはエラーが発生するまで、繰り返し呼び出される。

## リンクコントローラ・インタフェースの使用

リンクコントローラは、Newtonデバイスとインターネット間のリンクの作成と管理に用いる。リンクコントローラは、複数のアプリケーションに対して1つのリンクを同時に管理することができる。つまり、1つのアプリケーションがリンクを確立すると、他のアプリケーションもそのリンクを利用できるということである。

リンクを確立する（接続する）には、InetGrabLink関数を呼び出す。リンクの最初の接続は、時間がかかる場合がある。通常は次の手順を踏む。Inetツールソフトウェアがモデムをダイヤルし、インターネットセッションを確立するための接続のネゴシエイト(negotiate)を行う。次にInetツールは、ユーザがInternet Setupアプリケーションを用いて構成したログイン・プロシージャと初期化プロシージャを実行する。これらすべての実行には、かなりの時間を要する場合がある。

新たなリンクの接続にはかなりの時間を要する場合があるため、Newton Internet Enablerでは、各リンクのユーザの参照数（ユーザカウント）を維持することにより、別のアプリケーションが、既に確立済のリンクを接続するのは簡単に行えるようになっている。アプリケーションがリンクを接続すると、リンクコントローラは、そのリンクのユーザカウントを増やす。すべてのユーザがそのリンクを解放した後に（ユーザカウントが0になったときに）、フィジカルリンクが切断される。

リンクコントローラ関数の大半は、非同期的に動作する。Inetツールは、オペレーションを開始した後、制御をアプリケーションへ戻す。オペレーションが完了すると、あるいは場合によってはオペレーションの進行中に、Inetツールは、設定してあるコールバック関数を呼び出して、通知を行う。コールバック関数を用いれば、リンクのステータスを監視し、オペレーションが成功したか否かを決定できる。

リンクの接続が進行している間、Inetはコールバック関数を頻繁に呼び出して、リンクのカレント・ステータスに関する情報を提供する。このステータス情報をInetStatusDisplay関数へ送れば、接続プロセスに関する視覚的フィードバックをユーザに提供できる。

次に、Newton Internet Enablerによるインターネットセッションの間に、どのようなオペレーションが発生するのか、その流れの典型的な例を示す。

1. アプリケーション（My_Application）がInetGrabLink関数呼出しを行う。リンクコントローラがモデムをダイヤルし、インターネットプロバイダとのインターネットセッションを開始する。
2. 接続操作実行中に、 Inetツールは、My_Applicationが接続操作のために定義したコールバック関数を、定期的に呼び出す。このコールバック関数が、InetStatusDisplay関数を呼び出して、リンクのカレント・ステータスの画面表示を更新する。
3. 接続操作が完了すると、コールバック関数が画面からステータス表示を削除する。
4. My_Applicationが、そのリンクで使用する1つまたは1つ以上のエンドポイントをインスタンス化し、結合する。各エンドポイントは、TCPまたはUDPトランスポートサービスを使用できる。各エンドポイントは、出力接続を開始する（connect）か、あるいは入力接続を受信することで結合できる。
5. My_Applicationが、そのエンドポイントを使用してコミュニケーションオペレーションを実行する。その際、OutputやSetInputSpecなどのエンドポイント・メソッドを呼び出す。
6. 別のアプリケーション（Your_Application）が、My_Applicationと同じサービスプロバイダを利用するために、InetGrabLink関数呼出しを行う。Inetツールが、ステップ1で確立したのと同じリンクを戻す。
7. Your_Applicationがエンドポイントを作成し、それを使用してコミュニケーションオペレーションを実行する。
8. Your_Applicationがリンクの使用を終了し、InetReleaseLink関数を呼び出す。リンクコントローラが、リンクしているユーザのカウントを減らす。
9. 3つめのアプリケーション（Their _Application）がリンクを接続し、リンクで使用するエンドポイントを作成した後、リンクを解放する。
10. My_Applicationがリンクの使用を終了し、InetReleaseLink関数を呼び出す。リンクコントローラが、リンクしているユーザのカウントを減らす。ユーザカウントが0になると、リンクは切断される。つまり、インターネットセッションが終了し、モデムがハングアップし、リンクのために使用したリソースが解放される。

## リンクの確立

まず最初に、リンクを確立（接続）しなければならない。リンクを確立するには、InetGrabLink関数を呼び出す。InetGrabLinkには、リンクID、コールバック関数、およびコールバック・コンテキストフレームが必要である。

```
InetGrabLink(linkID, clientContext, clientCallback)
```

リンクIDについては、nilを指定してデフォルト・リンクを使うこともできるし、あるいは、このパラメータの値として、InetAddNewLinkEntry関数により戻される識別子を使うこともできる。nilを指定すると、システムソフトウェアは、デフォルト・リンクIDとして設定されているリンクIDを使用する。殆どの場合は、このようにした方が都合がよい。

InetGrabLinkオペレーションは、完了するまで若干の時間を要する。このオペレーションの進行中は、Inetツールがコールバック関数を繰り返し呼び出して、リンクの接続に関するカレント・ステータスを報告する。Inetツールは、エラーが発生するか、あるいはステータスが 'connectedになるまで、コールバック関数の呼出しを繰り返す。

コールバック内のステータス値はステータスフレームである。このフレームは、カレントのリンクステータス値と（場合によっては）他の情報を含む。コールバック内にInetDisplayStatus関数を使用すれば、カレント・ステータスをユーザに対して表示できる。次節の“リンクステータス情報の検索と表示”では、ユーザに対してステータスを表示する方法を解説する。

次に、InetGrabLink関数のコールバック関数の一例を示す。

```
myApp.GrabLinkCallback := func(linkID, stat, err)
  begin
  myLinkID := linkID;                    // save the link ID
    // during grab processing, display status & return
  if err = nil and status.linkStatus <> 'connected then
    return InetDisplayStatus(linkID, myStatView, stat);

    // at this point, either we are connected or we
    //  got an error, so close the status display
  InetDisplayStatus(linkID, myStatView, nil);

  if err then begin
    print("link failed");               // handle the error

    :endGrabLink(err);                 // end connect attempt
    end

  else       // status.linkStatus = 'connected, so resolve name
    DNSGetAddressFromName("apple.com", self, 'DNSCallback);
  end;
```

最初のステートメントのmyLinkID:=linkIDは、InetGrabLinkが接続しつつあるリンクのIDを、変数の1つに保存する。アプリケーションの他の部分で使用するために、リンクIDをストアしたい場合に有用である。

リンクの接続がエラーもなく進行していれば、コールバック関数が呼び出されて、進行状況を報告する。上記の例に示したように、InetDisplayStatus関数を呼び出すこともできる。この例で使われているmyStatViewビューは、リンクの接続が初期化される前に作成されている。

リンクの接続は、接続完了時あるいはエラー発生時に終了する。どちらの場合も、その時点でステータス表示を削除できる。そのためには、InetDisplayStatus関数を、ステータス・パラメータの値をnilに設定して呼び出す。

InetGrabLinkがエラーに遭遇すると、エラーコードは非ゼロ値になるので、アプリケーションはそのエラーを処理しなければならない。上記の例においては、メッセージが表示され、接続の試みが終了する。

InetGrabLinkが成功すると、コールバックは、linkStatusの値として'connectedを受け取る。この時点では、適切なオペレーションを何であれ実行できる。上記の例においては、この時点で、リモートエコー・ホスト名をIPアドレスへと変換し、それをDNSCallback関数がローカル変数内に保存する。

## リンクステータス情報の検索と表示

多くのアプリケーションは、ネット接続の確立中に、ユーザに対してステータスを表示するよう設計される。Newton Internet Enablerは、InetDisplayStatus関数により、この作業を容易にしている。この関数は、Newton画面上にリンクステータス情報を表示する。この関数の一例を次に示す。

```
statusView InetDisplayStatus(linkID, statusView, status)
```

InetDisplayStatus関数には、次の3つの使い方がある。

- 新たなステータスビューを作成するには、各パラメータの値としてnilを渡す。

```
myStatusView := InetDisplayStatus(nil, nil, nil);
```

- 現在のステータスビュー内にリンクステータスを表示するには、リンクID、ステータスビュー、およびコールバック関数へ送ったステータスフレームを渡す。

```
InetDisplayStatus(myLinkID, myStatusView, myStatus);
```

- ステータスビューを削除して廃棄するには、ステータスフレームの値としてnilを渡す。

```
InetDisplayStatus(myLinkID, myStatusView, nil);
```

InetStatusDisplay関数は、protoStatusTemplateに基づくビューを作成し、使用する。このプロトについては、Newton Programmer's Guideの“ Additional System Services ”の章を参照のこと。

ステータス表示を初期化するには、ステータスビューをオープンしなければならない。これを行うのに最も便利な場所は、InetGrabLink関数呼出しの直前である。例えば、次の関数は、ステータスビューを作成し、それを後で使用するときのためにmyStatView内にストアした後、InetGrabLink関数を呼び出す。

```
DoGrabLink := func()
  begin
  myStatView := InetDisplayStatus(nil, nil, nil);
  InetGrabLink(nil, self, 'GrabLinkCallback);
  end;
```

接続操作の進行中は、コールバック関数が呼び出されるたびに、ステータス表示を更新できる。例えば、次に示す、接続リンク・コールバック関数からのコードセグメントは、エラーが発生しないか、あるいはリンクステータスが 'connectedにならなければ、ステータス表示を更新する。

```
if err = nil and status.linkStatus <> 'connected then
  InetDisplayStatus(linkID, myStatView, stat);
```

接続操作が完了すると、ステータス表示を削除できる。次に示す、接続リンク・コールバック関数からのコードセグメントは、リンクステータスが'connectedになったときに、ステータス表示を削除する。

```
if err = nil and status.linkStatus = 'connected then
  InetDisplayStatus(linkID, myStatView, nil);
```

InetDisplayStatus関数が表示するビューは、ユーザがタップすることによりInetCancelLink関数を呼び出せるボタンを含んでいる。この関数は現在進行中の接続操作を取り消す。

## エンドポイント用Newton Internet Enablerの構成

Newton Internet Enablerのリンクを接続した後は、エンドポイントをインスタンス化しなければならない。アプリケーション用にNewton Internet Enablerを構成するのに必要なオプションを、Instantiateメッセージと共にエンドポイントへ送る。

Instantiateメッセージには、次の3つのオプションを設定しなければならない。

- 'inetサービス識別子オプションは、Newtonシステムソフトウェアに対し、エンドポイントでNewton Internet Enablerを使用するよう通知する。

- Inetツールフィジカルリンク（'ilid'）オプションは、エンドポイントにどのリンクIDを使用するか、Newton Internet Enablerに対し通知する。InetGrabLink関数が戻すリンクIDを使用すること。

- Inetツール・トランスポートサービスタイプ（'itsv'）オプションは、エンドポイントにどのトランスポートタイプ（例えば、UDPあるいはTCP）を使用するか、Newton Internet Enablerに対し通知する。

## Newton Internet Enablerによるエンドポイントの結合

エンドポイントをインスタンス化した後は、それをアドレスへ結合しなければならない。エンドポイントは、接続するか（出力接続を開始する）、あるいは入力接続を受信するか、そのどちらかのために結合する。受信するためのエンドポイントを結合する場合は、エンドポイントへBindメッセージを送るときに必ずInetローカルポート（'ilpt'）オプションを渡さなければならない。接続するためのエンドポイントを結合する場合は、Inetローカルポート・オプションは、TCPリンク用ではなく、UDPリンク用を渡さなければならない。

Inetローカルポート・オプションは、指定可能な2つのデータスロットをもつ。すなわち、短い値のInetPortNumberとブール値のuseDefaultPortである。useDefaultPort値は、UDPリンク接続でエンドポイントを結合するときのみ、適用される。ローカルポート・オプションをBindリクエストとともに送るときは、表1-1に示す値をInetPortNumberに割り当てること。

*表1-1　Newton Internet Enablerによる結合のためのローカルポート番号*

| 結合のタイプ | トランスポートサービスタイプ | ローカルポート番号 |
|---|---|---|
| 接続用 | TCP | システムが常にローカルポート番号を選択するため、このオプションは設定しないこと。ただし、Bindと共にこのオプションのゲット(opGetCurrent)を送り、システムが割り当てたポート番号を検索することはできる。 |
| 接続用 | UDP | useDefaultPortにtrueを指定すると、Newton Internet Enablerは使用するローカルポートを選択し、その値をオプション内へ戻す。useDefaultPortにnilを指定すると、使用していないポート番号を割り当てなければならない。さもないと、Bindは失敗する。 |
| 待ち受け用 | TCP | IETF RFC 1700: Assigned Numbers（1994年10月）に定義されている通りに、待ち受け用のポート番号を指定する。 |
| 待ち受け用 | UDP | IETF RFC 1700: Assigned Numbersに定義されている通りに、待ち受け用のポート番号を指定する。 |

## Newton Internet Enablerによるエンドポイントの接続

エンドポイントをインスタンス化し、結合した後は、それを接続しなければならない。TCPリンクを使用している場合は、エンドポイントへConnectメッセージを送るときに、TCPリモートソケット（'itrs'）オプションを渡さなければならない。このオプションは、TCPが接続するホストアドレスを設定する。ドメインネームサーバを使用して、このアドレスを得ることができる。

UDPリンクを使用している場合は、Connectメッセージ内にいかなるオプションも渡す必要はない。

入力接続を待ち受けるためにエンドポイントを使用している場合は、Listenメッセージと共にいかなるオプションも送る必要はない。

## データの送信

Newton Internet Enabler では、他のNewton コミュニケーションツールとまったく同じようにデータ送信ができる。接続を確立すれば、出力指定フレームをセットアップして、エンドポイントへOutputメッセージを送ることができる。

UDP接続の場合は、UDPデータグラムの宛先を確立するために、Inet UDP宛先ソケット（'iuds'）オプションを指定しなければならない。UDP出力指定は、sendFlagsスロット内に2つのフラグ、すなわちkPacketフラグとkEOPフラグを含んでいなければならない。例えば、UDPリンクで“Hello World!”という文字列を送信するには、次のようなコードセグメントになる。

```
local myUDPstreamOutputSpec := {
  form:         'string,
  sendFlags:    'kPacket+'kEOP,
}

local myUDPOptions :=
[{
  label:      "iuds",
  type:       'option,
  opCode:     opSetCurrent,
  result:     nil,
  form:       'template,
  data:
  {
    arglist:
  [
    130,          // byte 1 of host address
    43,           // byte 2 of host address
    2,            // byte 3 of host address
    2,            // byte 4 of host address
    7,            // destination port number
  ],
    typelist:
    [
      'struct,
      'byte,
      'byte,
      'byte,
      'byte,
      'short
    ]
  }
}];

try
  ep:Output("Hello World!", myUDPOptions,
                          myUDPstreamOutputSpec);
onexception |evt.ex.comm| do
  return :DoDisconnect();
```

TCPリンクの場合は、Outputメッセージ内にいかなるオプションも指定する必要はない。また、出力指定フレーム内のsendFlagsにいかなる値も指定する必要はない。例えば、TCPリンクで“Hello World!”という文字列を送信するには、次のようなコードセグメントになる。

```
local myTCPstreamOutputSpec := {
  form:     'string,
}

try
  ep:Output("Hello World!", nil, myTCPstreamOutputSpec);
onexception |evt.ex.comm| do
  return :DoDisconnect();
```

上記2つの例においては、データ送信中に例外処理が起これば、アプリケーションのDoDisconnect関数が呼び出される。

TCPリンクで優先データを送信することもできる。優先データとは、即時送信される1バイトのデータである。このデータバイトは、リモートエンド上の、受信済だが未処理であるデータの先頭に挿入される。例えば、大量のデータを伝送している途中で、ブレークを送信しなければならない場合がある。こういう場合は、Outputメッセージと共にInet優先データ・オプションを使用すればよい。

出力指定フレームおよびOutputメソッドの詳細については、Newton Programmer's Guideの“Endpoint Interface”の章を参照のこと。

## データの受信

Newton Internet Enabler では、他のNewton コミュニケーションツールとまったく同じようにデータ受信ができる。通常、これは、入力指定フレームをセットアップし、エンドポイントへSetInputSpecメッセージを送ることを意味する。

UDPリンクの場合、入力指定フレームはkPacket受信フラグを含んでいなければならない。また終了スロット内に、useEOP:trueを含んでいなければならない。さらに次の場合は、rcvOptionsスロット内に2つのオプションを指定できる。すなわち、データグラム送信側のアドレスを取り込みたければ、UDPソースソケット・オプションを指定し、また、受信したパケットの正確な送信先アドレスを取り込みたければ、 UDP宛先ソケット・オプションを指定する。パケットがブロードキャストアドレス(broadcast address)へ送信された場合は、宛先アドレスはローカルアドレスとは異なるかも知れない。

