

私のMacはWebサーバだ

愛しのHyperTalk

田中求之 mact@antares.ecn.fpu.ac.jp

注意：以下の文章は、技術評論社の雑誌 Macintosh Developer Journal に連載していた「私のMacはWebサーバだ」の原稿として書かれたものです（Dec, 1997、28号に掲載されました）。

はじめに

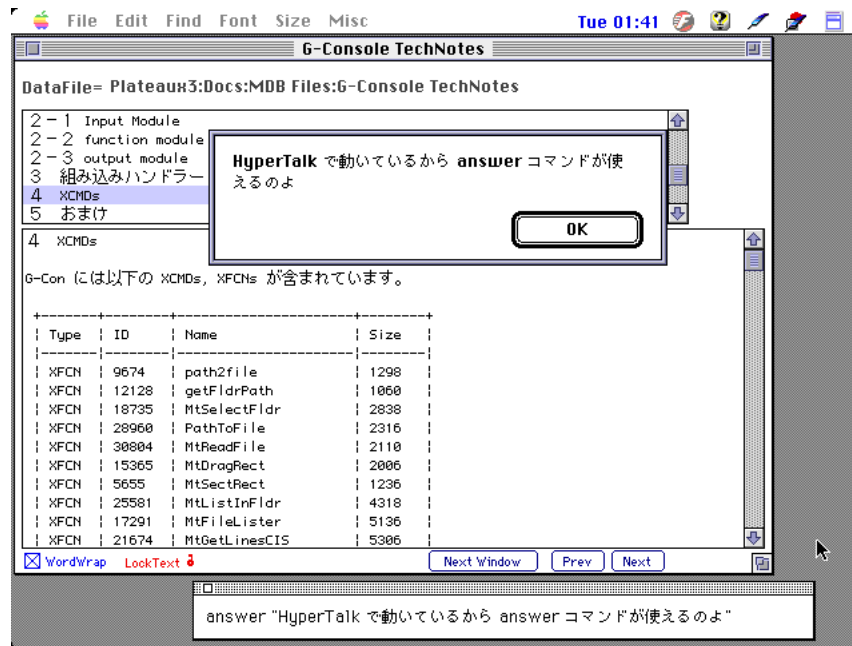
今回は筆者の長年の開発言語であるHyperTalkについての話をしようと思います。すでにCGIの解説の中でも少し触れたのですが（気が付かなかった方も少なくないかもしれませんが）、筆者はCGI作成の過程で必要になったosaxをHyperTalkで作成しています。筆者はCもC++も書けません。かろうじてPascalやCのソースから自分に必要な知識ぐらいは読み取ることができますが（なにせInside Macintoshのサンプルコードを読むのに必要ですからね）、自分でソースコードを書いたことはありません。HyperTalkとAppleScriptしかできない、根っからのスクリプト野郎なのです。そんな筆者でもosaxを作ってAppleScriptでコントロールできる範囲を広げることができたからこそ、色々なCGIをMacOSのWebサーバで動かすことができたのです。

CGIをAppleScriptで作れると言っても、システムに付属のAppleScriptのままではたいしたことはできません。Formに書き込んでもらったメッセージをファイルに記録するといった、CGIとしてはかなり単純な部類に入るものでさえ、osaxを追加しなければ、使い物になるものは作れないのです。特に日本では漢字コードの問題がありますから、純正のセットのままではCGIを作るのは無理だと言ってもよいでしょう。筆者がCGI作成に取り組み始めて最初にぶつかった壁がこのことでした。そして、自分でosaxを作れさえすればAppleScriptでもかなりのことがCGIとして可能になると思い、それまでXCMD作りに愛用していたコンパイラを使って、CGIのためのosax作りを始めたのです。まずは、そのへんの経緯から話を始めます。

HyperCardからWebへ

筆者は、3年前にLC475でWebサーバを立ち上げるまでは、ずっとHyperCardに夢中になっていました。といっても、マルチメディアやハイパーテキストの作品を作るのではなく、もっぱらHyperTalkによるプログラミング（スクリプティング）環境としてHyperCardを使っており、スタックをいわばshellとして使い、ファイルのキーワード検索とか草稿の管理といった仕事を行わせるツールをあれこれと作るのに使っていました。HyperTalkでできないことを補うためにXCMD作りにも手を染め、揚げ句の果てはHyperTalkでアプリケーションを作ったりもしていました¹。図1は今でも愛用しているHyperTalkで作ったファイルブラウザです。またパソコン通信もやっていたのですが、興味の中心はHyperCardで、自作のスタックやXCMD集をNIFTY-Serveにアップロードしたり、夜ごと会議室でHyperCardについて語り合うのを楽しみ毎日過ごしていました。このように、HyperCardにどっぷりと漬かった日々だったのです。そんなある日、勘違いがきっかけで、手元にあったLC475がWebサーバになってしまったのでした。