UDPリンクでデータグラム・パケットを受信するには、次のようなコードセグメントになる。

```
local streamInputSpec := {
  form:           'string,
  termination:    {useEOP: true},
  discardAfter:   256,
  rcvFlags:       kPacket,
  rcvOptions:     {
                label:   "iuss",
                type:      'option,
                opCode:    opGetCurrent,
                result:    nil,
                form:      'template,
                data: {
                     arglist:
                     [
                       0,// host addr - byte 1
                       0,// host addr - byte 2
                       0,// host addr - byte 3
                       0,// host addr - byte 4
                       0,// host port number
```

```
                    ],
                    typelist: kPortAddrStruct,
                    [
                      'struct,
                      'byte,
                      'byte,
                      'byte,
                      'byte,
                      'short
                    ]
                  }
                },

  inputScript: func(ep, data, terminator, options)
    begin
    // do something with data
    end,

  completionScript: func(ep, options, result)
    begin
      // skip error handling for canceled requests
    if result <> kCommAbortErr then
      begin
      print("Error: " && result);
      ep:DoDisconnect();
      end;
    end,
}

try
  ep:SetInputSpec(streamInputSpec);
onexception |evt.ex.comm| do
  return :DoDisconnect();
```

上記の例の入力指定フレームは、Newton Internet Enablerに対し、UDPリンクからのデータ・パケットを受信するよう指定し、2つのスクリプトを提供する。すなわち、データ受信の正常終了を処理するinputScript関数と、データ受信の予期せぬ終了を処理するcompletionScript関数である。さらに、この入力指定には、UDPソースソケット・アドレスの“get（取り込み）”が含まれており、ここには、こちらのアプリケーションへデータグラムを送信したホストのIPアドレスが入る。

TCPリンクの場合、入力指定フレーム内には、いかなるオプションも受信フラグも指定する必要はない。例えば、TCP接続から、改行コードで終了した文字列を受信するには、次のようなコードセグメントになる。

```
local streamInputSpec := {
  form:           'string,
  termination:    {endSequence: UnicodeCR},
  discardAfter:   256,

  inputScript: func(ep, data, terminator, options)
    begin
    // do something with data
    end,

  completionScript: func(ep, options, result)
    begin
      // skip error handling for canceled requests
    if result <> kCommAbortErr then
      begin
      print("Error: " && result);
      ep:DoDisconnect();
      end;
    end,
}

try
  ep:SetInputSpec(streamInputSpec);
onexception |evt.ex.comm| do
  return :DoDisconnect();
```

上記の例の入力指定フレームは、Newton Internet Enablerに対し、Unicodeの改行コード文字を受信したら、入力を終了するよう指定し、2つのスクリプトを提供している。すなわち、データ受信の正常終了を処理するinputScript関数と、データ受信の予期せぬ終了を処理するcompletionScript関数である。

TCPリンクで優先データを受信することもできる。優先データを受信すると、アプリケーションは即座に通知を受け取る。すなわち、リンクコントローラがアプリケーション・イベントフレームを送る。このイベントフレームのeventCodeスロットは値kEventToolSpecificをもち、dataスロット値は受信したバイト数である。

入力指定、SetInputSpecメソッド、コミュニケーションイベントのハンドリング、およびエンドポイントを用いるその他のスタイルのデータ受信については、Newton Programmer's Guideの“Endpoint Interface”の章を参照のこと。

## エンドポイントの切断

エンドポイントの使用を終了したら、それを切断し、接続解除し、廃棄しなければならない。エンドポイントの使用を終了するには、次のような関数を使う。

```
MyApp.DoDisconnect := func()
  begin
  if ep then begin     // ignore all disconnect errors
    try
        ep:Disconnect(true, nil);
    onexception |evt.ex.comm|  do
        nil;

    try
        ep:UnBind(nil)
    onexception |evt.ex.comm| do
        nil;

    try
        ep:Dispose()
    onexception |evt.ex.comm| do
        nil;
    end;
  end;
```

## リンクの解放

アプリケーションがリンクの使用を完全に終了した後は、あるいはリンクを長い間使用しない場合は（およそ15分以上）、InetReleaseLink関数を呼び出して、リンクを解放しなければならない。他のアプリケーションがリンクを使用していなければ、Newtonシステムソフトウェアがリンクを遮断する。

InetReleaseLinkには、リンクID、コールバック関数、およびコールバック・コンテキストフレームを指定しなければならない。

```
InetReleaseLink(linkID, clientContext, clientCallback)
```

## ドメインネームサービスインタフェースの使用

Newton Internet Enablerドメインネームサービス関数を使用すれば、ホスト名とインターネットアドレスを相互に変換できる。Newton Internet Enablerには、次のドメインネームサービス・グローバル関数が用意されている。

- DNSCancelRequests関数は、未完のDNSリクエストを取り消す。
- DNSGetAddressFromName関数は、ドメインネームを、対応するインターネットアドレスへ変換する。
- DNSGetMailAddress関数は、ドメインネームを、そのドメインのメールサーバ用インターネットアドレスへ変換する。
- DNSGetMailServerNameFromDomainName関数は、ドメインネームを、そのドメインのメールサーバ用ドメインネームへ変換する。
- DNSGetNameFromAddress関数は、インターネットアドレスを、対応するドメインネームへ変換する。

リンクコントローラ関数の場合と同様、どのDNS関数に対しても

clientContextパラメータと clientCallbackパラメータを指定しなければならない。ただし、DNSコールバック関数は、リンクコントローラ関数とは異なるパラメータを使用して呼び出す。

DNSCancelRequestsのコールバック関数は、パラメータを受け取らない。

その他のDNS関数のコールバック関数はすべて、2つのパラメータを受け取る。すなわち、DNSリザルトフレームの配列とリザルトコードである。各リザルトフレームには、実行されたDNSオペレーションを記述する多数のスロットが含まれている。例えば、DNSGetAddressFromName関数は、次のように宣言する。

```
DNSGetAddressFromName(addr, clientContext, clientCallback)
```

この関数のコールバックの例を次に示す。

```
myApp.DNSGetAddrcallback := func(results, error)
  begin
  if error or length(results) < 1 then
    begin
    print("DNS error: " && error);
    // do something with the error
    return;
    end;

    // save the resolved address
  myRemoteIpAddr := results[0].resultIPAddress;
  end;
```

各リザルトフレームには、タイプスロットと少なくとも1つのリザルトスロットが含まれている。リザルトフレームには、たいていtargetDomainNameスロットが含まれている。しかし、これは保証されていない。表1-2は、各DNSオペレーションに対して、必ず有効であるスロットを示す。

*表1-2： 各DNSオペレーションに対するリザルトスロット*

| DNSオペレーション | リザルトフレームスロット |
|---|---|
| DNSGetAddressFromName | resultIPAddress |
| DNSGetNameFromAddress | resultDomainName |
| DNSGetMailServerNameFromDomainName | resultDomainName |
| DNSGetMailAddressFromName | resultIPAddress |

例えば、DNSGetAddressFromName関数は、次のようなリザルト配列を戻す。

```
[ {
type: kDNSAddressType,
targetDomainName: "newton.apple.com.",
resultIPAddress: [155,227,54,3]
} ]
```

逆に、DNSGetNameFromAddress関数は、次のようなリザルト配列を戻す。

```
[{
type: kDNSDomainNameType,
targetDomainName: "newton.apple.com.",
resultIPAddress: [155,227,54,3]
} ]
```

DNSオペレーションには、1つ以上のリザルトフレームを含むリザルト配列を戻すものもある。例えば、メール交換オペレーションは、複数のメール交換リザルトフレームを生成し得る。

## Newton Internet Enablerオプションの使用

Newton Internet Enablerのリンクは、さまざまなコミュニケーションオプションを用いて構成する。他のコミュニケーションツールの場合と同様、これらのオプションをエンドポイント・メソッド呼出しと共に送信すればよい。表1-3は、Newton Internet Enablerのオプションである。どのオプションをどのエンドポイント・メソッドと共に送信するかを示す。

*表1-3： Newton Internet Enablerのオプション*

| オプション名 | 説明 | いつ使用するか |
|---|---|---|
| 優先データ転送（'iexp'） | TCPリンクでの優先データ伝送。 | TCPエンドポイント上のデータを転送するOutput呼出しに、このオプションを設定する。 |
| フィジカルリンク識別子（'ilid'） | 使用するリンクIDを識別する。 | エンドポイントのインスタンス化時にこのオプションを設定する。 |
| ローカルポート（'ilpt'） | TCP結合のローカルポート番号を設定する。 | 待ち受け（Listen）を実行するための結合にこのオプションを設定する。（エンドポイントのインスタンス化時または結合時）このオプションは、Connectに対して設定する必要はない。 |
| | UDP結合のローカルポート番号を設定する。 | このオプションは、エンドポイントのインスタンス化時または接続時に取り込む。 |
| | TCPまたはUDPで使用するローカルポート番号を取り込む。 | 接続またはデータの送受信時に、このオプションの値を取り込む。 |
| TCPリモートソケット（'itrs'） | TCPの接続先ソケットを設定する。 | このオプションの値は、接続を使用する前（エンドポイントのインスタンス化時、結合時、または接続時）に設定する。 |
| | TCPリンクで受信したデータの送信側アドレスを取り込む。 | このオプションの値は、TCP接続のデータを待ち受けするときに取り込む。 |
| トランスポートサービスタイプ（'itsv'） | トランスポートサービスタイプを設定する（TCPまたはUDP）。 | このオプションは、エンドポイントのインスタンス化時に設定する。 |
| UDP宛先ソケット（'iuds'） | UDPリンクで送信したデータの宛先アドレスを設定する。 | このオプションは、UDP接続でデータを送信するときに設定する。 |
| | UDPリンクで受信したデータの宛先アドレスを取り込む。 | このオプションは、UDP接続のデータを受信するときに取り込む。 |
| UDPソースソケット（'iuss'） | UDPリンクで受信したデータのソースアドレスを取り込む。 | このオプションの値は、 UDP接続のデータを受信するときに取り込む。 |

## Newton Internet Enablerの優れた接続性

Newton Internet Enablerにより、開発中のNewtonScriptアプリケーションにインターネットへの接続性を付与するのが容易になった。データの送受信を行うNewtonコミュニケーションエンドポイントの使用法を理解すれば、インターネットでのデータ送受信を行うInetツールの使用法はわかるはずである。

さらにもう1つ、Newton Internet Enablerを使用するために理解しておかねばならないことは、リンクの概念である。所定のインターネットサービスプロバイダとのインターネットセッションを確立するために、リンクを確立する。各リンクは、所定のトランスポートサービス（TCPまたはUDP）を扱い、所定のリンクレベル・プロトコル（PPPまたはSLIP）を使用するように構成される。単一のNewtonデバイス上では、同一時点では、ただ1つのリンクのみがアクティブにできるが、複数のアプリケーションがそのリンクを共用できる。したがって、ユーザは様々なインターネット関連のタスクを並列処理することができる。リンクを確立するには、InetGrabLink関数を呼び出す。InetGrabLink関数は、Newton Internet Enablerが提供する、リンク関連のグローバル関数の1つである。

リンクがオープンされたなら、標準のエンドポイント・メソッドを使用して、データの送受信を行う。リンクの使用が終了したなら、InetReleaseLink関数を呼び出す。リンクを使用するすべてのアプリケーションが終了すると、Newtonシステムはリンクをクローズし、そのリンクに関連するすべてのリソースを開放する。

Newton Internet Enablerは、ドメインネームサービス変換関数を持ち、

ドメインネームとそれに対応するIP番号を相互に交換できる。

Newton Internet Enablerにより、Newtonデベロッパは、非常に短期間でInternetをうまく利用したプログラムを、Newtonユーザ向けに開発することができる。このことにより、Newtonプラットフォームは、ますます利用価値があり、魅力のあるものとなる。皆さん、頑張ってください。

NTJ

Communications Technology

# イネーブラキット（Enabler Kits）について

*by Eileen Tso, Apple Computer, Inc.*

Newton OS 2.0の出現により、コミュニケーションの能力が今までになく重要になります。Newton Systems Groupは、有用なコミュニケーション用のアプリケーションが着実にNewtonプラットフォーム用に開発されますよう、デベロッパの皆様からのサポートとフィードバックに期待しています。本号の“Newton 2.0のコミュニケーション戦略”では、我々が作成中の数多くのツールと“Kits（キット）”に言及しています。デベロッパの皆様は、これらを利用して、 Newtonプラットフォームとそのソリューションのために我々が計画していますビジョンを成し遂げて頂けるものと思います。

## eMail Enabler（eMailイネーブラ）

初めに、Newton eMail Enablerについては多くの方がご存じのように、Newton OS 2.0が達成した重要なことの１つが、トランスポートと汎用の送受信箱（Universal In/Out Box）であり、これらを用意することで、ユーザは電子メールの選択とこれに付随する面倒な手順から解放されました。しかし、MessagePadにeWorldのソリューションを組み込んだことで、デベロッパにとってはとても簡単に、これを“サンプルコード”として利用でき、他の電子メールソリューション（例えば、Internet POP/SMTP、LANベースの電子メールアクセス、ダイヤル・イン、等）用のクライアントとして転用できるようになったのではないか、と思います。もはやeWorldは存在しなくなり、クライアントも基本的に不要となった現状で、eMail Enablerが最初から、eWorldのあるなしにかかわらず提供されるべきだった、ということが明白になりました。

“Newton 2.0のコミュニケーション戦略”の記事で触れたように、eMail Enablerは、NTKのストリームファイル、電子メールトランスポートの要素、ステーショナリ、ビューなどを含んでいます。ご記憶の方もいらっしゃると思いますが、DTSで昨年、既存のデベロッパのうち何人がこのようなツールを必要としているかが問われました。これには大きな反響が寄せられ、eWorldコードに替わる選択肢の提供を要求したのです。この夏には、電子メールソリューションが完成し、組み込み型のクライアントや“サンプルコード”が不要となるはずです。

## Messaging Enabler（Messagingイネーブラ）

Enabler キットの２番目の要素が、Newton Messaging Enablerです。今までの噂や話題の中から、このツールのことを“Paging Enabler”としてお聞きになられた方もいらっしゃると思います。しばらく前にも、ページカードのようなデバイスを使って、もっとシームレスにデータの転送を行う必要が感じられていました。この業界が、技術的な障害を克服しはじめ、ポケベルがメッセージ機能付きデバイスあるいはポケット型無線機のようになるにつれ、Paging Enablerとして始まったものが、はるかに強力かつ有用なMessaging Enablerとなったのです。eMail Enablerもそうですが、 Messaging Enablerも、すべてのデベロッパが必要とするわけではありません。これは、魔法のブラックボックスとミドルウェアを組み合わせたコードではなく、既存のアプリケーションが呼び出しメッセージを送ることができるようなエキサイティングなアプリケーションでもありません。しかし、このような話の流れからそれほどかけ離れたものとしてではなく、選択肢としてMessagingデバイスに対し音声で通信することができるのです。そこからは、 Newton 2.0のトランスポート機能が引き継ぎ、さらにアプリケーションへと引き渡すことになります。

## Internet Enabler（Internetイネーブラ）

“Newton 2.0のコミュニケーション戦略”の記事で言及され、Gary Hillersonの“Newton Internet Enabler”で詳述された最後のイネーブラが、Newton Internet Enabler (NIE)です。簡単にいえば、NIEはNewton TCP/IPです。他のイネーブラと同様、NIEはすべてのコミュニケーションアプリケーションのデベロッパに必要というわけではありません。しかし、他のイネーブラと同様、このイネーブラもこの夏にはデベロッパが自由に入手できるようになります。

最後に、私共はデベロッパの皆様のたゆまぬご支援とNewtonプラットフォームへの貢献を心から感謝しております。そして、いつもながら、私共は、皆様との協同作業の成果が実りあるものとなることを望んでおります。

NTJ

# Newton 2.0のコミュニケーション戦略

*by Barney Dewey, Apple Computer, Inc.*

今日、“作業環境”の定義は急速に変わりつつある。人々は、仕事のホームベースが会社のデスクであれ、自分の家にあるホームオフィスであれ、そのホームベースを、ますます遠く離れて動きまわるようになった。仕事に関する情報は、今や自分のデスクトップのハードディスクに格納されるのみならず、社内のMIS担当者が管理するサーバにも、さらには遠隔地にある情報サービスプロバイダのサーバにも格納されている。仕事をする場所を選ぶのは、今までになく容易になったが、必要な情報との接触を保ち、何時でもアクセスできるためには、異なるプロトコル、インタフェース、ソフトウェアを持った、複数のコミュニケーション手段を駆使することが必要である。これは複雑なプロセスにもなり得る。