¹ スタンドアロン形式のスタックのことではありません。ちゃんとした(?)アプリケーションです。作れるって知ってました？ HyperTalkのサブセットが走るshellアプリケーション(Double-XX)にXCMDとインターフェース拡張ツール(WindowScript)を組み合わせてアプリケーションに仕立てるんですよ。こういったHyperCardの開発環境については、かつて技術評論社から出ていたMacJapanという雑誌に短期連載の形で紹介記事を書いたことがあります（MacJapan 1992年5月号、および1992年11月号~1993年1月号）。



今でこそ、こうやって偉そうにWebサーバやCGIについて語っていますが、3年前にWebサーバを始めたときには、筆者はHTTPが何のことも正確には知らないという状態でした。インターネットはメールやニュースを使っていましたが、こちらでもHyperCard関連のニュースグループやメーリングリストしか見ていませんでしたので、サーバソフトウェアの動向など知りませんでした。Mosaic（当時はNetscapeはまだ出ていませんでした）でホームページを覗いたりもしてましたが、こういうもののサーバなんてものはUNIXのワークステーションでしかできないと思ってましたので（今でもそう思っている人もいらっしゃるようですが）、プロトコルなんてものに関心もなかったのです。そんな筆者が、研究室に入ったばかりだったLC475で使うインターネット関連のツールをそろえるためにInfo-macを漁っていたとき、MacHTTPというシェアウェアのWebサーバソフトを、URLにつかうhttpが付くくらいだからMosaic関連のユーティリティだろうなと「勘違いして」ダウンロードしたのです。Read Meを読んでみると、これを動かすとWebサーバになると書いてあります。ほんまかいなと半信半疑でMacHTTPをダブルクリックし、MosaicにLC475のIPアドレスを打ち込んでみると、確かに目の前のソフトがサーバとして動いている！それでも何となく信じられなくて、同僚のSunのワークステーションのMosaicで試してみたのですが、そこからでもちゃんとアクセスできる！その時の、「そうか、こんなもので、こんなに簡単にWebサーバってできるんだ」という驚きと感動は今でも覚えています。

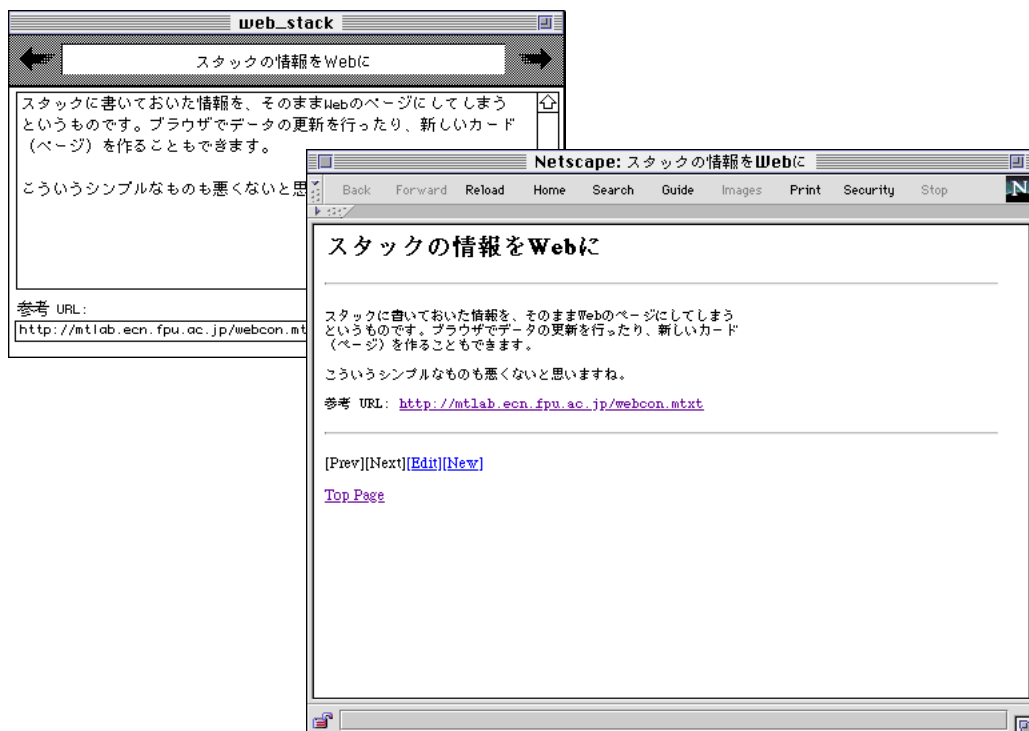
こうして、LC475がひょんなことからWebサーバになって以来、次第にサーバ運用にのめり込んでいくことになったのですが、それでも、最初のうちはまだHyperCardが中心でした。そもそも自分のWebサーバを一般に公開しようと思ったきっかけの一つは、NIFTY-Serveで知りあったHyperCard仲間のスタック作品をインターネットで広く知ってもらうための場を提供しようと考えたからでした。当時はプロバイダが日本でもようやくでき始めたという状況で、個人が簡単にホームページを持てるような環境は整っていませんでした。そういうわけで、仲間の作品を紹介したり、自作のスタックの載せたり、色々と揃えていたHyperCard関連のツールの紹介などをしたりして、HyperCardの情報やスタックを集めたサイトにでもしようかと始めたのが私のサーバ（サイト）の出発点なのです。

HyperCardとCGI

WebサーバではCGIというプログラムを使うことで色々な機能（仕掛け）を追加できるということも、もちろんWebサーバの運用を始めてから知りました。MacHTTPではAppleScriptやHyperCardなどのAppleEventが扱えるスクリプティング環境で開発が可能になっていたため、筆者は一気にのめり込むことになりました。というのも、筆者は絵を描いたり画面のデザインをカッコよくしたりするアートのセンスはありません。だからこそ、HyperCardでももっぱらスクリプトに凝り、そういう楽しみ方もできるところがHyperCardの素晴らしさだと言っていたのですが、Webでも同様で、筆者のようなアートのセンスのない人間でも楽しめる部分があったのです！

「HTML+CGI=インターネットのスタック」という発見（大げさですが）によって、筆者はHTMLの習得はそっちのけで、CGIの作成にのめり込んでいくことになりました²。

AppleEventが使える開発環境ならCGIが作れるということで、最初は、当然のようにHyperCardでCGIを作ってみました。図2は筆者がHyperCardで作ったCGIスタックです。ブラウザでは図3のように表示されるようになってます。このように、URLのデコードなどはXFCNを作って処理を行うようにすれば、確かにHyperCardでもCGIは作れました。また、リスト1がCGIのためのハンドラーの基本形ですが、パラメーターを順に取り出していくのは面倒にせよ、スクリプトとしては決して難しいものではありません。それでも、筆者は、これでは使い物にならないと思いました。CGIを動かすためにはHyperCardをサーバで動かさなければならぬためメモリをかなり必要とすることと、何よりも最大の問題は実行速度が遅いということでした。LC475をサーバーにしていたこと、またThread Managerが本格的に用いられるようになる前だった（システムにも組み込まれておらず、MacHTTPも未対応）という状況もあるのですが、とてもCGIとして使える速度ではなかったのです。



² おかげで、筆者のHTMLの知識はNetscape1.0当時のままで止まってしまい、いまだにテーブルを書くときには参考書がなければタグを間違えます。ちなみに、筆者はHTMLはエディタでしか書かない（自分で書けないようなタグは使わない）主義です（^_^;;

```

on appleEvent class, eventID, sender
  if class & eventID is "WWWλsdoc" then

    --- path_args を取得
    request appleEvent data with keyword "----"
    put it into myPathArgs

    --- search_args を取得
    request appleEvent data with keyword "kfor"
    put it into mySearchArgs

    --- post_args を取得
    request appleEvent data with keyword "post"
    put it into myPostArgs

    --- 以下、同様に request コマンドで必要なパラメータを取り出したう
    --- えで、それをもとにCGI の具体的な処理を行う

    --- 最後にページを reply (myMsg にページ用の HTML が入っているとす)
    put return & linefeed into CRLF
    reply "HTTP/1.0 200 OK" & CRLF & "MIME-Version: 1.0" & CRLF ツ
    & "Content-Type: text/html; charset=Shift_JIS" & CRLF & CRLF ツ
    & myMsg

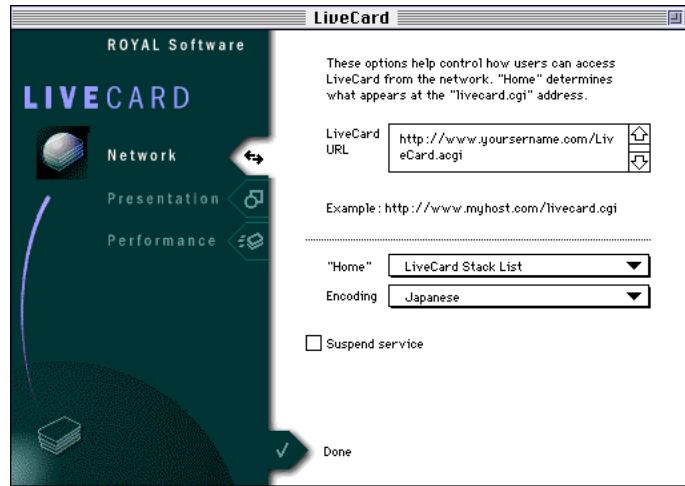
  else
    pass appleEvent
  end if
end appleEvent