Newton OS 2.0は、広範囲のコミュニケーション・テクノロジーをサポートしている。モバイルデバイスのユーザが、有線であれ無線であれ情報交換の手段を必要とする時、Newton OS 2.0はそれぞれの分野で数多くのコミュニケーション・プロトコルと標準をサポートしている。その中には以下のようなものがある：

- ファクスの送受信（組み込み）
- 電子メール（組み込みおよびサードパーティ製）
- 一方向および双方向のページングとメッセージング
- ネットワーキング（組み込みのAppleTalk）
- インターネットのプロトコル（TCP/IP, UDP, PPP）
- パケット無線ネットワーク（ARDIS, CDPD, RAM）
- 携帯電話によるファクスとデータ通信（AMPS, digital, GSM）
- PCS ファクスとデータ通信（CDMA, PCS-1900, TDMA）
- ワイヤレスLANのサポート

## Newtonプラットフォームのコミュニケーション戦略

Newtonプラットフォームの基本的なコミュニケーションの戦略は3つの部分よりなる：

核となるコミュニケーション、無線通信、およびインターネットへの接続性である。

Newton Systems Groupは、この戦略を実現するためにさまざまなプログラムを実施している。いくつかの例を以下に示す：

- 最も重要な機能を組み込む（例えば、シリアル接続、ファクス）
- デベロッパが組み込み機能を拡張し、新機能を開発するサポート
- コミュニケーション用のAPIとイネーブラをデベロッパに提供するサポート
  (イネーブラについては以下に触れる。)

Newton OS 2.0は、完全なコミュニケーション用APIの集合を持ち、それにはトランスポート(Transport)、ルーティング(Routing)、送受信箱(In/Out Box)ならびにコミュニケーション用スクリプトが含まれる。多くのコミュニケーション用アプリケーションは、1つあるいはそれ以上の共通機能を持つので、高レベルのプログラムインタフェースを提供すれば、多くのコミュニケーション用アプリケーション開発が推進される。このような高レベルのインタフェースを提供するデベロッパ向けの機能をenabler（イネーブラ）と呼ぶ。最初に提供されたのがModem Enablerである。この記事では、次の3つの新しいイネーブラを紹介する：Newton eMail Enabler、Newton Messaging Enabler、およびNewton Internet Enablerである。

イネーブラのいくつかは、例えばNewton eMail EnablerやNewton Messaging Enablerは、高レベルのAPIとコードを提供することにより、デベロッパの開発過程を合理化する。他のイネーブラ、例えばNewton Internet Enablerは、新機能を提供する。(詳細については本号の“イネーブラキット（Enabler Kits）”の記事を参照のこと。)

アップルのNewton Systems Groupには、コミュニケーションの分野を担当している次の5つのグループがある：デベロッパ・リレーションズ(Developer Relations)、デベロッパ・テクニカルサポート(Developer Technical Support)、コミュニケーション用OS開発(Communications OS Engineering)、コミュニケーション・ソリューション開発(Communications Solutions Engineering)、およびプロダクト・マーケティング(Product Marketing)である。このなかの、デベロッパ・リレーションズとデベロッパ・テクニカルサポートの2つのグループは、Newtonのコミュニケーション戦略でもっとも重要な地位を継続的に占めるサードパーティ・デベロッパのサポートに専念している。この記事で述べるイネーブラキットは、コミュニケーション用OS開発とコミュニケーション・ソリューション開発の2つのグループが開発した主要となるイネーブラ技術を、デベロッパが利用できるようにする方法の1つである。

## 核となるコミュニケーション機能

われわれの第一の戦略目標は、 Newtonデバイスに、すべてのPDA(Personal Digital Assistance)製品のなかで最も優秀なコミュニケーション機能を実現することである。核となるコミュニケーション機能は、基本的なコミュニケーション機能をはるかにこえたものである：すなわち、ユーザがラップトップやデスクトップ型コンピュータに期待するすべてのコミュニケーション機能が、核となるコミュニケーション機能に含まれる。今日、Newtonデバイスには、多くのラップトップやデスクトップより、もっと広範囲なコミュニケーション機能が組み込まれている。この主導的地位を、将来のNewton PDAにおいて、さらに拡張するつもりである。

以下、核となるコミュニケーションの要素ならびに新しい電子メール機能について述べる。

*汎用送受信箱（Universal In/Out Box)*：この機能により、多くのコミュニケーションモードに対し共通のアクセスと制御を行う。対象としては、電子メール、ファクス（送受信)、赤外線ポートでの通信、ワイヤレスメッセージング、および印刷がある。

*モデムサポート*：Newton OS 2.0では、ユーザは、シリアルポート経由またはPCMCIA標準のPCカードのスロットを使用してモデムに接続できる。Newton 2.0は、モデム経由のデータ転送速度として、28.8 Kbps以上をサポートしている。NewtonのModem Enablerにより、デベロッパやモデムのメーカは、ほとんどのモデムをNewton 2.0プラットフォームに接続できるよう、スクリプト（われわれは、これを”setups”と呼んでいる。)を書くことが容易にできる。

*ネットワークへの接続性*：アップルのNewtonプラットフォームの戦略的な方向の1つとして、もっとも人気のあるPCネットワークへの接続性を実現することがあげられる。このため、Newton OS 2.0では、AppleTalkが使われている。AppleTalkをサポートすることにより、ほとんどのMacintoshとWindows NTのネットワークに接続できる。将来のNewton Internet Enabler（本号の“イネーブラキット(Enabler Kits)について”の記事を参照のこと。）のバージョンでは、TCP/IPによるEthernetネットワークへの接続がサポートされる予定である。

*企業内LANへの接続*：Newton OS 2.0は、拡張AppleTalkスタックを持つ。したがって、ユーザは、どのノードであれ、そのノードのAppleTalkのコネクタに接続するだけで、AppleTalkのLANにアクセスできる。Windows NTも、AppleTalkを標準のネットワークの1つ

(AppleTalk ADSP)としてサポートしており、ほとんどのWindowsネットワークにもアクセスできる。

*Ethernet LANへの接続*：われわれのネットワーク戦略は、AppleTalkを拡張し、TCP/IPでEthernetに対する接続性を実現することである。アップルのイネーブラのバージョンアップとサードパーティのデベロッパによるEthernetドライバならびにEthernetアクセスカードで、来年はこの機能を実現する計画である。

*ファクス送受信*：ファクスを完全にサポートすることは、Newtonプラットフォームのコミュニケーション機能を拡張するわれわれの戦略の一部である。Newton OS 2.0では、ファクスを送信するだけでなく、受信する能力をも持たせることができ、オプションとして、受信したファクスを他人に転送する前に注釈を付けることができる。コミュニケーション機能を継続して更新するわれわれの戦略の一部として、クラス２のファクス機能（例：速度の遅いG2）を追加している。これは、北米以外の市場でとくに重要である。

名前とファクス番号情報は、Nameファイルに格納され、Newton OS 2.0のファクス送信機能と密接に統合されている。Newton OS 2.0は、複雑なダイヤル呼び出しコードと複数の電話番号をサポートする。

*印刷*：Newton PDA戦略では、ネットワークでシリアル接続したプリンタによる印刷をサポートしている。AppleTalk経由の印刷もサポートされる。多くのアップル製プリンタのドライバソフトは組み込まれている。よく使われるアップル以外のプリンタのサポートは、Newton Print Packに入っている。TCP/IPでEthernetに接続して行う印刷は来年中にサポートされる計画である。

*電子メール*：Newtonの電子メール戦略は、広範囲の電子メールをサポートするためのアーキテクチャと開発環境を提供することである。サポートの対象となるものには、次のようなものが含まれる；オンラインで電子メールのクライアントとして、CompuServeやAmerica Onlineなどのサービスに接続する；リモートのメール・クライアント経由でLotusのcc:Mail、CE QuickMail、Microsoft MailなどのLAN電子メールに接続する；無線電子メールでRadioMailとWyndMailに接続する；QualcommのEudoraなどのアプリケーションを通じ、インターネットのメールに接続する。

上記の戦略的方向をサポートするため、現在Newton eMail Enablerを開発中である。このeMail Enablerは、次の要素からなる：

- Newtonプラットフォーム上に、 電子メールとメッセージングのためのクライアントを構築する目的で使用する、公式に記録され、推奨される、ユーザインタフェース
- 公式に記録され、検証されたAPI
- NTKのストリーム・ファイルで、電子メールのアプリケーションとの共通要素。

その中には、ルーティングスリップ、接続スリップ、送受信箱ヘッダ、電子メールステーショナリ、および設定スリップが含まれる。

## 無線通信

最も優れた無線通信機能を備えることは、Newtonプラットフォームの将来にとって重要である。仕事をする人達が、ますます電子サービス（例えば、電子メール、本社のデータベースへのアクセス、インターネットへのアクセス）に依存し、自分のデスクから離れて過ごす時間が増えるにつれ、無線によるアクセスは、多くの産業と職業において必要不可欠となるであろう。われわれは、ライセンス先との密接な協力関係のもとに統合化されたワイヤレスソリューションを提供しようとしている。われわれは、さらにType IIのPCカードによる様々なワイヤレスネットワークのサポートのような、アップル製品(MessagePadも含めて）に対しての拡張手段をも提供している。

次に、Newtonプラットフォームのキーとなるコミュニケーション機能と新しい一方向および双方向のワイヤレスページングとメッセージングについて述べる。

*携帯電話への接続*：われわれは、人気の高い、cellular/faxモデム付きの携帯電話への接続性を今後もサポートしていく。Newton Modem Enablerは多くの携帯電話搭載のモデムをサポートする機能をもっている。プロフェッショナルが、ダイヤル呼出し（例えば、社内のLANの電子メールやインターネットなど）を行うニーズは引き続き増えている。多くのユーザはNewtonプラットフォームと組み合わせて使うことにより、携帯電話を有効化しようと望んでいる。

現在、われわれは、CDPD (Cellular Digital Data Packet)の開発に注目している。

このセルラー技術は、セルラー・ボイス・ネットワークでデータ転送ができ、無線データのスループットが高い。いままでのところ、CDPDは特定の企業向けのアプリケーションに主として使われており、全米にわたる一般的な標準になるとは考えられていない。

*GSM*：GSMは、デジタル携帯電話の標準として北米以外の地域でよく使われている。もしGSMがPCSバンドのPCS-1900標準として広範囲に採用されるなら、北米でも人気がでる可能性がある。Newtonプラットフォームの GSM戦略では、多くのブランドの電話に対応する内蔵データアダプタを組み込み、その他の機器用にはModem Enablerとデバイスドライバを持つことで、引き続きサポートしている。

*PCS (Personal Communication Services)*：Newton 2.0は、将来のPCS-1900、TDMA、およびCDMAデータシステムをサポートするコミュニケーションのアーキテクチャを持っている。これらの技術のいくつかは、今後数年の内に北米で実施されるだろう。そのことにより、無線通信のコストが下がり、パーソナル電話番号（特定の機器に番号が付くのではなく、個人についてまわる番号のこと）や高速データ通信(64 kbps又はそれ以上）などの高度化されたサービスが実現するだろう。

*ワイヤレスLAN*：ユーザの中には、エリア内をたえず動きまわらねばならないが、ローカル・エリア・ネットワーク（例えば、倉庫や広いキャンパスなど）に接続している必要のある人達がいる。Newton 2.0では、このようなモバイルユーザが、Newton PDAのワイヤレス機能を通じてLANとの接続を維持し、高い転送速度を維持できる。われわれは、数多くのバーティカルユーザに、Newtonのデバイス・ドライバ・キットを提供することで、真の携帯性を与える“ワイヤレスEthernet”タイプの機能をサポートしている。2.4 GHzと赤外線ポートによるソリューションは、サードパーティのデバイス提供者から出荷されている。

*パケット無線*：今日では、Newton 2.0 PDAユーザがARDISやRAMなど、双方向のパケット無線サービスに接続できる、PCカード・ソリューションが存在する。これらのサービスにより、米国国内では全国的な無線データ通信が可能である。ヨーロッパでは、Mobitexネットワーク（Ericsson)とDataTAC（Motorola)が、広範囲に使用可能である。われわれの戦略では、現在および将来にわたって、このようなネットワークとデバイスに対するサポートを開発するため、デベロッパやデバイス・メーカと密接な協力関係を維持することである。

*赤外線 (IR)*：Newton 2.0には赤外線ビーム伝送機能が組み込まれているので、Newton PDAを使って、ユーザがお互いに“beaming（赤外線でビームする）”で、Notes、名刺、呼出し記録などのNewtonのデータの交換を行える。われわれのIR戦略は、標準のIRトランスポートが幅広く利用されるようになるにつれて、これらの標準に準拠して行こう、というものである。今年の終わりまでには、IrDAをサポートする製品を出す計画である。デベロッパにとって、Newtonプラットフォーム用に、IrDA経由でデスクトップのアプリケーションやその他のデバイス（例えば、プリンタや特化したデバイス）にデータを転送する、ドライバと接続用アプリケーションを作成するといった沢山のビジネスチャンスがある。

*一方向および双方向ページングならびにメッセージング*：Newton 2.0コミュニケーション・アーキテクチャにより、モバイルコンピューティングのプロフェッショナル達は、Newton PDAでページ（英数字と双方向ページの両方）を送受信できる。Newton OS 2.0は、複数のページングカードをサポートし、メッセージはすべて汎用送受信箱(Universal In/Out Box)で受け取るので、集中した、効率の高い、メッセージ交換ができる。より広範囲なページングテクノロジーのサポートとデベロッパのアプリケーションにより、Newtonプラットフォームは、無線によるデータベースの自動更新、株式や債権の取引、リモートシステムのチェック等、メッセージング以外のアプリケーションをサポートできる。

ページングとメッセージングにおけるこのような戦略的方向をサポートするため、われわれは、Newton Messaging Enablerを開発中である。

このMassaging Enablerは、デベロッパにとっては開発ツールであると同時に、エンドユーザにとっては、アプリケーションである。これは次の要素からなる：

- “ routing slip ”と“ PutAway ”関数を通じて、デベロッパ・アプリケーションとの統合を容易にするための、ドキュメント化されたAPI。
- テキストとデータを読み込み、送信する簡単なエンドユーザ・アプリケーション（このアプリケーションは、デベロッパにより、公式に記録されたAPIを使用して拡張可能である。）
- 一方向のページャ用デバイスドライバ(SocketのPageCardとMotorolaのNewsCard)、および双方向ページャ用ドライバ(MotorolaのTango)。デバイスのデベロッパは、新しいページングデバイスが利用可能になる度に、追加のドライバを書くことができる。

## インターネットコミュニケーション

インターネットのテクノロジーは、Newtonにおけるコミュニケーションの戦略に不可欠である。これらのテクノロジーは、EthernetとAppleTalkの２つのネットワークの組み合わせにより、より優れたネットワーク機能を発揮する。おそらく、もっともエキサイティングなのは、メーカ固有のクライアントにかわるものとしてのインターネット・テクノロジーの動向であろう。（例えば、PPP、TCP/IP、HTML、GIF、POP/SMTP）

その動向の一例として、Lotus cc:Mail for World Wide Webリリース1.0があげられる。このソフトウェアは、cc:Mail クライエントのソフトを必要とせずに、完全な cc:Mail をサポートすることができる。もう1つの例は、Newtonプラットフォーム上ではLotus NotesやOracle SQLデータベース･クライアント･ソフトウェアを使用せずに、Lotus InterNotesまたはOracle WebServerを使って企業内プロセスおよびデータベースにアクセスする方法である。

モバイル機器のプラットフォームをインターネット・テクノロジーに企業が使用する利点は、次の通りである：

- 企業側のデータベースの変更は必要ない。
- PDAデバイスのメーカ固有のアプリケーションやプロトコルの制約のない、モバイル機器のプラットフォームは、“ Universal Client ”になり、モバイル機器側にソフトを必要とせずに、企業のデータベースに完全なアクセスができる。

インターネットのテクノロジーをNewtonプラットフォーム上で実現するため、現在4つの戦略的な開発が行われている：サードパーティによるインターネット・アプリケーション開発、新しいハードウェア機能の強化、イネーブラソフトウェア開発、およびアップルブランドのソリューション開発である。

インターネット・アプリケーション開発。われわれは、HTML、電子メール、newsgroups、Telnet、およびFTPなどの中心的なインターネット・アプリケーションについては、デベロッパに依存しながらも、かつ密接な共同作業をしている。これらの分野でも、又その他のインターネットの分野でも、デベロッパには多くのビジネスチャンスがある。

新しいハードウェア機能の強化。Newtonプラットフォームをインターネット・テクノロジー実現のために強化する目的で、われわれは、これから来年（訳注：本記事は1996年6月号であり、この時点で来年とは1997年をさす。）にかけて、ハードウェア分野で3つの変更を行う：

- *メモリ*：メモリを拡張するとコストが増加するのが常である。われわれは、アップル製品に組み込むシステムメモリを拡張し、TCP/IPコミュニケーション機能とインターネット・アプリケーションを同時に実行する要求を満たそうとしている（例えば、MessagePad 130)。さらに、われわれは、モジュラタイプのメモリを提供する方法も検討している。

- *グラフィックス*：グラフィックスは、急速に、インターネット環境の重要な部分となりつつある。われわれは、スクリーン機能を改善して、Newtonプラットフォームがグラフィック・ディスプレイを持つことを検討中である。

- *速度*：グラフィックをサポートし、同時に複数のアプリケーションを実行し、さらにTCP/IPによるコミュニケーション機能を実現するとなると、プラットフォームの性能強化が必要となる。現在(訳注：本記事は1996年6月号である。)われわれは、NewtonプラットフォームにStrongARMテクノロジーを使用することを検討中である。

*イネーブラのテクノロジー*：インターネットのアプリケーションを実行するには、一連の、インターネット共通のテクノロジーが必要である。それらは、TCP、UDP、IP DNR、PPP、SLIPなどである。したがって、われわれは、Newton Internet Enabler (NIE)を開発中である。NIE 1.0は、上記のプロトコルとISPへの接続を確立するスクリプティング・エンジンを含む、完全なデベロッパ用キットを提供する。NIEの将来のバージョンは、この1.0のインターネット・テクノロジーの基盤の上に構築される。(NIEは、デベロッパにだけ、デベロッパ・アプリケーションに組み込むツールとして提供される。その場合のライセンスと取扱いは、DILと同様な方法で行われる。)

*アップルブランドのソリューション*：ユーザがNewtonプラットフォーム上で、インターネット・ソリューションを実現するのに必要な要素を入手しやすくするため、われわれは、Apple Internet Connection Kitのような、アップルブランドの製品をNewton向けに開発することを考慮中である。この記事で触れたその他の部分も一緒に必要となるので、アップル社内での開発とデベロッパによる開発の両方を考慮中である。

## まとめ

Newtonプラットフォームにとって、高度なコミュニケーション機能は非常に重要である。Newton Systems Groupのメンバーは、Newtonプラットフォームのコミュニケーション戦略に大いなる熱意を持っている。組み込み機能（例えば、シリアル接続とファクス機能）とイネーブラのような開発ツールの組み合わせにより、Newtonデベロッパが開発する製品に新たなコミュニケーション機能を付け加える作業が推進され、PDAプラットフォームとしてのNewtonの先進性を維持するであろう。

NTJ


AppleのNewton デベロッパプログラムに関するお問い合わせ、またはお申し込みは下記へご連絡ください。
アップルコンピュータ株式会社
Newton Developer Support事務局
電　話：03-5334-2480
ＦＡＸ：03-5334-2781
email：jnewtondev@asia.apple.com

# NewtAppフレームワークの技法

*by Greg Christie, Apple Computer, Inc.*

NewtAppフレームワークは、システムが用意するプロトの集まりであり、このプロトを組み合わせることにより、完全なアプリケーションを形成することができる。かつては、Newton PDA用アプリケーションを作成するには、完全なアプリケーションを形成するために、多数のプロトをアセンブルしなければならなかった。NewtAppの特色は、NewtAppフレームワークに組み込まれた関数の多さである。基本的なシステムサービス―ファイリング、検索、ルーティング、オーバビュー管理、ステーショナリなど―に対するサポートが、NewtAppプロトに用意されている。NewtAppを用いてアプリケーションを作成するプロセスは、主としてアプリケーションのデータとその閲覧方法を記述する作業から成る。その後は、データのまわりにアプリケーションのシェルをアセンブルするという、どちらかというと単純な作業である。このシェルの基礎としてDTSサンプルコードを利用してもよいし、あるいは、独自のシェルを作成してもよい。いずれにせよ、まずNewtAppを作成してしまえば、他のアプリケーション用に他のデータタイプを追加して、そのNewtAppを再使用するのはたやすい。将来のDTSサンプルは、デベロッパによるアプリケーション開発を促進するために、組込み型の再使用可能なシェルを含むようになるかも知れない。

すべてのタイプのアプリケーションが、NewtApp処理に適しているわけではない。 NewtAppが適しているのは、通常、一度に1つのスープエントリを扱い、すべてのエントリが同一のスープに属すようなアプリケーションである。ステーショナリ機能も利用すれば、1つのエントリに対して、多数の異なるビューを提供できる。ステーショナリを利用すれば、1種類のデータに対して複数のビューを提供できるばかりでなく、異なるタイプのデータを使用できるようにアプリケーションを拡張できる。例えば、1つのアプリケーションであるNotesは、罫線付き、チェックリスト、およびアウトラインを扱える。同じスープ内に、異なるステーショナリが作成したアイテムを入れることができるし、NewtAppは、それらのアイテムをすべて表示できるのである。

ステーショナリを使っても使わなくても、NewtAppを用いてアプリケーションを作成することはできる。しかし、Newton OS 2.0におけるルーティングはステーショナリにより処理されるし、また、しばしば、個々のデータアイテムに対して複数のビューを提供した方が望ましいため、ステーショナリはきわめて有用である。ただし、選択するのはデベロッパ自身である。アイテムの印刷やファクス、複数のビューの提供、アプリケーションの拡張などの必要がなくても、NewtAppを使用してかまわない。DTSの“Checkbook”サンプルには、ステーショナリを使用しないNewtAppフレームワークの使用法が紹介されている。さらに付け加えれば、ステーショナリは、NewtAppを使用しないアプリケーションでも使用可能である。この記事はNewtAppの使用法のみを扱うが、ステーショナリを使う場合でも、使わない場合でも、NewtAppのガイドとして役立つはずである。

現在のNewtApp設計には、2つの主な制約がある。第1に、NewtAppはスープに基づくデータに対してのみ使用可能である。第2に、NewtAppは一度に1つのスープからのエントリしか扱えない。したがって、計算機あるいは汎用ユーティリティのようなアプリケーションには適していない。さらに、組込みCalendarのように、同時にいくつかのスープからのデータを扱わねばならないアプリケーションも、NewtAppには向かないだろう。しかし、開発中のアプリケーションが普通のNewtonアプリケーションに似ているか、あるいはアプリケーションのためのアイデアをすぐにプロトタイプ化したいのであれば、NewtAppは適切なアプローチである。

## なぜNewtAppを用いるのか？

NewtAppは当初、Newton Systems GroupがROMにアプリケーションを作成する方法として、またサードパーティ製アプリケーション開発でサポートされるフレームワークとして設計された。Names、Notes、Calls、およびIn&OutBoxアプリケーションは、すべてNewtAppを用いて作成された。Newton 2.0用の組込みアプリケーションが進化するにつれ、NewtAppフレームワークも進化した。フレームワークが使用可能になると、アプリケーションを書くのに必要なプロシージャ体系が、文書化された。このアプリケーション・フレームワークを使用するためのプロシージャは、Newton Programmer's Guideに詳細にわたって文書化されており、またDTSサンプルコードにも実例が紹介されている。

現在、NewtAppプロトをさまざまに組み合わせることにより、アプリケーションとして機能させることは可能だが、別の構成技術や配列方法がサポートされていないことに注意していただきたい。しかし、NewtAppフレームワークが進化するにつれ、将来的には新たな方法が出現する可能性がある。

NewtAppには、数多くの設計目標があった。そのうちのいくつかを紹介しよう。

- Newton共通のアプリケーション展開を容易にし、そのスピードを増す。
- Newtonアプリケーションにおける共通コードを排除する。
- システムサービスを“コストなしで”提供する。
- Newtonアプリケーション間の画面表示の標準化を促進する。
- ステーショナリのための簡便なコンテナを提供する。
- サードパーティ製アプリケーションをシステムと共に“アップグレード”可能とする。

スープ関連の制約を除けば、NewtAppフレームワークはこれらの目標を達成しており、Newtonプログラミングが初めての人は、ここからスタートするのがよいだろう。また、NewtAppを使用すれば、1.xアプリケーションから2.0への変換も容易である。変換に必要な手順については、この記事の最後で解説する。

## NewtAppの使用

NewtAppによるアプリケーションの作成は、従来のやり方とやや異なっているかもしれない。NewtAppフレームワークはいくつかのレイヤをもつため、最初は、どれがどのレイヤに属すのか、判断するのにまごつくかもしれない。本節では、まずNewtAppアプリケーションを作成するための一般的技法を解説し、次に各レイヤを説明する。ステーショナリは、NewtAppを使用する上で重要であり、またアプリケーションを拡張する強力な機能でもあるため、この記事では、ステーショナリ固有の機能についても触れる。NewtAppにおけるステーショナリの使用については、Newton Programmer's Guideの第5章に詳しく解説してある。また、ステーショナリに基づき完成されたNewtAppのデモンストレーションについては、DTSサンプルコードの“Who-Owes-Whom”を参照していただきたい。

## 始める前に

NewtAppは、Newtonアプリケーションの3つの主要機能をサポートしている。すなわち、card、pageおよびrollの3つである。cardアプリケーションは、Namesと同様、一度に1つのエントリを表示し、エントリの高さはすべて同じである。pageアプリケーションは、一度に1つのエントリのみを表示するが、エントリの高さは任意である。このアプリケーションの特徴を持つ１つの例が、Callsアプリケーションである。roll機能は、Notesアプリケーションと同様、任意の数のエントリを任意の

高さで表示できる。NewtAppは、これら3つの特徴的機能をサポートすることにより、アプリケーションに対し異なるスタイルを実験できる、というフレキシビリティをもつ。

始める前にもう１つやっておくべきことは、データ構造と、そのデータをどのように視覚的に表現したいのかを、おおまかに把握しておくことである。ステーショナリを使用する場合、慎重にプランをたて、データ特定コードをステーショナリ内にカプセル化するよう努めること。アプリケーションがどのようなデータを表示するのかについて、推測は避けること。ステーショナリを設計するときは、そのステーショナリがコンテナ・アプリケーションの外部で使用される可能性もあることを念頭に置いておく。アプリケーションとステーショナリの間の参照をハードコード化しないこと。その代わり、フレームワークに用意されている、スロットならびにレイヤ間の参照への継承を使用すること。

NewtAppをビルドする前に、さらにもう１つ覚えておきたいことは、作業の大半は、プロトのアセンブリングとスロットの値の設定から成るということである。フレームワークはその残りをやるわけである。これにより、使いやすく視覚表現されたソリッドデータ設計を作成する手間が省ける。この機能を利用するためには、自身のコードを書くときに、システムが用意したプロト内にある、継承されたメソッドを呼び出すこと。

最後にもう１つ、インストールあるいは削除スクリプト内には、必須のステートメントを入れること。以上述べた注意事項を守れば、アプリケーションとそのステーショナリは確実に登録され、すべての必要なシステムサービスが利用できるはずである。

## レイヤ

NewtAppフレームワークはレイヤの集まりである。レイヤは上から順にアプリケーション、レイアウト、エントリ、スロットとなっている。ステーショナリを使用する場合は、エントリとスロットの間にレイヤが追加される。１番上に位置するアプリケーション・レイヤは、さまざまな部分が連結する場所である。レイアウト・レイヤは、データのオーバビューとデフォルトビューを提供する。エントリ・レイヤは、個々のスープエントリが表示される場所である。スロットビューは、ユーザがスープエントリ内の個々のスロットを閲覧、編集するためのものである。ステーショナリを使用する場合は、ステーショナリ・レイヤがデータの`viewDef`を含むため、これらの`viewDef`にスロットビューが含まれることになる。

## アプリケーション・レイヤ

NewtApplicationプロトは、NewtAppのベースビューの役割を果たす。もし、開発中のアプリケーションにファイリングを加えたければ、フォルダタブ･プロト、すなわち`newtFolderTab`または`newtClockFolderTab`を加えればよい。このベースビュー内に、アプリケーションのステータスバーを置くこともできる。`newtStatusBar`プロトは、ステータスバー用のボタンを入れる2つのスロット、すなわち`menuLeftButtons`と`menuRightButtons`を含んでいる。このボタンは、実行時に自動的にレイアウトされる。オーバビューとデフォルトビューで異なるボタンを表示したい場合も、NewtAppはこれをサポートしている。ベースビュー内に`statusBarSlot`というスロットを作成し、そのスロット内に`newtStatusBar`の宣言された名前を入れればよい。各レイアウト内には（以下のレイアウトの節を参照のこと）、`menuLeftButtons`および`menuRightButtons`スロットを置くこと。アプリケーションがレイアウトを切り替えると、適切なボタンがステータスバー上に表示されるはずである。

NewtApp内の多くのボタンは、システムが提供している。ステーショナリに関しては、`newtNewStationeryButton`であるNewボタンと、`newtShowStationeryButton`であるShowボタンがある。どのNewtAppアプリケーションにおいても、ルーティングとファイリングは、`newtActionButton`と`newtFilingButton`が提供する。すべての必須動作がこれらのボタンに組み込まれている。したがって、デベロッパは、これらのボタンを`menuLeftButtons`および`menuRightButtons`スロット内の適切な場所へ入れるだけでよい。

`newtApplication`プロト内には、スクロール、検索、およびファイリングに対するアプリケーションの動作を決定する、いくつかのスロットを設定しなければならない。さらに、アプリケーション・レイヤもまた、アプリケーションを機能させる少数のキースロットを含んでいる。これらは、`allSoups`、`allLayouts`、`allDataDefs`および`allViewDefs`である。これらのスロットを、アプリケーション固有のデータのプレゼンテーションのために、NewtAppをカスタマイズするのに用いる。

### allSoupsスロット

アプリケーション・レイヤ内の`allSoups`スロットは、アプリケーションスープを定義する典型的なフレームである。`allSoups`の各サブフレームは、システムの`newtSoup`プロト化されたものであり、アプリケーション内で扱いたいスープをここで指定する。単純な`allSoups`スロットは次のようになる。

```
{ appSoup:{_proto: newtSoup,
      ...}}
```

アプリケーションがオープンしているとき、これらのサブフレームは、アプリケーションの実スープを含む。`newtSoup`の必須スロットは、Newton Programmer's Guideに列挙されているので、ここではもっと高度な問題を少し取り上げる。

アプリケーションがいくつかの異なるスープを扱うようにしたい場合、それらのスープはここで定義する。実行時において（後述のレイアウトレベルの節を参照のこと）、閲覧するスープを決定できる。これと同じテクニックを使うことにより、アプリケーションのスープに対し、異なるquery指定あるいはソート順序を提供することができる。そのためには、異なるものをそれぞれ、`allSoups`の各専用サブフレームに入れるだけでよい。また、スープ上で機能するメソッドはすべて、これらのフレーム内に置かなければならない。もし、アプリケーションが、Pickerの値あるいは他の関連情報として、他のスープを必要とするならば、ここで各スープを指定する。アプリケーションがオープンしているときは必ず、これらの二次スープの作成、登録および維持が行われる。

最後にもう１つ覚えておきたいことは、実行時においてこれらのフレームはスープであるため、エントリの追加または削除が可能であり、また、これらのフレームから、任意のスープ関連タスクを直接実行できることである。また、アプリケーション・ベースビューは、`allSoups`フレーム内で定義したメソッドにより継承できないことを忘れてはならない。これらのスロットは、ベースビュー内に存在しているが、`newtSoup`からプロト化されるのであって、ベースビューの子ではない。`newtSoup`内のメソッドからベースビューを参照する必要があれば、`GetRoot().(kAppSymbol)`を使用して、アプリケーション・ベースビューへ戻ればよい。

### allLayoutsスロット

`allLayouts`フレーム内には、アプリケーション用の2つのレイアウトがセットアップされている。これらのレイアウトは、実行時にビルドされて、ベースビューの子ビューとしてインスタンス化される。これが、Newton 1.x用アプリケーションのビルドとやや異なる点である。以前は、いくつかのアプリケーションにおいて、データのベーシックビューとオーバビューは、アプリケーション・ベースビューの異なる2つの子ビュー(Child)だった。そのため、必要に応じて、片方を表示したり隠したりしなければならなかった。NewtAppは、オーバビュー管理をすべてサポートしているため、ベースビューに対してこれらのレイアウトを宣言する必要はない。2つのスロット、すなわちデフォルトおよびオーバビュー・スロットをもつフレームを作成し、`GetLayout`(filename)を使用して、各ビューに使用するNTKレイアウトファイルを指定するだけでよい。その他はすべてが自動化されている。