```

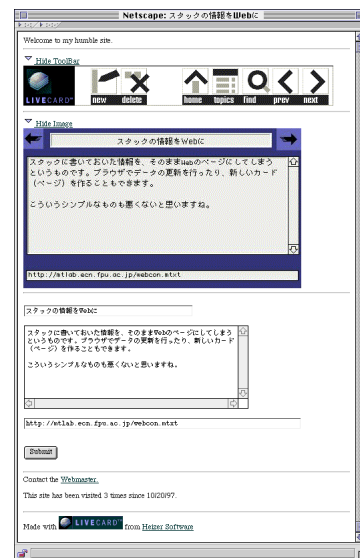
処理速度の問題さえなければ、CGIの開発環境としてのHyperCardは、かなりの可能性がある
と、今でも思っています。スタックというオブジェクトの集まりを土台としてプログラムが作れ
るということは、簡単に言ってしまうと、スクリプトで自由自在に操作できるデータベース上で
プログラムを書くということです。また、Webもスタックも画像と文章を1枚の画面にレイア
ウトしてページを構成するわけですから、スタックのデザインをそのままWeb上に移すことも不可
能ではないと考えていました。

実際、Royal Software社³より、LiveCardという、スタックをそのままWeb経由で使えるよ
うにするツールが販売されています(図4)。

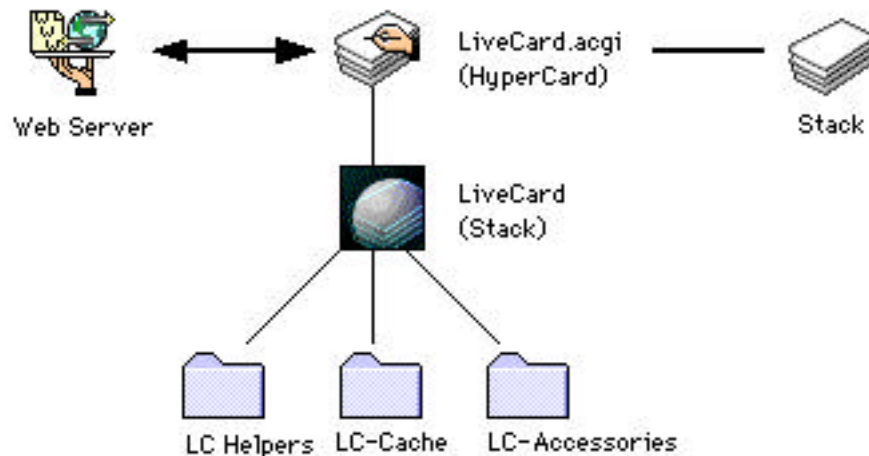
³ HyperCard関連の各種ツールの販売で有名だったHeizer Software社は、買収されて現在ではRoyal Software社になっています。
Heizer Softwareのツールの販売やサポートはそのまま引き継がれると共に、XCMDやHyperCard用スクリプトエディタなどの新しい製品も
販売しています。
<<http://www.royalsoftware.com/>>



これを使うと、CGIとして特別なスクリプトを書く必要すらくなく、自分の手元できちんと動くスタックさえ作れば、それが、リンクやボタンのスクリプトの機能なども含めて、そのままWeb上で動かせるようになります。先程のCGIのサンプルのスタックと同じデザインでカラーのスタックを作り（ただし、カード移動用ボタン以外には一切スクリプトを書いていません）（図5），それをLiveCardを使ってWebで表示させたのが図6です。



LiveCardでは、カードの絵をXCMDでJPEG画像にして表示し、一方でフィールドはFORMに変換することで入力や編集を可能にしています。画像部分はクリックマッピングになっており、ボタンのある位置をクリックすると、スタックのボタンに対してmouseUpメッセージが送られ、その実行結果がちゃんと表示されます。図6の場合ですと、矢印の部分をクリックすると、ちゃんとカード（ページ）が移動します。図7のような仕組みになっており、LiveCardというスタックがライブラリに追加された状態で動き、Webサーバとスタックとの間の仲介役としての処理を引き受けるようになっています。



このように、スタックをきちんと作れば、それがそのままWebでCGIになってしまうという、ある意味で画期的なツールなのですが、やはり実行速度が遅いのです。カード画面のJPEG変換こそXCMDを用いていますが、その他の処理はHyperTalkで書かれています。ここまでのことがHyperTalkでもできるということで、HyperTalkのプログラムとして見たら筆者は感動すら覚えますが、処理速度がどうしても遅いという現実はいかんともしがたいわけです。最新のPowerMacをこのLiveCardを動かすための専用のWebサーバに用いれば、そこそこ使えるものになるのかもしれませんが、少なくとも筆者の環境では使い物になりませんでした。

このように、筆者の環境では、どうしてもHyperCardのCGIを使う気にはなれなかったのです。HyperCardが好きでMacを使っていたようなものですから、遅いのを覚悟で、HyperCardでCGIを動かしていること自体に意義を持たせ、そのことに満足することも考えました。筆者がHyperTalkしかできなければ、そうしていたかもしれませんが。しかし、筆者が使えるもう一つのスクリプト言語であるAppleScriptが、まさにCGIを作るのには手ごろで、かつ速度も実用的であるということ、そして必要に応じてAppleScript対応の他のソフトウェアを呼び出せるというメリットがあったため、こちらでCGIを作っていくことにしたのです。こうして、次第に、HyperCardよりもAppleScriptのスクリプトを書くことの方が多くなり、スタック作りから遠ざかっていくことになったのでした。