### allDataDefsスロット

アプリケーションがステーショナリを使用する場合、このスロット内に、アプリケーション用の`dataDef`、およびシステムに登録するその他の`dataDef`を指定する。これらの`dataDef`は、パッケージのインストール時にインストールと登録が行われ、パッケージの削除時に登

録取消しと削除が行われる。これの実行は、パッケージのInstallScript内のNewtInstallScriptへの必須呼出しにより行う。allLayoutsの場合と同様、これも典型的なフレームである。各dataDefのスロット名は、そのdataDef内のシンボルスロットと一致しなければならない。dataDefの必須スロットについては、Newton Programmer's Guideを参照のこと。allDataDefsフレームは、フレームセットを明確にリストしたり、GetLayout(filename)関数を用いてNTKレイアウトを参照したりできる。

### allViewDefsスロット

allViewDefsスロットは、viewDefがアプリケーションのdataDefへリンクされる場所である。このスロットは、プリントフォーマット、またはその他の使用予定のルーティングフォーマットを含む。このフレームの構造は、allDataDefsスロットと同じものである。allDataDefs内の各dataDefスロットに対して、そのdataDefのviewDefを含むフレームが1つ必要である。viewDefの1つをデフォルトとして指定するステーショナリ要件は別として、1つのdataDefに対して、いくつでもviewDefを関連づけることができる。例えば、allDataDefsスロットが次のdataDefを含むとする。

```
{ dataDef1: GetLayout("dataDef1"),
  dataDef2: GetLayout("dataDef2")},
```

これに対応するallViewDefsスロットは次のようになる。

```
{ dataDef1:{default:GetLayout("dd1DefaultViewDef"),
      notes: GetLayout("notesViewDef"),
      frameFormat: protoFrameFormat},
  dataDef2:{default: GetLayout("dd2DefaultViewDef"),
      frameFormat: protoFrameFormat}}
```

赤外線ビームとメールについては、システムが用意した、protoFrameFormatを使用すれば、コストをかけずルーティングができる。また、各dataDefに対して同じ個数のviewDefを指定する必要はない。アプリケーションがあるアイテムを1つのviewDefで表示中であり、別のアイテムはそのviewDefを使用できない場合でも、NewtAppフレームワークは、必要に応じてデフォルトviewDefへ切り替える。また、newtShowStationeryButtonは、現在表示中のアイテムに対して使用可能なviewDefのみをリストする。

## レイアウトレベル

われわれがNewtApp内のレイアウトと言うとき、NTKにおけるレイアウトとは異なる意味あいでこの語を使っている。確かに、NewtApp内のレイアウトはNTKレイアウトファイルだが、NewtAppのその他のレベルもNTKレイアウトファイルなのである。NewtAppにおけるレイアウト・レイヤは、アプリケーションの外観全体を制御する。したがって、ここで言うレイアウトとは、アプリケーションの視覚的レイアウトと考える方がわかりやすい。NewtAppは、1つのアプリケーション内で2つのレイアウトをサポートする。この2つは、アプリケーション・レイヤのallLayoutsスロット内に列挙されているレイアウトで、デフォルトとオーバビューと呼ばれている。デフォルト・レイアウトは、アプリケーションスープ内の個々のエントリを表示するのに用いる。オーバビュー・レイアウトは、おなじみの、スープのNewtonオーバビューを提供する。現在、デフォルト・レイアウトには3つのプロトがあり、オーバビュー・レイアウトには2つのプロトがある。どのプロトを使用するかは、ビルドするアプリケーションの機能（card、pageまたはroll）により決まる。

レイアウト・プロトの中で最も重要なスロットは、masterSoupSlotである。このスロットは、アプリケーション・ベースビュー内にあるallSoupsフレームのスロットのうちの1つである。allSoupsの節で示した例においては、レイアウトは 'appSoupのmasterSoupSlotをもつ。さらに、アプリケーション・ベースレイヤのstatusBarSlot機能を使う場合は、このレベルにおいてmenuLeftButtonsとmenuRightButtonsを作成し、ステータスバー・ボタンをこれらで置き換えることができる。他のキースロットおよびメソッドについては、Newton Programmer's Guideに概略が説明されている。また、DTSサンプルコードにも実例が示されている。覚えておくと便利なメソッドは、DoRetarget()である。このメソッドは、スープデータのレイアウトを更新する。したがって、特定のエントリへ進む必要がある場合は、レイアウトのmasterSoupSlotにより参照されたスープエントリへ行き、DoRetarget()を呼び出して、ビューを更新すればよい。

### デフォルト・レイアウト

アプリケーション内のデフォルト・レイアウトについては、newtLayout、newtPageLayoutまたはnewtRollLayoutを使用する。cardアプリケーションの場合は、newtLayoutを使う。pageおよびrollアプリケーションの場合は、それぞれnewtPageLayout、newtRollLayoutを使わなければならない。このレイアウトレベルは、ユーザがスープエントリを閲覧したり、編集したりする場所である。したがって、デフォルト・レイアウトには、スープエントリを含むビューが追加される。newtLayoutプロトについては、NTKを使用して、このテンプレートの子ビューとしてnewtEntryView（後述のエントリ・レイヤの節を参照のこと）を描画する。newtPageLayoutまたはnewtRollLayoutについては、レイアウトレベルprotoChildスロット内のエントリビューに対する参照である、GetLayout(filename)-styleを用いると、必要に応じてエントリビューをシステムが作成する。エントリの背景として凝ったグラフィックスを使いたい場合、あるいはスープワイドで機能する制御を使用したい場合、それらを設定する場所としては、レイアウト・レイヤが適している。

### オーバビュー・レイアウト

オーバビューとして使用できるレイアウトには2種類ある。cardアプリケーション用のnewtOverLayoutと、pageあるいはrollアプリケーション用のnewtRollOverLayoutである。これらのオーバビュー・プロトは、コスト効率の良い機能性を提供する。これらは、アプリケーションにNewton OS 2.0の標準オーバビューを提供するものであり、スクローリング、チェックボックス、dataDefに基づくアイコン、および各エントリの1、2行の概要を完備している。さらに、アイテムをタップすると、デフォルト・レイアウト内の対応するエントリが表示される。チェックボックスを抑制したり、概要を変えたり、あるいはアイテムをタップしたときの動作を変更したりできる。newtOverLayoutおよびnewtRollOverLayoutプロトについては、Newton Programmer's Guideを参照していただきたい。ただし、標準的なオーバビューの動作を得るには、masterSoupSlotを設定するだけでよい。

## エントリ・レイヤ

エントリ・レイヤは、アプリケーション内で実スープエントリが表現され、表示される場所である。ここでは、個々のスープエントリを扱うことができる。また、アプリケーションにおける、全スープエントリ内のスロットに対応する制御を表示できる。このレベル又はそれ以下において、異なるアプリケーション用にNewtAppをカスタマイズするために必要な、プログラミングの大部分を行う。エントリビュー・プロトには3種類ある。cardアプリケーションにはnewtEntryViewを使い、pageまたはrollアプリケーションにはnewtRollEntryViewを使用する。3つめのタイプは、newtFalseEntryViewで、これは、非NewtAppベースのアプリケーション内のステーショナリまたはスロットビューに使用する。newtFalseEntryViewの使用は重要なことがらであるが、ここではこれ以上触れないことにする。Newton Programmer's Guideに詳しく解説されているので、参照していただきたい。

entryViewはスープエントリと1対1対応であるため、変更されたスープデータからのビューの更新、およびスープエントリの更新を行う。すべてのエントリビューは、カレント・スープエントリを含む、ターゲットというスロットをもつ。データを書き込んだり、エントリからデータを得るときは、常にこのターゲットを使用できる。レイアウトレベルの場合と同様、エントリビューもまたDoRetarget()メソッドをもつ。ターゲットを変更した場合、このメソッドを使えば、エントリビューを更新できる。

エントリレイヤでは、ターゲットからのビューの更新の他に、ビューの変更をターゲット・スープエントリへ書き戻す処理も行う。実際には、変更そのものは、個々のスロットビューにより行われるが（以下を参照のこと）、変更されたエントリは、即座にストアへ書き戻されるわ

けではない。その代わり、エントリレベルメソッドのStartFlush()が呼び出され、それがアイドルタイマをスタートさせる。数秒後、あるいはエントリのスクロール時、レイアウトの変更時、アプリケーションのクローズ時に、変更されたエントリがストアへ書き戻される。スロットビューの外で、エントリを直接変更する場合は、必ずStartFlush( )を呼び出すこと。

ターゲットの他にも、実行時にカレント・コンテキストを扱うための、エントリレベルのスロットがある。例えば、currentDataDefとcurrentViewDefである。これらのスロットは、ターゲットと共に使われるカレント・ステーショナリ構成要素を含む。これらスロット内のviewDefとdataDefは、パッケージ内に用意されているテンプレートである。実行時において、カレントのviewDefは“ライブビュー”へとインスタンス化される。viewDef内の実行時ビューを得るためには、インスタンス化されたviewDefを含む、currentStatViewというスロットがある。

組込みNotesアプリケーションのように、エントリビュー内にヘッダをつけたい場合もあろう。このプロトには2つのバージョンがある。ヘッダ上にアクションボタンとファイリングボタンを加えるには、newtEntryRollHeaderを用いる。単純なタイトルとアイコンのヘッダにするには、newtEntryRollHeaderを用いる。pageおよびrollスタイルのアプリケーションについては、これらのヘッダ・プロト内のサイズ変更可能スロットを用いて、ドラッグによるエントリのサイズ変更を行うことにより、ヘッダを制御できる。エントリビューには、このレベルに追加するヘッダとエントリレベルの制御全般の他に、個々のスロットビューとステーショナリ・コンテナを追加することができる。

## ステーショナリレベル

ステーショナリは、viewDefとdataDefを一括して指すときの用語だが、NewtAppのレイヤにおいては、viewDefのみを指す。NewtAppにおけるviewDefの使い方は非常に簡単である。エントリレベル・プロト内にnewtStationeryViewを入れるだけでよい。entryView内にviewDefを直接レイアウトしなくてよい。システムが、現在選ばれているviewDefを、カレント・スープエントリまたはターゲットに対応するdataDefとマッチさせる。viewDefそのものの設計は、実際にそれを使用するより、ずっと手間がかかる。entryViewが、スープエントリ全般に関係する制御により、1つのスープエントリに対応しているならば、viewDefはサブエントリビューのようになり、そのスープエントリのdataDefクラスに関連するデータを表示する。

データの複数ビュー（例えばinfoビューとnotesビュー）を提供したいならば、1つのエントリの異なるスロットであるとわかるような、2つのviewDefを作成するだけでよい。アプリケーションにnewtShowStationeryButtonを入れてしまえば、ユーザは簡単に異なるビュー間の切り替えができる。このようなアプリケーションにおいては、付属パッケージから追加したviewDefはすべて、Show Picker内に表示される。任意のviewDef内のエントリにある、任意のスロットの内容を表示させることは可能だが、viewDefは、それに対応するdataDefにより作成されたスロットのみを表示するように設計したほうが良い。スープエントリの個々のスロットを表示したり、編集したりできるように、viewDefにはスロットビューが含まれている。viewDefに関する詳細については、 Newton Programmer's Guideを参照のこと。

## スロットレベル

NewtAppフレームワークには、スープエントリの単一スロットの表示と編集に用いるプロトが数多くある。これらは、スロットビューとして知られており、多くの類似点をもつ。スロットビューには3つの基本的なクラスがある。すなわち、普通、ラベル付き、および雑スロットビューである。普通スロットビューは、newtTextViewまたはnewtRONumberViewのように、スープエントリ・スロットの値を単に表示したり、更新したりする。ラベル付きスロットビューの例としては、newtLabelInputLine、newtSmartNameViewおよびnewtLabelPhoneInputLineがある。これらのスロットは、protoLabelInputLineに関連している。ラベル付きスロットビューは、普通スロットビューの機能をもつ他、ラベルを含み、可能な値のポップアップリストを表示できる。その他のスロットビューには、newtEditViewとnewtCheckBoxがある。

スロットビューは、数多くのタイプの共通データに対して適用される。適切なスロットビューを使用すれば、テキスト、整数、実数、記号、およびnotes全体を表示させ、編集することができる。スロットビュー・プロトの名前は、たいていは長くて扱いにくいが、それを使うべき場所のヒントを示している。例えば、newtLabelPhoneInputLineは、長くてタイプしづらいが、いつそれを使うかがわかりやすい。名前の中には、データのタイプのみならず、その他の情報も示されている。プロト名が“RO”の文字を含めば（例えばnewtRONumberView）、そのプロトが読み取り専用であり、データ表示にのみ使用できることを示す。

スロットビューはフォーマットも機能もさまざまだが、すべて共通のインタフェースをもつ。スロットビューはすべてパススロットを使用する。パススロットは、そのスロットビューに対応する、スープエントリ内のスロットへのパス参照値を含む。このパス参照値は、'myDataSlotのような単純なシンボル、または[pathExpr:'myClassFrame,'myDataSlot]のようなパス式である。スロットビューはすべて、ReTargetメソッドを実行する。このメソッドが実行されると、指定したパス内の値でそのスロットビューの値を強制的に更新する。スロットビューは変化すると、エントリレベル内のフラッシュタイマにトリガをかける。スロットビューの大半は、jamSlotにより制御されるJamFromEntryメソッドを実行できる。

## ジャミング

ジャミングとは、スロットビューが、他のスープのエントリからのデータを、自分のエントリへ組み込むために用いるテクニックである。ある種のラベル付きスロットビューは、別のスープに対し、ポップアップの内容を参照したり、ユーザに選択させるスープエントリのPickerを表示することができる。中でも最も便利なのが、newtSmartNameViewである。アプリケーション内で、組込みnamesスープからのデータを使用するためには、単にnewtSmartNameViewを用いるだけでよい。ユーザがこのスロットビューをタップすると、標準NamesリストPickerが表示される。このPickerからユーザが1つの名前を選択すると、アプリケーション内のカレント・エントリビューが、選択したnamesエントリとジャミングされる。そして、各スロットビュー内のJamFromEntry(otherEntry)メソッドが呼び出される。JamFromEntryは、そのビューのjamSlotが設定されているかをチェックする。jamSlotがnilでなければ、それがフォーリン・エントリ、つまり選択された外部エントリ、内のパス式として使われる。そして、スロットビューのパスの内容は、フォーリン・エントリのjamSlotの内容で置き換えられる。例えば、あるカスタマの名前と肩書きを自分のアプリケーションのエントリ内に入れたければ、'customerNameのパスでnewtSmartNameViewを使用すればよい。もう１つのスロットビューにおいては（それはnewtTextViewまたはnewtLabelInputLineなのだが、パスは 'customerTitleに、jamSlotを 'titleに設定する。なぜならnamesエントリのtitleスロットにそのカスタマの肩書きが入っているからである。newtSmartNameViewから１つの名前を選択すると、自分のスープエントリには、選択された名前の入ったcustomerNameスロットと、肩書きの入ったcustomerTitleスロットが、組込まれる。

ラベル・スタイルのスロットビューはデフォルトでジャミングする。自分のエントリのデータがフォーリン・エントリから上書きされたくなければ、自分のスロットビューのJamFromEntryメソッドを上書きし、適切な継承メソッドを呼べば良い。フォーリン・データをストアあるいは表示する前に処理する必要があれば、このメソッドを完全に上書きすることができる。その場合、フォーリン・エントリから必要なデータを処理し、新しい値をターゲットのパスに入れる。良く使われるステートメントとしてtarget.(path):=newDataがある。新しいデータの表示 (retargetting) は、システムが行う。

## 自分のスロットビューのローリング

システムがスロットビューを提供しないデータの表示を行いたい場合がある。

例えば、ROMはnewtPictureViewsあるいはnewtLabelPickersを持っていない。NewtAppの範囲を拡大するには、自分のスロットビューを書く必要がまれにおこる。この作業は単純であり、使用するメソッ