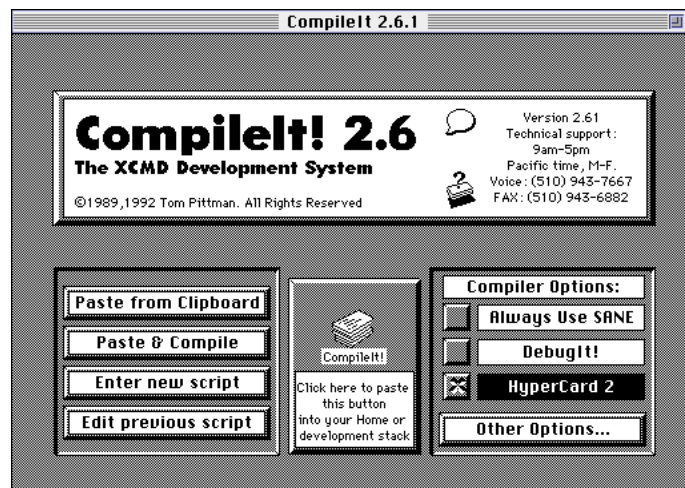
CGIにはosaxが必要

AppleScriptでCGIを作り始めてみて、筆者は、ようやくAppleScriptのスクリプティングが十分に楽しめる対象をみつけたと思いました。鳴り物入りで登場したAppleScriptでしたが、1994年の時点では対応アプリケーションの数も少なく、普段の仕事や作業の中で活かす(今風に言うとソリューションを提供する)ツールとは言い難い状況だったのです。筆者は、AppleScriptがリリースされるとすぐにDeveloper Kitを購入し、HyperCardの時からお世話になっているダニー・グッドマン氏の解説書を読んだりして、一通りはAppleScriptが使えるようになっていた(つもり)でしたが、だからといって何かに使うという状況ではありませんでした。しかし、ようやく、CGIという、AppleScriptによって具体的に何かを行うことができる領域が見つかった、これが筆者の気持ちでした。

しかしながら、すぐに壁にぶつかることになりました。それが、冒頭に述べた、osaxを追加しなければ、非常に限られたものしか作ることができないということです。AppleScript自体がosaxがあればこそ使いものになる環境であるという側面は持っており、最初から数多くのosaxが付属しているわけですが、これらの純正のセットだけでは、CGIを作るのには不十分なのです。ア

クセスカウンターやクリックابلマップのようなものであれば問題ないのですが、FORMに書き込まれたメッセージの処理がからんでくると、メッセージのデコードなどにosaxを追加する必要があります。このため、CGIのためのosaxがInfo-mac等に登録公開されはじめていたのですが、いくら待っても出てきそうもない、それでいてメッセージ処理のためには不可欠なosaxがありました。それが漢字コード変換だったのです。

最初のうちは、漢字コード変換を行うアプリケーションを呼び出して処理させていたのですが、呼び出しを行う分遅かったり、処理を思い通りにコントロールできるわけではない（アプリケーションが提供してくれる機能をやり繰りして使うしかない）ので、これはやはりosaxを作るしかないんだろうな、と考え始めました。とはいえ、osaxに関する資料がまったく見つからず、最初のうちは何をどうすればいいのか皆目見当が付かない状態でした。資料を探しているうちに、Appleが作成した、XCMDをAppleScriptで使えるようにするosaxというのを見つけましたので、とりあえず漢字コード変換のXFCNを一式作り、それをosaxとして使ってしのぐという状況が1年ほど続きました。HyperCardにハマっていたときにXCMDやXFCNは山のように自作してしまっていたのでそれらが使えて重宝していたのですが、HyperCardの環境にあわせて作られたXCMDでは色々と制約も多かったということと、このXCMD用osaxは配付ができないということ（CGIでの漢字コード変換をどうやっているんだ？ ツールを分けてくれといった問い合わせが増えていたのです）が不満でした。やはり自分でちゃんとしたosaxを作るしかないな、やっぱりCodeWarriorでCをやらないと作れないのだろうか？ などと思っていたところ、ようやくosaxについての解説が載った『Macintosh プロフェッショナル・プログラミング』（デーブ・マーク著、トッパン）が国内で出版されました。読んでみると、osaxは、タイプ'osax'のコードリソースを作ればよいと書いてあります。な～んだ、コードリソースだったのかぁ、それだったらコイツでばっちり作れるはずだ、と筆者が立ち上げたのが、XCMD作りに長年愛用してきたCompileIt!というHyperTalkのコンパイラです（図8）。



HyperTalkによるCGI作成

ようやく今回の記事の本題にたどり着きました(^_^;; Webサーバを始めたことでHyperCardから離れ、スタック作りから遠のいてしまった筆者でしたが、AppleScriptでCGIを作り始めたことで、ふたたびHyperTalkでスクリプトを書くことになったのです。ただし、今度はスタックのスクリプトではなく、osaxのためのスクリプトです。そしてosaxが作れるようになったことで、それまでアイデアだけにとどまっていたCGIを具体的に作ることが可能になりました。以前に紹介した会議室用CGI（EasyBBS）は、osaxが自分で作れたからこそ形にできたCGIの一つです。こ

のように、CompileIt!によって、HyperTalkとAppleScriptを組み合わせたCGI開発環境を筆者は手に入れたのです。こういう環境で開発を行っている人間というのは、おそらく非常に少ないだろうと思っています。AppleScriptに関係するメーリングリストにいくつか参加していますが、そもそもosaxを作っている人が少ないうえに、作っている人もCodeWarriorあたりを使ってCでソースを書いているようです。本誌をお読みの方でも、CompileIt!なんて知らないという方も多いでしょうし、HyperTalkでosaxを作るってどうやっているの?とお思いの方も少なくないでしょう。そこで、実際にHyperTalkでどのようなソースを書くのかといった例をお見せします。

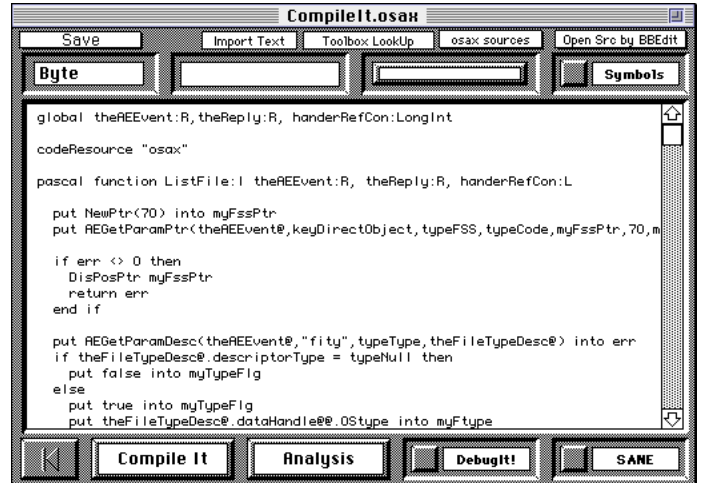
まず、簡単にCompileIt!を紹介しておきます。Royal Software(Heizer Software)が販売しているCompileIt!は、HyperTalkによってXCMDを作ることができるHyperTalkコンパイラです。HyperTalkでスクリプトを書いて、Compile Itボタンを押せば、それがXCMDになります(図9)。

CompileIt!自体がスタックですのでHyperCardだけですべての開発が行えますし(ソースレベルデバッガも備えています)、開発環境を自在にカスタマイズすることも可能

です(普通のスタックですから)。開発環境の特徴として、HyperCardの「普通の」HyperTalkをコンパイルできる以外に、スクリプトの中でToolboxが呼び出せるようになっており、Toolboxを使ったXCMDをHyperTalkだけで作成できるようになっている点をあげることができます。このToolboxをHyperTalkで呼び出せるという点こそが、CompileIt!の最大のメリットだと筆者は思っています。もちろん、HyperTalkというのはHyperCardに即して作られた言語ですから、Toolboxの世界とはなじみにくい点もあります。たとえば、HyperTalkはすべてのデータを文字列として扱うというデータの型のことを考えなくてよい世界ですが、一方、Toolboxの世界は、データの型が厳密に決まっている世界です。こうした世界の違いは、CompileIt!ができるかぎり面倒を見てくれるようになっているのです。もちろん、Toolboxを呼び出すスクリプトを書く時点で、変数などのデータの型は理解しておく必要がありますが、文脈上明らかに型変換が必要な個所ではCompileIt!が型変換を行う処理を自動的に組み込んでくれるなど、HyperTalkにどっぷり使った筆者のような人間にも、比較的簡単にToolboxが扱えるようになっているのです。そして、バージョン2.5以降では、XCMD以外のコードリソース一般が作成可能になっています。

では、具体的に、osaxを作る際に、どのようなソースコードを書くことになるのかをお見せしましょう。サンプルとして、指定したフォルダー内にあるファイルをすべて(サブフォルダーに入っているものも含めて)リストアップするosaxを作るとします。List Filesというコマンド名(イベント名)にします。ダイレクトパラメーターとしてフォルダーのファイル指定(File Specification)を受け取り、そのフォルダーの中に入っているファイルのファイル指定のリストを返す、という仕様になります。また、オプションとして、リストアップするファイルのタイプを指定できるようにもします。つまり、フォルダーからPICTファイルだけをリストアップさせるといった処理を行えるようにします。

本誌をお読みの方にはいまさら説明するほどのことではないかもしれませんが、フォルダー内のファイルをすべてリストアップするには、PBGetCatInfoを、Indexの数値を増やしなが