ドとスロットも最低ですむ。まず、スープデータを指定するにはパスのメカニズムを用いる。つぎに、ビューのデータはviewSetupFormScriptに設定する。新しいスープデータ、あるいは変更されたスープデータからのビューデータの更新は、ReTargetメソッドを実行して行う。ユーザがビューのデータを変更したなら、スープエントリを更新し、フラッシュ・タイマーをスタートさせる。必要なら、jamSlotとJamFromEntryを実行する。この技法の例は、DTSサンプルコードの”Who-Owes-Whom”に示されているので、参照のこと。

## 自分のAppからNewtAppへ

既存の1.xアプリケーションを移植するのは、NewtAppの設計上の制約条件内である限り、比較的簡単である。このためには、まず、スープ関連のコードを分離し、allSoupsスロットに入れる。つぎに、エントリ関連のコードを（現在どのビューにスープエントリを入れてあるかにかかわらず）分離し、個々のビューをスロットビュープロトで置き換える。最後に、これらの新しいエントリレベルのコントroll、スロットならびにメソッドを、エントリビューに入れる。移植前のアプリケーションにおける、レイアウトやスロットに対するその他特定の参照は、NewtAppでの参照に変更すること。野心的なアプローチを取りたいと思うなら、データ構造とエントリレベルのビューをステーショナリに持ち込むために、dataDefと１つまたは１つ以上のviewDefをそのdataDefのために使うことを考えてみるのも良い。そうすれば、既存のルートフォーマットをNewton OS 2.0のステーショナリに基づくルーティングに変換する時の手順を単純化でき、後で手間が省ける。何人かのデベロッパが、1.xアプリケーションをNewtAppに移植した経験を持っているが、彼らの経験によると、いったんエントリレベルのコンポーネントを分離してしまえば、その後のプロセスは簡単であり、時間もわずかですんだ、とのことである。

## 結　論

以上、NewtAppフレームワークの使用について概要を述べた。NewtAppは、十分にカプセル化され、完全な機能性を持つNewtonアプリケーションを作成する強力なツールである。同じ外側のレイヤを様々なエントリレベルとステーショナリのプロトに使用できるので、一度NewtAppを作ってしまえば、２回目以降は、新しいアプリケーションを作成する度に、１回目のコストを償却することができるのである。アプリケーションシェルの外側のレイヤをコピーし、allSoups, allDataDefsとallViewDefsスロットをカスタマイズしさえすれば、新しいアプリケーションのステーショナリを作成することができる。１つまたは１つ以上のレイアウトを移植し、プロトスロットの名前をいくつか付け替えると、以前のcardスタイルのアプリケーションがrollあるいはpageスタイルのアプリケーションとしてどのように見えるか、またその逆もわかる。NewtAppを使えば、開発時間が節約され、多くのことがNewtAppのプロトによって実行されるので、パッケージのサイズもずっと小さくできる。1.xアプリケーションのあるものは、NewtApp 2.0バージョンでは、サイズが半分になる。NewtAppフレームワークに新機能が追加されると、デベロッパのアプリケーションも自動的に新機能が追加される。

NTJ

# ビューのスクロール性能を最大化するキャッシュの使用法

*by Jeffrey C. Schlimmer, Washington State University*

より速い処理速度を実現するコードを書くテクニックは、常にNewtonScriptのデベロッパの関心事であろう。処理のスピードが増せば、新しい機能をもったアプリケーションが誕生する道が開ける。つまり、以前は信じがたいほど遅かった動作が、市場に出せるレベルまで速くなるからである。一部のスピードに関するテクニックは、実行環境の詳細事項を利用する。例えば、NewtonScriptインタープリタでは、ループの使用より、むしろforeachコマンドを使用して配列の要素を反復したほうが、スピードが速い [McKeehan & Rhodes, 1995]。これらの詳細事項に細心の注意を払えば、パフォーマンスが著しく向上する可能性がある。スピードに関するその他のテクニックは、いかなる実行環境に対しても適用可能である。例えば、変化しないステートメントをループの外へ移動すれば、不要な反復を避けられるためスピードが速くなる。この記事では、ビューのスクロールのスピードを４倍程度向上させるこのようなテクニックの組み合わせを検討する。

## スクローリング

Newtonアプリケーションに共通なタスクの１つとして、ユーザがアイテムをチェックしたり選択したりできるように、アイテムのスクロール可能なリストを表示する機能がある。図1は、この記事における考察の具体例として使用したサンプルアプリケーションを示す。Newton MessagePad 120に搭載されているこのアプリケーションは、100アイテムのうち14アイテムを表示する。ユーザが上向き矢印または下向き矢印をタップすると、アイテムが1つずつスクロールされる。（現行のNewtonユーザインタフェース・ガイドラインは、1画面分のアイテムから1アイテム引いた個数によるスクローリングを推奨している。このポリシーは、スクロールスピードの上限を推測して決められたものかも知れない。）各アイテムは、チェックボックス、ボタンの形をした1つのアイコン、2種類のテキストエレメント、および目盛を含んでいる。仮に、ユーザがあるアイテムをタップすると、アプリケーションが応答するものとする。おそらく、チェックボックスを切り換えたり、アイテムをより詳細に表示するレイアウトに切り換えることによって、応答することになろう。サンプルアプリケーションでは、アイテムのデータが1つの配列内にあるものとする。データをスープエントリに格納しているアプリケーションの場合は、やや異なるテクニックが必要となる。スープを扱うこの種の問題については、Newton DTSサンプルコードの“ True Grid ”に具体例が示されている。

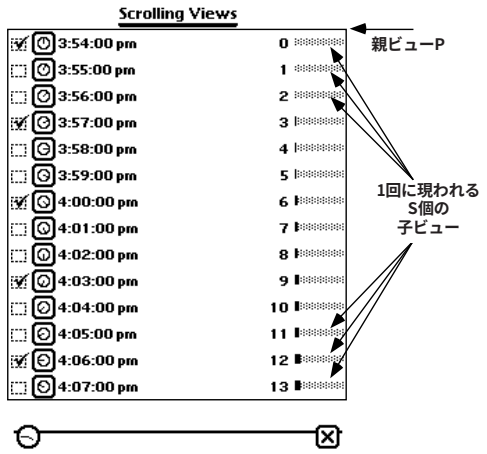


*図1.　スクロールするビューをもつサンプルアプリケーション*

## 集合体プロトとsetOriginを使う方法

アイテムのリストをスクロールするための、単純で強力な方法は、1つのアイテムを表わす1つのプロトを構築することである。1つのプロトは5つの子供、すなわち、1つのチェックボックス、1つのピクチャビュー、2つのスタティックテキスト、および1つの目盛を含む。これらの子は、viewSetupFormScriptを用いて自身を初期化する。このスクリプトにおいて、これらの子供は、プロトのアイテムからスロット値を継承し、それに従って自らの表示をセットアップする。この方法では、表示すべきアイテム（N個のアイテム、例えば100のアイテムが存在すると仮定しよう）1つにつき1つの子をもつ、1つの親ビュー（以下これをPと呼ぶ）を構築しなければならない。Pは、自身のクリッピングフラグ・セットをもち、1画面分のアイテム（S個のアイテム、例えば14のアイテムと仮定しよう）を表示できるサイズをもつ。ベースアプリケーション・ビューは、viewScrollUpScriptおよびviewScrollDownScriptメッセージに対し、PへsetOriginを送ることにより応答する。これにより、Pの垂直オフセットは、1つの子ビューの高さ分だけ変更される。Newtonのビューシステムは、ユーザがスクロールした場所に応じて、それぞれ異なるアイテムのサブセットを自動的に表示する。

この方法のスピードをテストするため、アプリケーションに対してNewton Toolkit Inspectorウィンドウから一連のコマンドを送った。すなわちopen、viewScrollDownScriptを10回、viewScrollUpScriptを10回、そしてcloseである。Newtonのビューシステムを強制的に更新させるため、各メッセージの後ろにRefreshViews呼出しを付随させた。これら一連のコマンドは強調表示され、全てを同時に評価した。方法としては、実行プロファイルを作成し、かかった時間を100分の1秒単位で測定し、四捨五入して0.1秒の精度で報告している。

この方法では、すばやいスクロール表示が可能である。図1と同様のデータを使用したテストでは、10のアイテムをスクロールダウンしてから元の位置までスクロールアップするのに、9.8秒しかかからなかった。各アイテムに対応するN個のビューが、Pのオープン時に構築されるため、ビューシステムは、Pのクリッピング領域の内部あるいは外部へ、子ビューをすばやく移動できる。また、この方法では、正確な実行が簡単にできる。Pは、表示すべき1つのアイテムに対し1つの子供を割り当てることにより、そのviewSetupChildrenScriptの中で、N 個の子ビューを1回でセットアップできる。各子プロトは、viewSetupFormScriptにより自身を初期化するだけでよい。このパフォーマンスの代価は、Pのオープン時にN個の子ビューを作成することに関わるオーバヘッドである。この作成に要する時間とヒープスペースは、オープンに要する3.6秒と32Kのヒープ（GCおよびStatにより測定し、最も近いKの単位に四捨五入した場合）である。これらの値から判断する限り、この方法は、本記事で検討する方法の中で最悪である。

## 集合体プロトとsyncChildrenを使う方法

別な方法として、S個（つまり、1回で表示できる個数）の子ビューのみをもつ親ビューPをビルドする方法がある。前述の方法と同様、Pは、1画面分のアイテムを表示できるサイズをもつが、クリップする必要がない。ベースビューは、viewScrollUpScriptとviewScrollDownScriptメッセージをPへ送る。すると、Pが自身へsyncChildrenメッセージを送る。これに対する応答として、NewtonのビューシステムがPへviewSetupChildrenScriptメッセージを送り、Pの子ビューを同期化しstepChildren配列とマッチングさせる。具体的に言えば、stepChildren配列内にもはや存在しないテンプレートをもつ子ビューはすべてクローズされ、stepChildren配列内の新たなテンプレー

トに対しては子ビューが構築され、残りのビューは、そのviewBoundsテンプレートが変更されているなら、再描画される。概念的には、Pを1アイテムずつスクロールダウンさせるためには、Pが、stepChildren配列から先頭のアイテムのテンプレートを削除し、表示すべき新たなアイテムの新たなテンプレートをstepChildrenの末尾に追加するだけでよい。こうすると、ビューシステムが不要な子ビューをクローズし、残りの子ビューを上へシフトさせて、画面の１番下に新たなビューをオープンする。

実際には、この方法はスクロールダウンの場合にうまく機能する。PのviewSetupChildrenScriptは、ArrayMungerへのシングルコールを用いて、stepChildren配列の先頭の要素を除くすべての要素を、新たなテンプレートのみを含む新たな配列の先頭に接合する。Newtonのビューシステムは、最上部のビューをクローズし、残りのビューを元の位置の１つ上に再描画し、画面の底辺に新たなビューをオープンする。しかし、スクロールアップは期待したようにはうまくいかない。スクロールダウンのパターンに従って、Pは、SetLength呼出しによりstepChildren配列から末尾のテンプレートを削除し、ArrayMungerを用いて配列の先頭に新たなテンプレートを追加する。ところが、驚くべきことに、ビューシステムは底辺の不要なビューをクローズすると、残りのビューを下方向へシフトさせる代わりに、その場所に新たなビューを描画するのである。これがうまく行くために、Pは、スクロールアップ時にstepChildren配列全体を再作成できる。よって、ビューシステムは各子ビューをクローズしてから再度オープンせねばならない。

このアプローチは、最初に紹介した方法の最も不都合な属性を緩和する。Pのオープンに要する時間は大幅に短縮され（わずか1.1秒）、所要ヒープも大幅に減少する（わずか5K）。この方法の欠点はスクロールのスピードが非常に遅いことである。10のアイテムをスクロールアップしてから元の位置までスクロールダウンするのに、32.4秒かかる。この所要時間は、すばやいスクロールダウン（すなわち、1つのビューがクローズして、1つのビューがオープンする）と遅いスクロールアップ（すなわち、S個のビューがクローズして、オープンする）という、2つの不均等な部分から成る。

## 集合体プロトとアップデートメッセージを使う方法

既にお気づきと思うが、3番目のアプローチとして、子ビューのクローズと再オープンを避ける方法がある。前述の方法と同様にS個の子ビューをもつ親ビューPを用いて、PがviewScrollUpScriptとviewScrollDownScriptメッセージを受け取ると、S個の子ビューの各々に対しアップデートメッセージを送り、表示すべき新たなアイテムの引き数を渡すようにする。表示画面を1つスクロールアップするには、各子ビューに対し、それがもつ引き数より1大きい引き数を送る。スクロールダウンするには、1小さい引き数を送る。

これを実施するのは、setOriginを用いる１番目のアプローチよりもやや複雑である。Pは、その時点で表示されている先頭のアイテムの引き数を常に保持しなければならない。また子プロトは、アップデートメッセージに対し応答し、その構成要素（チェックボックス、ピクチャビュー、スタティックテキスト、および目盛）を適切に初期化しなければならない。これを実行するには、プロトが、自分自身へredoChildrenメッセージを送り、構成要素にviewSetupFormScriptsで自身を初期化させるという方法が考えられる。ややスピードの速い別の方法としては、SetValueを呼び出すことにより、各構成要素に、表示画面の１部をリセットするアップデートメソッドをもたせる方法が考えられる。ユーザが末尾のアイテムを表示画面の１番上に来るまでスクロールできる（そして、画面の残りの部分は空白になる）ようにする場合は、プロトが、表示すべきアイテムがないケースも扱えるようにしなければならない。

この複雑な方法は、syncChildrenを用いる方法と比べると、オープンのスピードはほぼ同じであり、ヒープの使用領域はより少ない（5Kに対し4K）。しかし、スクロールスピードが遅くなる。アイテムの配列を端から端まで事前に計算し、PのviewSetupDoneScriptにchildViewFramesをキャッシュしたとしても、10のアイテムをスクロールダウンしてから元の位置までスクロールアップするのに、36.5秒もかかる。S個の子ビューとその孫ビューに対して送られる、SetValue呼出しなど、余分なメッセージの直接の結果としてスクロールスピードが低下する。

これまでに述べたアプローチは、両極端の特徴を示している。setOriginを用いるアプローチは、積極的に演算するという特徴を示す。すなわち、最初の機会にすべての結果を算出した後、これらの結果をキャッシュに保持して使う。キャッシュは構築するのに時間がかかるし、またスペースを多く使うが、その後の使用に際してはすばやいアクセスが可能である。これとは対照的に、syncChildrenとアップデートメッセージを用いるアプローチは、消極的に演算するという特徴を示す。すなわち、最初において最小限の演算を行い、その後は必要に応じて計算する。キャッシュに関するデメリットを免れる代わりに、そのメリットを享受することもない。

アップデートメッセージを用いるアプローチで、より良いスクローリングパフォーマンスを得るためには、アップデートメッセージの各ラウンドに伴う再計算を避ける必要がある。ユーザがスクロールすると、1つの新たなアイテムのみが表示され、古いアイテムのうち1つを除くすべてのアイテムは再表示される。ここで取り上げた例では、1回に14のアイテムが表示される。つまり、ユーザがスクロールするたびに、13の古いアイテムが再表示される。実に、演算の約93%が反復されているのである！

## 単純プロトとviewDrawScriptを使う方法

ビューをより単純化すれば、消極的に演算を行ってもスクロールスピードを速くできる。つまり、複数の子ビューで1つのプロトをビルドする代わりに、viewDrawScriptの中でdrawShapeメッセージを使用し、同等の視覚的表現を描画するのである。以前に子チェックボックスビューであったものが、今度は2つのビットマップ、すなわちチェックされているビットマップとチェックされていないビットマップのうちのどちらかになる。アイコンは1つのビットマップになり、アイコンの境界線はMakeRoundRect形状になる。スタティックテキストはMakeText形状になる。目盛は1組のMakeRect形状になる。

この方法をもう少し最適化するために、drawShapeメッセージが形状の配列を一度に描画できるだけの柔軟性をもつという点を利用できる。そうすると、プロトは、7個の形状すべてをイメージ化するdrawShapeメッセージを1回送るので、メッセージ送信の反復に関わるオーバヘッドを省ける。描画をさらに最少化するために、チェックされているビットマップとチェックされていないビットマップを、丸みを帯びた矩形状のビットマップと組み合わせることもできる。こうすれば、それぞれ別個にMakeRoundRect呼出しを行う必要がなくなる。アプリケーションの幅が固定しており、目盛とチェックボックスの距離がコンパイル時にわかるようであれば、目盛の下にあるグレーの矩形を組み合わせることもできるだろう。

ベースビューは、やはり親ビューPに対しviewScrollDownScriptとviewScrollUpScriptメッセージを渡さなければならない。前述の方法と同様、PはS個の子各々に対しアップデートメッセージを渡さなければならない。アップデートメッセージは、特定の子ビューが表示すべきアイテムの引き数を含む。プロトは、その構成要素へアップデートメッセージを送る代わりに、自分自身へ“ dirty ”メッセージを送るだけである。すると、その“ dirty ”メッセージが、プロトのviewDrawScriptを呼び出す。

このアプローチは、アプリケーションのオープンスピードは非常に速いが（1.3秒）、スクロールスピードはsetOriginを用いる方法の約半分に遅くなる（9.8秒に対し21.9秒）。しかし、集合体プロトとアップデートメッセージを使う方法より、はるかに速い。Newtonのビューシステムは、同じ数のビューを扱う場合で比較すると、ビューを更新するよりも、描画コマンドを実行するほうが速い。ビューシステムはプロトの各子ビューに対し、viewSetupFormScript、viewSetupChildrenScript、viewSetupDoneScript、およびその他のメッセージを送るという、非常に多くの作業を行うため、集合体プロトの使用がスピードの低下を招いても驚くにはあたらない。単純プロトとviewDrawScriptを使う方法のもう１つのメリットは、アクティブなビューがわずかしか存在しないため、ヒープ使用領域がごくわずかで済むという点である。図1に示した例で言えば、集合体プロトを用いる方法では少なくとも5×14＝70のビューを使用するが、単純プロトを用いる方法では14しか使用しない。つまり、ヒープの使用は5分の１に節約できると見積れる。これが4Kに対して1Kという実際の相違になる。

## 単純プロトとviewDrawScriptをキャッシングとともに使う方法

viewDrawScriptを伴う単純プロトは、スピードは速いものの、いまだ多くの作業を反復する。MakeTextなどのshape creation呼出しは、ユーザがスクロールアップやスクロールダウンするとき、同一のアイテムに対して反復される。これを補正するために、各アイテムの形状をキャッシュにセーブしておき、これらを再使用することもできる。こうすると、Pは、キャッシュ自体を構築するために、各アイテムに対し1つ、つまりN個のエントリをもつ配列を割り当てる。作成した個々の形状にアイテムの引き数を付けて、キャッシュにセーブすることもできるが、Newton OSには、MakePICTという、より優れた機能が用意されている。この機能はdrawShapeと同じ形状とスタイルの配列を受け入れる。ただし、この機能は、ストアしておいて必要なときにdrawShapeへ渡せるような、PICTピクチャを構築する。この方法を用いると、アイテムの描画コマンドのすべてを1つのピクチャとしてキャッシュできる。

キャッシュを使用するには、まず最初にプロトのviewDrawScriptが、キャッシュされたピクチャの配列をチェックする。キャッシュされたピクチャが1つもなければ、形状とスタイルとともにMakePICTを呼び出し、その結果をキャッシュする。次にviewDrawScriptは、キャッシュされたピクチャとともにdrawShapeメッセージを送る。

キャッシュを使用すると、アプリケーションのオープンの速度が遅くなるとは予測しないが、実際には1.3秒に対し1.8秒と、わずかに遅くなる。ピクチャを構築した後にイメージ化すると、何らかの時間的な不利益が生じる。アイテムをキャッシュにストアするには、ごくわずかな時間しかかからない。この場合キャッシュは、アイテムの引き数を付けられた配列であるため、ストアされた形状を検索するのにもごくわずかな時間しかかからない。その結果、スクローリングは21.6秒から14.7秒へと少し速くなる。オーバヘッドのうち、shape creationに起因するものは、全体の一部のみであることは明らかである。

キャッシュの使用がもたらす最大の相違点は、ヒープの使用領域が1Kから12Kへと増加する点である。キャッシュは、アイテムの各ピクチャにつき0.4Kのスペースを使用する。仮にユーザが100アイテムのリストの末尾までスクロールダウンした後に、元の位置までスクロールアップした場合を想定すると、キャッシュが膨らむために、このアプリケーションが44K以上のヒープを使用することになってしまう。この問題に対する単純かつ効果的な解決方法は、ユーザがスクロールするたびに1つのピクチャのキャッシュを解放することである（つまり、配列の適切な要素をNILに戻す）。スクロールのたびに、1つの新たなアイテムがビューに追加され、1つのアイテムがビューから削除されることを思い出していただきたい。viewDrawScriptを用いれば、新たなアイテムのピクチャが確実にキャッシュされるようにできる。またviewScrollUpScriptとviewScrollDownScriptを用いれば、隠されたアイテムのピクチャのキャッシュを解放することができる。これらの変更はスクロールスピードを0.5秒低下させるが、アプリケーションによるヒープ使用領域は8K以下になる。

スペースをさらに節約するために、S個の子ビューと同じ数のS個のピクチャを格納するように、キャッシュを構築することもできる。モジュロ演算を用いると、各子ビューの引き数は、キャッシュの特定の格納位置にマップすることができる。例えば、16番目のアイテムが15という引き数値をもつとする。これをS（=14）に対しモジュロ演算をすると、キャッシュには1という引き数が生じる。この位置は、（1という引き数をもつ）2番目のアイテムと共有することになるが、2番目のアイテムと16番目のアイテムが同時に表示されないのは明白である。この方法はNマイナスS個の空配列要素を節約するため、アイテムN個が多数の場合は、重要になりうる。

以上述べたアプローチは、演算所要時間と必要スペースの間の一般的なトレードオフを示している。ピクチャをキャッシングすれば、ヒープの使用スペースが7K増えるかわりに、スクロール時間を6.4秒短縮できる。setOriginを使用すると、ヒープの使用スペースがさらに24K増えるかわりに、スクロール時間をさらに5.6秒短縮できる。個々のアプリケーションの必要条件を考慮することは、このようなトレードオフにおいてどちらが望ましいかを決定する手引きになる。しかし、結局は、魅力に乏しい選択肢の中から選ばねばならないことも多い。

## 単純プロトとビットマップ・キャッシングを使う方法

Newton OS 2.0は、スクロールするアプリケーションにおける時間とスペースのトレードオフに関して、もう1つの選択肢を与える便利な機能を備えている。単純プロトとviewDrawScriptを用いると、アイテムの視覚的表現のビットマップ全体をキャッシュできる。前の方法では、ピクチャをキャッシュしたので、ビューをイメージ化するために、Newtonのビューシステムが一連の描画指令を実行しなければならなかった。新しい方法では、各ピクセルの値を指定することによって、より速くイメージ化されるビットマップを、キャッシュできるようになった。

まずアイテムが描画されるとき、以前と同様にdrawShapeとshape creation関数を使用するが、次に、結果の画像は、viewIntoBitmapメッセージを送ることによりビットマップに変換される。その次に、結果のビットマップは、アイテムの引き数を付けられた配列にキャッシュされる。アイテムを再表示するときは、形状を描画せずビットマップで表示する。

この作業を効果的に行うためには、viewDrawScriptを3つの個別の条件とともに用いる必要がある。第1の条件は、キャッシュがこの特定アイテムのためのビットマップを含むかどうか確認する。キャッシュを実現する配列がこのアイテムの引き数に対して非NILならば、ビットマップが存在するため、viewDrawScriptはdrawShapeを一緒に呼び出す。第2の条件は、キャッシュされたビットマップが存在しない場合に呼び出される。この場合、第2の条件は、空のビットマップを作成し、自身に対しviewIntoBitmapとdrawShapeを送り、その結果をキャッシュする。viewIntoBitmapメッセージがビューのviewDrawScriptを呼び出すと、第2の条件は、無限ループを回避するフラグを設定し、それをチェックする。第3の条件は、フラグが設定されていると実行され、viewIntoBitmapメッセージに対する応答としてビューをイメージ化する。この場合、形状は、キャッシングしないviewDrawScriptの場合と同様に描画される。参考までに、ここで取り上げた例において、単純プロトのビットマップをキャッシュするviewDrawScriptを、表1に示す。（このコードでは、丸みを帯びた矩形は、MakeRoundRectに対する別個の呼出しではなく、チェックボックス・ビットマップの一部として作成されていることに注意していただきたい。）

*表1.　ビットマップイメージをキャッシュする単純プロトのためのviewDrawScript*

```
func() begin
  if index then begin
    local B := madeBitmaps[index];
    // Draw the cached bitmap if there is one.
    if B then begin
      :drawShape(B,nil);
    end;
    // Otherwise, trigger viewIntoBitmap and cache result.
    else if NOT cachingInProcess then begin
      cachingInProcess := true;
      B := MakeBitmap(itemWidth,viewBounds.bottom,nil);
      :viewIntoBitmap(nil,nil,B);
      madeBitmaps[index] := B;
      :drawShape(B,nil);
      cachingInProcess := nil;
    end;
    // Called by viewIntoBitmap to image bitmap.
    else begin
      local Item := kItemArray[index];
      :copyBits(if Item.check then kCheckIcon else
        kUncheckIcon,0,1,modeCopy);
      :copyBits(Item.icon,19,5,modeCopy);
      :drawShape(
        MakeText(Item.text1,36,1,text1R,12),
        kTextStyle);
      :drawShape(
        MakeText(Item.text2,text2L,1,text2R,12),
        kTextStyle);
      :drawShape(
        MakeRect(sliderL,6,itemWidth,12),
        kBarBackgroundStyle);
      :drawShape(
        MakeRect(sliderL,6,
          itemWidth-RIntToL(36-Item.value*36/100),
          12),
        kBarForegroundStyle);
    end;
  end;
end
```

ピクチャのキャッシングの場合と同様、ビットマップのキャッシングは、アプリケーションのオープンのスピードを大幅に低下させることはない。この方法はオープンに1.5秒を要するため、これまでのところ、setOriginを用いる方法を除き、すべての方法のなかで１番速い。われわれの本来の目的は、スクロールのスピードを速めることだった。ビットマップのキャッシングは、すべての方法の中で最も速いスクロールスピードを実現している。10のアイテムをスクロールダウンしてから元の位置までスクロールアップする時間は、8.5秒である。このスピードは、setOriginを用いる方法よりも速く、アップデートメソッドを用いる方法の4倍以上速い。画面からスクロールオフされたアイテムのビットマップのキャッシュを解放する方法では、所要ヒープは12K以下であり、これはsetOriginを用いる方法の場合の半分だが、ピクチャをキャッシュする単純プロトを用いる方法の場合よりも約50%遅い。各ビットマップは約0.6Kを使用する。

スピードの向上とヒープの消費の関係は、ピクチャを描画するコマンドとピクチャのイメージを描画するコマンドとの相違を反映している。これらのコマンドは、ストアするためのスペースはより少なくてすむが、ピクチャを描画するたびに実行しなければならない（例えば、キャッシュされた形状の場合）。ピクチャのイメージは、ストアするためのスペースをより多く必要とするが、すばやく表示させることができる（例えば、キャッシュされたビットマップの場合）。

## プロトなしでviewDrawScriptを使う方法

論理的完全性を実現するには、子ビューを使用しないアプローチも考慮しなくてはならない。ストアされた引き数を変更し、自身に対し“dirty”メッセージを送ることにより、ビューPがviewScrollUpScriptおよびviewScrollDownScriptメッセージに応答する可能性もある。PのviewDrawScriptが、子ビューを用いることなく、すべての表示状態のアイテムをイメージ化する可能性もある。

このアプローチは、オープンに1.0秒を要する（僅差で最も速い）。またヒープ使用領域もごくわずかである（1/2K未満）。しかし、このアプローチは、形状もdrawShapeの結果もキャッシュしないため、スクロールスピードは他の方法の約半分である。drawShapeメソッドは、形状とスタイル・フレームの入れ子の配列（nested array）をイメージ化するため、キャッシングの追加は比較的簡単である。画面上にスクロールされたばかりのアイテムの形状を、適切な格納位置にキャッシュし、スクロールオフされたアイテムの形状はただちにキャッシュを解放するという可能性が考えられる。そしてキャッシュ全体をdrawShapeへ渡せばよい。この考え方をもう一歩進めれば、前述のキャッシュされたPICTとともにdrawShapeを送るという可能性がある。さもなければ、前述のキャッシュされたビットマップとともにdrawShapeを送るという可能性もある。これらのバリエーションは、キャッシングしないアプローチから比べると、前に検討した方法と同等のスクロールスピードの向上を示すはずである。子ビューを使用しないことによるヒープ上のメリットは、キャッシュのサイズと比べると、ごくわずかである。

子ビューを使用しないアプローチで、唯一大きな問題が生じるのは、ユーザが単一アイテムをタップすれば何らかのアクションを指示できるように、設計したい場合である。この場合、ビューPは、ユーザがどのアイテムをタップしたかを決定するためにhitShapeメソッドを使用する必要があろう。さらにPは、タップに対して適切な視聴覚的応答、すなわち強調表示とクリック音をフィードバックする必要もあるだろう。Newtonのビューシステムを利用せずに、このフィードバック機能をゼロからプログラミングしていく煩雑さと、その結果得られる、スクロールスピードとヒープスペースに関するパフォーマンスの向上を考え合わせると、この方法は割に合わないと言えるかもしれない。

## まとめ

表2は主な結果をまとめたものである。集合体プロトを使用すると、アプリケーションのオープン時またはスクロール時のスピードが低下する。また、ヒープの使用領域は非常に多いか、または非常に少ないかのどちらかになる可能性がある。単純プロトをviewDrawScriptと一緒に用いる方法は、アプリケーションのオープン時とスクロール時ともに、スピードが速くなる可能性があるが、キャッシュを使用しなければならない。キャッシュは、ヒープを消費し過ぎないように、慎重に維持しなければならない。

*表2.　スクロールの実施方法による、スピード、ヒープおよびアプリケーションサイズの比較*

| | 時間（秒） | | サイズ（K） | | 行数 |
|---|---|---|---|---|---|
| | オープン/クローズ | スクロール | ヒープ | パッケージ | ソースコード |
| 集合体プロトとsetOrigin | 3.6 | 9.8 | 32 | 10 | 157 |
| 集合体プロトとsyncChildren | 1.1 | 32.4 | 5 | 10 | 202 |
| 集合体プロトとアップデートメソッド | 1.6 | 36.5 | 4 | 11 | 206 |
| 単純プロトとviewDrawScript | 1.3 | 21.9 | 1 | 10 | 152 |
| 単純プロトとPICTのキャッシュ | 1.8 | 14.7 | 12 | 11 | 169 |
| 単純プロトとPICTのキャッシュを解放 | 1.8 | 15.2 | 8 | 11 | 173 |
| 単純プロトとビットマップのキャッシュ | 1.5 | 8.5 | 18 | 11 | 194 |
| 単純プロトとビットマップのキャッシュを解放 | 1.5 | 8.9 | 12 | 11 | 197 |
| プロト非使用 | 1.0 | 17.0 | 0.4 | 11 | 155 |

この分析の結果から、多数の子ビューで構築されたプロトは、スクロールを行うアプリケーションに適さないと言えるかもしれない。このような印象のバランスをとる手だてとして、集合体プロトを使用するアプローチとviewDrawScriptを使用するアプローチのどちらにするか決定するために、次の2つの経験則を考え合わせてみるとよい。

- データ構成要素が、それぞれ異なる方法で表示され、ユーザによる入力に対し異なった反応をする場合、アプリケーションは集合体プロトを使用すべきである。
- データ構成要素が、類似の方法で表示され、ユーザによる入力に対し同様に反応する場合、アプリケーションはviewDrawScriptを使用すべきである。

上記の経験則は、今回分析したスクロールの例について、次のことを指摘している。すなわち、もしユーザがそれぞれのアイテムを１つのまとまりとしてのみタップできれば良い場合は、viewDrawScriptをキャッシングと共に使う２つの方法のどちらかを選択すべきである。これとは異なり、もしユーザがアイテム上の様々な部分をタップできるようにする必要があるなら、集合体プロトとアップデートメソッドを使う方法が好ましいかもしれない。

NTJ

*1ページからの続き*

# Windows用のNewton Toolkitのリリース

ある種のコンパイルされたC++ルーチンを、NewtonScriptに移植することなく、直接Newtonアプリケーションにリンクできる。これらの製品はすべて、Newtonソフトウェア・デベロッパのニーズに応える広範囲な一連のツールを提供しようとする、アップルの決意の表われである。

NTK for Windowsは、CD-ROM版で出荷されており、完全なサンプル・コードとオンライン・ドキュメンテーションを含んでいる。ベータ・バージョンは、WWWで http://dev.info.apple.com/newtonからダウンロードできる。最終版は、株式会社バイス DeveloperDepot Japan（本文中での略称：DDJ）を通じて出荷される。