り返し呼んで、片っ端からファイルを拾い上げていく、ということになります。サブフォルダーの中身をリストアップするためには再帰呼び出しを使うのが普通なのかもしれませんが、筆者は再帰呼び出しが好きではないので、スキャンすべきフォルダーのIDを順番にならべておく変数を作り、フォルダーを見つけるたびにIDをその変数の末尾に追加し、変数の最後のフォルダーに達するまで順に処理を続けるという方法を取りました。

リスト2がCompileIt!でコンパイルするためのHyperTalkのソースです。低レベルのFile Managerを呼ぶときに必要なパラメータ・ブロックは自分でNewPtr()を呼んで確保しなければならないなど、CompileIt!ならでは(?)の処理が入っていますので、今どきの開発環境をお使いの方から見ればごちゃごちゃとしたものに見えるのではないかと思います。でも、ちゃんとHyperTalkでしょ？

```
global theAEEEvent:R,theReply:R, handerRefCon:LongInt

codeResource "osax"

pascal function ListFile:I theAEEEvent:R, theReply:R, handerRefCon:L

-- ダイレクトパラメータの取得
put NewPtr(70) into myFssPtr
put AEGgetParamPtr(theAEEEvent@,keyDirectObject,typeFSS,typeCode,myFssPtr,70,myactSize)
into err
if err <> 0 then
    DisposPtr myFssPtr
    return err
end if

-- オプションのファイルタイプのパラメータの取得
put AEGgetParamDesc(theAEEEvent@,"fity",typeType,theFileTypeDesc@) into err
if theFileTypeDesc@.descriptorType = typeNull then
    put false into myTypeFlg
else
    put true into myTypeFlg
    put theFileTypeDesc@.dataHandle@@.OSType into myFtype
end if
put AEDisposeDesc(theFileTypeDesc@) into err

-- ダイレクトパラメータのファイル指定のチェックのためにPBGetCatInfoを呼ぶ
put NewPtrClear(256) into myPbPtr
put myFssPtr@.IntegerType into myPbPtr@.ioVRefNum
put myFssPtr+2 into aPtr
put aPtr@.LongIntType into myPbPtr@.ioDrDirID
put myFssPtr+6 into myPbPtr@.ioNamePtr

put PBGetCatInfo(myPbPtr) into err
if err <> 0 then
    DisposPtr myPbPtr
    DisposPtr myFssPtr
    return err
end if

-- ファイル指定がフォルダーを指していなければエラーコードを返して終了
put myPbPtr + 30 into aPtr
if not BitTst(aPtr,3) then
    DisposPtr myPbPtr
    DisposPtr myFssPtr
    return -1708
end if
```

```

-- reply 用のリストを作成
put AECreatelist(nil,0,false,theDesList@) into err
if err <> 0 then
  DisposPtr myPbPtr
  DisposPtr myFssPtr
  return err
end if

put NewPtr(4) into digTempPtr
put NewPtr(70) into newFSSPtr
put NewPtr(32) into myNmPtr

-- digHandle にスキャンすべきフォルダーのIDを並べていくことにする
put 1 into DigIndex
put NewHandle(4) into digHandle
put myPbPtr@.ioDrDirID into digHandle@.longIntType[1]

put 0 into Merr

repeat
  -- スキャンするフォルダーの ID を取り出す
  put digHandle@.longIntType[DigIndex] into DirToScan

  -- PBGetCatInfo を呼ぶ際のIndexの初期化
  put 1 into dirIndex

  repeat
    put "" into myNmPtr@.Str255type
    put myNmPtr into myPbPtr@.ioNamePtr
    put dirIndex into myPbPtr@.ioFDirIndex
    put DirToScan into myPbPtr@.ioDrDirID

    put PBGetCatInfo(myPbPtr) into err
    if err <> 0 then
      exit repeat
    end if

    put myPbPtr + 30 into aPtr

    if BitTst(aPtr,3) then

      -- フォルダーを見つけた場合は、そのIDをdigHandleの末尾に追加
      put myPbPtr@.ioDrDirID into digTempPtr@.longIntType
      put PtrAndHand(digTempPtr,digHandle,4) into Merr

    else

      -- オプションでファイルタイプの指定があったときにはタイプが
      -- 一致するかどうかを判定
      if myTypeFlg then
        put myPbPtr + 32 into aPtr
        if (aPtr@.fdType.OStype) <> myFtype then
          add 1 to dirIndex
          next repeat
        end if
      end if

      -- 見つけたファイルの FileSpec をつくって、リストに追加
      put myPbPtr@.ioVRefNum into tgVref
      put myNmPtr@.Str255Type into myFname
      put FSMakeFSSpec(tgVref,DirToScan,myFname,newFSSPtr@) into Merr

      if Merr = 0 then
        put AEPutPtr(theDesList@,0,typeFSS,newFSSPtr,70) into Merr
      end if
    end if
  end repeat
end repeat