NTJ

# 高品質のNewtonアプリケーションを作りだすには

*by Peter Murray, Apple Computer, Inc.*

## はじめに

品質はまさに違いを生みだす。ユーザにとっての違い、そして究極的にはデベロッパの利益の違いを生みだす。しかし品質保証は、単にバグを発見し取り除くという行為を越えた事柄である。しかし、テストをどんなに行っても、テストだけではアプリケーションに品質を確保することはできない。品質とは、ユーザが経験する全ての側面においてユーザの期待に応える、あるいはそれ以上のものを提供することである。デベロッパが成功する秘訣は、アプリケーションが簡潔で動作が安定し、実用的であることが最も大切である。

小規模なデベロッパであれ大規模なデベロッパであれ、徹底した品質保証プログラムによって製品に大きな価値を与えることができる。十分にテストされた製品を出荷するという自信のほかに、ソフトウェア品質保証プログラム(SQA)は、ますますユーザ志向の製品を作り出すのに必要な、あるいはユーザからのフィードバックを得る中心的な役割を果たすことになる。

Newtonプラットフォーム上でのソフトウェアの品質保証は、他のプラットフォームのものと大差ない。すぐれたNewtonプログラムの開発に成功する必要条件は、次の３つである。

- ソフトウェア開発を理解していること
- 一般的SQA手順が実施されていること
- Newtonプラットフォーム上でのテストの詳細を知っていること

## ソフトウェア開発

ソフトウェア開発においては、プログラムを作成する前にまず計画を立て、その計画に従って開発を行う必要がある。マーケティン部門は、ターゲットとなるユーザを想定し、ユーザの要望を収集し、開発部門に伝えねばならない。設計チームはこれに対し、案件がソフトウェアによってどのように満たされるかを、ヒューマン・インタフェースおよび低レベル言語で記述しなければならない。マーケティングと設計の要求に基づき、SQAでは、どのようにテストを実行し、高品質の製品にまとめ上げて行くかというテストプランを作成する。チームの全員が、最終製品がどのような形をとり、それが誰のためであり、どのような問題を解決するのかについて、議論しなければならない。木を見て森を見ないように、詳細にとらわれすぎて大きな展望を見失ってはならない。

ソフトウェア開発は対話型のプロセスである。ソフトウェアが、アルファ、ベータ、ファイナルそしてゴールデン・マスターという重要な各段階に到達するのはどんな意味があるのだろうか。どんな尺度で開発進度を測ればよいのだろうか。そしてプロジェクトの現況を判断する各重要段階での基準は何だろうか。これらの疑問に対する答えは、プロジェクトの目標に大きく依存している。尺度と判断基準を注意深く選択しよう。ものごとを簡便にやろうという最小限主義は、すべて規則にしばられた官僚主義によってプロジェクトの進行を遅くしてしまうより良い。

十分な計画性を持つ製品開発のプロセスには、作業を遂行するガイドラインを含むが、ソフトウェア開発はもともとダイナミックなものである。日常行う決定がプロジェクトの目標を反映したものであり、ユーザの要求を満足させるようにしなければならない。

## SQAの一般的な方法

良いSQAプログラムは、常に以下の考えが土台となっている。

**• テスト範囲に優先順位を付ける。**

全てのことをテストをすることはできない。したがって、優先順位を付けることが最も重要である。

ユーザの観点から見て、必ずうまく機能する必要があり、製品を成功させるのに必要なのは、何であろうか。重要性の度合いに応じてリソースを集中せよ。

テスト範囲は開発技術部門によってある程度決定されるべきである。

開発エンジニアひとりひとりに、どの部分のコードを担当し、何に１番心を悩ませているかを聞くこと。

相互依存している部分や仮定を含む部分を探すこと。これらの領域には、バグが多く含まれる可能性があるからである。

エラーチェック自体を、全てテストせよ。エラーをシミュレートし、エラー処理の妥当性を検証せよ。

開発と品質保証のエンジニアが協力して、コードをブレークする独創的な方法を発見すること。

弱点は、コード自体にも、またデザイン・レビュー時にも、発見される可能性がある。

経験不足のプログラマは、より広範囲のテストを必要とするかもしれない。特に、システム固有の機能にこれがあてはまる。

**• 知識の共有**

開発と品質保証のエンジニアは、ソフトウェアの機能とそれを支えるコードの経路について、出来る限り共通理解を持たねばならない。

**• バグを発見する科学的方法**

優秀な品質保証と開発のエンジニアは、正式な科学的方法を習得した上で、デバッグを行う時には、それを無意識に使っている。“ 科学的方法の真の目的は、実際には知らないことを、知っている、というような誤解に、陥らないようにすることである ” [1]

以下、科学的手法をSQAに適用する場合の要素を、簡略に示す。

1) 問題の記述
2) 問題の原因についての仮説
3) 個々の仮説をテストするためのテストケース
4) テストケースの予期される結果
5) テストケースからの観察された結果
6) テストケースの結果から得られる結論

**• 管理されたリスク**

製品の品質検査と出荷にかかわる決定の多くは、様々な程度のリスクを伴う。難しいのは、正確なリスク分析に基づき、リスクを最小化するステップをとることである。リスク分析は真剣に行われるべきであり、全ての判断は、事実に基づくことであり、推量に基づいてはならない。常に、リスクを最小化するための創造的な方法を考え、努力をすること。次にその例をいくつか示す。

***コードの変更***

プロジェクトがかなり進行した後で、また重要な区切りの段階に近い所でコードを変更する場合、変更のリスクを最小化する効果的な方法が、いくつかある。コード変更をする場合は、コードを書いた本人とは別の開発エンジニアにより、コードレビューを行うこと。

変更を行うデベロッパは、リリースノートを書き、その中で、変更がシステムの他の部分に及ぼす影響並びに実行すべき重要なテストについて述べること。

SQA手順により、変更部分と関連分野の検証とテストを行うこと。

### バグ

全ての事項をテストすることはできないし、現実のユーザが実行しうる全ての方法で製品を使用してみることもできない。従って、全てのバグは発見できないし、殆どの場合、既知のバグがある状態で出荷を行うことになる。ユーザから見た重要度により、取り除くべきバグには、必ず優先順位を付けること。バグが、曖昧なケースでしか発生しない場合は、除去するに値しないかもしれない。大多数のユーザに影響するバグや、使い勝手の問題は、解決すること。悪い宣伝をされる可能性のあるものにも、注意を払うこと。時には、予想が外れることもある。解決を延期するバグは記録を取り、実際にユーザがレポートするかを見る。そして、次のバージョンでは段階的に対応していき除去する。

## • カスタマフィードバック

製品開発の改善は、フィードバック・ループに基づいて行なわれる。プロダクトサイクルの異なる部分でユーザからのフィードバックを取り入れる３つのプログラムがある：ユーザテスト、ベータテスト、そして、“ あれれ？（uh oh) ” というユーザの声に対して行うサポートである。

### ユーザテスト

開発サイクルの初期の段階で、ターゲット・ユーザに製品についてフィードバックしてもらうことにより、デザインの直感的な部分に関するデータが得られ、組込まれようとしている機能の取り合わせが正しいか、検証できる。SQAエンジニアは、現実のユーザがどのように製品を使うかを観察することによっても、テストを改善できる。新鮮な観点で眺めることには常に価値がある。なぜなら、開発チームは、製品にあまりに近すぎるからである。

### ベータテスト

現実の環境でソフトウェアを使うことにより、ベストなフィードバックが得られたり、バグが発見されることも多い。Newton OS 2.0プロジェクトの期間中、Newtonチームのメンバーの多くは、最新バージョンのソフトウェアを搭載したフラッシュ・ROMを搭載したMessage Padを持って歩いた。多くの良いバグ・レポートは、これらのMessage Padをミーティングで使った人々から出された。ここに、1.xと2.0の違いがある。2.0では、実際にミーティングの中でノートを取ることが出来るのである。

### ユーザサポート

製品開発は、製品を出荷した時点では終わらない。ユーザサポートの担当者は、ユーザの反応を定量化し、サービスコールの頻度により、将来のバージョンアップでどのバグを除去するか、優先順位を付けることが可能になる。機能についてのユーザからの要望は、製品を実際に使っている人々から良いアイデアを集める情報源である。Newton OS 2.0の開発は、ユーザの要望と使用上の問題により、大きくつき動かされ、ソフトウェア開発チームの熱意ある設計作業が、これを現実のものとした。

## Newtonでのテストの詳細

Mac OS, Windows あるいはNewton アプリケーションのテストは、全て本質的に同じである。ソフトウェア開発の要求とSQAの要求とをバックグラウンドに、プラットフォームの技術的な詳細を重ね合わせて見る。Newtonのアプリケーションをテストする際は、いかにNewtonが動作し、どういったアーキテクチャであるのかを知っておくことである。最も良い情報源は、ユーザーズマニュアル、Newton Programmer’s Guide (NPG), Newton User Interface Guidelines である。Newton用に書かれたアプリケーションは、システムサービスのレイヤの1つに、プラグインされる。したがって、ソフトウェア機能を隔離した状態でテストするほかに、システムサービスとのインタフェース部分のテストが必要である。

以下、テストすべき重要事項のいくつかを列挙する。

- 認識ビューのフラグを、入力の種類に合わせて、正しく設定すること。
- データのストアとファイリングをロックしないカードおよびロックしたカードについてテストし、内部的にスープのコードを検証すること。
- アプリケーションをExtrasから削除してremoveScriptとオプションのdeletionScriptをテストすること。アプリケーションが間違いなく自身を様々なレジストリから削除し、“ 死に神に取りつかれ ” ないようにすること。（注：無効参照が発生し “ にっちもさっちも行かなくなる ” のを避けること。）
- アプリケーションデータの入ったカードをイジェクトし、その際、スクリーンがリフレッシュされることを確認すること。
- アプリケーションが組込み型のプロトを変更する際に、どの様な違いを生むか、テストすること。
- アプリケーションがランドスケープ（landscape）をサポートする場合は、全てのダイアログが適合するようにすること。
- アプリケーションによるFindの使用をテストすること。特に、エントリの強調表示のような、変わった動作をする場合には、テストすること。
- アプリケーションが、バックドロップ(backdrop)として正しく動作するか、確認すること。
- カスタムディクショナリにリンクする場合、様々な状態でテストすること。
- スープの変更その他の通知内容が、アプリケーションに反映されているか、確認すること。
- プログラムによるフレームメモリとシステムメモリの使用を最適化すること。
- 絶対的に必要でない限り、アプリケーションがグローバル関数を作成しないようにすること。
- 最新バージョンのシステムで、テストすること。（システムアップデートを削除する最も簡単な方法は、バッテリーを全て抜き取り、バックアップ・ターミナルを硬貨でショートさせることである。）
- アプリケーションがscreen-relative bounds（スクリーン内の相対表示位置を定義する関数）を使用し、異なるスクリーンサイズとの互換性を持たせること。
- アプリケーションのユーザ･インタフェースは、ガイドラインに準拠すること。
- スープをストアフォルダから削除し、アプリケーションがこれを適切に処理するか、確認すること。
- アプリケーションが1.xと2.0の両方のプラットフォームをサポートするか、テストすること。
- 2.0環境で、アプリケーションが、read-onlyの1.xカード上の自分のデータおよびロックされた2.0.のフォーマット済みカード上の自分のデータを、どう処理するか、テストすること。
- Newtonキーボードでアプリケーションを動かし、ビューのためのタブの順序が正しいかどうか、確認すること。
- 一般公開されたアプリケーション・インタフェースは、全てテストすること。
- サポートする全ての周辺機器をテストすること。
- アプリケーションが1.xと2.0の両方をサポートする場合、`gestalt`を使用して個々のNewton機能があるかないかをテストすること。
- アプリケーションが、組込みアプリケーションの動作を変更するなら、他の既存のアプリケーションで、他人が自分のやっているのと同じ事をやっていないか調べる。非互換性が発見された場合は、正式な書類にすること。
- アプリケーションが、正式に記録されたAPIとDTS認定のメソッドのみを使っていることを確認のこと。例えば、組込み型アプリケーションのスープ・データにアクセスする際にである。こうすれば、非互換性が将来発生するのを避けることができる。

バックグラウンド情報およびNewton固有の情報を適用し、以下の５つのステップを踏むことで、Newtonアプリケーションを、開発サイク

ルを通じてテストする枠組みが得られる：

1. 公式記録、プロトタイプの結果、ならびに開発エンジニアとのコミュニケーションの結果にもとづいた、製品の機能一覧表を、詳細かつ階層構造的に作成し、ユーザがアクセスできる全ての機能を記述すること。ただし、あまりに詳細にする必要はない。機能一覧表は、製品の現在の状態を反映するよう更新すること。

2. 前述の機能一覧表からテストケースを作成する。個々の機能に関し、１つまたはそれ以上のテストケースを設ける。基本機能をテストするケースは、各ケースにつき１機能のみをテストすること。こうすれば、バグの分離が容易になり、テストケースとバグを関連づけやすい。システムサービスのリストを使用してテストケースを作成し、アプリケーションがNewtonのその他の部分と適切にインタフェースを取れるかテストする。

   例：Newton 2.0のTime Zoneアプリケーションの１機能としてcity（都市名）を削除できる。この機能のテストケースのいくつかを列挙する：
   a. 組込みROMのcityを削除する。
   b. ユーザによって追加されたcityを削除する。
   c. 仕事場所(worksite)として指定したcityを削除する。
   d. あるcityを削除してからundoをタップする。
   e. 組込みROMのcityを削除し、次にExtrasのストアフォルダからTime Zoneスープを削除する。

3. テストケースの中から“手早く見る(quicklooks)”ための短いリストを作成する。この短いリストに基づき、製品の主要機能をテストする。新開発のソフトウェアがSQAプログラムに渡されると、構造化されたテストをこの短いリストにより開始する。アプリケーションの主要部分が、テスト開始後最初の３０分でブレークした方が、２日たってからコードブレークが発見されるより生産的である。

4. 通常の開発中のソフトウェアと、アルファあるいはベータなどの重要な区切りにおけるソフトウェアの両方に対し、最も効率の高いテストサイクルを決定すること。すべてのビルド（コード生成）において全テストケースを実行する必要はないだろう。短いリストのアイテムを全てテストし、更にその上で、変更分野のテストやビルド・リリース・ノートに正式に記録されたバグの解決に集中した方が、より効果的だ。テストケースの全てを実行するのは、アルファ、ベータ、ファイナル等の重要な区切りの時期である。

   各テストサイクルの１部分は、必ず、計画外の、必要に応じたテストに当てねばならない。その際は、アプリケーションの動作の知識と、ユーザがどの様に機能を使うかの知識を活用する。１度でもユーザテストを観察したことがあるならすぐに気づくことであるが、多くのユーザは、最も簡単な機能を使う時ですら、ルールに従わない。Newton OS 2.0のテストの最終段階では、このような状況に応じたテストに一層重点がおかれた。すなわち、テストチームを２つのグループに分け、１つは“検証グループ（Validator）”もう１つは（他にこれより良い名前が見つからなかったので）“壊滅者グループ（Exterminator）”と名付けた。検証グループは、製品の全機能に渡り、設定されたテストケースの全てを体系的にテストした。他方、壊滅者グループは、全く無計画に、システム全体を自由にテストしたのである。

5. プロジェクトが進展し、ソフトウェアの機能が安定してくるにつれ、テストを拡張してストレスとバウンダリ条件の分野に及ぼさねばならない。それは、予期せざる条件に対してソフトウェアが安定した処理を行い、動作の安定性を増強させる目的のためである。ユーザが製品を使用する全ての方法についてテストを行うことは不可能なので、このような方法を取ることによって、フィールドで重大なバグが発生するリスクを最小化することに役立つ。Newtonアプリケーションにストレスを与える単純な方法の１つは、データ量を増やすことである。例えば、平均的なNewtonのユーザは、Namesアプリケーションでは、おそらく100から200のエントリを持つであろう。しかし我々は、2.0では1500以上のエントリを使ってテストを行ったのである。

## 結　論

皆さんがこの記事に書かれた全てのアドバイスを採用しても、高品質のアプリケーションを出荷できる、という保証はない。ソフトウェア開発における全ての要素を１つにまとめる絆は、仕事を大切にする心である。自らの仕事に誇りを持ち、最善を尽くすことに満足を見い出す、心である。“仕事への愛と品質は同じものが内と外に現れたものである。仕事をしながら品質を見、感じる人は、仕事を愛する人間である。自分が見たり、したりすることを大切にする人は、必ず自己の人格になんらかの資質を実現した人である”[2]

## 参照文献

1. Pirsig Robert. Zen and the Art of Motorcycle Maintenance. 1974. Page 94.
2. Idem. Page 247.

NTJ