```

```

end if

if Merr <> 0 then
  put false into myDigFlg
  exit repeat
end if

add 1 to dirIndex

end repeat

if Merr <> 0 then exit repeat

-- digHandle のすべてのフォルダーをチェックし終わったら抜ける
if GetHandleSize(digHandle) = DigIndex*4 then exit repeat

add 1 to DigIndex

end repeat

-- 使用したメモリの後始末
DisposPtr myNmPtr
DisposPtr myPbPtr
DisposPtr myFssPtr
DisposPtr newFSSPtr
DisposHandle digHandle
DisposPtr digTempPtr

-- 処理の中でエラーが起きていたらエラーメッセージを返して終了
if Merr <> 0 then
  put AEDisposeDesc(theDesList@) into err
  return Merr
end if

-- reply を返す
if theReply@.descriptorType <> typeNull then
  put AEPutParamDesc(theReply@, keyDirectObject,theDesList@) into err
end if

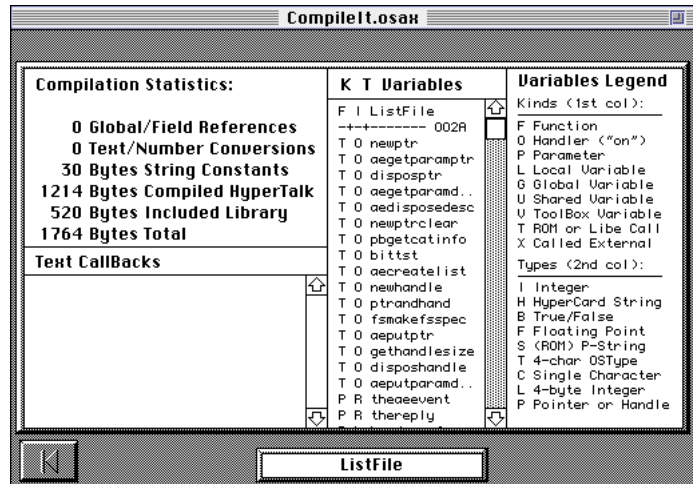
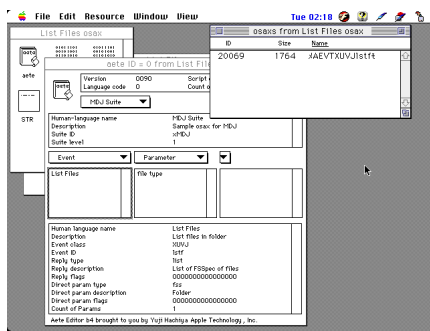
put AEDisposeDesc(theDesList@) into err

return err

end ListFile

```

このソースをコンパイルするとListFileというosaxリソースが作られますので、ResEditでosax用のファイルにコピーし、名前をosaxの規則にしたがって付け直し、aeteリソースに辞書情報を記入すれば、osaxが出来上がるわけです(図10)。コンパイルの際にスクリプトがどのように解釈されたか(データの型など)はAnalysisというカードに記録されますので、これをチェックすればたいいのミスは見つかります(図11)。Toolboxを呼ぶ際のパラメータのミスなどはコンパイル時にエラーになります。



このように、CompileIt!を使えば、HyperTalkでosaxが作れるのです。もっとも、Macでのプログラミングは、どの言語を用いようとも、結局のところ、Toolboxを呼ぶということが中心になります。ですから、Toolboxが呼び出せて、コンパイルしてコードリソースが作れるようになっていれば、HyperTalkでosaxが作れることに何の不思議もないとも言えます。また、XCMDにせよosaxにせよ、ウィンドウやらメニューやら、イベント処理やらのことは考える必要がなく、限定された枠の中でゴリゴリと必要な処理を行うコードさえ書けばよいものですので、ある意味で簡単な世界であるとも思います。必要なToolboxさえ割り出せたらこっちのもの、という感じで、結構気軽にToolboxと付き合える⁴と筆者は感じています。そうして作ったosaxがCGIと結びついて、インターネット上の仕掛けになっていくところが面白くてしょうがない今日この頃です。

終わりに

今回は、HyperTalkを軸にして、HyperCardオタクからCGIオタクへ、という筆者の経過を語ってしまいました。まあ、こういう人間も世の中にはいるということを知っていただければよいかなと思います。

HyperCardが持っている自分の情報を手軽に共有するためのツールという側面は、今やホームページ(Web)が、ある意味で一つの理想的な形(世界中の人々と、機種の違いを越えて共有可能)として実現したとも言えます。筆者は、CGIのデバッグをしながら、ふと、結局、自分はいつも同じことをしてるよなあと感じることがあります。かつて、AppleがHyperCardの傍らに掲げていたキャッチフレーズ、Everyone is an Author!、この周りを衣装を替えながら、それでいてHyperTalkという導きの糸をしっかりと握り締め、ぐるぐると回っているようにも思えるのです。迷宮から抜け出すどころか、ますます深みにはまる一方なんですけどね。

⁴ 者は、いまだに『インサイドマック徹底ガイド』(最新機種としてMac SEとMac II があげられているような古い本です)と、インサイドマック旧6巻(System 7関連の新機能がひとまとまりなので便利なんです)で見当をつけてから、Toolbox Assistantで調べという方法で、Toolboxに取り組んでいます。